

V 部 例

第5部は、本書の各章で説明した概念の例を挙げることに専念します。EJB仕様はベンダに依存しない方法で関心事に対処しますが、実際のEJBコンテナでコードを動作させるには起動やデプロイなどの操作を行うためのベンダ固有のフックを使う必要があります。これ以降は、コードを特定の実装に合わせて調整します。

本文に関するオンラインでの手引きは以下で参照できます。

<http://community.jboss.org/groups/oreilleyjb6th>

JBoss Community が主催しているこのサイトでは、実際にEJB概念をさらに深く理解するために読者はフォーラムでの議論、議題の追跡、記事へのコメントに自由に参加できます。

本書のすべての例は LGPL オープンソースソフトウェアであり、コードを自分で実行しようとするユーザへの手引きになります。これらの例は本書の出版日以降でも通用し、ユーザから求められた説明、新機能、マニュアルの更新、バグ修正に対応しています。

ソースからの構築 要件

| 製品 | バージョン | 必要性 |
|----------------------------|-----------------------|---------------------|
| JDK (Java Development Kit) | 1.6.0+ (ただし 1.7.0 以前) | あり |
| Apache Maven | 2.0.9 以降 | あり |
| Git クライアント | | |
| Eclipse IDE | | |
| m2eclipse | 0.10.0 | Eclipse を使っている場合は推奨 |
| IntelliJ IDEA | | |

Subversion SCM を使ったソースの取得 匿名アクセス

```
$> git clone  
git://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples.git
```

コミッタ

```
$> git clone  
git@github.com:jbossejb3/oreilly-ejb-6thedition-book-examples.git
```

チェックアウト位置への移動

```
$> cd oreilly-ejb-6thedition-book-examples
```

上記のリンクはプロジェクトへの最新（未リリース）の更新を取得します。最新の安定版を取得したければ、tags にある最新バージョンを取得してください。

```
http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples
```

本書のためにリリースされたバージョンは「1.0.0」ですが、その後のリリースには追加機能やバグ修正が含まれ、最新バージョンの EJB コンテナを使うように更新されているでしょう。

ビルドとテスト

ローカルの Maven レポジトリにインストールし、ビルドとテストを実行します。

```
myLocation $> mvn clean install
```

注記

- 最初のビルドでは、必要なすべての依存関係を JBoss Nexus レポジトリからダウンロードするので多少時間がかかる可能性があります。
- すべての例を完全に実行するには、環境固有の設定が必要なプロジェクトもあります。その場合の詳細な情報については、各プロジェクトの説明ページを参照してください。

テストハーネス

EJB の純粋な POJO プログラミングモデルの例を示すために、多くの章には EJB コンテナのスコープ外で動作するテストを使った例があります。この設定では、本物の EJB ではなく通常のオブジェクトインスタンスに含まれるビジネスロジックだけをテストします。このようなテストはベンダに依存することなく迅速に実行することを目的としており、テスト設定では注入などのサービスを手動でモックします。

すべてのテストは優れた JUnit テスティングフレームワーク (<http://www.junit.org/>) を使って記述され

ています。JUnit では、ライフサイクルコールバックやテストメソッドなどを指定するためのアノテーションを追加した簡単なクラスをコーディングできます。JUnit は有名なテスト標準になっており、手動でのテスト設定と比べて効率が改善するでしょう。

どのテストも JUnit 固有の機能に依存していないので (API のみ)、TestNG (<http://testng.org>) などの代替のフレームワークに交換したい場合にはわずかなりファクタリングで実現できるはずですが、どちらにしても、何らかのテストフレームワークを使って、できる限り配管コードの記述を減らし宣言的な振る舞いを活用するというテーマを追い続けることをお勧めします。

基礎をなすテストフレームワークに加え、これらの例では JBoss.org コミュニティ内の 2 つの胸躍る若いプロジェクトを活用しています。これらのプロジェクトはどちらも、テスト可能な統合テストの記述をできるだけ簡単で苦痛の伴わない作業にすることに重点を置いています。

ShrinkWrap

<http://jboss.org/shrinkwrap>

ビルドの省略

ShrinkWrap は、Java の JAR、WAR、EAR などのアーカイブを作成するための簡単な API です。

説明

Java Archives (JAR) や Enterprise Archives (EAR) などのパッケージング仕様は、一連のクラスやリソースを 1 つの単位として宣言するための標準メカニズムです。多くの場合、アーカイブは Web サーバやアプリケーションサーバで直接実行またはデプロイされることを目的としています。アーカイブはその構造に関する暗黙的なメタデータを備え、そこからクラスローディングスコープやマニフェストパラメータを推測します。

アーカイブは便利ではありますが、通常はスクリプトや別のツールを使って開発ライフサイクル内に構築工程を追加する必要があります。ShrinkWrap プロジェクトは、コード内でアーカイブをプログラムで作成するための簡単な API を提供し、オプションで ZIP や Exploded File 形式にエクスポートできます。これにより、クラスパス、ファイルシステム、またはリモート URL に点在するリソースから「仮想」アーカイブをプロトタイピングするのが非常に迅速になります。

本章の例では、デプロイの内容を定義するために主に ShrinkWrap を使います。テストでファイルベースの JAR や EAR を作成するのに完全なビルドを実行する必要はありません。ShrinkWrap アーカイブのデプロイをサポートする EJB コンテナを使うと、IDE のインクリメンタルコンパイルを活用し、変更後に何も作業することなくすぐにテストできます。

ShrinkWrap と対応するコンテナを接続するアダプタは、テストクラスと統合するライブラリです。EJB がサーバサイドビジネスオブジェクトに対する強力なバックエンドとなるのと同様に、Arquillian フレームワークはサービスと EJB コンテナ抽象化を使って POJO テストを強化します。

Arquillian

<http://jboss.org/arquillian>

コンテナ内でのテスト

Arquillian は、アプリケーションコードをリモートや埋め込みのコンテナ内でテストするか、コンテナのクライアントとしてやり取りしてテストするための簡単なメカニズムを提供します。

目的

Arquillian は、Java アプリケーション（たいていはエンタープライズアプリケーション）のさまざまな統合テストのための簡単なテストハネスを開発者に提供します。テストケースはコンテナ内で実行されるか、テストのコードと一緒にデプロイされるか、またはコンテナと連携してデプロイされたコードに対するクライアントとして機能します。

Arquillian は、リモートと埋め込みの 2 種類のコンテナを定義します。リモートコンテナはテスト実行環境とは別の JVM に存在します。リモートコンテナのライフサイクルは Arquillian で管理されるか、または Arquillian がすでに起動しているコンテナと結合します。埋め込みコンテナは同じ JVM に存在し、たいていは Arquillian が管理します。コンテナはその機能によってさらに分類できます。例としては、完全準拠の Java EE アプリケーションサーバ（GlassFish、JBoss AS、埋め込み GlassFish など）、サーブレットコンテナ（Tomcat、Jetty など）、Bean コンテナ（Wslid SE など）があります。Arquillian は、テストに使うコンテナが接続可能であることを保証するので、開発者は独自のテスト環境に縛られません。

Arquillian は、以下のようなテスト環境のすべての側面に対処し、開発者が統合テストを実行する際の負担を最小限にすることを目指しています。

1. コンテナのライフサイクルの管理（起動 / 停止）
2. 依存するクラスやリソースを持つテストクラスのデプロイ可能なアーカイブへの統合
3. テストクラスの補強（@Inject、@EJB、@Resource 注入の解決など）
4. テストのためのアーカイブのデプロイ（デプロイ / アンデプロイ）
5. 結果とエラーの取得

開発者のビルド環境を不必要に複雑にしないために、Arquillian はなじみのあるテストフレームワーク（JUnit 4、TestNG 5 など）と透過的に連携し、既存の IDE、Ant、Maven テストプラグインを使ってアドオンなしにテストを実行できるようにします。

Arquillian は統合テストを簡単にします。

付録 A

FirstEJB の例

A.1 説明

この例では、追加という極めて簡単なビジネスプロセスをモデル化します。主なロジックはコンピュータ科学の学部生レベルの授業で記述するロジックと同じであり、少数のアノテーションを使ってこれらの小さなクラスから本物の EJB を作成します。さらに、この例はビジネスインタフェース、EJB 2.x レガシーコンポーネント、EJB 3.1 の新しいインタフェースなしのビューなどのさまざまな「ビュー」を介して EJB を公開する方法も示します。

このセクションのテストは、単体テストと統合テストの両方を提供します。単体テストでは単にクラスを POJO オブジェクトとしてインスタンス化してビジネスメソッドを直接呼び出しますが、統合テストでは実際のコンテナを使って EJB を作成し、プロキシ参照で呼び出します。これは EJB POJO プログラミングモデルの用途の広さを示しています。

A.2 オンライン上の関連情報

ウィキ記事：<http://community.jboss.org/docs/DOC-15566>

ソースの場所：<http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples/tree/master/ch04-firstejb/>

A.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

A.3.1 実装リソース

A.3.1.1 CalculatorBeanBase.java

```
package org.jboss.ejb3.examples.ch04.firstejb;

import org.jboss.logging.Logger;

/**
 * CalculatorEJB の Bean 実装クラスのベース
 * 必要な規約のためのビジネスロジックを提供する
```

```
*
* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
*/
public class CalculatorBeanBase implements CalculatorCommonBusiness
{
    // -----||
    // クラスメンバー -----||
    // -----||

    /**
     * □ガー
     */

    private static final Logger log = Logger.getLogger(CalculatorBeanBase.class);
    // -----||
    // 必要な実装 -----||
    // -----||

    /**
     * {@inheritDoc}
     * @see org.jboss.ejb3.examples.ch04.firstejb.CalculatorCommonBusiness#add(int[])
     */
    @Override
    public int add(final int... arguments)
    {
        // 初期化
        final StringBuffer sb = new StringBuffer();
        sb.append("Adding arguments: ");
        int result = 0;

        // すべての引数を加算する
        for (final int arg : arguments)
        {
            result += arg;
            sb.append(arg);
            sb.append(" ");
        }

        // 戻す
        log.info(sb.toString());
        log.info("Result: " + result);
        return result;
    }
}
```

A.3.1.2 CalculatorCommonBusiness.java

```

package org.jboss.ejb3.examples.ch04.firstejb;

/**
 * CalculatorEJB のすべてのビジネスインタフェースに
 * 共通する処理の規約を含む
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface CalculatorCommonBusiness
{

    // -----||
    // 規約 -----||
    // -----||
    /**
     * すべての引数を加算する
     *
     * @return すべての引数の合計
     */
    int add(int... arguments);
}

```

A.3.1.3 CalculatorLocal.java

```

package org.jboss.ejb3.examples.ch04.firstejb;

import javax.ejb.EJBLocalObject;

/**
 * CalculatorEJB の EJB 2.x ローカルコンポーネントインタフェース
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface CalculatorLocal extends CalculatorCommonBusiness, EJBLocalObject
{

}

```

A.3.1.4 CalculatorLocalBusiness.java

```

package org.jboss.ejb3.examples.ch04.firstejb;

/**
 * CalculatorEJB のローカルビジネスインタフェース
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>

```

```
*/
public interface CalculatorLocalBusiness extends CalculatorCommonBusiness
{
}
}
```

A.3.1.5 CalculatorLocalHome.java

```
package org.jboss.ejb3.examples.ch04.firstejb;

import javax.ejb.CreateException;
import javax.ejb.EJBLocalHome;

/**
 * CalculatorEJB の EJB 2.x ローカルホーム
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface CalculatorLocalHome extends EJBLocalHome
{
    // -----||
    // create<METHOD> メソッド -----||
    // -----||

    /**
     * CalculatorEJB のローカルコンポーネントビューへの参照を返す
     */
    CalculatorLocal create() throws CreateException;
}
}
```

A.3.1.6 CalculatorRemote.java

```
package org.jboss.ejb3.examples.ch04.firstejb;

import javax.ejb.EJBObject;

/**
 * CalculatorEJB の EJB 2.x リモートコンポーネントインタフェース
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface CalculatorRemote extends CalculatorCommonBusiness, EJBObject
{
}
}
```

A.3.1.7 CalculatorRemoteBusiness.java

```
package org.jboss.ejb3.examples.ch04.firstejb;

/**
 * CalculatorEJB のリモートビジネスインタフェース
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface CalculatorRemoteBusiness extends CalculatorCommonBusiness
{

}
```

A.3.1.8 CalculatorRemoteHome.java

```
package org.jboss.ejb3.examples.ch04.firstejb;

import java.rmi.RemoteException;

import javax.ejb.CreateException;
import javax.ejb.EJBHome;

/**
 * CalculatorEJB の EJB 2.x リモートホーム
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface CalculatorRemoteHome extends EJBHome
{
    // -----||
    // create<METHOD> メソッド -----||
    // -----||

    /**
     * CalculatorEJB のリモートコンポーネントビューへの参照を返す
     */
    CalculatorRemote create() throws CreateException, RemoteException;
}
```

A.3.1.9 ManyViewCalculatorBean.java

```
package org.jboss.ejb3.examples.ch04.firstejb;

import javax.ejb.Local;
import javax.ejb.LocalBean;
import javax.ejb.LocalHome;
import javax.ejb.Remote;
```

```
import javax.ejb.RemoteHome;
import javax.ejb.Stateless;

/**
 * ローカルとリモートのビジネスおよびコンポーネントビューと、
 * EJB 3.1 のインタフェースなしのビューを公開する
 * CalculatorEJB の Bean 実装クラス
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@Stateless
@Local(CalculatorLocalBusiness.class)
@Remote(CalculatorRemoteBusiness.class)
@LocalHome(CalculatorLocalHome.class)
@RemoteHome(CalculatorRemoteHome.class)
@LocalBean // No-interface view
public class ManyViewCalculatorBean extends CalculatorBeanBase implements CalculatorCommonBusiness
{
    /**
     * 共通ベースクラスで提供される実装
     */
}
```

A.3.1.10 NoInterfaceViewCalculatorBean.java

```
package org.jboss.ejb3.examples.ch04.firstejb;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;

/**
 * インタフェースなしのビューを持つ
 * CalculatorEJB の Bean 実装クラス
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@Stateless
@LocalBean
public class NoInterfaceViewCalculatorBean extends CalculatorBeanBase
{
    // ベースクラスでの実装
}
```

A.3.1.11 SimpleCalculatorBean.java

```

package org.jboss.ejb3.examples.ch04.firstejb;

import javax.ejb.Local;
import javax.ejb.Stateless;

/**
 * ローカルビジネスビューを公開する
 * CalculatorEJB の Bean 実装クラス
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@Stateless
@Local(CalculatorLocalBusiness.class)
public class SimpleCalculatorBean extends CalculatorBeanBase implements CalculatorCommonBusiness
{
    /**
     * 共通ベースクラスで提供される実装
     */
}

```

A.3.2 テストリソース

A.3.2.1 CalculatorAssertionDelegate.java

```

package org.jboss.ejb3.examples.ch04.firstejb;

import junit.framework.TestCase;

import org.jboss.logging.Logger;

/**
 * {@link CalculatorCommonBusiness} の実装が
 * 期待どおりに機能していることを証明する
 * 関数を含む
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
class CalculatorAssertionDelegate
{
    // -----||
    // クラスメンバー -----||
    // -----||

    /**
     * ログー
     */
}

```

```
private static final Logger log = Logger.getLogger(CalculatorAssertionDelegate.class);

// -----||
// 機能メソッド -----||
// -----||

/**
 * 提供された Calculator インスタンスを使って
 * 加算アルゴリズムをテストする
 */
void assertAdditionSucceeds(final CalculatorCommonBusiness calc)
{
    // 初期化
    final int[] arguments = new int[]
    {2, 3, 5};
    final int expectedSum = 10;

    // 加算
    final int actualSum = calc.add(arguments);

    // テスト
    TestCase.assertEquals("加算が、期待した結果を返しませんでした ", expectedSum, actualSum);

    // ログに記録
    final StringBuffer sb = new StringBuffer();
    sb.append("Obtained expected result, ");
    sb.append(actualSum);
    sb.append(", from arguments: ");
    for (final int arg : arguments)
    {
        sb.append(arg);
        sb.append(" ");
    }
    log.info(sb.toString());
}
}
```

A.3.2.2 CalculatorIntegrationTestCase.java

```
package org.jboss.ejb3.examples.ch04.firstejb;

import java.net.MalformedURLException;

import javax.ejb.EJB;

import org.jboss.arquillian.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
```

```

import org.jboss.logging.Logger;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.runner.RunWith;

/**
 * 1つのビジネスビューを公開する
 * CalculatorEJBの統合テスト
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@RunWith(Arquillian.class)
public class CalculatorIntegrationTestCase
{
    // -----||
    // クラスメンバー -----||
    // -----||

    /**
     * □ガー
     */
    private static final Logger log = Logger.getLogger(CalculatorIntegrationTestCase.class);

    /**
     * CalculatorEJBのEJB 3.x ローカルビジネスビュー
     */
    @EJB
    private static CalculatorLocalBusiness calcLocalBusiness;

    /**
     * 取得したCalculatorが期待どおりに機能することを確認するためのデリゲート
     */
    private static CalculatorAssertionDelegate assertionDelegate;

    /**
     * デプロイの定義
     */
    @Deployment
    public static JavaArchive createDeployment() throws MalformedURLException
    {
        final JavaArchive archive = ShrinkWrap.create("firstejb.jar", JavaArchive.class).
addPackage(CalculatorBeanBase.class.getPackage());
        log.info(archive.toString(true));
        return archive;
    }
}

```

```

    }

    // -----||
    // ライフサイクルメソッド -----||
    // -----||

    /**
     * テストの前に 1 回動作する
     */
    @BeforeClass
    public static void beforeClass() throws Throwable
    {
        // アサーションデリゲートを作成する
        assertionDelegate = new CalculatorAssertionDelegate();
    }

    // -----||
    // テスト -----||
    // -----||

    /**
     * EJB 3.x ビジネスビューを使って、
     * CalculatorEJB が期待どおりに加算することを確認する
     */
    @Test
    public void testAdditionUsingBusinessReference() throws Throwable
    {
        // テスト
        log.info("Testing EJB via business reference...");
        assertionDelegate.assertAdditionSucceeds(calcLocalBusiness);
    }
}

```

A.3.2.3 CalculatorUnitTestCase.java

```

package org.jboss.ejb3.examples.ch04.firstejb;

import junit.framework.TestCase;

import org.jboss.logging.Logger;
import org.junit.BeforeClass;
import org.junit.Test;

/**
 * CalculatorEJB のビジネスメソッドが期待どおりに機能していることを
 * 確認するためのテスト
 */

```

```

* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
*/
public class CalculatorUnitTestCase
{
    // -----||
    // クラスメンバー -----||
    // -----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(CalculatorUnitTestCase.class);

    /**
     * テストする POJO インスタンス
     */
    private static CalculatorCommonBusiness calc;

    // -----||
    // ライフサイクルメソッド -----||
    // -----||

    @BeforeClass
    public static void beforeClass()
    {
        // CalculatorCommonBusiness 規約に従う
        // POJO インスタンスを生成する
        calc = new SimpleCalculatorBean();
    }

    // -----||
    // テスト -----||
    // -----||

    /**
     * CalculatorEJB の背後にあるビジネスロジックを
     * 純粋な POJO として使ったときに
     * 期待どおりに機能することを確認する
     */
    @Test
    public void testAddition()
    {
        // 初期化
        final int[] arguments = new int[]
        {3, 7, 2};
        final int expectedSum = 12;
    }
}

```

```

// 加算
final int actualSum = calc.add(arguments);

// テスト
TestCase.assertEquals(" 加算が、期待した結果を返しませんでした ", expectedSum, actualSum);

// ログに記録
final StringBuffer sb = new StringBuffer();
sb.append("Obtained expected result, ");
sb.append(actualSum);
sb.append(", from arguments: ");
for (final int arg : arguments)
{
    sb.append(arg);
    sb.append(" ");
}
log.info(sb.toString());
}
}

```

A.3.2.4 MultiViewCalculatorIntegrationTestCase.java

```

package org.jboss.ejb3.examples.ch04.firstejb;

import java.net.MalformedURLException;

import javax.naming.Context;
import javax.naming.InitialContext;

import org.jboss.arquillian.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.logging.Logger;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.runner.RunWith;

/**
 * CalculatorEJB の多くのビューをテストする統合テスト
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@RunWith(Arquillian.class)
public class MultiViewCalculatorIntegrationTestCase
{
    // -----||

```

```
// クラスメンバー -----||
// -----||

/**
 * ロガー
 */
private static final Logger log = Logger.getLogger(MultiViewCalculatorIntegrationTestCase.class);

/**
 * JNDI ネーミングコンテキスト
 */
private static Context namingContext;

/**
 * CalculatorEJB の EJB 3.x ローカルビジネスビュー
 */
private static CalculatorLocalBusiness calcLocalBusiness;

/**
 * CalculatorEJB の EJB 2.x ローカルコンポーネントビュー
 */
private static CalculatorLocal calcLocal;

/**
 * 取得した Calculator が期待どおりに機能することを確認するためのデリゲート
 */
private static CalculatorAssertionDelegate assertionDelegate;

/**
 * ローカルビジネス参照の JNDI 名
 */
// グローバル JNDI 構文を使うための用意
private static final String JNDI_NAME_CALC_LOCAL_BUSINESS = ManyViewCalculatorBean.class.
getSimpleName() + "Local";

/**
 * ローカルホーム参照の JNDI 名
 */
// グローバル JNDI 構文を使うための用意
private static final String JNDI_NAME_CALC_REMOTE_HOME = ManyViewCalculatorBean.class.
getSimpleName() + "LocalHome";

/**
 * デプロイの定義
 */
@Deployment
```

```

public static JavaArchive createDeployment() throws MalformedURLException
{
    final JavaArchive archive = ShrinkWrap.create("firstejb.jar", JavaArchive.class).addPackage(
        CalculatorBeanBase.class.getPackage());
    log.info(archive.toString(true));
    return archive;
}

// -----||
// ライフサイクルメソッド -----||
// -----||

/**
 * テストの前に 1 回動作する
 */
@BeforeClass
public static void beforeClass() throws Throwable
{
    // CP の jndi.properties を使ってネーミングコンテキストを作成
    namingContext = new InitialContext();

    // EJB 3.x ビジネス参照を取得
    calcLocalBusiness = (CalculatorLocalBusiness)
        namingContext.lookup(JNDI_NAME_CALC_LOCAL_BUSINESS);

    // アサーションデリゲートを作成
    assertionDelegate = new CalculatorAssertionDelegate();

    // ホームを介して EJB 2.x コンポーネント参照を取得
    final Object calcLocalHomeReference = namingContext.lookup(JNDI_NAME_CALC_REMOTE_HOME);
    final CalculatorLocalHome calcRemoteHome = (CalculatorLocalHome) calcLocalHomeReference;
    calcLocal = calcRemoteHome.create();
}

// -----||
// テスト -----||
// -----||

/**
 * EJB 3.x ビジネスビューを使って、
 * CalculatorEJB が期待どおりに加算することを確認
 */
@Test
public void testAdditionUsingBusinessReference() throws Throwable
{
    // テスト

```

```
        log.info("Testing remote business reference...");
        assertionDelegate.assertAdditionSucceeds(calcLocalBusiness);
    }

    /**
     * EJB 2.x コンポーネントビューを使って、
     * CalculatorEJB が期待どおりに加算することを確認
     */
    @Test
    public void testAdditionUsingComponentReference() throws Throwable
    {
        // テスト
        log.info("Testing remote component reference...");
        assertionDelegate.assertAdditionSucceeds(calcLocal);
    }
}
```

A.3.2.5 jndi.properties

OpenEJB ネーミングプロバイダとのローカルのやり取りのための JNDI プロパティ

```
java.naming.factory.initial=org.apache.openejb.client.LocalInitialContextFactory
```


付録 B

ステートレスセッション EJB : 暗号化の例

B.1 説明

クライアント呼び出し間のサーバ側の状態を取得する必要のないビジネスプロセスには、ステートレスセッション Bean 実装を選択するのが最適です。要求を処理するのに必要なすべての情報を要求そのものを含めることができれば、SLSB は非常に効率的なエンドポイントとして使えます。この例では、一方向キャッシングと暗号化 / 復号処理の両方を実行できる暗号化サービスを構築します。

さらに、ここでは要求の間にクライアントに漏洩しなければ SLSB が実は独自の内部状態を持てることも示します。最後に、XML で環境エンティティを使い、サービスの設定可能な要素を外部化する方法を紹介します。

B.2 オンライン上の関連情報

ウィキ記事 : <http://community.jboss.org/docs/DOC-15567>

ソースの場所 : <http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples/tree/master/ch05-encryption/>

B.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

B.3.1 実装リソース

B.3.1.1 EncryptionBean.java

```
package org.jboss.ejb3.examples.ch05.encryption;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.spec.AlgorithmParameterSpec;
import java.security.spec.KeySpec;

import javax.annotation.PostConstruct;
```

```
import javax.annotation.Resource;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.PBEParameterSpec;
import javax.ejb.Local;
import javax.ejb.Remote;
import javax.ejb.SessionContext;
import javax.ejb.Stateless;

import org.apache.commons.codec.binary.Base64;
import org.jboss.logging.Logger;

/**
 * EncryptionEJB の Bean 実装クラス。ライフサイクルコールバック
 * の実装方法 (@PostConstruct) と、
 * 外部化した環境エントリを取得する 2 つの方法
 * を示す。
 *
 * @author <a href="mailto:alr@jboss.org">ALR</a>
 */
@Stateless(name = EncryptionBean.EJB_NAME)
@Local(EncryptionLocalBusiness.class)
@Remote(EncryptionRemoteBusiness.class)
public class EncryptionBean implements EncryptionLocalBusiness, EncryptionRemote
Business
{
    // -----||
    // クラスメンバー -----||
    // -----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(EncryptionBean.class);

    /**
     * この EJB に付けた名前は対応する META-INF/ejb-jar.xml ファイル
     * で参照される
     */
    static final String EJB_NAME = "EncryptionEJB";

    /**
     * ejb-jar.xml で提供される暗号パスフレーズを表す
     * 環境エントリの名前

```

```

*/
private static final String ENV_ENTRY_NAME_CIPHERS_PASSPHRASE = "ciphersPassphrase";

/**
 * ejb-jar.xml で提供されるメッセージダイジェストアルゴリズム
 * を表す環境エントリの名前
 */
private static final String ENV_ENTRY_NAME_MESSAGE_DIGEST_ALGORITHM = "messageDigestAlgorithm";

/**
 * 一方方向ハッシングのダイジェストが使うデフォルトアルゴリズム
 */
private static final String DEFAULT_ALGORITHM_MESSAGE_DIGEST = "MD5";

/**
 * 文字列とバイト表現間の符号化 / 復号に使う文字セット
 */
private static final String CHARSET = "UTF-8";

/**
 * 対称暗号化の暗号キーに使うデフォルトアルゴリズム
 */
private static final String DEFAULT_ALGORITHM_CIPHER = "PBESWithMD5AndDES";

/**
 * 対称暗号化 / 復号のためのデフォルトパスフレーズ
 */
private static final String DEFAULT_PASSPHRASE = "LocalTestingPassphrase";

/**
 * 対称暗号化 / 復号に使うソルト
 */
private static final byte[] DEFAULT_SALT_CIPHERS = {(byte) 0xB4, (byte) 0xA2,
    (byte) 0x43, (byte) 0x89, 0x3E, (byte) 0xC5, (byte)0x78, (byte) 0x53};

/**
 * 対称暗号化 / 復号に使う反復回数
 */
private static final int DEFAULT_ITERATION_COUNT_CIPHERS = 20;

// -----||
// インスタンスメンバー -----||
// -----||

/*
 * 以下のメンバーがサービスの内部状態を表す。

```

```
* これはエンドユーザ API を介して漏洩しないため、
* 「内部状態」の一部であって「会話状態」ではないこと
* に注意する。
*/

/**
 * この EJB の SessionContext : @Resource が付いているので
 * EJB コンテナによって注入される。
 */
@Resource
private SessionContext context;

/**
 * 暗号化処理で鍵として使うパスフレーズ
 * SessionContext.lookup で遅延初期化ロード
 */
private String ciphersPassphrase;

/**
 * メッセージダイジェスト (ハッシュ) 処理に使うアルゴリズム
 * env-entry 名に等しい name プロパティを持つ @Resource アノテーションで注入
 */
@Resource(name = ENV_ENTRY_NAME_MESSAGE_DIGEST_ALGORITHM)
private String messageDigestAlgorithm;

/**
 * 一方方向ハッシングで使うダイジェスト
 */
private MessageDigest messageDigest;

/**
 * 対称暗号化に使う暗号
 */
private Cipher encryptionCipher;

/**
 * 対称復号に使う暗号
 */
private Cipher decryptionCipher;

// -----||
// ライフサイクル -----||
// -----||

/**
 * 要求に対処する前にこのサービスを初期化
```

```

*
* @throws Exception 予期せぬエラーが発生した場合
*/
@PostConstruct
public void initialize() throws Exception
{
    // 現在の状況をログに記録
    log.info(" 初期化中、" + PostConstruct.class.getName() + " のライフサイクルの一部 ");

    /*
    * 対称暗号化
    */

    // 暗号を初期化するのに使うパラメータを取得
    final String cipherAlgorithm = DEFAULT_ALGORITHM_CIPHER;
    final byte[] ciphersSalt = DEFAULT_SALT_CIPHERS;
    final int ciphersIterationCount = DEFAULT_ITERATION_COUNT_CIPHERS;
    final String ciphersPassphrase = this.getCiphersPassphrase();

    // 暗号の鍵とパラメータ仕様を取得する
    final KeySpec ciphersKeySpec = new PBEKeySpec(ciphersPassphrase.toCharArray(), ciphersSalt,
        ciphersIterationCount);
    final SecretKey ciphersKey = SecretKeyFactory.getInstance(cipherAlgorithm).
        generateSecret(ciphersKeySpec);
    final AlgorithmParameterSpec paramSpec = new PBEPParameterSpec(ciphersSalt,
        ciphersIterationCount);

    // 暗号を作成し初期化する
    this.encryptionCipher = Cipher.getInstance(ciphersKey.getAlgorithm());
    this.decryptionCipher = Cipher.getInstance(ciphersKey.getAlgorithm());
    encryptionCipher.init(Cipher.ENCRYPT_MODE, ciphersKey, paramSpec);
    decryptionCipher.init(Cipher.DECRYPT_MODE, ciphersKey, paramSpec);

    // ログに記録
    log.info("Initialized encryption cipher: " + this.encryptionCipher);
    log.info("Initialized decryption cipher: " + this.decryptionCipher);

    /*
    * 一方方向ハッシング
    */

    // メッセージダイジェストのアルゴリズムを取得する
    final String messageDigestAlgorithm = this.getMessageDigestAlgorithm();

    // メッセージダイジェストを生成する
    try

```

```
    {
        this.messageDigest = MessageDigest.getInstance(messageDigestAlgorithm);
    }
    catch (NoSuchAlgorithmException e)
    {
        throw new RuntimeException("Could not obtain the " + MessageDigest.class.getSimpleName() + "
for algorithm: " + messageDigestAlgorithm, e);
    }
    log.info("Initialized MessageDigest for one-way hashing: " + this.messageDigest);
}

// -----||
// 必要な実装 -----||
// -----||

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch05.encryption.EncryptionCommonBusiness#compare(java.lang.String,
 * java.lang.String)
 */
@Override
public boolean compare(final String hash, final String input)
    throws IllegalArgumentException, EncryptionException
{
    // 前提条件を確認
    if (hash == null)
    {
        throw new IllegalArgumentException("hash is required.");
    }
    if (input == null)
    {
        throw new IllegalArgumentException("Input is required.");
    }

    // 提供された入力のハッシュを取得する
    final String hashOfInput = this.hash(input);

    // 等しいかどうかを判断する
    final boolean equal = hash.equals(hashOfInput);

    // 戻す
    return equal;
}

/**
 * {@inheritDoc}
```

```
* @see org.jboss.ejb3.examples.ch05.encryption.EncryptionCommonBusiness#decrypt(java.lang.String)
*/
@Override
public String decrypt(final String input) throws IllegalArgumentException, IllegalStateException,
    EncryptionException
{
    // 暗号を取得する
    final Cipher cipher = this.decryptionCipher;
    if (cipher == null)
    {
        throw new IllegalStateException("Decryption cipher not available, has this service been
            initialized?");
    }

    // 暗号化を実行する
    byte[] resultBytes = null;;
    try
    {
        final byte[] inputBytes = this.stringToByteArray(input);
        resultBytes = cipher.doFinal(Base64.decodeBase64(inputBytes));
    }
    catch (final Throwable t)
        throw new EncryptionException("Error in decryption", t);
    }
    final String result = this.byteArrayToString(resultBytes);

    // ログに記録
    log.info("Decryption on ¥" + input + "¥": " + result);

    // 戻す
    return result;
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch05.encryption.EncryptionCommonBusiness#encrypt(java.lang.String)
 */
@Override
public String encrypt(final String input) throws IllegalArgumentException, EncryptionException
{
    // 暗号を取得する
    final Cipher cipher = this.encryptionCipher;
    if (cipher == null)
    {
        throw new IllegalStateException("Encryption cipher not available, has this service been
            initialized?");
    }
}
```

```
    }

    // 文字列からバイトを取得する
    byte[] inputBytes = this.stringToArray(input);

    // 暗号化を実行する
    byte[] resultBytes = null;
    try
    {
        resultBytes = Base64.encodeBase64(cipher.doFinal(inputBytes));
    }
    catch (final Throwable t)
    {
        throw new EncryptionException("Error in encryption of: " + input, t);
    }
}

// ログに記録
log.info("Encryption on ¥" + input + "¥": " + this.byteArrayToString(resultBytes));

// 戻す
final String result = this.byteArrayToString(resultBytes);
return result;
}

/**
 * 注 :
 *
 * これは不十分な実装だが、この例には十分である。
 * 実際の世界を考えるなら、少なくとも以下が必要である。
 *
 * 1) ランダムなソルトを取り入れ、ハッシュ結果と共に格納する
 * 2) N 回再ハッシュするための反復カウントを追加で実装する
 */
/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch05.encryption.EncryptionCommonBusiness#hash(java.lang.String)
 */
@Override
public String hash(final String input) throws IllegalArgumentException, EncryptionException
{
    // 前提条件を確認
    if (input == null)
    {
        throw new IllegalArgumentException("Input is required.");
    }

    // 入力からバイトを取得する
```

```

byte[] inputBytes = this.stringToByteArray(input);

// メッセージダイジェストを取得する
final MessageDigest digest = this.messageDigest;

// 入力を使って更新し、ハッシュを取得してメッセージダイジェストを設定しなおす。
digest.update(inputBytes, 0, inputBytes.length);
final byte[] hashBytes = digest.digest();
final byte[] encodedBytes = Base64.encodeBase64(hashBytes);

// 入力を適切な形式に戻す
final String hash = this.byteArrayToString(encodedBytes);
log.info("One-way hash of ¥" + input + "¥": " + hash);

// 戻す
return hash;
}

/**
 * 暗号のパスフレーズをサーバ上の安全な場所で定義できるように
 * パスフレーズの取得方法をオーバーライドする。これで運用システムは
 * 符号化に開発サーバとは異なるキーを使うようになり、
 * プログラムが開発中にデフォルトのパスフレーズを
 * 透過的に使えるようにしたまま、セキュリティ侵害が起こる可能性を
 * 抑制できる
 *
 * env-entry として提供されない場合は、デフォルトを使う。
 *
 * なお、実際のシステムはこのメソッドをパブリック API で決して公開しない。
 * ここではテストと例示のために公開している。
 *
 * @see org.jboss.ejb3.examples.ch05.encryption.EncryptionBeanBase#getCiphersPassphrase()
 */
@Override
public String getCiphersPassphrase()
{
    // 現在のパスフレーズを取得する
    String passphrase = this.ciphersPassphrase;

    // 設定されていない場合
    if (passphrase == null)
    {

        // SessionContext を介してルックアップを行う
        passphrase = this.getEnvironmentEntryAsString(ENV_ENTRY_NAME_CIPHERS_PASSPHRASE);
    }
}

```

```
// 提供されているかどうかを確認する
if (passphrase == null)
{
    // 警告をログに記録
    log.warn("No encryption passphrase has been supplied explicitly via " + "an env-entry,
            falling back on the default...");

    // 設定
    passphrase = DEFAULT_PASSPHRASE;
}

// このパスフレーズを使うように設定するので、再び遅延初期化を行う必要はない
this.ciphersPassphrase = passphrase;
}

// 安全なシステムではこれをログに記録しない
log.info("Using encryption passphrase for ciphers keys: " + passphrase);

// 戻す
return passphrase;
}

/**
 * ejb-jar.xml で定義された env-entry 要素から注入されたメッセージアルゴリズム
 * を取得する。指定されていない場合は、デフォルトを使い、
 * 警告メッセージをログに記録する
 *
 * @see org.jboss.ejb3.examples.ch05.encryption.EncryptionRemoteBusiness#getMessageDigestAlgorithm()
 */
@Override
public String getMessageDigestAlgorithm()
{
    // まず注入 / 設定されているかどうかを確認する
    if (this.messageDigestAlgorithm == null)
    {
        // 警告をログに記録
        log.warn("No message digest algorithm has been supplied explicitly via " + "an env-entry,
                falling back on the default...");

        // 設定
        this.messageDigestAlgorithm = DEFAULT_ALGORITHM_MESSAGE_DIGEST;
    }

    // ログに記録
```

```

log.info("Configured MessageDigest one-way hash algorithm is: " + this.messageDigestAlgorithm);

// 戻す
return this.messageDigestAlgorithm;
}

// -----||
// 内部ヘルパーメソッド -----||
// -----||

/**
 * 指定された名前を持つ環境エントリを取得して文字列にキャストし、
 * その結果を返す。エントリが文字列にキャストできない場合は、
 * {@link IllegalStateException} が発行される。指定された環境エントリが
 * 見つからない場合は、警告メッセージをログに記録して
 * null を返す。
 *
 * @param envEntryName
 * @return
 * @throws IllegalStateException
 */
private String getEnvironmentEntryAsString(final String envEntryName) throws IllegalStateException
{
    // SessionContext があるかどうかを確認する
    final SessionContext context = this.context;
    if (context == null)
    {
        log.warn("No SessionContext, bypassing request to obtain environment entry: " + envEntryName);
        return null;
    }

    // 注入された SessionContext を使ってプライベート JNDI ENC をルックアップする
    Object lookupValue = null;
    try
    {
        lookupValue = context.lookup(envEntryName);
        log.debug("取得した環境エントリ「" + envEntryName + "」: " + lookupValue);
    }
    catch (final IllegalArgumentException iae)
    {
        // この EJB のコンポーネント環境内の定義が見つからないので、
        // null を返して呼び出し側に対処させる
        log.warn("次の名前の環境エントリが見つかりません: " + envEntryName);
        return null;
    }
}

```

```
// キャスト
String returnValue = null;
try
{
    returnValue = String.class.cast(lookupValue);
}
catch (final ClassCastException cce)
{
    throw new IllegalStateException(" 指定された環境エントリ " +
        lookupValue + " を " + String.class.getName() +
        " として表現できません ", cce);
}

// 戻す
return returnValue;
}

/**
 * {@link EncryptionBeanBase#getCharset()} の文字セットを使って
 * 指定されたバイト配列の文字列表現を返す。
 * 無効な文字セットを使った結果の {@link UnsupportedEncodingException}
 * を {@link RuntimeException} にラップする。
 *
 * @param bytes
 * @return
 * @throws RuntimeException 文字セットが無効であるか、その他の未知のエラーが発生した場合
 * @throws IllegalArgumentException バイト配列が指定されていない場合
 */
private String byteArrayToString(final byte[] bytes)
    throws RuntimeException, IllegalArgumentException
{
    // 前提条件を確認
    if (bytes == null)
    {
        throw new IllegalArgumentException("Byte array is required.");
    }

    // 文字列として表現する
    String result = null;
    final String charset = this.getCharset();
    try
    {
        result = new String(bytes, charset);
    }
    catch (final UnsupportedEncodingException e)
    {

```

```
        throw new RuntimeException("Specified charset is invalid: " + charset, e);
    }

    // 戻す
    return result;
}

/**
 * {@link EncryptionBeanBase#getCharset()} の文字セットを使って
 * 指定された文字列のバイト配列表現を返す。
 * 無効な文字セットを使った結果の {@link UnsupportedEncodingException}
 * を {@link RuntimeException} にラップする。
 *
 * @param input
 * @return
 * @throws RuntimeException 文字セットが無効であるか、その他の未知のエラーが発生した場合
 * @throws IllegalArgumentException 入力が指定されていない場合 (null)
 */
private byte[] stringToByteArray(final String input)
    throws RuntimeException, IllegalArgumentException
{
    // 前提条件を確認
    if (input == null)
    {
        throw new IllegalArgumentException("Input is required.");
    }

    // バイト配列として表現する
    byte[] result = null;
    final String charset = this.getCharset();
    try
    {
        result = input.getBytes(charset);
    }
    catch (final UnsupportedEncodingException e)
    {
        throw new RuntimeException("Specified charset is invalid: " + charset, e);
    }

    // 戻す
    return result;
}

/**
 * 文字列とバイト表現間の符号化 / 復号に使う
 * 文字セットを取得する

```

```

*
* @return 文字セット
*/
private String getCharset()
{
    return CHARSET;
}
}

```

B.3.1.2 EncryptionCommonBusiness.java

```

package org.jboss.ejb3.examples.ch05.encryption;

/**
 * EncryptionEJB のすべてのビジネスインタフェースに共通する処理
 * のための規約を含む
 *
 * @author <a href="mailto:alr@jboss.org">ALR</a>
 */
public interface EncryptionCommonBusiness
{
    // -----||
    // 規約 -----||
    // -----||

    /**
     * 指定された文字列を暗号化し、その結果を返す
     *
     * @param input
     * @return
     * @throws IllegalArgumentException 入力提供されなかった場合 (null)
     * @throws EncryptionException 暗号化で問題が発生した場合
     */
    String encrypt(String input) throws IllegalArgumentException, EncryptionException;

    /**
     * 指定された文字列を復号し、その結果を返す。一般的な規約として、
     * {@link EncryptionCommonBusiness#encrypt(String)} で暗号化された
     * 文字列の復号の結果は、元の入力に対する値と
     * 同じになる (ラウンドトリップ)
     *
     * @param input
     * @return
     * @throws IllegalArgumentException 入力提供されなかった場合 (null)
     * @throws EncryptionException 復号で問題が発生した場合
     */
    String decrypt(String input) throws IllegalArgumentException, EncryptionException;
}

```

```
/**
 * 指定された引数の一方方向ハッシュを返す。
 * パスワードを安全に格納するのに役立つ。
 *
 * @param input
 * @return
 * @throws IllegalArgumentException 入力提供されなかった場合 (null)
 * @throws EncryptionException ハッシュの作成で問題が発生した場合
 */
String hash(String input) throws IllegalArgumentException, EncryptionException;

/**
 * 指定された入力が指定されたハッシュと一致するかどうかを返す。
 * 安全に格納されたハッシュでパスワードを
 * 検証するのに役立つ。
 *
 * @param hash
 * @param input
 * @return
 * @throws IllegalArgumentException ハッシュまたは入力提供されなかった場合 (null)
 * @throws EncryptionException ハッシュの作成で問題が発生した場合
 */
boolean compare(String hash, String input) throws IllegalArgumentException, EncryptionException;

/*
 * このコメントはこれ以降のすべてに適用される
 *
 * 実際の世界では、このような内部メソッドを公開するのは
 * セキュリティリスクであるが、テストのためと
 * 例で提供されている機能を示すためにここに配置する。
 */

/**
 * 対称暗号化 / 復号の鍵に使う
 * パスフレーズを取得する
 *
 * @return
 */
String getCiphersPassphrase();

/**
 * 一方方向ハッシングに使う
 * アルゴリズムを取得する
 *
 * @return
 */
```

```
String getMessageDigestAlgorithm();
}
```

B.3.1.3 EncryptionException.java

```
package org.jboss.ejb3.examples.ch05.encryption;

import javax.ejb.ApplicationException;

/**
 * 暗号化処理での予期せぬ問題を示す
 * 検査アプリケーション例外
 *
 * @author <a href="mailto:alr@jboss.org">ALR</a>
 */
@ApplicationException
// 明示的なアノテーション。ただし、Exception を拡張しているので、
// デフォルトでアプリケーション例外と推論される。
public class EncryptionException extends Exception
{
    // -----||
    // クラスメンバー -----||
    // -----||

    /**
     * JVM に対する明示的なシリアライゼーションヒントを満たすため
     */
    private static final long serialVersionUID = 1L;

    // -----||
    // コンストラクタ -----||
    // -----||

    /**
     * すべてのコンストラクタはスーパークラス実装に委譲する
     */

    public EncryptionException()
    {
        super();
    }

    public EncryptionException(String message, Throwable cause)
    {
        super(message, cause);
    }
}
```

```

public EncryptionException(String message)
{
    super(message);
}

public EncryptionException(Throwable cause)
{
    super(cause);
}
}

```

B.3.1.4 EncryptionLocalBusiness.java

```

package org.jboss.ejb3.examples.ch05.encryption;

/**
 * EncryptionEJB の EJB 3.x ローカルビジネスビユー
 *
 * @author <a href="mailto:alr@jboss.org">ALR</a>
 */
public interface EncryptionLocalBusiness extends EncryptionCommonBusiness
{
    // 階層内の規約
}

```

B.3.1.5 EncryptionRemoteBusiness.java

```

package org.jboss.ejb3.examples.ch05.encryption;

/**
 * EncryptionEJB の EJB 3.x リモートビジネスビユー
 *
 * @author <a href="mailto:alr@jboss.org">ALR</a>
 */
public interface EncryptionRemoteBusiness extends EncryptionCommonBusiness
{
    // 階層内の規約
}

```

B.3.1.6 META-INF/ejb-jar.xml

```

<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/ejb-jar_3_1.xsd"
    version="3.1">

    <enterprise-beans>

```

```

<!--
    この部分では、EncryptionEJB にメタデータを追加し、
    アノテーションで定義された情報を
    補足
-->
<session>

    <!--
        これは Bean 実装クラスの @Stateless.name
        の値と一致
    -->
    <ejb-name>EncryptionEJB</ejb-name>

    <!-- 暗号のデフォルトのパスフレーズをオーバーライド -->
    <env-entry>
        <env-entry-name>ciphersPassphrase</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>OverriddenPassword</env-entry-value>
    </env-entry>

    <!-- デフォルトの一方方向ハッシュメッセージダイジェストアルゴリズムをオーバーライド -->
    <env-entry>
        <env-entry-name>messageDigestAlgorithm</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>SHA</env-entry-value>
    </env-entry>

</session>
</enterprise-beans>
</ejb-jar>

```

B.3.2 テストリソース

B.3.2.1 EncryptionIntegrationTestCase.java

```

package org.jboss.ejb3.examples.ch05.encryption;

import java.net.MalformedURLException;
import java.net.URL;

import javax.ejb.EJB;

import junit.framework.TestCase;
import org.jboss.arquillian.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.logging.Logger;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;

```

```

import org.junit.Test;
import org.junit.runner.RunWith;

/**
 * EncryptionEJB の統合テスト
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@RunWith(Arquillian.class)
public class EncryptionIntegrationTestCase extends EncryptionTestCaseSupport
{
    // -----||
    // クラスメンバー -----||
    // -----||

    /**
     * □ガー
     */
    private static final Logger log = Logger.getLogger(EncryptionIntegrationTestCase.class);

    /**
     * EncryptionEJB の EJB 3.x ローカルビジネスビュー
     */
    @EJB
    private static EncryptionLocalBusiness encryptionLocalBusiness;

    /**
     * ejb-jar.xml 内の env-entry に関連し、デフォルトのオーバーライドとして使う
     */
    private static final String EXPECTED_CIPHERS_PASSPHRASE = "OverriddenPassword";

    /**
     * ejb-jar.xml 内の env-entry に関連し、デフォルトのオーバーライドとして使う
     */
    private static final String EXPECTED_ALGORITHM_MESSAGE_DIGEST = "SHA";

    /**
     * デプロイの定義
     */
    @Deployment
    public static JavaArchive createDeployment() throws MalformedURLException
    {
        final JavaArchive archive = ShrinkWrap.create("slsb.jar", JavaArchive.class).
            addClasses(EncryptionBean.class, EncryptionCommonBusiness.class, EncryptionLocalBusiness.class,
                EncryptionRemoteBusiness.class, EncryptionException.class).addManifestResource(new URL(Encryption
                IntegrationTestCase.class.getProtectionDomain().getCodeSource().getLocation(), "../classes/META-

```

```
INF/ejb-jar.xml"), "ejb-jar.xml");
// SHRINKWRAP-141 を実行し、ejb-jar の冗長性を低くする
log.info(archive.toString(true));
return archive;
}

// -----||
// テスト -----||
// -----||

/*
 * 以下のテストはテストの初期化で設定した EJB を使う
 */

/**
 * @see {@link EncryptionTestCaseSupport#assertHashing(EncryptionCommonBusiness)}
 */
@Test
public void testHashing() throws Throwable
{
    // ログに記録
    log.info("testHashing");

    // スーパークラスを使ってテスト
    this.assertHashing(encryptionLocalBusiness);
}

/**
 * @see {@link EncryptionTestCaseSupport#assertEncryption(EncryptionCommonBusiness)}
 */
@Test
public void testEncryption() throws Throwable
{
    // ログに記録
    log.info("testEncryption");

    // スーパークラスを使ってテスト
    this.assertEncryption(encryptionLocalBusiness);
}

/**
 * ハッシングアルゴリズムが ejb-jar.xml で宣言された
 * 環境エントリからオーバーライドされていることを確認
 *
 * @throws Throwable
 */
```

```

@Test
public void testMessageDigestAlgorithmOverride() throws Throwable
{
    // ログに記録
    log.info("testMessageDigestAlgorithmOverride");

    // 利用するアルゴリズムを取得
    final String algorithm = encryptionLocalBusiness.getMessageDigestAlgorithm();
    log.info("Using MessageDigest algorithm: " + algorithm);

    // 予想どおりであることを確認
    TestCase.assertEquals("MessageDigest algorithm should have been overridden
from the environment entry", EXPECTED_ALGORITHM_MESSAGE_DIGEST, algorithm);
}

/**
 * 暗号パスワードが ejb-jar.xml で宣言された
 * 環境エントリからオーバーライドされていることを確認
 *
 * @throws Throwable
 */
@Test
public void testCiphersPassphraseOverride() throws Throwable
{
    // ログに記録
    log.info("testCiphersPassphraseOverride");

    // 使用するパスフレーズを取得
    final String passphrase = encryptionLocalBusiness.getCiphersPassphrase();
    log.info("Using Encryption passphrase: " + passphrase);

    // 予想どおりであることを確認
    TestCase.assertEquals("Encryption passphrase should have been overridden
from the environment entry", EXPECTED_CIPHERS_PASSPHRASE, passphrase);
}
}

```

B.3.2.2 EncryptionTestCaseSupport.java

```

package org.jboss.ejb3.examples.ch05.encryption;

import junit.framework.TestCase;

import org.jboss.logging.Logger;

/**
 * Encryption POJO と EncryptionEJB に使うテストロジックを

```

```

* 集中させるための共通ベース
*
* @author <a href="mailto:alr@jboss.org">ALR</a>
*/
public class EncryptionTestCaseSupport
{
    // -----||
    // クラスメンバー -----||
    // -----||

    /**
     * ロガー
     */
    private static final Logger log = Logger.getLogger(EncryptionTestCaseSupport.class);

    /**
     * テストに使う簡単な文字列
     */
    private static final String TEST_STRING = "EJB 3.1 Examples Test String";

    // -----||
    // テストサポート -----||
    // -----||

    /**
     * ハッシング機能が期待どおりに機能していることを確認
     *
     * 1) ハッシングを行うと入力と等しくない結果が返される
     * 2) ハッシュ結果と比較し、元の入力が一致する
     *
     * @param service 使用するサービス (POJO または EJB)
     * @throws Throwable
     */
    protected void assertHashing(final EncryptionCommonBusiness service) throws Throwable
    {
        // ログに記録
        log.info("assertHashing");

        // 入力を宣言
        final String input = TEST_STRING;

        // ハッシュ
        final String hash = service.hash(input);
        log.info("「" + input + "」のハッシュ：" + hash);

        // ハッシュ機能に何らかの効果があることを検査

```

```
    TestCase.assertNotSame("The hash function had no effect upon the supplied input", input, hash);

    // 比較結果を取得
    final boolean equal = service.compare(hash, input);

    // 入力取得したハッシュと一致することを確認
    TestCase.assertTrue("The comparison of the input to its hashed result failed", equal);
}

/**
 * 暗号化機能が期待どおりに機能していることを確認
 *
 * 1) 暗号化を行うと入力と等しくない結果が返される
 * 2) 再び復号でラウンドトリップすると元の入力と等しい結果が返される
 *
 * @param 使用するサービス (POJO または EJB)
 * @throws Throwable
 */
protected void assertEncryption(final EncryptionCommonBusiness service) throws Throwable
{
    // ログに記録
    log.info("assertEncryption");

    // 入力を宣言
    final String input = TEST_STRING;

    // 暗号化
    final String encrypted = service.encrypt(input);
    log.info("Encrypted result of ¥" + input + "¥: " + encrypted);

    // 暗号化機能に何らかの効果があることを確認
    TestCase.assertNotSame("The encryption function had no effect upon the supplied input", input,
        encrypted);

    // ラウンドトリップの結果を取得
    final String roundTrip = service.decrypt(encrypted);

    // 結果が元の入力と一致することを確認
    TestCase.assertEquals("The comparison of the input to its encrypted result failed", input,
        roundTrip);
}
}
```

B.3.2.3 EncryptionUnitTestCase.java

```

package org.jboss.ejb3.examples.ch05.encryption;

import org.jboss.logging.Logger;
import org.junit.BeforeClass;

import org.junit.Test;

/**
 * EncryptionEJB のビジネスメソッドが期待どおりに
 * 機能していることを確認するためのテスト
 *
 * @author <a href="mailto:alr@jboss.org">ALR</a>
 */
public class EncryptionUnitTestCase extends EncryptionTestCaseSupport
{
    // -----||
    // クラスメンバー -----||
    // -----||

    /**
     * □ガー
     */
    private static final Logger log = Logger.getLogger(EncryptionUnitTestCase.class);

    /**
     * POJO 暗号化サービス
     */
    private static EncryptionBean encryptionService;

    // -----||
    // ライフサイクル -----||
    // -----||

    /**
     * テストスイートを初期化する。テストが実行される前に一度だけ呼び出される。
     */
    @BeforeClass
    public static void initialize() throws Throwable
    {
        // 暗号化サービスを POJO として作成する
        encryptionService = new EncryptionBean();
        encryptionService.initialize(); // ここで初期化を手動で呼び出す
    }

    // -----||

```

```
// テスト -----||
// -----||

/**
 * 以下のテストはテスト初期化で設定された POJO を使う
 */

/**
 * @see {@link EncryptionTestCaseSupport#assertHashing(EncryptionCommonBusiness)}
 */
@Test
public void testHashing() throws Throwable
{
    // ログに記録
    log.info("testHashing");

    // スーパークラスを使ってテストする
    this.assertHashing(encryptionService);
}

/**
 * @see {@link EncryptionTestCaseSupport#assertEncryption(EncryptionCommonBusiness)}
 */
@Test
public void testEncryption() throws Throwable
{
    // ログに記録
    log.info("testEncryption");

    // スーパークラスを使ってテストする
    this.assertEncryption(encryptionService);
}
}
```


付録 C

ステートフルセッション EJB : FTP クライアントの例

C.1 説明

サーバは、要求の間に特定のクライアントに関する情報を覚えておく必要があることが少なくありません。これは「対話状態」を持つコンポーネントでモデル化します。例えば、ファイル転送プロトコル (FTP) は「ステートフル」です。また、サーバは各クライアントが使っている現在の作業ディレクトリを知っています。この例では、ステートフルセッション Bean を使って FTP クライアントをモデル化します。

リソース (RAM) を節約するために、SFSB には「パッシブ化」という過程があり、インスタンスとクライアントのセッションがタイムアウト後にメモリから削除されてディスクに記録されます。クライアントセッションが再び必要になると、状態がアクティブ化され使えるようになります。クライアントにパッシブ化が全く気付かれないように、シリアライズ / デシリアライズ工程で状態を適切に管理する必要があります。

C.2 オンライン上の関連情報

ウィキ記事 : <http://community.jboss.org/docs/DOC-15568>

ソースの場所 : <http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples/tree/master/ch06-filetransfer/>

C.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

C.3.1 実装リソース

C.3.1.1 FileTransferBean.java

```
package org.jboss.ejb3.examples.ch06.filetransfer;

import java.io.IOException;
import java.io.Serializable;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
```

```
import javax.ejb.PostActivate;
import javax.ejb.PrePassivate;
import javax.ejb.Remote;
import javax.ejb.Remove;
import javax.ejb.Stateful;

import org.apache.commons.net.ftp.FTPClient;
import org.apache.commons.net.ftp.FTPFile;
import org.apache.commons.net.ftp.FTPReply;
import org.jboss.logging.Logger;

/**
 * ステートフルセッション Bean でモデル化した
 * FileTransferEJB の Bean 実装クラス
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@Stateful(name = FileTransferBean.EJB_NAME)
@Remote(FileTransferRemoteBusiness.class)
public class FileTransferBean implements FileTransferRemoteBusiness, Serializable
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * シリアルバージョン UID
     */
    private static final long serialVersionUID = 1L;

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(FileTransferBean.class);

    /**
     * グローバル JNDI アドレスで使われる、EJB の名前
     */
    public static final String EJB_NAME = "FileTransferEJB";

    /**
     * 接続するホスト名。
     * 本番システムでは、通常は構成可能な環境エントリを使って外部化。
     */
    private static String CONNECT_HOST = "localhost";
```

```

/**
 * 接続するポート。
 * 本番システムでは、通常は構成可能な環境エントリを使って外部化。
 * FTP ポートの IANA 標準は 21 だが、テストのために *nix への
 * ルートアクセスが必要なので、非標準の 12345 を使用。
 */
private static int CONNECT_PORT = 12345;

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * 基礎をなす FTP クライアント。
 * パッシブ化の際には状態をシリアライズさせたくない。
 * アクティブ化の際にこのクライアントと接続を再初期化。
 */
private FTPClient client;

/**
 * 現在の作業ディレクトリの名前。
 * パッシブ化するときこれが指定されていれば、
 * アクティブ化の際にこのディレクトリに移動
 */
private String presentWorkingDirectory;

//-----||
// ライフサイクルコールバック -----||
//-----||

/**
 * インスタンスがパッシブ化されるか、完全に使われなくなるときに
 * コンテナによって呼び出される。
 *
 * @see org.jboss.ejb3.examples.ch06.filetransfer.FileTransferCommonBusiness#
disconnect()
 */
@PrePassivate
@PreDestroy
@Override
public void disconnect()
{
    // FTP クライアントを取得
    final FTPClient client = this.getClient();

    // 存在する場合

```

```
if (client != null)
{
    // 接続している場合
    if (client.isConnected())
    {
        // ログアウト
        try
        {
            client.logout();
            log.info("Logged out of: " + client);
        }
        catch (final IOException ioe)
        {
            log.warn("Exception encountered in logging out of the FTP client", ioe);
        }

        // 切断
        try
        {
            log.debug("Disconnecting: " + client);
            client.disconnect();
            log.info("Disconnected: " + client);
        }
        catch (final IOException ioe)
        {
            log.warn("Exception encountered in disconnecting the FTP client", ioe);
        }

        // このフィールドをシリアライズしないように、null に設定
        this.client = null;
    }
}

/**
 * インスタンスの作成時や再アクティブ化時（パッシブ化状態からの遷移時）に
 * コンテナによって呼び出される。基礎をなす FTP クライアントを作成し、
 * 適切なすべての接続をオープン
 *
 * @see org.jboss.ejb3.examples.ch06.filetransfer.FileTransferCommonBusiness#connect()
 */
@PostConstruct
@PostActivate
@Override
public void connect() throws IllegalStateException, FileTransferException
{
```

```
/*
 * 前提条件を確認
 */
final FTPClient clientBefore = this.getClient();
if (clientBefore != null && clientBefore.isConnected())
{
    throw new IllegalStateException("FTP クライアントがすでに初期化されています");
}

// 接続プロパティを取得
final String connectHost = this.getConnectHost();
final int connectPort = this.getConnectPort();

// クライアントを作成
final FTPClient client = new FTPClient();
final String canonicalServerName = connectHost + ":" + connectPort;
log.debug("Connecting to FTP Server at " + canonicalServerName);
try
{
    client.connect(connectHost, connectPort);
}
catch (final IOException ioe)
{
    throw new FileTransferException("Error in connecting to " + canonicalServerName, ioe);
}

// 設定
log.info("Connected to FTP Server at: " + canonicalServerName);
this.setClient(client);

// 直前の処理が成功したことを確認
this.checkLastOperation();
try
{
    // ログイン
    client.login("user", "password");

    // 直前の処理が成功したことを確認
    this.checkLastOperation();
}
catch (final Exception e)
{
    throw new FileTransferException("Could not log in", e);
}

// pwd が定められている場合は、そこに cd する
```

```
        final String pwd = this.getPresentWorkingDirectory();
        if (pwd != null)
        {
            this.cd(pwd);
        }
    }

//-----||
// 必要な実装 -----||
//-----||

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch06.filetransfer.FileTransferCommonBusiness#cd(java.lang.String)
 */
@Override
public void cd(final String directory)
{
    // クライアントを取得
    final FTPClient client = this.getClient();

    // cd を実行
    try
    {
        // cd を実行
        client.changeWorkingDirectory(directory);

        // 成功を確認
        this.checkLastOperation();
    }
    catch (final Exception e)
    {
        throw new FileTransferException("Could not change working directory to ¥" + directory + "¥",
e);
    }

    // (アクティブ化時に使う) pwd を設定
    log.info("cd > " + directory);
    this.setPresentWorkingDirectory(directory);
}

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch06.filetransfer.FileTransferCommonBusiness#mkdir(java.lang.String)
 */
@Override
public void mkdir(final String directory)
```

```
{
    // クライアントを取得
    final FTPClient client = this.getClient();

    // cd を実行
    try
    {
        // mkdir を実行
        client.makeDirectory(directory);

        // 成功を確認
        this.checkLastOperation();
    }
    catch (final Exception e)
    {
        throw new FileTransferException("Could not make directory ¥" + directory + "¥", e);
    }
}

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch06.filetransfer.FileTransferCommonBusiness#pwd()
 */
@Override
public String pwd()
{
    // クライアントを取得
    final FTPClient client = this.getClient();

    // pwd を実行
    try
    {
        final FTPFile[] files = client.listFiles();
        for (final FTPFile file : files)
        {
            log.info(file);
        }

        // pwd を実行
        return client.printWorkingDirectory();
    }
    catch (final IOException ioe)
    {
        throw new FileTransferException("Could not print working directory", ioe);
    }
}
```

```
//-----||
// 内部ヘルパーメソッド -----||
//-----||

/**
 * 直前の処理が正の応答コードを伴って成功していることを確認します。
 * 成功していない場合は {@link FileTransferException} が発行され、
 * 応答コードはエラーを示します。
 *
 * @throws FileTransferException
 */
protected void checkLastOperation() throws FileTransferException
{
    // クライアントを取得
    final FTPClient client = this.getClient();

    // 接続から応答を取得して調べる
    final int connectReply = client.getReplyCode();
    if (!FTPReply.isPositiveCompletion(connectReply))
    {
        // 問題を知らせる
        throw new FileTransferException("Did not receive positive completion code from server, instead
code was: " + connectReply);
    }
}

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch06.filetransfer.FileTransferRemoteBusiness#endSession()
 */
@Remove
@Override
public void endSession()
{
    log.info("Session Ending...");
}

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return connectHost
 */
public String getConnectHost()
{
    return CONNECT_HOST;
}
```

```
    }

    /**
     * @return connectPort
     */
    public int getConnectPort()
    {
        return CONNECT_PORT;
    }

    /**
     * @return client
     */
    protected final FTPClient getClient()
    {
        return client;
    }

    /**
     * @param client 設定するクライアント
     */
    private void setClient(final FTPClient client)
    {
        this.client = client;
    }

    /**
     * @return presentWorkingDirectory
     */
    private String getPresentWorkingDirectory()
    {
        return presentWorkingDirectory;
    }

    /**
     * @param presentWorkingDirectory 設定する presentWorkingDirectory
     */
    private void setPresentWorkingDirectory(String presentWorkingDirectory)
    {
        this.presentWorkingDirectory = presentWorkingDirectory;
    }
}
```

C.3.1.2 FileTransferCommonBusiness.java

```
package org.jboss.ejb3.examples.ch06.filetransfer;

/**
 * FileTransferEJB のすべてのビジネスインタフェースに共通する
 * 処理のための規約を含む
 *
 * 現在の作業ディレクトリの変更、現在の作業ディレクトリの表示、
 * ディレクトリの作成に対するサポートを含む
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface FileTransferCommonBusiness
{
    // -----||
    // 規約 -----||
    // -----||

    /**
     * 指定された名前のディレクトリを作成
     *
     * @throws IllegalStateException クライアント接続が初期化されていない場合
     */
    void mkdir(String directory) throws IllegalStateException;

    /**
     * 指定されたディレクトリに移動
     *
     * @param directory
     * @throws IllegalStateException クライアント接続が初期化されていない場合
     */
    void cd(String directory) throws IllegalStateException;

    /**
     * 現在の作業ディレクトリの名前を取得
     *
     * @return
     * @throws IllegalStateException クライアント接続が初期化されていない場合
     */
    String pwd() throws IllegalStateException;

    /**
     * このサービスを使ったクライアントが完了したことを示す。
     * 保留中のすべての処理を取り消し、適切なクリーンアップを実行
     * すでに切断されている場合は何も実行しない。
     */
}
```

```

    */
    void disconnect();

    /**
     * 対象となる FTP サーバへの接続をオープンし、
     * コマンド（ログインなど）を送信する前に必要なタスクを実行
     *
     * @throws IllegalStateException すでに初期化 / 接続されている場合
     */
    void connect() throws IllegalStateException;
}

```

C.3.1.3 FileTransferException.java

```

package org.jboss.ejb3.examples.ch06.filetransfer;

/**
 * ファイル転送処理中に問題が発生した
 * ことを示す例外
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public class FileTransferException extends RuntimeException
{

    //-----||
    // クラスメンバー -----||
    //-----||
    private static final long serialVersionUID = 1L;

    //-----||
    // コンストラクタ -----||
    //-----||

    public FileTransferException()
    {
        super();
    }

    public FileTransferException(final String message, final Throwable cause)
    {
        super(message, cause);
    }

    public FileTransferException(final String message)
    {

```

```

        super(message);
    }

    public FileTransferException(final Throwable cause)
    {
        super(cause);
    }
}

```

C.3.1.4 FileTransferRemoteBusiness.java

```

package org.jboss.ejb3.examples.ch06.filetransfer;

import javax.ejb.Remove;

/**
 * FileTransferEJB のリモートビジネスインタフェース
 * これは EJB 環境だけで使うので、
 * 現在のセッションを終了するメソッドを定義
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface FileTransferRemoteBusiness extends FileTransferCommonBusiness
{
    // -----||
    // 規約 -----||
    // -----||

    /**
     * 現在のセッションを終了する。Bean 実装クラスで
     * {@link javax.ejb.Remove} アノテーションが付けられるので、
     * これが SFSB の @Remove となる。
     */
    void endSession();
}

```

C.3.2 テストリソース

C.3.2.1 FileTransferIntegrationTestCase.java

```

package org.jboss.ejb3.examples.ch06.filetransfer;

import java.io.File;

import javax.ejb.EJB;
import javax.ejb.NoSuchEJBException;

```

```

import junit.framework.TestCase;

import org.jboss.arquillian.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.logging.Logger;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.runner.RunWith;

/**
 * FileTransferEJB が EJB コンテナからのステートフルセッション Bean
 * として機能していることを確認するテストケース
 *
 * {@link FileTransferTestCaseBase} からのテストサポートを継承し、
 * プロキシ上の EJB 固有のタスクも追加でテスト
 * セッションが独立して機能し、セッションを削除すると
 * そのセッションを使えなくなることを示す。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@RunWith(Arquillian.class)
public class FileTransferIntegrationTestCase extends FileTransferTestCaseBase
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログー
     */
    private static final Logger log = Logger.getLogger(FileTransferIntegrationTestCase.class);

    /**
     * FTP サーバユーザのための設定ファイル名
     */
    private static final String FTP_SERVER_USERS_CONFIG_FILENAME = "ftpusers.properties";

    /**
     * FTP サーバが接続すべきポート
     */
    private static final int FTP_SERVER_BIND_PORT = 12345;

```

```
/**
 * FTP サーバ
 */
private static FtpServerPojo ftpServer;

/**
 * デプロイ
 * @return
 */
@Deployment
public static JavaArchive createDeployment()
{
    final JavaArchive archive = ShrinkWrap.create("ftpclient.jar", JavaArchive.class).
addPackage(FileTransferBean.class.getPackage());
    log.info(archive.toString(true));
    return archive;
}

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * EJB のビュー。プロキシのリモートビジネスインタフェース型
 */
@EJB
private FileTransferRemoteBusiness client1;

/**
 * 別の FTP クライアントセッション
 */
@EJB
private FileTransferRemoteBusiness client2;

//-----||
// ライフサイクル -----||
//-----||

/**
 * FTP サーバの作成と起動
 */
@BeforeClass
public static void startFtpServer() throws Exception
{
    // 作成
    final FtpServerPojo server = new FtpServerPojo();
}
```

```
// 設定
server.setUsersConfigFileName(FTP_SERVER_USERS_CONFIG_FILENAME);
server.setBindPort(FTP_SERVER_BIND_PORT);

// 起動と設定
server.initializeServer();
server.startServer();
ftpServer = server;
}

/**
 * FTP サーバの停止
 * @throws Exception
 */
@AfterClass
public static void stopFtpServer() throws Exception
{
    ftpServer.stopServer();
}

/**
 * FTP クライアント SFSB プロキシのセッションを終了して
 * リセット
 */
@After
public void endClientSessions() throws Exception
{
    // クライアント 1 のセッションを終了
    try
    {
        client1.endSession();
    }
    // すでに終了している場合
    catch (final NoSuchEJBException nsee)
    {
        // 無視
    }

    // クライアント 2 のセッションを終了
    try
    {
        client2.endSession();
    }
    // すでに終了している場合
    catch (final NoSuchEJBException nsee)
    {
```

```
        // 無視
    }
}

//-----||
// テスト -----||
//-----||

/**
 * 2つの別個のセッションが互いに独立して動作することを確認
 *
 * @throws Exception
 */
@Test
public void testSessionIsolation() throws Exception
{
    // ログに記録
    log.info("testSessionIsolation");

    // テストライフサイクルで作成された既存のクライアントを取得
    final FileTransferRemoteBusiness session1 = this.getClient();

    // 別のクライアントを使う
    final FileTransferRemoteBusiness session2 = this.client2;

    // それぞれのホームディレクトリに移動
    final String ftpHome = getFtpHome().getAbsolutePath();
    session1.cd(ftpHome);
    session2.cd(ftpHome);

    // 各セッションで新しいディレクトリを作成し、そこに移動
    final String newDirSession1 = "newDirSession1";
    final String newDirSession2 = "newDirSession2";
    session1.mkdir(newDirSession1);
    session1.cd(newDirSession1);
    session2.mkdir(newDirSession2);
    session2.cd(newDirSession2);

    // 各セッションの現在の作業ディレクトリを取得
    final String pwdSession1 = session1.pwd();
    final String pwdSession2 = session2.pwd();

    // 各セッションが正しい作業ディレクトリにいることを確認
    TestCase.assertEquals("セッション1のpwdが予期しない結果である", ftpHome + File.separator +
        newDirSession1, pwdSession1);
    TestCase.assertEquals("セッション2のpwdが予期しない結果である", ftpHome + File.separator +
        newDirSession2, pwdSession2);
}
```

```

// セッション2を手動で終了（セッション1はテストライフサイクルによって終了される）
session2.endSession();
}

/**
 * {@link FileTransferRemoteBusiness#endSession()} の呼び出しにより
 * SFSB の対応するインスタンスが削除され、その後の操作が
 * {@link NoSuchEJBException} になることを確認
 *
 * @throws Exception
 */
@Test
public void testSfsbRemoval() throws Exception
{
    // ログに記録
    log.info("testSfsbRemoval");

    // テストライフサイクルで作成された既存のクライアントを取得
    final FileTransferRemoteBusiness sfsb = this.getClient();

    // ホームディレクトリに移動
    final String ftpHome = getFtpHome().getAbsolutePath();
    sfsb.cd(ftpHome);

    // pwd を実行してテスト
    final String pwdBefore = sfsb.pwd();
    TestCase.assertEquals("セッションは FTP ホームディレクトリにいるべき ",

    // セッションを終了すると、Bean 実装クラスに付けられた
    // @Remove アノテーションにより、対応するインスタンスが削除される
    sfsb.endSession());

    // 別の操作を試し、NoSuchEJBException が発行されることを確認
    boolean gotExpectedException = false;
    try
    {
        // @Remove メソッドを呼び出し済みなので、これは成功しないはず
        sfsb.pwd();
    }
    catch (final NoSuchEJBException nsee)
    {
        gotExpectedException = true;
    }
    TestCase.assertTrue("セッションを終了する呼び出しによって、SFSB Bean インスタンスが削除されて
    いない ", gotExpectedException);
}

```

```

//-----||
// 必要な実装 -----||
//-----||

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch06.filetransfer.FileTransferTestCaseBase#getClient()
 */
@Override
protected FileTransferRemoteBusiness getClient()
{
    return this.client;
}
}

```

C.3.2.2 FileTransferTestCaseBase.java

```

package org.jboss.ejb3.examples.ch06.filetransfer;

import java.io.File;

import junit.framework.TestCase;

import org.jboss.logging.Logger;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

/**
 * ファイル転送テストクラスのベーステスト
 * 単体テストまたは統合テストで拡張
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public abstract class FileTransferTestCaseBase
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * □ガー
     */
    private static final Logger log = Logger.getLogger(FileTransferTestCaseBase.class);

    /**
     * テストのホームとしての役割を果たす書き込み可能な

```

```

    * 一時ファイルシステムの下のディレクトリ名
    */
private static final String RELATIVE_LOCATION_HOME = "ejb31_ch06-example-ftpHome";

/**
 * 入出力一時ディレクトリを示すシステムプロパティ名
 */
private static final String SYS_PROP_NAME_IO_TMP_DIR = "java.io.tmpdir";

/**
 * FTP 操作の書き込み可能なホームとして使うファイル
 * テストライフサイクルに従って作成され破棄される
 */
private static File ftpHome;

//-----||
// ライフサイクル -----||
//-----||

/**
 * FTP 操作の書き込み可能なホームとして使うディレクトリを作成
 * 各テストの実行前に呼び出す。
 *
 * @throws Exception
 */
@Before
public void createFtpHome() throws Exception
{
    final File ftpHome = getFtpHome();
    if (ftpHome.exists())
    {
        throw new RuntimeException("Error in test setup; FTP Home should not yet exist: " + ftpHome.
getAbsolutePath());
    }
    final boolean created = ftpHome.mkdir();
    if (!created)
    {
        throw new RuntimeException("Request to create the FTP Home failed: " + ftpHome.
getAbsolutePath());
    }
    log.info("Created FTP Home: " + ftpHome.getAbsolutePath());
}

/**
 * FTP 操作の書き込み可能なホームとして使うディレクトリを削除
 * 各テストの実行後に呼び出す。
 *

```

```
* @throws Exception
*/
@After
public void deleteFtpHome() throws Exception
{
    final File ftpHome = getFtpHome();
    if (!ftpHome.exists())
    {
        throw new RuntimeException("Error in test setup; FTP Home should exist:" + ftpHome.
getAbsolutePath());
    }
    final boolean removed = this.deleteRecursive(ftpHome);
    if (!removed)
    {
        throw new RuntimeException("Request to remove the FTP Home failed: " + ftpHome.
getAbsolutePath());
    }
    log.info("Removed FTP Home: " + ftpHome.getAbsolutePath());
}

//-----||
// テスト -----||
//-----||

/**
 * 新しいディレクトリを作成してそのディレクトリに移動し、
 * 新たに作成したディレクトリを現在の作業ディレクトリとして
 * 取得できることを確認
 */
@Test
public void testMkdirCdAndPwd() throws Exception
{
    // ログに記録
    log.info("testMkdirAndPwd");

    // クライアントを取得
    final FileTransferCommonBusiness client = this.getClient();

    // ホームに移動
    final String home = getFtpHome().getAbsolutePath();
    client.cd(home);

    // ホームにいることを確認
    final String pwdBefore = client.pwd();
    TestCase.assertEquals("Present working directory should be our home", home, pwdBefore);
}
```

```

// ディレクトリを作成
final String newDir = "newDirectory";
client.mkdir(newDir);

// 新しいディレクトリに移動
client.cd(newDir);

// 新しいディレクトリにいることを確認
final String pwdAfter = client.pwd();
TestCase.assertEquals("現在の作業ディレクトリは新しいディレクトリに設定されるべき", home + File.
separator + newDir, pwdAfter);
}

//-----||
// 規約 -----||
//-----||

/**
 * テストに使うクライアントを取得
 */
protected abstract FileTransferCommonBusiness getClient();

//-----||
// 内部ヘルパーメソッド -----||
//-----||

/**
 * 指定されたルート（ルートそのものも含む）のすべての中身を
 * 再帰的に削除する。指定されたルートが存在しない場合は、
 * 何も行わない。
 *
 * @param root
 * @return true 削除した場合、false その他の場合
 */
protected boolean deleteRecursive(final File root)
{
    // 存在することを確認
    if (!root.exists())
    {
        return false;
    }

    // すべての子要素を取得
    final File[] children = root.listFiles();

    // ディレクトリの場合

```

```
    if (children != null)
    {
        // すべての子要素を削除
        for (final File child : children)
        {
            this.deleteRecursive(child);
        }
    }

    // それ自身を削除
    final boolean success = root.delete();
    log.info("Deleted: " + root);
    return success;
}

/**
 * テストのための書き込み可能なホームを取得し、
 * 入出力一時ディレクトリの名前空間に設定する
 */
protected static File getFtpHome() throws Exception
{
    // ホームが定義されていない場合
    if (ftpHome == null)
    {
        // プロパティを取得
        final String sysPropIoTempDir = SYS_PROP_NAME_IO_TMP_DIR;
        final String ioTempDir = System.getProperty(sysPropIoTempDir);
        if (ioTempDir == null)
        {
            throw new RuntimeException("I/O temp directory was not specified by system property: " +
sysPropIoTempDir);
        }

        // ファイルを作成
        final File ioTempDirFile = new File(ioTempDir);
        if (!ioTempDirFile.exists())
        {
            throw new RuntimeException("I/O Temp directory does not exist: " + ioTempDirFile.
getAbsolutePath());
        }

        // ホーム用の接尾辞を付加する
        final File home = new File(ioTempDirFile, RELATIVE_LOCATION_HOME);
        ftpHome = home;
    }
}
```

```

        // 戻す
        return ftpHome;
    }
}

```

C.3.2.3 FileTransferUnitTestCase.java

```

package org.jboss.ejb3.examples.ch06.filetransfer;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.ObjectInput;
import java.io.ObjectInputStream;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;

import javax.ejb.PostActivate;
import javax.ejb.PrePassivate;

import junit.framework.TestCase;

import org.jboss.logging.Logger;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

/**
 * ファイル転送ビジネスロジックが損なわれていないことを
 * コンテナ外から確認するためのテストケース
 *
 * これは厳密には SFSB 例の一部ではないが、
 * この例に含まれるすべてが期待どおりに機能していることを
 * 確かめるために存在する。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public class FileTransferUnitTestCase extends FileTransferTestCaseBase
{

    //-----||
    // クラスメンバー -----||
    //-----||

}

```

```
* ログガー
*/
private static final Logger log = Logger.getLogger(FileTransferUnitTestCase.class);

/**
 * 接続する FTP サービス
 */
private static FtpServerPojo ftpService;

/**
 * FTP サービスが接続するポート
 */
private static final int FTP_SERVICE_BIND_PORT = 12345;

/**
 * サーバ用のユーザ設定ファイル名
 */
private static final String FILE_NAME_USERS_CONFIG = "ftpusers.properties";

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * FTP クライアント
 */
private FileTransferBean ftpClient;

//-----||
// ライフサイクル -----||
//-----||

/**
 * テストクライアントが接続する FTP サービスを
 * 生成して初期化し、起動する。
 * 各テストの実行前に呼び出す。
 */
@BeforeClass
public static void createFtpService() throws Exception
{
    // FTP サービスを生成する
    final FtpServerPojo service = new FtpServerPojo();

    // 設定
    service.setBindPort(FTP_SERVICE_BIND_PORT);
    service.setUsersConfigFileName(FILE_NAME_USERS_CONFIG);
}
```

```
// 初期化
service.initializeServer();

// 起動
service.startServer();

// 設定 (成功時)
log.info("Started up test FTP Service: " + service);
ftpService = service;
}

/**
 * FTP サービスを停止してリセットする。
 * すべてのテストが完了した後に一度だけ呼び出す。
 *
 * @throws Exception
 */
@AfterClass
public static void destroyFtpService() throws Exception
{
    // 初期化が終了している場合にだけ実行する
    if (ftpService == null)
    {
        return;
    }

    // サーバを停止
    ftpService.stopServer();

    // リセット
    ftpService = null;
    log.info("Brought down test FTP Service");
}

/**
 * テストで使う FTP クライアントを作成して初期化
 * 各テストの実行前に呼び出す。
 */
@Before
public void createFtpClient() throws Exception
{
    // クライアントを作成
    final FileTransferBean ftpClient = new FileTransferBean();

    // 接続
    ftpClient.connect();
}
```

```
// 設定
this.ftpClient = ftpClient;
log.info("FTP クライアントを設定 : " + ftpClient);
}

/**
 * FTP クライアントを切断してリセット
 * 各テストが完了した後に呼び出す。
 *
 * @throws Exception
 */
@After
public void cleanup() throws Exception
{
    // クライアントを取得
    final FileTransferBean ftpClient = this.ftpClient;

    // クライアントを取得
    if (ftpClient != null)
    {
        // 切断してリセット
        ftpClient.disconnect();
        this.ftpClient = null;
    }
}

//-----||
// テスト -----||
//-----||

/**
 * {@link PrePassivate} と {@link PostActivate} ライフサイクルコールバックを
 * 手動で呼び出して、パッシブ化 / アクティブ化の処理をモック。
 * 呼び出し後は、クライアントが正しく機能すべき。
 * 期待どおりに再接続し、現在の作業ディレクトリで再開。
 *
 * @throws Exception
 */
@Test
public void testPassivationAndActivation() throws Exception
{
    // ログに記録
    log.info("testPassivationAndActivation");

    // クライアントを取得
    final FileTransferCommonBusiness client = this.getClient();
}
```

```

// ホームに移動
final String home = getFtpHome().getAbsolutePath();
client.cd(home);

// pwd をテスト
final String pwdBefore = client.pwd();
TestCase.assertEquals("現在の作業ディレクトリはホームに設定されるべき", home, pwdBefore);

// @PrePassivate をモック
log.info("モック@" + PrePassivate.class.getName());
client.disconnect();

// パッシブ化をモック
log.info("アクティブ化をモック");
final ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
final ObjectOutput objectOut = new ObjectOutputStream(outputStream);
objectOut.writeObject(client);
objectOut.close();

// アクティブ化をモック
log.info("アクティブ化をモック");
final InputStream inputStream = new ByteArrayInputStream(outputStream.toByteArray());
final ObjectInput objectIn = new ObjectInputStream(inputStream);

// パッシブ化 / アクティブ化から新しいクライアントを取得
final FileTransferCommonBusiness serializedClient = (FileTransferCommonBusiness)
    objectIn.readObject();
objectIn.close();

// @PostActivate をモック
log.info("モック@" + PostActivate.class.getName());
serializedClient.connect();

// pwd をテスト
final String pwdAfter = serializedClient.pwd();
TestCase.assertEquals("現在の作業ディレクトリはパッシブ化 / アクティブ化の前と同じであるべき",
    home, pwdAfter);
}

//-----||
// 必要な実装 -----||
//-----||

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch06.filetransfer.FileTransferTestCaseBase#getClient()
 */

```

```

@Override
protected FileTransferCommonBusiness getClient()
{
    return this.ftpClient;
}
}

```

C.3.2.4 FtpServerPojo.java

```

package org.jboss.ejb3.examples.ch06.filetransfer;

import java.net.URL;
import java.security.AccessController;
import java.security.PrivilegedAction;

import org.apache.ftpserver.FtpServer;
import org.apache.ftpserver.FtpServerFactory;
import org.apache.ftpserver.ftplet.FtpException;
import org.apache.ftpserver.ftplet.UserManager;
import org.apache.ftpserver.listener.ListenerFactory;
import org.apache.ftpserver.usermanager.ClearTextPasswordEncryptor;
import org.apache.ftpserver.usermanager.PropertiesUserManagerFactory;
import org.jboss.logging.Logger;

/**
 * 埋め込まれた FTP サーバの開始と停止
 * を担う POJO
 *
 * これはテスト実行環境の一部と考えるべきであり、
 * 本当は SFSB 例そのものの一ではない。
 * この例の SFSB は、この簡単な Bean で開始される
 * FTP サーバのクライアントである。
 *
 * スレッドセーフではない。シングルスレッド環境で使う
 * (または独自の外部同期を実行する) ことを意図している。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public final class FtpServerPojo
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー

```

```

*/
private static final Logger log = Logger.getLogger(FtpServerPojo.class);

/**
 * サーバのデフォルトリスナ名
 */
private static final String LISTENER_NAME_DEFAULT = "default";

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * FTP サーバが接続するポート
 */
private int bindPort;

/**
 * 基礎をなすサーバ。エクスポートしなければいけない。
 */
private FtpServer server;

/**
 * ユーザ / パスワード設定ファイルの名前
 */
private String usersConfigFileName;

//-----||
// ライフサイクルメソッド -----||
//-----||

/**
 * 基礎をなすサーバの作成と初期化を行う。
 * この POJO を作成したときにライフサイクルに沿って呼び出すべきである。
 *
 * @throws IllegalStateException サーバのプロパティが正しく
 * 初期化されていない場合
 */
public void initializeServer() throws IllegalStateException
{
    // プロパティを取得
    final int bindPort = this.getBindPort();

    /**
     * 前提条件を確認
     */

```

```
        if (bindPort <= 0)
        {
            throw new IllegalStateException("Property for bind port has not been set to a valid value
above 0.");
        }

        // 初期化
        final FtpServerFactory serverFactory = new FtpServerFactory();
        final ListenerFactory factory = new ListenerFactory();

        // プロパティを設定
        log.debug("Using FTP bind port: " + bindPort);
        factory.setPort(bindPort);

        // サーバファクトリにデフォルトリスナを追加
        serverFactory.addListener(LISTENER_NAME_DEFAULT, factory.createListener());

        // 現在のクラスローダを取得
        final ClassLoader tccl = AccessController.doPrivileged(new PrivilegedAction<ClassLoader>()
        {
            @Override
            public ClassLoader run()
            {
                return Thread.currentThread().getContextClassLoader();
            }
        });

        // プロパティファイルを読み込んで URI を取得
        final String usersConfigFileName = this.getUsersConfigFileName();
        log.info("Using users configuration file: " + usersConfigFileName);
        final URL usersConfigUrl = tccl.getResource(usersConfigFileName);
        if (usersConfigUrl == null)
        {
            throw new RuntimeException("Could not find specified users configuration file upon the
classpath: " + usersConfigFileName);
        }

        // ユーザ認証メカニズムを設定
        final PropertiesUserManagerFactory userManagerFactory = new PropertiesUserManagerFactory();
        userManagerFactory.setUrl(usersConfigUrl);
        userManagerFactory.setPasswordEncryptor(new ClearTextPasswordEncryptor());
        final UserManager userManager = userManagerFactory.createUserManager();
        serverFactory.setUserManager(userManager);

        // サーバを作成
        final FtpServer server = serverFactory.createServer();
```

```
        this.setServer(server);
        log.info("Created FTP Server: " + server);
    }

    /**
     * サーバを開始
     *
     * @throws IllegalStateException サーバが初期化されていない場合、または
     * サーバがすでに開始されている場合
     * @throws FtpException サーバの開始時にエラーが発生した場合
     */
    public void startServer() throws IllegalStateException, FtpException
    {
        // サーバを取得
        final FtpServer server = this.getServer();

        /**
         * 前提条件を確認
         */

        // 初期化されていることを確認
        if (server == null)
        {
            throw new IllegalStateException("The server has not yet been initialized");
        }

        // すでに稼働していたり他の状態になっていたりしないことを確認
        if (!server.isStopped())
        {
            throw new IllegalStateException("Server cannot be started if it is not currently stopped");
        }

        // 開始
        log.debug("Starting the FTP Server: " + server);
        server.start();
        log.info("FTP Server Started: " + server);
    }

    /**
     * サーバを停止
     *
     * @throws IllegalStateException サーバがすでに停止している場合、または
     * サーバが初期化されていない場合
     * @throws FtpException
     */
    public void stopServer() throws IllegalStateException
```

```
{
    // サーバを取得
    final FtpServer server = this.getServer();

    /*
     * 前提条件を確認
     */

    // 初期化されていることを確認
    if (server == null)
    {
        throw new IllegalStateException("The server has not yet been initialized");
    }

    // すでに稼働していたり他の状態になっていたりしないことを確認
    if (server.isStopped())
    {
        throw new IllegalStateException("Server cannot be stopped if it's already stopped");
    }

    // 停止
    log.debug("Stopping the FTP Server: " + server);
    server.stop();
    log.info("FTP Server stopped: " + server);
}

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * 接続するポートを取得
 */
public int getBindPort()
{
    return bindPort;
}

/**
 * 接続するポートを設定
 *
 * @param bindPort
 */
public void setBindPort(final int bindPort)
{
    this.bindPort = bindPort;
}
```

```
    }

    /**
     * 基礎をなす FTP サーバを取得
     *
     * @return
     */
    protected FtpServer getServer()
    {
        return server;
    }

    /**
     * 基礎をなす FTP サーバを設定
     *
     * @param server
     */
    private void setServer(final FtpServer server)
    {
        this.server = server;
    }

    /**
     * ユーザ設定ファイル名を取得
     *
     * @return usersConfigFileName
     */
    public String getUsersConfigFileName()
    {
        return usersConfigFileName;
    }

    /**
     * ユーザ設定ファイル名を設定
     *
     * @param usersConfigFileName 設定する usersConfigFileName
     */
    public void setUsersConfigFileName(final String usersConfigFileName)
    {
        this.usersConfigFileName = usersConfigFileName;
    }
}
```

C.3.2.5 ftpusers.properties

```
# ユーザ / パスワードファイル
ftpserver.user.user.idletime=0
```

```
ftpserver.user.user.userpassword=password  
ftpserver.user.user.homedirectory=/  
ftpserver.user.user.writepermission=true  
ftpserver.user.user.enableflag=true
```

付録 D

シングルトンセッション EJB : RSS キャッシュの例

D.1 説明

プール (SLSB) やキャッシュ (SFSB) の代わりに1つのインスタンスでモデル化するのが最適なビジネスプロセスも少なくありません。@Singleton EJB という EJB 3.1 の新機能は、対応する1つのインスタンスを生成して受信したすべての要求に対処します。

@Singleton EJB を使うと2つの重要な結果をもたらします。まず、このインスタンスにはサービス (@Startup) アノテーションが付けられるので、アプリケーションライフサイクルイベント (@PreConstruct / @PostConstruct) が発生します。

次に、すべての要求が1つのインスタンスを共有するので、EJB 開発者には書き込み可能な共有状態の同時実行性に対処する必要が初めて生じます。SLSB や SFSB Bean は、同時に複数のスレッドからアクセスされることはありません (制限によるスレッドセーフ)。@Singleton は同時に複数スレッドからアクセスされる可能性があるため、仕様では書き込み / 読み込みロックモデルに基づいたコンテナ管理の同時実行性アノテーションを導入し、開発者に宣言型のスレッドセーフ機能を提供しています。

この例では、RSS フィードに備わっている簡単なキャッシュをモデル化しています。読み込み操作はブロックされませんが、キャッシュを更新すると完了するまですべての購読者がブロックされます。かなり高い割合の読み込みが更新されるとしても、このキャッシュは有効です。

D.2 オンライン上の関連情報

ウィキ記事 : <http://community.jboss.org/docs/DOC-15569>

ソースの場所 : <http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples/tree/master/ch07-rsscache/>

D.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

D.3.1 実装リソース

D.3.1.1 ProtectExportUtil.java

```
package org.jboss.ejb3.examples.ch07.rsscache.impl.rome;
```

```
import java.net.MalformedURLException;
import java.net.URL;

/**
 * 可変状態がエクスポートされるのを防ぐための
 * パッケージプライベートユーティリティ
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
class ProtectExportUtil
{
    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * 内部コンストラクタ：インスタンス化を防ぐ
     */
    private ProtectExportUtil()
    {
    }

    //-----||
    // 機能メソッド -----||
    //-----||

    /**
     * 指定された URL のコピーを返す。可変の内部状態が
     * クライアントに漏洩しないようにするために使う。
     * @param url
     * @return
     */
    static URL copyUrl(final URL url)
    {
        // null の場合は、null を返す
        if (url == null)
        {
            return url;
        }

        try
        {
            // コピー
            return new URL(url.toExternalForm());
        }
        catch (final MalformedURLException e)
    }
}
```

```

        {
            throw new RuntimeException("URL コピー時のエラー ", e);
        }
    }
}

```

D.3.1.2 RomeRssEntry.java

```

package org.jboss.ejb3.examples.ch07.rsscache.impl.rome;

import java.net.MalformedURLException;
import java.net.URL;

import org.jboss.ejb3.examples.ch07.rsscache.spi.RssEntry;

import com.sun.syndication.feed.synd.SyndContent;
import com.sun.syndication.feed.synd.SyndEntry;

/**
 * RSS エントリの java.net ROME 実装
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public class RomeRssEntry implements RssEntry
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * エントリの作者
     */
    private String author;

    /**
     * エントリの簡単な説明
     */
    private String description;

    /**
     * エントリのタイトル
     */
    private String title;

    /**
     * エントリへのリンク

```

```
    */
private URL url;

//-----||
// コンストラクタ -----||
//-----||

/**
 * コンストラクタ
 *
 * @param entry ROME API の RSS エントリ表現
 * @throws IllegalArgumentException エントリが指定されていない場合
 */
RomeRssEntry(final SyndEntry entry) throws IllegalArgumentException
{
    // プロパティを設定する
    this.author = entry.getAuthor();
    final SyndContent content = entry.getDescription();
    this.description = content.getValue();
    this.title = entry.getTitle();
    final String urlString = entry.getLink();
    URL url = null;
    try
    {
        url = new URL(urlString);
    }
    catch (final MalformedURLException murle)
    {
        throw new RuntimeException("ROME RSS エントリから無効な URL を取得 : " + entry, murle);
    }
    this.url = url;
}

//-----||
// 必要な実装 -----||
//-----||

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch07.rsscachе.spi.RssEntry#getAuthor()
 */
@Override
public String getAuthor()
{
    return this.author;
}
```

```
/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch07.rsscach.spi.RssEntry#getDescription()
 */
@Override
public String getDescription()
{
    return this.description;
}

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch07.rsscach.spi.RssEntry#getTitle()
 */
@Override
public String getTitle()
{
    return this.title;
}

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch07.rsscach.spi.RssEntry#getUrl()
 */
@Override
public URL getUrl()
{
    return ProtectExportUtil.copyUrl(this.url);
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/* (non-Javadoc)
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    final StringBuilder sb = new StringBuilder();
    sb.append(this.getTitle());
    sb.append(" - ");
    sb.append(this.url.toExternalForm());
    return sb.toString();
}
}
```

D.3.1.3 RssCacheBean.java

```

package org.jboss.ejb3.examples.ch07.rsscach.impl.rome;

import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import javax.annotation.PostConstruct;
import javax.ejb.ConcurrencyManagement;
import javax.ejb.ConcurrencyManagementType;
import javax.ejb.Lock;
import javax.ejb.LockType;
import javax.ejb.Remote;
import javax.ejb.Singleton;
import javax.ejb.Startup;

import org.jboss.ejb3.examples.ch07.rsscach.spi.RssCacheCommonBusiness;
import org.jboss.ejb3.examples.ch07.rsscach.spi.RssEntry;
import org.jboss.logging.Logger;

import com.sun.syndication.feed.synd.SyndEntry;
import com.sun.syndication.feed.synd.SyndFeed;
import com.sun.syndication.fetcher.FeedFetcher;
import com.sun.syndication.fetcher.FetcherException;
import com.sun.syndication.fetcher.impl.HttpClientFeedFetcher;
import com.sun.syndication.io.FeedException;

/**
 * アプリケーションデプロイ時にインスタンス化され、
 * RSS フィードのキャッシュされたビューを公開するシングルトン
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@Singleton
@Startup
@Remote(RssCacheCommonBusiness.class)
// コンテナ管理の並列性を明示的に宣言しているが、これは不要。これがデフォルト。
@ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)
public class RssCacheBean implements RssCacheCommonBusiness
{
    //-----||
    // クラスメンバー -----||
    //-----||

```

```

/**
 * ログガー
 */
private static final Logger log = Logger.getLogger(RssCacheBean.class);

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * RSS フィードを指す URL
 */
private URL url;

/**
 * フィードに対する、キャッシュされた RSS エントリ
 */
private List<RssEntry> entries;

//-----||
// 必要な実装 -----||
//-----||

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch07.rsscachspi.RssCacheCommonBusiness#getEntries()
 */
@Override
@Lock(LockType.READ)
public List<RssEntry> getEntries()
{
    return entries;
}

/* (non-Javadoc)
 * @see org.jboss.ejb3.examples.ch07.rsscachspi.RssCacheCommonBusiness#getUrl()
 */
@Override
@Lock(LockType.READ)
public URL getUrl()
{
    // 可変状態をクライアントにエクスポートしないように、コピーを返す。
    return ProtectExportUtil.copyUrl(this.url);
}

/**
 * @see org.jboss.ejb3.examples.ch07.rsscachspi.RssCacheCommonBusiness#refresh()

```

```
* @throws IllegalStateException URL が設定されていない場合
*/
@PostConstruct
@Override
// 完了するまで、ここですべての読み書きをブロック。省略可能なメタデータ、WRITE がデフォルト
@Lock(LockType.WRITE)
public void refresh() throws IllegalStateException
{

    // URL を取得
    final URL url = this.url;
    if (url == null)
    {
        throw new IllegalStateException(" フィード URL が設定されていません ");
    }
    log.info(" リクエストされた URL : " + url);

    // フィードを取得
    final FeedFetcher feedFetcher = new HttpClientFeedFetcher();
    SyndFeed feed = null;
    try
    {
        feed = feedFetcher.retrieveFeed(url);
    }
    catch (final FeedException fe)
    {
        throw new RuntimeException(fe);
    }
    catch (final FetcherException fe)
    {
        throw new RuntimeException(fe);
    }
    catch (final IOException ioe)
    {
        throw new RuntimeException(ioe);
    }

    // エントリの新しいリストを作成
    final List<RssEntry> rssEntries = new ArrayList<RssEntry>();

    // 各エントリに対して
    @SuppressWarnings("unchecked")
    // ROME API は総称 (Generics) を提供しないので、警告を抑制
    final List<SyndEntry> list = (List<SyndEntry>) feed.getEntries();
    for (final SyndEntry entry : list)
    {
```

```

        // 新しいエントリを作成
        final RssEntry rssEntry = new RomeRssEntry(entry);

        // リストに配置
        rssEntries.add(rssEntry);
        log.debug(" 新規 RSS エントリを発見: " + rssEntry);
    }

    // クライアントビューへの公開による変更からエントリを保護
    final List<RssEntry> protectedEntries = Collections.unmodifiableList(rssEntries);

    // キャッシュにエントリを設定する
    this.entries = protectedEntries;
}

//-----||
// 内部ヘルパーメソッド -----||
//-----||

/**
 * フィードを指す URL を設定する
 *
 * @param url
 * @throws IllegalArgumentException URL が null の場合
 */
void setUrl(final URL url) throws IllegalArgumentException
{
    // URL を設定する
    this.url = url;

    // 更新
    this.refresh();
}
}

```

D.3.1.4 RssCacheCommonBusiness.java

```

package org.jboss.ejb3.examples.ch07.rsscache.spi;

import java.net.URL;
import java.util.List;

/**
 * RSS フィードのキャッシュされたビューを公開する
 * 共通ビジネスインタフェース (EJB コンテナ)
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>

```

```

*/
public interface RssCacheCommonBusiness
{
    // -----||
    // 規約 -----||
    // -----||

    /**
     * {@link RssCacheCommonBusiness#getUrl()} で表される
     * RSS フィードのすべてのエントリを返す。
     * このリストは可変ではなく、読み取り専用である。
     */
    List<RssEntry> getEntries();

    /**
     * RSS フィードの URL を返す
     *
     * @return
     */
    URL getUrl();

    /**
     * キャッシュを消去し、フィードのエントリを更新
     */
    void refresh();
}

```

D.3.1.5 RssEntry.java

```

package org.jboss.ejb3.examples.ch07.rsscache.spi;

import java.net.URL;

/**
 * 1 つの RSS エントリの規約を定める
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface RssEntry
{
    // -----||
    // 規約 -----||
    // -----||

    /**
     * エントリの作者を取得する
     *

```

```
    * @return
    */
    String getAuthor();

    /**
     * エントリのタイトルを取得する
     *
     * @return
     */
    String getTitle();

    * エントリへの URL リンクを取得する
    *
    * @return
    */
    URL getUrl();

    /**
     * エントリの簡単な説明を取得する
     *
     * @return
     */
    String getDescription();
}
```

D.3.2 テストリソース

D.3.2.1 RssCacheTestCaseBase.java

```
package org.jboss.ejb3.examples.ch07.rsscachec;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import junit.framework.Assert;

import org.jboss.ejb3.examples.ch07.rsscache.spi.RssCacheCommonBusiness;
import org.jboss.ejb3.examples.ch07.rsscache.spi.RssEntry;
import org.jboss.logging.Logger;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import org.mortbay.jetty.Handler;
import org.mortbay.jetty.Server;
import org.mortbay.jetty.handler.AbstractHandler;

/**
 * RssCache @Singleton テストクラスのベーステスト
 * 単体テストや統合テスト
 * で拡張される
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public abstract class RssCacheTestCaseBase
{
    //-----||
    // クラスメンバー -----||
    //-----||

    private static final Logger log = Logger.getLogger(RssCacheTestCaseBase.class);

    /**
     * デフォルト RSS フィードで予想される RSS エントリ数
     */
    private static final int EXPECTED_15_RSS_ENTRIES = 15;

    /**
     * 5 つのエントリを持つ RSS フィードで予想される RSS エントリ数
     */
    private static final int EXPECTED_5_RSS_ENTRIES = 5;

    /**
     * テストで使うモック RSS フィードを含むファイル名
     */
    static final String FILENAME_RSS MOCK_FEED_15_ENTRIES = "15_entries.rss";

    /**
     * テストで使うモック RSS フィードを含むファイル名
     */
}
```

```

static final String FILENAME_RSS MOCK_FEED_5_ENTRIES = "5_entries.rss";

/**
 * HTTP サーバで要求される対象 RSS フィードのファイル名
 */
static final String FILENAME_RSS_FEED = "feed.rss";

/**
 * テスト HTTP サーバが接続すべきポート
 */
static final int HTTP_TEST_BIND_PORT = 12345;

/**
 * RSS フィードのコンテンツタイプ
 */
private static final String CONTENT_TYPE_RSS = "text/rss";

/**
 * モック RSS ファイルを配信するのに使う HTTP サーバ
 */
static Server httpServer;

/**
 * 改行文字
 */
private static final char NEWLINE = '\n';

//-----||
// ライフサイクル -----||
//-----||

/**
 * 内蔵 HTTP サーバを起動し、
 * モック RSS ファイルを配信するする RSS file
 * (ROME FeedFetcher は file:/URL からの取得はサポートしない)
 */
@BeforeClass
public static void startHttpServer()
{
    // 内蔵 HTTP サーバを起動
    final Handler handler = new StaticFileHandler();
    final Server httpServer = new Server(HTTP_TEST_BIND_PORT);
    httpServer.setHandler(handler);
    try
    {
        httpServer.start();
    }
}

```

```
    }
    catch (final Exception e)
    {
        throw new RuntimeException("Could not start server");
    }
    log.info("HTTP Server Started: " + httpServer);
    RssCacheUnitTestCase.httpServer = httpServer;
}

/**
 * デフォルトのモックテンプレートから RSS フィードファイルを作成
 */
@BeforeClass
public static void createRssFeedFile() throws Exception
{
    writeToRssFeedFile(getMock15EntriesRssFile());
}

/**
 * 内蔵 HTTP サーバをシャットダウンして削除
 */
@AfterClass
public static void shutdownHttpServer()
{
    if (httpServer != null)
    {
        try
        {
            httpServer.stop();
        }
        catch (final Exception e)
        {
            // 受け入れる
            log.error("Could not stop HTTP Server cleanly", e);
        }
        log.info("HTTP Server Stopped: " + httpServer);
        httpServer = null;
    }
}

/**
 * RSS フィードファイルを削除
 */
@AfterClass
public static void deleteRssFeedFile() throws Exception
{

```

```

    final File rssFile = getRssFeedFile();
    boolean deleted = rssFile.delete();
    if (!deleted)
    {
        log.warn("RSS Feed File was not cleaned up properly: " + rssFile);
    }
}

//-----||
// テスト -----||
//-----||

/**
 * RSS エントリが初期化され、予想どおりに解析されていることを確認
 * さらに、{@link RssCacheCommonBusiness#refresh()} がキャッシュを消去し、
 * 期待どおりに機能することを確認
 */
@Test
public void testRssEntries() throws Exception
{
    // ログに記録
    log.info("testRssEntries");

    // RSS キャッシュ Bean を取得
    final RssCacheCommonBusiness rssCache = this.getRssCacheBean();

    // すべてのエントリを取得
    final List<RssEntry> rssEntries = rssCache.getEntries();
    log.info("Got entries: " + rssEntries);

    // エントリが初期化され、適切なサイズに解析されることを確認
    this.ensureExpectedEntries(rssEntries, EXPECTED_15_RSS_ENTRIES);

    // RSS フィードファイルの内容を書き出すので、更新すると新しい内容になる
    writeToRssFeedFile(getMock5EntriesRssFile());

    // 更新
    rssCache.refresh();

    // すべてのエントリを取得
    final List<RssEntry> rssEntriesAfterRefresh = rssCache.getEntries();
    log.info("Got entries after refresh: " + rssEntriesAfterRefresh);

    // エントリが初期化され、適切なサイズに解析されることを確認
    this.ensureExpectedEntries(rssEntriesAfterRefresh, EXPECTED_5_RSS_ENTRIES);
}

```

```

// 元の 15 個のモックエントリに戻す
writeToRssFeedFile(getMock15EntriesRssFile());
rssCache.refresh();

// 通常に戻っていることを確認
final List<RssEntry> rssEntriesAfterRestored = rssCache.getEntries();
log.info("Got entries: " + rssEntriesAfterRestored);
this.ensureExpectedEntries(rssEntriesAfterRestored, EXPECTED_15_RSS_ENTRIES);
}

//-----||
// 規約 -----||
//-----||

/**
 * このテストに使う RssCache Bean を取得
 */
protected abstract RssCacheCommonBusiness getRssCacheBean();

//-----||
// 内部ヘルパーメソッド -----||
//-----||

/**
 * RSS エントリを解析し、指定された数であることを確認
 * @param entries
 * @param expectedSize
 */
private void ensureExpectedEntries(final List<RssEntry> entries, final int expectedSize)
{
    // エントリが初期化され、適切なサイズに解析されることを確認
    Assert.assertNotNull("RSS Entries was either not initialized or is returning null", entries);
    final int actualSize = entries.size();
    Assert.assertEquals("Wrong number of RSS entries parsed out from feed", expectedSize,
actualSize);
    log.info("Got expected " + expectedSize + " RSS entries");
}

/**
 * コードソースのベースを取得
 */
private static URL getCodebaseLocation()
{
    return RssCacheUnitTestCase.class.getProtectionDomain().getCodeSource().getLocation();
}

```

```
/**
 * テンプレートファイルの内容を RSS フィードファイルに書き込む
 *
 * @param templateFile
 * @throws Exception
 */
private static void writeToRssFeedFile(final File templateFile) throws Exception
{
    // 対象ファイルへのライタを取得
    final File rssFile = getRssFeedFile();
    final PrintWriter writer = new PrintWriter(new BufferedWriter(new FileWriter(rssFile)));

    // デフォルトのモックテンプレートファイルのリーダを取得
    final BufferedReader reader = new BufferedReader(new FileReader(templateFile));

    // 読み取りと書き込み
    String line = null;
    while ((line = reader.readLine()) != null)
    {
        writer.write(line);
        writer.write(NEWLINE);
    }

    // フラッシュして閉じる
    writer.flush();
    writer.close();
    reader.close();
}

/**
 * サーバが提供する RSS フィードファイルを取得
 * @return
 * @throws Exception
 */
private static File getRssFeedFile() throws Exception
{
    final File baseFile = getBaseDirectory();
    final File rssFile = new File(baseFile, FILENAME_RSS_FEED);
    return rssFile;
}

/**
 * 15 個のエントリを持つモック RSS テンプレートファイルを取得
 * @return
 * @throws Exception
 */
```

```
*/
private static File getMock15EntriesRssFile() throws Exception
{
    return getFileFromBase(FILENAME_RSS MOCK_FEED_15_ENTRIES);
}

/**
 * 5 個のエントリを持つモック RSS テンプレートファイルを取得
 * @return
 * @throws Exception
 */
private static File getMock5EntriesRssFile() throws Exception
{
    return getFileFromBase(FILENAME_RSS MOCK_FEED_5_ENTRIES);
}

/**
 * 指定された名前のファイルをベースディレクトリから取得
 *
 * @param filename
 * @return
 * @throws Exception
 */
private static File getFileFromBase(final String filename) throws Exception
{
    final File baseFile = getBaseDirectory();
    final File mockTemplateFile = new File(baseFile, filename);
    return mockTemplateFile;
}

/**
 * テストファイルが存在するベースディレクトリを取得
 * @return
 */
private static File getBaseDirectory() throws Exception
{
    final URL baseLocation = getCodebaseLocation();
    final URI baseUri = baseLocation.toURI();
    final File baseFile = new File(baseUri);
    return baseFile;
}

//-----||
// 内部クラス -----||
//-----||
```

```
/**
 * Web ルートから静的文字ファイルを提供するための Jetty ハンドラ
 */
private static class StaticFileHandler extends AbstractHandler implements Handler
{
    /**
     * (non-Javadoc)
     * @see org.mortbay.jetty.Handler#handle(java.lang.String, javax.servlet.http.HttpServletRequest,
     javax.servlet.http.HttpServletResponse, int)
     */
    public void handle(final String target, final HttpServletRequest request,
        final HttpServletResponse response, final int dispatch) throws IOException, ServletException
    {
        // ストリームに書き込む前にコンテンツタイプと状態を設定
        response.setContentType(CONTENT_TYPE_RSS);
        response.setStatus(HttpServletResponse.SC_OK);

        // Web ルートに対して相対的な要求ファイルを取得
        final URL root = getCodebaseLocation();
        final URL fileUrl = new URL(root.toExternalForm() + target);
        URI uri = null;
        try
        {
            uri = fileUrl.toURI();
        }
        catch (final URISyntaxException urise)
        {
            throw new RuntimeException(urise);
        }
        final File file = new File(uri);

        // ファイルが見つからない場合は 404 エラー
        if (!file.exists())
        {
            response.setStatus(HttpServletResponse.SC_NOT_FOUND);
            log.warn("Requested file is not found: " + file);
            return;
        }

        // 各行を書き出す
        final BufferedReader reader = new BufferedReader(new FileReader(file));
        final PrintWriter writer = response.getWriter();
        String line = null;
        while ((line = reader.readLine()) != null)
        {
            writer.println(line);
        }
    }
}
```

```

    }

    // フラッシュして閉じる
    writer.flush();
    reader.close();
    writer.close();
  }
}
}
}

```

D.3.2.2 RssCacheUnitTestCase.java

```

package org.jboss.ejb3.examples.ch07.rsscachec;

import java.net.MalformedURLException;
import java.net.URL;

import org.jboss.ejb3.examples.ch07.rsscachec.impl.rome.TestRssCacheBean;
import org.jboss.ejb3.examples.ch07.rsscachec.spi.RssCacheCommonBusiness;
import org.jboss.logging.Logger;
import org.junit.AfterClass;
import org.junit.BeforeClass;

/**
 * RssCache の単体テスト
 * EJB コンテナのコンテキスト外で POJO として使う
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public class RssCacheUnitTestCase extends RssCacheTestCaseBase
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * □ガー
     */
    private static final Logger log = Logger.getLogger(RssCacheUnitTestCase.class);

    /**
     * テストする Bean (POJO) インスタンス。@Singleton EJB をモック
     */
    private static RssCacheCommonBusiness bean;

    //-----||
    // ライフサイクル -----||

```

```

//-----||

/**
 * テストを実行する前に本物のコンテナ EJB @Singleton
 * をモックする POJO インスタンスを生成
 */
@BeforeClass
public static void createPojo()
{
    // 初期化と設定
    final TestRssCacheBean bean = new TestRssCacheBean();
    RssCacheUnitTestCase.bean = bean;
    log.info("Created POJO instance: " + bean);

    // モック RSS ファイルの URL を設定
    URL url = null;
    try
    {
        url = new URL(getBaseConnectUrl(), FILENAME_RSS_FEED);
    }
    catch (final MalformedURLException murle)
    {
        throw new RuntimeException("Error in test setup while constructing the mock RSS feed URL",
murle);
    }
    bean.setUrl(url);

    // Mock container initialization upon the bean
    bean.refresh();
}

/**
 * すべてのテストの実行後に POJO インスタンスを null にリセット
 */
@AfterClass
public static void clearPojo()
{
    // null に設定するので、テスト実行間にインスタンスは漏洩しない
    bean = null;
}

//-----||
// 必要な実装 -----||
//-----||

/**

```

```

    * {@inheritDoc}
    * @see org.jboss.ejb3.examples.ch07.envinfo.EnvironmentInformationTestCaseBase#getEnvInfoBean()
    */
    @Override
    protected RssCacheCommonBusiness getRssCacheBean()
    {
        return bean;
    }

    //-----||
    // 内部ヘルパーメソッド -----||
    //-----||

    /**
     * コードソースのベースを取得
     */
    private static URL getBaseConnectUrl()
    {
        try
        {
            return new URL("http://localhost:" + HTTP_TEST_BIND_PORT);
        }
        catch (final MalformedURLException e)
        {
            throw new RuntimeException("Error in creating the base URL during set setup", e);
        }
    }
}

```

D.3.2.3 TestRssCacheBean.java

```

package org.jboss.ejb3.examples.ch07.rsscachel.impl.rome;

import java.net.URL;

/**
 * テスト用にフィード URL の設定をサポートする
 * RSS キャッシュ Bean の拡張
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public class TestRssCacheBean extends RssCacheBean
{

    //-----||
    // オーバーライドされた実装 -----||
}

```

```
//-----||  
  
/* (non-Javadoc)  
 * @see org.jboss.ejb3.examples.ch07.rsscach.impl.rome.RssCacheBean#setUrl(java.net.URL)  
 */  
@Override  
public void setUrl(final URL url) throws IllegalArgumentException  
{  
    super.setUrl(url);  
}  
}
```

D.3.2.4 jndi.properties

```
# JBoss アプリケーションサーバネーミングサービスとリモートでやり取りするための JNDI プロパティ  
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory  
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces  
java.naming.provider.url=jnp://localhost:1099
```


付録 E

メッセージ駆動型 EJB : 状態更新リスナの例

E.1 説明

これまで説明してきたように、セッション Bean はクライアントの要求に対処するのに最適です。しかし、多くのエンタープライズシステムでは、要求をアプリケーション間で非同期に渡すためにメッセージングレイヤを使います。Java では、Java Message Service という抽象概念を使ってキューやトピックからメッセージをやり取りし、JMS と EJB を統合するのがメッセージ駆動型 Bean です。

この例では、コンシューマ / パブリッシャトピックを介してソーシャルネットワーキング状態の更新を実装します。このトピックをリスンしているすべてのユーザは状態更新を受信し、ここでは2つの受信者を作成します。1つの簡単な受信者はコマンドラインやログファイルへログを書き出し、もう1つは更新を Twitter へ書き出します。

注：テスト実行中に Twitter 更新を活用するために、実行前に以下の環境変数を設定してください。

```
OREILLY_EJB_BOOK_CH08_TWITTER_USERNAME  
OREILLY_EJB_BOOK_CH08_TWITTER_PASSWORD
```

さらに、この例では、テストとサーバに1つのJVMを使って共有メモリロック（`java.util.concurrent`）を可能にし、非同期コンポーネントが期待どおりに完成していることを確実にテストできる方法も示します。

E.2 オンライン上の関連情報

ウィキ記事：<http://community.jboss.org/docs/DOC-15570>

ソースの場所：<http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples/tree/master/ch08-statusupdate/>

E.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

E.3.1 実装リソース

E.3.1.1 StatusUpdate.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.api;

import java.io.Serializable;

/**
 * 1つの汎用的な状態更新をカプセル化する
 *
 * この実装は JMS と EJB MDB の処理の例を示すためだけに使うので、
 * {@link Serializable} の厳密な規約には従わず、
 * デフォルトのシリアライズされた形式
 * を受け入れる。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public class StatusUpdate implements Serializable
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * serialVersionUID
     */
    private static final long serialVersionUID = 1L;

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 初期状態
     */
    private final String status;

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * 指定された新しい状態で、新たな状態更新を作成
     *
     * @throws IllegalArgumentException 状態またはユーザ名が指定されていない場合
     */
    public StatusUpdate(final String status) throws IllegalArgumentException

```

```
{
    // 前提条件を確認
    if (status == null || status.length() == 0)
    {
        throw new IllegalArgumentException("Status must be specified");
    }

    // 設定
    this.status = status;
}

//-----||
// 機能メソッド -----||
//-----||

/**
 * 新しい状態を返す
 * @return 状態
 */
public String getText()
{
    return status;
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    return this.getClass().getSimpleName() + " [status=" + status + "];"
}

/**
 * {@inheritDoc}
 * @see java.lang.Object#hashCode()
 */
@Override
public int hashCode()
{
    final int prime = 31;
```

```

        int result = 1;
        result = prime * result + ((status == null) ? 0 : status.hashCode());
        return result;
    }

    /**
     * {@inheritDoc}
     * @see java.lang.Object#equals(java.lang.Object)
     */
    @Override
    public boolean equals(Object obj)
    {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        StatusUpdate other = (StatusUpdate) obj;
        if (status == null)
        {
            if (other.status != null)
                return false;
        }
        else if (!status.equals(other.status))
            return false;
        return true;
    }
}

```

E.3.1.2 StatusUpdateConstants.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.api;

import javax.management.ObjectName;

/**
 * StatusUpdate MDB のクライアントが共有する
 * リソースを参照するのに使う定数を含む
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public interface StatusUpdateConstants
{
    //-----||
    // 定数 -----||
    //-----||
}

```

```

/**
 * 状態更新のための pub/sub トピックの JNDI 名
 */
String JNDI_NAME_TOPIC_STATUSUPDATE = "topic/StatusUpdate";

/**
 * StatusUpdate MDB が使う送信先の種類
 */
String TYPE_DESTINATION_STATUSUPDATE = "javax.jms.Topic";

/**
 * トピックの依存関係名として使う JMX {@link ObjectName}
 */
String OBJECT_NAME_TOPIC_STATUSUPDATE = "jboss.messaging.destination:service=Topic,name=StatusUpdate";
}

```

E.3.1.3 EnvironmentSpecificTwitterUtil.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import twitter4j.Twitter;

/**
 * {@link Twitter} クライアントの新しいインスタンスを生成するのに使う
 * ステートレスクラス。実際にはこの方法は決して使わないが、
 * この生成では環境から取得したユーザ名 / パスワードの認証情報を頼りにする。
 * この例では、プロパティを使う EJB を設定できるようにするために
 * プロパティを外部化する必要があるが、
 * セキュリティのためにデフォルト値を隠す必要がある。
 *
 * 実際のシステムではこの方法を使うべきではない。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public class EnvironmentSpecificTwitterClientUtil
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * Twitter ユーザ名の環境変数
     */
    private static final String ENV_VAR_NAME_TWITTER_USERNAME = "OREILLY_EJB_BOOK_CHO8_TWITTER_

```

```

USERNAME";

/**
 * Twitter パスワードの環境変数
 */
private static final String ENV_VAR_NAME_TWITTER_PASSWORD = "OREILLY_EJB_BOOK_CH08_TWITTER_PASSWORD";

/**
 * 環境が Twitter 統合をサポートしていないことを示すメッセージ
 */
static final String MSG_UNSUPPORTED_ENVIRONMENT = "Both environment variables
¥" + ENV_VAR_NAME_TWITTER_USERNAME + "¥" and ¥" + ENV_VAR_NAME_TWITTER_PASSWORD + "¥" must be
specified for this test to run";

//-----||
// インスタンスメンバー -----||
//-----||

//-----||
// コンストラクタ -----||
//-----||

private EnvironmentSpecificTwitterClientUtil()
{
    throw new UnsupportedOperationException("No instantiation allowed");
}

//-----||
// ユーティリティメソッド -----||
//-----||

static boolean isSupportedEnvironment()
{
    // ユーザ名とパスワードを取得する
    final UsernamePasswordCredentials creds = getCredentials();
    final String username = creds.username;
    final String password = creds.password;

    /*
     * ユーザ名とパスワードが指定されている場合だけ続ける。それ以外の場合は
     * 警告をログに記録してテストを省略する。通常は環境に基づいて
     * テストすべきではないが、セキュリティの制約のため SVN にユーザ名と
     * パスワードのペアを置けず、このテストは制御やローカルでのモックができない
     * 外部サービスとやり取りする (MDB を使って他のシステムと非同期に
     * 統合する方法を示すという目的を果たせなくなる)。
     */
}

```

```

    * 通常はまず実行環境で実施するが、この例のすべてのユーザが
    * Twitter アカウントを持っているとみなすことはできない。
    */
    if (username == null || password == null)
    {
        return false;
    }

    // すべてが適切である
    return true;
}

/**
 * 環境で指定されたユーザ名とパスワードの Twitter クライアントを取得する。
 * 環境が完全に設定されていない場合は、{@link IllegalStateException}
 * が発行され、環境変数を適切に設定すべきことを示す。ISE を避けるために、
 * {@link EnvironmentSpecificTwitterClientUtil#isSupportedEnvironment()}
 * を使って環境の完全性をまず調べる。
 *
 * @throws IllegalStateException 環境が Twitter クライアントの作成をサポートしていない場合
 */
static Twitter getTwitterClient() throws IllegalStateException
{
    // ユーザ名とパスワードを取得する
    final String username = SecurityActions.getEnvironmentVariable(ENV_VAR_NAME_TWITTER_USERNAME);
    final String password = SecurityActions.getEnvironmentVariable(ENV_VAR_NAME_TWITTER_PASSWORD);

    /*
     * ユーザ名とパスワードの両方が設定されている場合だけサポートされる
     */
    if (!isSupportedEnvironment())
    {
        throw new IllegalStateException(MSG_UNSUPPORTED_ENVIRONMENT);
    }

    // Twitter クライアントを取得する
    final Twitter twitterClient = new Twitter(username, password);

    // 戻す
    return twitterClient;
}

//-----||
// 内部ヘルパーメソッド -----||
//-----||

```

```

/**
 * 環境からユーザ名とパスワードの認証情報を取得する
 * @return
 */
private static UsernamePasswordCredentials getCredentials()
{
    // ユーザ名とパスワードを取得する
    final String username = SecurityActions.getEnvironmentVariable(ENV_VAR_NAME_TWITTER_USERNAME);
    final String password = SecurityActions.getEnvironmentVariable(ENV_VAR_NAME_TWITTER_PASSWORD);

    // 統一ビューとして戻す
    final UsernamePasswordCredentials creds = new UsernamePasswordCredentials();
    creds.username = username;
    creds.password = password;
    return creds;
}

//-----||
// 内部クラス -----||
//-----||

/**
 * ユーザ名 / パスワードのペアをカプセル化する簡単な値オブジェクト
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
private static class UsernamePasswordCredentials
{
    private String username;
    private String password;
}
}

```

E.3.1.4 LoggingStatusUpdateMdb.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import java.util.logging.Logger;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.MessageListener;

import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdate;
import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdateConstants;

```

```

/**
 * {@link MessageListener#onMessage(javax.jms.Message)} が INFO レベルで
 * 状態更新をログに書き出す MDB
 *
 * 例で明示的にはテストしないが（ロギングはテキストできないため）、
 * この利用方法が実例となるはずである
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@MessageDriven(activationConfig =
{
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
StatusUpdateConstants.TYPE_DESTINATION_STATUSUPDATE),
    @ActivationConfigProperty(propertyName = "destination", propertyValue = StatusUpdateConstants.
JNDI_NAME_TOPIC_STATUSUPDATE)})
public class LoggingStatusUpdateMdb extends StatusUpdateBeanBase implements MessageListener
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(LoggingStatusUpdateMdb.class.getName());

    //-----||
    // 必要な実装 -----||
    //-----||

    /**
     * INFO レベルで状態をログに記録
     * @see org.jboss.ejb3.examples.ch08.statusupdate.mdb.StatusUpdateBeanBase#updateStatus(org.jboss.
ejb3.examples.ch08.statusupdate.api.StatusUpdate)
     */
    @Override
    public void updateStatus(final StatusUpdate newStatus) throws IllegalArgumentException, Exception
    {
        // 前提条件を確認
        if (newStatus == null)
        {
            throw new IllegalArgumentException("status must be specified");
        }

        // 情報を取得する

```

```
        final String status = newStatus.getText();

        // ログに記録
        log.info("新しい状態を受信 : ¥" + status + "¥");
    }

}
```

E.3.1.5 SecurityActions.java

```
package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import java.security.AccessController;
import java.security.PrivilegedAction;

/**
 * このパッケージ以外に漏洩しないように保護されたセキュリティ処理
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
class SecurityActions
{

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * 外部での初期化はない
     */
    private SecurityActions()
    {

    }

    //-----||
    // ユーティリティメソッド -----||
    //-----||

    /**
     * スレッドコンテキストクラスローダを取得する
     */
    static ClassLoader getThreadContextClassLoader()
    {
        return AccessController.doPrivileged(GetTcclAction.INSTANCE);
    }
}
```

```
/**
 * 現在のスレッドのコンテキストに指定されたクラスローダを設定する
 *
 * @param cl
 * @throws IllegalArgumentException クラスローダが null の場合
 */
static void setThreadContextClassLoader(final ClassLoader cl) throws IllegalArgumentException
{
    if (cl == null)
    {
        throw new IllegalArgumentException("ClassLoader was null");
    }

    AccessController.doPrivileged(new PrivilegedAction<Void>()
    {
        public Void run()
        {
            Thread.currentThread().setContextClassLoader(cl);
            return null;
        }
    });
}

/**
 * 指定された名前の環境変数を取得する
 * 存在しない場合は null
 * @param envVarName
 * @return
 * @throws IllegalArgumentException 環境変数名が指定されていない場合
 */
static String getEnvironmentVariable(final String envVarName) throws IllegalArgumentException
{
    // 前提条件を確認
    if (envVarName == null || envVarName.length() == 0)
    {
        throw new IllegalArgumentException("Environment variable name was not specified");
    }

    // 戻す
    return AccessController.doPrivileged(new GetEnvironmentVariableAction(envVarName));
}

/**
 * 指定されたキーを持つシステムプロパティを取得する
 *
 * @param key
```

```
* @return
* @throws IllegalArgumentException キーが null の場合
*/
static String getSystemProperty(final String key) throws IllegalArgumentException
{
    // 前提条件を確認
    if (key == null)
    {
        throw new IllegalArgumentException("key was null");
    }

    // システムプロパティを取得する
    return AccessController.doPrivileged(new GetSystemPropertyAction(key));
}

//-----||
// 内部クラス -----||
//-----||

/**
 * TCCL を取得するための {@link PrivilegedAction} 処理
 */
private enum GetTcclAction implements PrivilegedAction<ClassLoader> {
    INSTANCE;

    @Override
    public ClassLoader run()
    {
        return Thread.currentThread().getContextClassLoader();
    }
}

/**
 * 環境変数にアクセスするための {@link PrivilegedAction}
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
private static class GetEnvironmentVariableAction implements PrivilegedAction<String>
{
    /**
     * 取得する環境変数名
     */
    private String envVarName;
```

```
/**
 * 指定された環境変数名を取得できる新しいインスタンスを生成する
 * @param envVarName
 */
public GetEnvironmentVariableAction(final String envVarName)
{
    this.envVarName = envVarName;
}

/**
 * {@inheritDoc}
 * @see java.security.PrivilegedAction#run()
 */
@Override
public String run()
{
    return System.getenv(envVarName);
}
}

/**
 * システムプロパティにアクセスするための {@link PrivilegedAction}
 *
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
private static class GetSystemPropertyAction implements PrivilegedAction<String>
{

    /**
     * 取得するシステムプロパティ名
     */
    private String sysPropName;

    /**
     * 指定されたシステムプロパティを名前で取得できる新しいインスタンスを生成する
     * @param sysPropName
     */
    public GetSystemPropertyAction(final String sysPropName)
    {
        this.sysPropName = sysPropName;
    }

    /**
```

```

    * {@inheritDoc}
    * @see java.security.PrivilegedAction#run()
    */
    @Override
    public String run()
    {
        return System.getProperty(sysPropName);
    }
}
}
}

```

E.3.1.6 StatusUpdateBeanBase.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import java.io.Serializable;
import java.util.logging.Logger;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;

import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdate;

/**
 * StatusUpdateEJB のベースサポート
 * 受信 JMS メッセージを消費し、
 * {@link StatusUpdateBeanBase#updateStatus(StatusUpdate)} に渡す役割を担う。
 * 子クラスはこのメソッドを特殊化する必要がある。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public abstract class StatusUpdateBeanBase implements MessageListener
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * □ガー
     */
    private static final Logger log = Logger.getLogger(StatusUpdateBeanBase.class.getName());

    //-----||
    // 規約 -----||
    //-----||

```

```

/**
 * 状態を指定された値に更新
 *
 * @throws IllegalArgumentException 新しい状態が指定されていない場合
 * @throws Exception 処理中にエラーが発生した場合
 */
public abstract void updateStatus(StatusUpdate newStatus) throws IllegalArgumentException, Exception;

//-----||
// 必要な実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see javax.jms.MessageListener#onMessage(javax.jms.Message)
 */
@Override
public void onMessage(final Message message)
{
    /**
     * 前提条件を確認
     *
     */
    // メッセージが指定されていることを確認
    if (message == null)
    {
        throw new IllegalArgumentException("メッセージが指定されていなければなりません");
    }

    // メッセージが期待どおりの形式であることを確認
    final ObjectMessage objMessage;
    if (message instanceof ObjectMessage)
    {
        objMessage = (ObjectMessage) message;
    }
    else
    {
        throw new IllegalArgumentException("指定されたメッセージは次の型でなくてはなりません：" +
            ObjectMessage.class.getName());
    }

    // 埋め込まれている状態更新を抽出
    final Serializable obj;
    try
    {
        obj = objMessage.getObject();
    }

```

```
    }
    catch (final JMSEException jmse)
    {
        throw new IllegalArgumentException("メッセージの内容を取得できません：" + objMessage);
    }

    // 期待どおりの型であることを確認
    final StatusUpdate status;
    if (obj instanceof StatusUpdate)
    {
        status = (StatusUpdate) obj;
    }
    else
    {
        throw new IllegalArgumentException("メッセージの内容は次の型でなくてはなりません：" +
            StatusUpdate.class.getName() + "：次の型でした：" + obj);
    }

    // 更新を処理
    try
    {
        this.updateStatus(status);
    }
    catch (final Exception e)
    {
        throw new RuntimeException("次の状態更新の処理で問題に遭遇しました：" + status, e);
    }
}
}
```

E.3.1.7 TwitterUpdateMdb.java

```
package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import java.util.logging.Logger;

import javax.annotation.PostConstruct;
import javax.jms.MessageListener;

import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdate;

import twitter4j.Twitter;

/**
 * Twitter API に対するアダプタの役割をする EJB 3.x MDB
 * 受信メッセージの Twitter 状態を更新
 *
 */
```

```

* 環境はまず、環境プロパティから入手できるユーザ名とパスワードのペア
* で Twitter を統合する必要がある。
* 詳細は {@link EnvironmentSpecificTwitterClientUtil} を参照。
*
* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
* @see http://twitter.com
* @see http://yusuke.homeip.net/twitter4j/en/index.html
*/
public class TwitterUpdateMdb extends StatusUpdateBeanBase implements MessageListener
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(TwitterUpdateMdb.class.getName());

    /**
     * EJB 名
     */
    static final String NAME = "TwitterUpdateMdb";

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * 新しいインスタンスを生成する。仕様では引数なしのコンストラクタである必要がある。
     */
    public TwitterUpdateMdb()
    {

    }

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * Twitter API を呼び出すことで Twitter を更新するために使う、基礎をなすクライアント
     */
    private Twitter client;

```

```

//-----||
// ライフサイクル -----||
//-----||

/**
 * ライフサイクルは提供された環境プロパティから Twitter クライアントを作成
 * ことから始まる（環境がそのように設定されている場合）。
 */
@PostConstruct
void createTwitterClient()
{
    if (!EnvironmentSpecificTwitterClientUtil.isSupportedEnvironment())
    {
        log.warning(EnvironmentSpecificTwitterClientUtil.MSG_UNSUPPORTED_ENVIRONMENT);
        return;
    }

    // クライアントを作成
    client = EnvironmentSpecificTwitterClientUtil.getTwitterClient();
    log.info("Twitter クライアントを作成：" + client);
}

//-----||
// 必要な実装 -----||
//-----||

/**
 * 受信した状態更新をコンテキストプロパティで設定された
 * Twitter アカウントに送信
 *
 * @see org.jboss.ejb3.examples.ch08.statusupdate.mdb.StatusUpdateBeanBase#updateStatus(org.jboss.
 * ejb3.examples.ch08.statusupdate.api.StatusUpdate)
 */
@Override
public void updateStatus(final StatusUpdate newStatus) throws IllegalArgumentException, Exception
{
    // クライアントが初期化されていることを確認（環境が許す場合）
    if (!EnvironmentSpecificTwitterClientUtil.isSupportedEnvironment())
    {
        // 何も実行せずに戻る
        return;
    }
    if (client == null)
    {
        throw new IllegalStateException("Twitter クライアントが初期化されていません");
    }
}

```

```

    // 状態を抽出
    final String status = newStatus.getText();

    // 状態を更新
    client.updateStatus(status);
}
}

```

E.3.1.8 hornetq-jms.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="urn:hornetq"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:hornetq /schema/hornetq-jms.xsd">

  <topic name="StatusUpdate">
    <entry name="/topic/StatusUpdate" />
  </topic>

</configuration>

```

E.3.2 テストリソース

E.3.2.1 MockObjectMessage.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import java.io.Serializable;
import java.util.Enumeration;

import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.ObjectMessage;

/**
 * {@link ObjectMessage#getObject()} メソッドだけをサポートする
 * モック {@link ObjectMessage}。テストで使う。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public class MockObjectMessage implements ObjectMessage
{
    //-----||
    // クラスメンバー -----||
    //-----||

    private static final String MESSAGE_UNSUPPORTED = "This mock implementation does not support this

```

```
operation";

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * このメッセージに含まれるオブジェクト
 */
private final Serializable object;

//-----||
// コンストラクタ -----||
//-----||

/**
 * {@link ObjectMessage#getObject()} で返される
 * 指定された対応オブジェクトを持つ新しいインスタンスを生成
 */
MockObjectMessage(final Serializable object)
{
    this.object = object;
}

//-----||
// 必要な実装 -----||
//-----||

/* (non-Javadoc)
 * @see javax.jms.ObjectMessage#getObject()
 */
@Override
public Serializable getObject() throws JMSEException
{
    return this.object;
}

//-----||
// 未サポート -----||
//-----||

/*
 * これ以降はすべて呼び出し時に例外を発行
 */

/* (non-Javadoc)
```

```

    * @see javax.jms.ObjectMessage#setObject(java.io.Serializable)
    */
    @Override
    public void setObject(Serializable object) throws JMSException
    {
        throw new UnsupportedOperationException(MESSAGE_UNSUPPORTED);
    }

    // ... 簡潔にするために省略
}

```

E.3.2.2 StatusUpdateIntegrationTest.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import java.util.concurrent.TimeUnit;
import java.util.logging.Logger;

import javax.jms.Message;
import javax.jms.ObjectMessage;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicPublisher;
import javax.jms.TopicSession;
import javax.naming.Context;
import javax.naming.InitialContext;

import org.jboss.arquillian.api.Deployment;
import org.jboss.arquillian.api.Run;
import org.jboss.arquillian.api.RunModeType;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdate;
import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdateConstants;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.runner.RunWith;

import twitter4j.Twitter;

/**
 * StatusUpdateEJB の統合テスト
 * MDB が EJB コンテナ内で動作するときに予想どおりに
 * 機能することを確認
 *
 */

```

```
* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
*/
@RunWith(Arquillian.class)
@Run(RunModeType.AS_CLIENT)
public class StatusUpdateIntegrationTest extends StatusUpdateTestBase
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(StatusUpdateIntegrationTest.class.getName());

    /**
     * テスト用にサーバにデプロイするアーカイブの名前
     */
    private static final String NAME_MDB_ARCHIVE = "statusUpdateEjb.jar";

    /**
     * 新しいStatusUpdate JMS トピックを作成するデプロイメント記述子のためのClassLoader リソース名
     */
    private static final String NAME_RESOURCE_TOPIC_DEPLOYMENT = "hornetq-jms.xml";

    /**
     * JNDI コンテキスト
     */
    private static Context NAMING_CONTEXT;

    /**
     * JNDI 内のキュー接続ファクトリの名前
     */
    private static final String JNDI_NAME_CONNECTION_FACTORY = "ConnectionFactory";

    /**
     * Arquillian を使ってサーバにデプロイする EJB JAR を作成
     * @return
     */
    @Deployment
    public static JavaArchive getDeployment()
    {
        final JavaArchive archive = ShrinkWrap.create(NAME_MDB_ARCHIVE, JavaArchive.class).
            addClasses(StatusUpdate.class,
                StatusUpdateConstants.class, LoggingStatusUpdateMdb.class, StatusUpdateBeanBase.class,
```

```

        TwitterUpdateBlockingTestMdb.class, SecurityActions.class, TwitterUpdateMdb.class,
        EnvironmentSpecificTwitterClientUtil.class).addResource(NAME_RESOURCE_TOPIC_DEPLOYMENT);
log.info(archive.toString(true));

return archive;
}

//-----||
// ライフサイクル -----||
//-----||

/**
 * この JVM に組み込まれた新しい JBossAS サーバを作成して起動
 */
@BeforeClass
public static void setNamingContext() throws Exception
{
    // ネーミングコンテキストを設定
    NAMING_CONTEXT = new InitialContext();
}

//-----||
// テスト -----||
//-----||

/**
 * {@link TwitterUpdateMdb} がリスンしている JMS トピックから新しい状態を
 * 受信したときに Twitter を更新することを確認
 */
@Test
public void testTwitterUpdateMdb() throws Exception
{
    // Twitter クライアントを取得し、環境を遂行
    final Twitter twitterClient;
    try
    {
        twitterClient = EnvironmentSpecificTwitterClientUtil.getTwitterClient();
    }
    catch (final IllegalStateException ise)
    {
        log.warning(ise.getMessage() + "; skipping...");
        return;
    }

    // 新しい状態を作成
    final StatusUpdate newStatus = this.getUniqueStatusUpdate();

```

```
// JMS トピックへ更新を送信する (MDB サブスクライバがその更新を消費する)
this.publishStatusUpdateToTopic(newStatus);

// 別のスレッドで処理するため、MDB が処理を行うのを待つ。
// これはテストと同じ JVM で MDB をテストするときのみ可能
try
{
    log.info("Waiting on the MDB...");
    TwitterUpdateBlockingTestMdb.LATCH.await(10, TimeUnit.SECONDS);
}
catch (final InterruptedException e)
{
    // フラグを削除して再発行する。準備でエラーが発生。
    Thread.interrupted();
    throw new RuntimeException("Thread was interrupted while waiting for MDB processing;
    should not happen in this test");
}

log.info("MDB signaled it's done processing, so we can resume");

// テスト
this.assertLastUpdateSentToTwitter(twitterClient, newStatus);
}

//-----||
// 内部ヘルパーメソッド -----||
//-----||

/**
 * 指定された状態を含む JMS {@link ObjectMessage} を
 * 指定された名前のキューに送信
 *
 * @param status
 * @param topicName
 * @throws Exception
 * @throws IllegalArgumentException どちらかの引数が提供されていない場合
 */
private void publishStatusUpdateToTopic(final StatusUpdate status) throws Exception,
IllegalArgumentException
{
    // 前提条件を確認
    if (status == null)
    {
        throw new IllegalArgumentException("状態が指定されていなければなりません");
    }
}
```

```

// JNDI からキューを取得
final Topic topic = (Topic) NAMING_CONTEXT.lookup(StatusUpdateConstants.JNDI_NAME_TOPIC_
STATUSUPDATE);

// JNDI から ConnectionFactory を取得
final TopicConnectionFactory factory = (TopicConnectionFactory) NAMING_CONTEXT.lookup(JNDI_NAME_
CONNECTION_FACTORY);

// 接続を作成
final TopicConnection connection = factory.createTopicConnection();
final TopicSession sendSession
= connection.createTopicSession(false, TopicSession.AUTO_ACKNOWLEDGE);
final TopicPublisher publisher = sendSession.createPublisher(topic);

// メッセージを作成
final Message message = sendSession.createObjectMessage(status);

// メッセージを送信
publisher.publish(message);
log.info(" 発行されたメッセージ " + message + " およびその内容 : " + status);

// クリーンアップ
sendSession.close();
connection.close();
}
}

```

E.3.2.3 StatusUpdateTestBase.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import java.util.List;
import java.util.UUID;
import java.util.logging.Logger;

import junit.framework.TestCase;

import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdate;

import twitter4j.Paging;
import twitter4j.Status;
import twitter4j.Twitter;
import twitter4j.TwitterException;

/**
 * POJO と JavaEE 環境の両方で StatusUpdate を
 * テストするためのベースサポート

```

```

*
* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
*/
public class StatusUpdateTestBase
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(StatusUpdateBeanBase.class.
getName());

    /**
     * 送信する状態更新
     */
    private static final String STATUS_UPDATE_PREFIX_TWITTER = "I'm testing Message-Driven EJBs using
JBoss EJB 3.x by @ALRubinger/@OReillyMedia! ";

    //-----||
    // 内部ヘルパーメソッド -----||
    //-----||

    /**
     * クライアントで表された Twitter アカウントの最後の更新が
     * 指定された最後に送信した状態更新と一致することを確認
     *
     * @throws TwitterException Twitter アカウントから最後の更新を取得するときにエラーが発生した場合
     */
    void assertLastUpdateSentToTwitter(final Twitter twitterClient, final StatusUpdate sent) throws
TwitterException, IllegalArgumentException
    {
        // 前提条件を確認
        if (twitterClient == null)
        {
            throw new IllegalArgumentException("Twitter クライアントが指定されていなければなりません");
        }
        if (sent == null)
        {
            throw new IllegalArgumentException("最後に送信した状態が指定されていなければなりません");
        }

        // 新しい状態を取得

```

```

    final List<Status> statuses = twitterClient.getUserTimeline(new Paging(1, 1));

    // 1つの状態を送信しており、それが適切であることを確認
    TestCase.assertEquals("Should have obtained one status (the most recent) back from request", 1,
statuses.size());
    final String roundtrip = statuses.get(0).getText();
    final String expected = sent.getText();
    log.info("Sent status update to Twitter: " + expected);
    log.info("Got last status update from Twitter: " + roundtrip);
    TestCase.assertEquals("Twitter API did not update with last sent status", expected, roundtrip);
}

/**
 * UUID に接頭辞 {@link StatusUpdateTestBase#STATUS_UPDATE_PREFIX_TWITTER}
 * 付けて一意の状態更新を取得する
 */
StatusUpdate getUniqueStatusUpdate()
{
    return new StatusUpdate(STATUS_UPDATE_PREFIX_TWITTER + UUID.randomUUID().toString());
}
}

```

E.3.2.4 StatusUpdateUnitTestCase.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import java.util.logging.Logger;

import javax.jms.MessageListener;
import javax.jms.ObjectMessage;

import junit.framework.TestCase;

import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdate;
import org.junit.Test;

import twitter4j.Twitter;
import twitter4j.TwitterException;

/**
 * StatusUpdate EJB の単体テスト。EJB コンテナのコンテキスト外での動作時に
 * ビジネスロジックが損なわれていないことを確認
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
public class StatusUpdateUnitTestCase extends StatusUpdateTestBase
{

```

```
//-----||
// クラスメンバー -----||
//-----||

/**
 * ログガー
 */
private static final Logger log = Logger.getLogger(StatusUpdateUnitTestCase.class.getName());

//-----||
// テスト -----||
//-----||

/**
 * {@link StatusUpdateBeanBase#updateStatus(StatusUpdate)} メソッドが
 * 受信メッセージによって呼び出されることを確認
 */
@Test
public void testUpdateStatusBase()
{
    // リスナを作成
    final StatusCachingMessageListener listener = new StatusCachingMessageListener();

    // 状態更新を作成
    final StatusUpdate newStatus = this.getUniqueStatusUpdate();

    // 送信
    this.sendMessage(newStatus, listener);

    // 送信された状態を抽出
    final StatusUpdate roundtrip = listener.getLastStatus();

    // 送信したものであることを確認
    TestCase.assertEquals("Status sent was not dispatched and received as expected", newStatus,
        roundtrip);
}

/**
 * {@link MessageListener#onMessage(javax.jms.Message)} が呼び出されたときに
 * {@link TwitterUpdateMdb} が Twitter を更新していることを確認
 */
@Test
public void testTwitterUpdateMdb() throws TwitterException
{
    // 環境が設定されているかどうかを判断
```

```

if (!EnvironmentSpecificTwitterClientUtil.isSupportedEnvironment())
{
    log.warning(EnvironmentSpecificTwitterClientUtil.MSG_UNSUPPORTED_ENVIRONMENT);
    return;
}

// リスナを作成する (POJO としての MDB Bean 実装クラス)
final TwitterUpdateMdb listener = new TwitterUpdateMdb();

// 自ら @PostConstruct を起動
listener.createTwitterClient();

// 状態更新を作成
final StatusUpdate newStatus = this.getUniqueStatusUpdate();

// 送信する
this.sendMessage(newStatus, listener);

/*
 * 注: このテストは、更新せずに 1 つの原子操作を調べるので不備がある。
 * 他の誰かがここに忍び込み、最後に受信した更新を調べるまでに別の
 * 状態更新を挿入できてしまう。これは例を示すためだけに使う。
 */

// Test
final Twitter twitterClient = EnvironmentSpecificTwitterClientUtil.getTwitterClient();
this.assertLastUpdateSentToTwitter(twitterClient, newStatus);
}

//-----||
// 内部ヘルパーメソッド -----||
//-----||

/**
 * 指定された状態更新を指定されたリスナに送信
 */
private void sendMessage(final StatusUpdate newStatus, final MessageListenerlistener)
{
    // この実装は直接送信 (POJO ベース)
    final ObjectMessage message = new MockObjectMessage(newStatus);

    // 手動で送信
    listener.onMessage(message);
}

//-----||

```

```

// 内部クラス -----||
//-----||

/**
 * POJO 環境で呼び出される {@link MessageListener}
 * {@link MessageListener#onMessage(javax.jms.Message)} で受信する最後の
 * 状態更新がキャッシュされ、検索に利用できる。これはシングルスレッド環境での
 * 使用を意図しているのでスレッドセーフではない
 */
private static class StatusCachingMessageListener extends StatusUpdateBeanBase
{
    private StatusUpdate lastStatus = null;

    /**
     * 指定された状態をキャッシュ
     * @see org.jboss.ejb3.examples.ch08.statusupdate.mdb.StatusUpdateBeanBase#updateStatus(org.
    jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdate)
     */
    public void updateStatus(final StatusUpdate newStatus) throws IllegalArgumentException
    {
        this.lastStatus = newStatus;
    }

    /**
     * 最後に受信した {@link StatusUpdate} を取得
     * @return
     */
    StatusUpdate getLastStatus()
    {
        return lastStatus;
    }
}
}

```

E.3.2.5 TwitterUpdateBlockingTestMdb.java

```

package org.jboss.ejb3.examples.ch08.statusupdate.mdb;

import java.util.concurrent.CountDownLatch;
import java.util.logging.Logger;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.MessageListener;

import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdate;
import org.jboss.ejb3.examples.ch08.statusupdate.api.StatusUpdateConstants;

```

```

/**
 * {@link TwitterUpdateMdb} の例を拡張し、テストだけで共有するラッチを追加する。
 * 先に進む前に処理が完了していることを
 * テストで確認できるようにする。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 */
@MessageDriven(name = TwitterUpdateMdb.NAME, activationConfig =
{
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = StatusUpdateConstants.
TYPE_DESTINATION_STATUSUPDATE),
    @ActivationConfigProperty(propertyName = "destination", propertyValue = StatusUpdateConstants.JNDI_
NAME_TOPIC_STATUSUPDATE)})
public class TwitterUpdateBlockingTestMdb extends TwitterUpdateMdb implements MessageListener
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * □ガー
     */
    private static final Logger log = Logger.getLogger(TwitterUpdateBlockingTestMdb.class.getName());

    /**
     * 共有ラッチ。テストは MDB が処理するまで待機できるようになる。
     * POJO テストではシングルスレッド環境なので全く必要ないが、
     * テストと同じ JVM で動作する EJB コンテナでテストするときには、
     * テストはこれを使って MDB が呼び出されるまで待機でき、テストの完全性が高まる。
     * これは運用版 EJB には搭載すべきではない。
     * 代わりに、これ（だけ）をサポートする EJB 拡張をテストする。
     */
    public static CountdownLatch LATCH = new CountdownLatch(1);

    //-----||
    // オーバーライドされた実装 -----||
    //-----||

    /**
     * {@inheritDoc}
     * 共有バリアの待機を追加し、先に進む前に処理が完了していることを
     * テストで確認できるようにする
     * @see org.jboss.ejb3.examples.ch08.statusupdate.mdb.TwitterUpdateMdb#updateStatus(org.jboss.ejb3.
examples.ch08.statusupdate.api.StatusUpdate)
     */

```

```
@Override
public void updateStatus(final StatusUpdate newStatus) throws IllegalArgumentException, Exception
{
    // スーパー実装を呼び出す
    try
    {
        super.updateStatus(newStatus);
    }
    finally
    {
        // ラッチをカウントダウンする
        log.info("Counting down the latch...");
        LATCH.countDown();
    }
}
}
```

付録 F

Java Persistence API : 従業員名簿の例

F.1 説明

エンタープライズアプリケーションでは、アプリケーションを再起動しても存続する状態を扱う必要があることが少なくありません。これを「永続状態」と呼び、通常はリレーショナルデータベース管理システム (RDBMS) というプログラムでモデル化されます。行ベースの RDBMS と Java オブジェクト間の変換やマッピングはアプリケーション開発者にとっては決まりきった処理なので、この作業は Java Persistence API (JPA) に任せます。すると、Java オブジェクトの通常のゲッターやセッターと自由にやり取りすれば、基礎をなすデータベースとのデータのやり取りに対処してくれます。

EJB はエンティティ Bean を使って JPA と統合でき、実際のデータベースを使ったデータの適切なモデル化、マッピング、永続化、検索に必要な API については 9 章から 14 章で詳しく説明しています。ここで例は従業員名簿であり、さまざまなテクニックを適用してマッピングされた型の関係を判断し、効率的に DB に問い合わせます。

F.2 オンライン上の関連情報

ウィキ記事 : <http://community.jboss.org/docs/DOC-15572>

ソースの場所 : <http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples/tree/master/ch09-14-employeeregistry/>

F.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

F.3.1 実装リソース

9 章を参照してください。

F.3.1.1 SimpleEmployee.java

```
package org.jboss.ejb3.examples.employeeregistry.ch09.entitymanager;  
  
import javax.persistence.Entity;
```

```
import javax.persistence.Id;

/**
 * システム内の従業員を表す。JPA アノテーションを
 * 追加した簡単な値オブジェクトとしてモデル化する。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
// EJB における Java Persistence API との統合点となる
// エンティティ Bean にする
public class SimpleEmployee
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * エンティティの主キー
     */
    @Id
    // このフィールドを主キーにする
    private Long id;

    /**
     * 従業員の名前
     */
    private String name;

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * JPA に必要なデフォルトコンストラクタ
     */
    public SimpleEmployee()
    {

    }

    /**
     * 便利なコンストラクタ
     */
}
```

```
public SimpleEmployee(final long id, final String name)
{
    // 設定
    this.id = id;
    this.name = name;
}

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return id
 */
public Long getId()
{
    return id;
}

/**
 * @param id 設定する id
 */
public void setId(final Long id)
{
    this.id = id;
}

/**
 * @return name
 */
public String getName()
{
    return name;
}

/**
 * @param name 設定する name
 */
public void setName(final String name)
{
    this.name = name;
}

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
```

```
    */
    @Override
    public String toString()
    {
        return SimpleEmployee.class.getSimpleName() + " [id=" + id + ", name=" + name + "];"
    }

    /**
     * {@inheritDoc}
     * @see java.lang.Object#hashCode()
     */
    @Override
    public int hashCode()
    {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        return result;
    }

    /**
     * {@inheritDoc}
     * @see java.lang.Object#equals(java.lang.Object)
     */
    @Override
    public boolean equals(Object obj)
    {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        SimpleEmployee other = (SimpleEmployee) obj;
        if (id == null)
        {
            if (other.id != null)
                return false;
        }
        else if (!id.equals(other.id))
            return false;
        return true;
    }
}
```

10 章を参照してください。

F.3.1.2 EmbeddedEmployeePK.java

```

package org.jboss.ejb3.examples.employeeregistry.ch10.mapping;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Embeddable;
import javax.persistence.EmbeddedId;

/**
 * {@link EmployeeWithExternalCompositePK} で {@link EmbeddedId} として使う
 * 複合主キークラス
 * 以下のインスタンスメンバーが一緒になって
 * データベースでの識別子（主キー）を構成
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Embeddable
// エンティティクラスに PK として埋め込むつもりであることを
// JPA に示す
public class EmbeddedEmployeePK implements Serializable
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * serialVersionUID
     */
    private static final long serialVersionUID = 1L;

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 姓
     */
    @Column
    private String lastName;

    /**
     * 社会保障番号（米国連邦 ID）
     */

```

```
@Column
private Long ssn;

//-----||
// 機能メソッド -----||
//-----||

/**
 * @return lastName
 */
public String getLastName()
{
    return lastName;
}

/**
 * @param lastName 設定する lastName
 */
public void setLastName(String lastName)
{
    this.lastName = lastName;
}

/**
 * @return ssn
 */
public Long getSsn()
{
    return ssn;
}

/**
 * @param ssn 設定する ssn
 */
public void setSsn(Long ssn)
{
    this.ssn = ssn;
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#hashCode()
```

```

    */
    @Override
    public int hashCode()
    {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((lastName == null) ? 0 : lastName.hashCode());
        result = prime * result + ((ssn == null) ? 0 : ssn.hashCode());
        return result;
    }

    /**
     * {@inheritDoc}
     * @see java.lang.Object#equals(java.lang.Object)
     */
    @Override
    public boolean equals(Object obj)
    {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        EmbeddedEmployeePK other = (EmbeddedEmployeePK) obj;
        if (lastName == null)
        {
            if (other.lastName != null)
                return false;
        }
        else if (!lastName.equals(other.lastName))
            return false;
        if (ssn == null)
        {
            if (other.ssn != null)
                return false;
        }
        else if (!ssn.equals(other.ssn))
            return false;
        return true;
    }
}

```

F.3.1.3 EmployeeType.java

```
package org.jboss.ejb3.examples.employeeregistry.ch10.mapping;
```

```

import javax.persistence.Enumerated;

/**
 * システム内の従業員の種類。エンティティ {@link EmployeeWithProperties} の
 * {@link Enumerated} を示すために使う。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public enum EmployeeType {
    MANAGER, PEON;
}

```

F.3.1.4 EmployeeWithEmbeddedPK.java

```
package org.jboss.ejb3.examples.employeeregistry.ch10.mapping;
```

```

import javax.persistence.EmbeddedId;
import javax.persistence.Entity;

/**
 * システム内の従業員を表す。
 * 識別子（主キー）は {@link EmbeddedEmployeePK} を使った
 * 埋め込みプロパティで決まる。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
// EJB における Java Persistence との統合点となる
// エンティティ Bean にする
public class EmployeeWithEmbeddedPK
{
    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 主キー。複合かつ埋め込み。
     */
    @EmbeddedId
    private EmbeddedEmployeePK id;

    //-----||
    // コンストラクタ -----||
    //-----||

```

```

/**
 * JPAに必要なデフォルトコンストラクタ
 */
public EmployeeWithEmbeddedPK()
{

}

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return id
 */
public EmbeddedEmployeePK getId()
{
    return id;
}

/**
 * @param id 設定する ID
 */
public void setId(final EmbeddedEmployeePK id)
{
    this.id = id;
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    return EmployeeWithEmbeddedPK.class.getSimpleName() + " [id=" + id + "];
}
}

```

F.3.1.5 EmployeeWithExternalCompositePK.java

```
package org.jboss.ejb3.examples.employeeregistry.ch10.mapping;
```

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.IdClass;

/**
 * システム内の従業員を表す。
 * 識別子（主キー）は {@link ExternalEmployeePK} を使った
 * 複合プロパティで決まる。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */

@Entity
// EJB における Java Persistence との統合点となる
// エンティティ Bean にする
@IdClass(ExternalEmployeePK.class)
// カスタム PK クラスを使った複合主キーを使う
public class EmployeeWithExternalCompositePK
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 姓
     */
    @Id
    private String lastName;

    /**
     * 社会保障番号（米国連邦 ID）
     */
    @Id
    private Long ssn;

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * JPA に必要なデフォルトコンストラクタ
     */
    public EmployeeWithExternalCompositePK()
    {
```

```
}

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return lastName
 */
public String getLastName()
{
    return lastName;
}

/**
 * @param lastName 設定する lastName
 */
public void setLastName(final String lastName)
{
    this.lastName = lastName;
}

/**
 * @return ssn
 */
public Long getSsn()
{
    return ssn;
}

/**
 * @param ssn 設定する ssn
 */
public void setSsn(final Long ssn)
{
    this.ssn = ssn;
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
```

```

@Override
public String toString()
{
    return EmployeeWithExternalCompositePK.class.getSimpleName() + " [lastName= " + lastName + ",
    ssn=" + ssn + " ]";
}

```

F.3.1.6 EmployeeWithMappedSuperClassId.java

```

package org.jboss.ejb3.examples.employeeregistry.ch10.mapping;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

import org.jboss.ejb3.examples.testsupport.entity.AutogenIdentityBase;
import org.jboss.ejb3.examples.testsupport.entity.IdentityBase;

/**
 * システム内の従業員を表す。
 * {@link IdentityBase#getId()}からの主キー
 * を継承する。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */

@Entity
// EJB における Java Persistence との統合点となる
// エンティティ Bean にする
@Table(name = "employees_with_autogen_pk")
// DB の表名を明示的に示す
public class EmployeeWithMappedSuperClassId extends AutogenIdentityBase
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 従業員の名前
     */
    // DB 内の列名を示すのに @Column.name を使える
    @Column(name = "employee_name")
    private String name;

```

```
//-----||
// コンストラクタ -----||
//-----||

/**
 * JPA に必要なデフォルトコンストラクタ
 */
public EmployeeWithMappedSuperClassId()
{

}

/**
 * 便利なコンストラクタ
 */
public EmployeeWithMappedSuperClassId(final String name)
{
    // 設定
    this.name = name;
}

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return name
 */
public String getName()
{
    return name;
}

/**
 * @param name 設定する name
 */
public void setName(final String name)
{
    this.name = name;
}

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
```

```

    public String toString()
    {
        return EmployeeWithMappedSuperClassId.class.getSimpleName() + " [id=" + this.getId() + ", name="
+ name + " ]";
    }
}

```

F.3.1.7 EmployeeWithProperties.java

```
package org.jboss.ejb3.examples.employeeregistry.ch10.mapping;
```

```

import java.util.Arrays;
import java.util.Date;

import javax.persistence.Basic;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Transient;

/**
 * JPA マッピングメタデータを示すための
 * 一連のプロパティを持つ
 * 従業員を表す。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
public class EmployeeWithProperties
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 主キー
     */
    @Id

```

```

@GeneratedValue
// PK の作成を自動的に管理してくれる
private Long id;

/**
 * 従業員が現在行っている作業の説明。
 * これは DB に格納する必要はない。
 */
@Transient
// これは永続化しない
private String currentAssignment;

/**
 * ID カードで使う従業員の写真
 */
@Lob
// これは大きなバイナリオブジェクトである
@Basic(fetch = FetchType.LAZY, optional = true)
// デフォルトではこれはロードしない。これはコストがかかる操作である。
// 要求されたときだけロードする。
private byte[] image;

/**
 * 従業員の種類
 */
@Enumerated(EnumType.STRING)

// これが列挙型の値であり、DB に格納される値は
// 列挙型の toString() の値であることを示す。
private EmployeeType type;

/**
 * 従業員の入社日
 */
@Temporal(TemporalType.DATE)
// これは SQL Date フィールドにマッピングすべきである
// SQL Time や Timestamp にもマッピングできる
private Date since;

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return id
 */

```

```
public Long getId()
{
    return id;
}

/**
 * @param id 設定する id
 */
public void setId(final Long id)
{
    this.id = id;
}

/**
 * @return currentAssignment
 */
public String getCurrentAssignment()
{
    return currentAssignment;
}

/**
 * @param currentAssignment 設定する currentAssignment
 */
public void setCurrentAssignment(final String currentAssignment)
{
    this.currentAssignment = currentAssignment;
}

/**
 * @return image
 */
public byte[] getImage()
{
    return image;
}

/**
 * @param image 設定する image
 */
public void setImage(final byte[] image)
{
    this.image = image;
}

/**
```

```

    * @return type
    */
    public EmployeeType getType()
    {
        return type;
    }

    /**
     * @param type 設定する type
     */
    public void setType(final EmployeeType type)
    {
        this.type = type;
    }

    /**
     * @return since
     */
    public Date getSince()
    {
        return since;
    }

    /**
     * @param since 設定する since
     */
    public void setSince(final Date since)
    {
        this.since = since;
    }

    //-----||
    // オーバーライドされた実装 -----||
    //-----||

    /**
     * {@inheritDoc}
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString()
    {
        return "EmployeeWithProperties [currentAssignment=" + currentAssignment + ", id=" + id + ",
            image=" + Arrays.toString(image) + ", since=" + since + ", type=" + type + "];"
    }
}

```

F.3.1.8 ExternalEmployeePK.java

```

package org.jboss.ejb3.examples.employeeregistry.ch10.mapping;

import java.io.Serializable;

import javax.persistence.IdClass;

/**
 * {@link EmployeeWithExternalCompositePK} で {@link IdClass} として使う
 * 複合主キークラス
 * 以下のインスタンスメンバーが一緒になって
 * データベースでの識別子 (主キー) を構成
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public class ExternalEmployeePK implements Serializable
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * serialVersionUID
     */
    private static final long serialVersionUID = 1L;

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 姓
     */
    private String lastName;

    /**
     * 社会保障番号 (米国連邦 ID)
     */
    private Long ssn;

    //-----||
    // 機能メソッド -----||
    //-----||

```

```
/**
 * @return lastName
 */
public String getLastName()
{
    return lastName;
}

/**
 * @param lastName 設定する lastName
 */
public void setLastName(String lastName)
{
    this.lastName = lastName;
}

/**
 * @return ssn
 */
public Long getSsn()
{
    return ssn;
}

/**
 * @param ssn 設定する ssn
 */
public void setSsn(Long ssn)
{
    this.ssn = ssn;
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#hashCode()
 */
@Override
public int hashCode()
{
    final int prime = 31;
    int result = 1;
    result = prime * result + ((lastName == null) ? 0 : lastName.hashCode());
}
```

```
        result = prime * result + ((ssn == null) ? 0 : ssn.hashCode());
        return result;
    }

    /**
     * {@inheritDoc}
     * @see java.lang.Object#equals(java.lang.Object)
     */
    @Override
    public boolean equals(Object obj)
    {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        ExternalEmployeePK other = (ExternalEmployeePK) obj;
        if (lastName == null)
        {
            if (other.lastName != null)
                return false;
        }
        else if (!lastName.equals(other.lastName))
            return false;
        if (ssn == null)
        {
            if (other.ssn != null)
                return false;
        }
        else if (!ssn.equals(other.ssn))
            return false;
        return true;
    }
}
```

11 章を参照してください。

F.3.1.9 Address.java

```
package org.jboss.ejb3.examples.employeeregistry.ch11.relationships;

import javax.persistence.Column;
import javax.persistence.Entity;

import org.jboss.ejb3.examples.testsupport.entity.AutoGenIdentityBase;
```

```

/**
 * 簡単な住所を表す。各 {@link Employee} は 1 つの住所を持つが、
 * 関係は双方向ではない。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
// EJB における Java Persistence との統合点となる
// エンティティ Bean にする
public class Address extends AutoGenIdentityBase
{
    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * ストリートアドレス
     */
    @Column(length = 100)
    // VARCHAR の長さ
    private String street;

    /**
     * 都市
     */
    @Column(length = 100)
    // VARCHAR の長さ
    private String city;

    /**
     * 州の郵便番号
     */
    @Column(length = 2)
    // VARCHAR の長さ
    private String state;

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * JPA に必要なデフォルトコンストラクタ
     */
    public Address()

```

```
{  
}  
  
/**  
 * 便利なコンストラクタ  
 */  
public Address(final String street, final String city, final String state)  
{  
    // 設定  
    this.setStreet(street);  
    this.setCity(city);  
    this.setState(state);  
}  
  
//-----||  
// アクセラレータ / ミューテータ -----||  
//-----||  
  
/**  
 * @return street  
 */  
public String getStreet()  
{  
    return street;  
}  
  
/**  
 * @param street 設定する street  
 */  
public void setStreet(String street)  
{  
    this.street = street;  
}  
  
/**  
 * @return city  
 */  
public String getCity()  
{  
    return city;  
}  
  
/**  
 * @param city 設定する city  
 */
```

```

public void setCity(String city)
{
    this.city = city;
}

/**
 * @return state
 */
public String getState()
{
    return state;
}

/**
 * @param state 設定する state
 */
public void setState(String state)
{
    this.state = state;
}

//-----||
// 必要な実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    return Address.class.getSimpleName() + " [city=" + city + ", state=" + state + ", street=" +
street + ", getId()=" + getId() + "];"
}
}

```

F.3.1.10 Computer.java

```

package org.jboss.ejb3.examples.employeeregistry.ch11.relationships;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.OneToOne;

import org.jboss.ejb3.examples.testsupport.entity.AutoGenIdentityBase;

```

```
/**
 * {@link Employee} のコンピュータを表す。
 * コンピュータをなくした場合や修理のために返す必要がある場合には
 * 関係は双方向である。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
public class Computer extends AutogenIdentityBase
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * コンピュータのメーカー
     */
    @Column(length = 100)
    // VARCHAR の長さ
    private String make;

    /**
     * コンピュータのモデル
     */
    @Column(length = 100)
    // VARCHAR の長さ
    private String model;

    @OneToOne
    // 双方向関係
    // 非所有側で mappedBy を宣言
    private Employee owner;

    //-----||
    // コンストラクタ -----||
    //-----||

    //-----||
    // アクセッサ / ミューテータ -----||
    //-----||

    /**
     * @return make
     */
}
```

```
public String getMake()
{
    return make;
}

/**
 * @param make 設定する make
 */
public void setMake(String make)
{
    this.make = make;
}

/**
 * @return model
 */
public String getModel()
{
    return model;
}

/**
 * @param model 設定する model
 */
public void setModel(String model)
{
    this.model = model;
}

/**
 * @return owner
 */
public Employee getOwner()
{
    return owner;
}

/**
 * @param owner 設定する owner
 */
public void setOwner(final Employee owner)
{
    this.owner = owner;
}

//-----||
```

```

// 必要な実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    return Computer.class.getSimpleName() + " [make=" + make + ", model=" + model + ", owner=" + owner + ",
    getId()=" + getId() + "];
}
}

```

F.3.1.11 Customer.java

```

package org.jboss.ejb3.examples.employeeregistry.ch11.relationships;

import javax.persistence.Entity;
import javax.persistence.ManyToOne;

import org.jboss.ejb3.examples.testsupport.entity.AutoGenIdentityBase;

/**
 * 顧客を表す。各顧客は取引の主担当である
 * {@link Employee} を持つが、
 * 関係は一方方向である。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
public class Customer extends AutoGenIdentityBase
{
    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 名前
     */
    private String name;

    /**
     * この {@link Customer} の主担当 {@link Employee}

```

```
    */
    @ManyToOne
    // 一方向
    private Employee primaryContact;

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * JPAに必要なデフォルトコンストラクタ
     */
    public Customer()
    {

    }

    /**
     * 便利なコンストラクタ
     */
    public Customer(final String name)
    {
        // 設定
        this.name = name;
    }

    //-----||
    // アクセッサ / ミューテータ -----||
    //-----||

    /**
     * @return name
     */
    public String getName()
    {
        return name;
    }

    /**
     * @param name 設定する name
     */
    public void setName(final String name)
    {
        this.name = name;
    }
}
```

```

/**
 * @return primaryContact
 */
public Employee getPrimaryContact()
{
    return primaryContact;
}

/**
 * @param primaryContact 設定する primaryContact
 */
public void setPrimaryContact(final Employee primaryContact)
{
    this.primaryContact = primaryContact;
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/* (non-Javadoc)
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    return Customer.class.getSimpleName() + " [name=" + name + ", getId()=" + getId() + "];"
}
}

```

F.3.1.12 Employee.java

```

package org.jboss.ejb3.examples.employeeregistry.ch11.relationships;

import java.util.ArrayList;
import java.util.Collection;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import javax.persistence.OneToOne;
import javax.persistence.OneToOne;

import org.jboss.ejb3.examples.testsupport.entity.AutoGenIdentityBase;

```

```

/**
 * システム内の従業員を表す。EJB と JPA アノテーション
 * を追加した簡単な値オブジェクトとしてモデル化する。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
// EJB における Java Persistence との統合点となる
// エンティティ Bean にする
public class Employee extends AutogenIdentityBase
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 名前
     */
    @Column(unique = true)
    // 同じ名前の従業員はいない。実際の世界では必ずしもこのような
    // 制約はないが、使い方を示す。
    private String name;

    /**
     * 従業員の住所
     */
    @OneToOne
    @JoinColumn(name="ADDRESS_ID")
    // 一方関係
    private Address address;

    /**
     * 従業員のコンピュータ
     */
    @OneToOne(mappedBy = "owner")
    // 双方向関係
    private Computer computer;

    /**
     * {@link Employee} のマネージャ
     */
    @ManyToOne
    private Employee manager;

```

```
/**
 * この {@link Employee} に報告が必要な {@link Employee}
 */
@OneToMany(mappedBy = "manager")
private Collection<Employee> peons;

/**
 * この {@link Employee} のすべての {@link Phone}
 */
@OneToMany
// 一方向関係
private Collection<Phone> phones;

/**
 * この {@link Employee} が属する {@link Team}
 */
@ManyToMany(mappedBy = "members")
private Collection<Team> teams;

//-----||
// コンストラクタ -----||
//-----||

/**
 * JPA に必要なデフォルトコンストラクタ
 */
public Employee()
{
    peons = new ArrayList<Employee>();
    phones = new ArrayList<Phone>();
    teams = new ArrayList<Team>();
}

/**
 * 便利なコンストラクタ
 */
public Employee(final String name)
{
    this();
    // 設定
    this.name = name;
}

//-----||
// アクセッサ / ミューテータ -----||
```

```
//-----||

/**
 * @return name
 */
public String getName()
{
    return name;
}

/**
 * @param name 設定する name
 */
public void setName(final String name)
{
    this.name = name;
}

/**
 * @return address
 */
public Address getAddress()
{
    return address;
}

/**
 * @param address 設定する address
 */
public void setAddress(final Address address)
{
    this.address = address;
}

/**
 * @return computer
 */
public Computer getComputer()
{
    return computer;
}

/**
 * @param computer 設定する computer
 */
public void setComputer(final Computer computer)
```

```
{
    this.computer = computer;
}

/**
 * @return manager
 */
public Employee getManager()
{
    return manager;
}

/**
 * @param manager 設定する manager
 */
public void setManager(final Employee manager)
{
    this.manager = manager;
}

/**
 * @return peons
 */
public Collection<Employee> getPeons()
{
    return peons;
}

/**
 * @param peons 設定する peons
 */
public void setPeons(final Collection<Employee> peons)
{
    this.peons = peons;
}

/**
 * @return teams
 */
public Collection<Team> getTeams()
{
    return teams;
}

/**
 * @param teams 設定する teams
```

```

    */
    public void setTeams(final Collection<Team> teams)
    {
        this.teams = teams;
    }

    /**
     * @return phones
     */
    public Collection<Phone> getPhones()
    {
        return phones;
    }

    /**
     * @param phones 設定する phones
     */
    public void setPhones(final Collection<Phone> phones)
    {
        this.phones = phones;
    }

    //-----||
    // オーバーライドされた実装 -----||
    //-----||

    /* (non-Javadoc)
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString()
    {
        return Employee.class.getSimpleName() + " [name=" + name + ", getId()=" + getId() + "];"
    }
}

```

F.3.1.13 Phone.java

```

package org.jboss.ejb3.examples.employeeregistry.ch11.relationships;

import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;

import org.jboss.ejb3.examples.testsupport.entity.AutoGenIdentityBase;

/**

```

```

* 電話番号を表す。{@link Employee} は複数の電話番号を持っている
* 可能性があるが、関係は一方方向である。
*
* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
* @version $Revision: $
*/

```

```
@Entity
```

```
// EJB における Java Persistence との統合点となる
```

```
// エンティティ Bean にする
```

```
public class Phone extends AutogenIdentityBase
```

```
{
```

```

//-----||
// インスタンスメンバー -----||
//-----||

```

```

/**
 * 電話番号
 */

```

```
private String number;
```

```

/**
 * 種類
 */

```

```
@Enumerated(EnumType.STRING)
```

```
private PhoneType type;
```

```

//-----||
// アクセッサ / ミューテータ -----||
//-----||

```

```

/**
 * @return number
 */

```

```
public String getNumber()
```

```
{
    return number;
}
```

```

/**
 * @param number 設定する number
 */

```

```
public void setNumber(String number)
```

```
{
    this.number = number;
}
```

```

/**
 * @return type
 */
public PhoneType getType()
{
    return type;
}

/**
 * @param type 設定する type
 */
public void setType(PhoneType type)
{
    this.type = type;
}

//-----||
// 必要な実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    return Phone.class.getSimpleName() + " [number=" + number + ", type=" + type + ", getId()=" +
getId() + "];"
}
}

```

F.3.1.14 PhoneType.java

```
package org.jboss.ejb3.examples.employeeregistry.ch11.relationships;
```

```

/**
 * {@link Phone}に関連する電話番号の種類
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public enum PhoneType {
    MOBILE, HOME, WORK
}

```

F.3.1.15 Task.java

```

package org.jboss.ejb3.examples.employeeregistry.ch11.relationships;

import java.util.ArrayList;
import java.util.Collection;

import javax.persistence.Entity;
import javax.persistence.ManyToOne;

import org.jboss.ejb3.examples.testsupport.entity.AutogenIdentityBase;

/**
 * 完了させるか課題として管理すべきタスクを表す。
 * タスクは任意の数の {@link Employee} に割り当てることができ、
 * {@link Employee} は任意の数の課題を持つことができる。しかし、
 * 関係はタスクから従業員への一方向である。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
public class Task extends AutogenIdentityBase
{
    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 名前
     */
    private String description;

    /**
     * この {@link Task} を担当する {@link Employee}
     */
    @ManyToOne
    private Collection<Employee> owners;

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * JPA に必要なデフォルトコンストラクタ
     */

```

```
public Task()
{
    owners = new ArrayList<Employee>();
}

/**
 * 便利なコンストラクタ
 */
public Task(final String description)
{
    this();
    // 設定
    this.description = description;
}

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return description
 */
public String getDescription()
{
    return description;
}

/**
 * @param description 設定する description
 */
public void setDescription(final String description)
{
    this.description = description;
}

/**
 * @return owners
 */
public Collection<Employee> getOwners()
{
    return owners;
}

/**
 * @param owners 設定する owners
 */
public void setOwners(final Collection<Employee> owners)
```

```

    {
        this.owners = owners;
    }

    //-----||
    // 必要な実装 -----||
    //-----||

    /**
     * {@inheritDoc}
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString()
    {
        return Task.class.getSimpleName() + " [description=" + description + ", owners=" + owners + ",
        getId()=" + getId() + "];"
    }
}

```

F.3.1.16 Team.java

```

package org.jboss.ejb3.examples.employeeregistry.ch11.relationships;

import java.util.ArrayList;
import java.util.Collection;

import javax.persistence.Entity;
import javax.persistence.ManyToMany;

import org.jboss.ejb3.examples.testsupport.entity.AutoGenIdentityBase;

/**
 * 主に同じ場所で働く {@link Employee} のチームを表す。
 * 従業員は複数のチームの一員になりえる。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
public class Team extends AutoGenIdentityBase
{
    //-----||
    // インスタンスメンバー -----||
    //-----||

```

```
/**
 * チームの名前
 */
private String name;

/**
 * この{@link Team}の{@link Employee}
 */
@ManyToMany
private Collection<Employee> members;

//-----||
// コンストラクタ -----||
//-----||

/**
 * JPAに必要なデフォルトコンストラクタ
 */
public Team()
{
    members = new ArrayList<Employee>();
}

/**
 * 便利なコンストラクタ
 */
public Team(final String name)
{
    this();
    this.name = name;
}

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return members
 */
public Collection<Employee> getMembers()
{
    return members;
}

/**
 * @param members 設定する members
```

```

    */
    public void setMembers(final Collection<Employee> members)
    {
        this.members = members;
    }

    /**
     * @return name
     */
    public String getName()
    {
        return name;
    }

    /**
     * @param name 設定する name
     */
    public void setName(String name)
    {
        this.name = name;
    }

    //-----||
    // 必要な実装 -----||
    //-----||

    /**
     * {@inheritDoc}
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString()
    {
        return Team.class.getSimpleName() + " [members=" + members + ", name=" + name + ", getId()=" +
            getId() + "];"
    }
}

```

F.3.1.17 Customer.java

```
package org.jboss.ejb3.examples.employeeeregistry.ch12.inheritance.joined;
```

```
import javax.persistence.Entity;
```

```

/**
 * 会社に関連する {@link Person} である顧客を表す。
 * 継承階層の中間に位置し、

```

```

* {@link Customer} の特殊な種類である
* 従業員タイプによって拡張される。
*
* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
* @version $Revision: $
*/
@Entity(name = "JOINED_CUSTOMER")
public class Customer extends Person
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * ストリートレベルの住所
     */
    private String street;

    /**
     * 都市
     */
    private String city;

    /**
     * 州
     */
    private String state;

    /**
     * 郵便番号
     */
    private String zip;

    //-----||
    // アクセッサ / ミューテータ -----||
    //-----||

    /**
     * @return street
     */
    public String getStreet()
    {
        return street;
    }
}

```

```
/**
 * @param street 設定する street
 */
public void setStreet(final String street)
{
    this.street = street;
}

/**
 * @return city
 */
public String getCity()
{
    return city;
}

/**
 * @param city 設定する city
 */
public void setCity(final String city)
{
    this.city = city;
}

/**
 * @return state
 */
public String getState()
{
    return state;
}

/**
 * @param state 設定する state
 */
public void setState(final String state)
{
    this.state = state;
}

/**
 * @return zip
 */
public String getZip()
{
    return zip;
}
```

```

    }

    /**
     * @param zip 設定する zip
     */
    public void setZip(final String zip)
    {
        this.zip = zip;
    }
}

```

F.3.1.18 Employee.java

```

package org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.joined;

import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;

/**
 * 従業員
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity(name = "JOINED_EMPLOYEE")
@PrimaryKeyJoinColumn(name = "EMP_PK")
public class Employee extends Customer
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 従業員の ID
     */
    private Integer employeeId;

    //-----||
    // アクセッサ / ミューテータ -----||
    //-----||

    /**
     * @return employeeId
     */
    public Integer getEmployeeId()
    {

```

```

        return employeeId;
    }

    /**
     * @param employeeId 設定する employeeId
     */
    public void setEmployeeId(final Integer employeeId)
    {
        this.employeeId = employeeId;
    }
}

```

F.3.1.19 Person.java

```
package org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.joined;
```

```

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

/**
 * 人物を表すエンティティのためのベースクラス
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity(name = "JOINED_PERSON")
@Inheritance(strategy = InheritanceType.JOINED)
public class Person
{
    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 主キー
     */
    @Id
    @GeneratedValue
    private Long id;

    /**
     * 人物の姓
     */

```

```
private String firstName;

/**
 * 人物の名
 */
private String lastName;

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return id
 */
public Long getId()
{
    return id;
}

/**
 * @param id 設定する id
 */
public void setId(final Long id)
{
    this.id = id;
}

/**
 * @return firstName
 */
public String getFirstName()
{
    return firstName;
}

/**
 * @param firstName 設定する firstName
 */
public void setFirstName(final String firstName)
{
    this.firstName = firstName;
}

/**
 * @return lastName
 */
```

```
public String getLastName()
{
    return lastName;
}

/**
 * @param lastName 設定する lastName
 */
public void setLastName(final String lastName)
{
    this.lastName = lastName;
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/**
 * 値の等価性は ID と種類だけに基づく
 */

/**
 * {@inheritDoc}
 * @see java.lang.Object#hashCode()
 */
@Override
public int hashCode()
{
    final int prime = 31;
    int result = 1;
    result = prime * result + ((id == null) ? 0 : id.hashCode());
    return result;
}

/**
 * {@inheritDoc}
 * @see java.lang.Object#equals(java.lang.Object)
 */
@Override
public boolean equals(final Object obj)
{
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
```

```

        return false;
    Person other = (Person) obj;
    if (id == null)
    {
        if (other.id != null)
            return false;
    }
    else if (!id.equals(other.id))
        return false;
    return true;
}

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    return this.getClass().getSimpleName() + " [firstName=" + firstName + ", id = " + id + ",
lastName=" + lastName + "];
}
}

```

F.3.1.20 Customer.java

```
package org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.singleclass;
```

```
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
```

```

/**
 * 会社に関連する {@link Person} である顧客を表す。
 * 継承階層の中間に位置し、
 * {@link Customer} の特殊な種類である
 * 従業員タイプによって拡張される。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity(name = "SINGLECLASS_CUSTOMER")
@DiscriminatorValue("CUSTOMER")
public class Customer extends Person
{

    //-----||
    // インスタンスメンバー -----||
}

```

```
//-----||  
  
/**  
 * ストリートレベルの住所  
 */  
private String street;  
  
/**  
 * 都市  
 */  
private String city;  
  
/**  
 * 州  
 */  
private String state;  
  
/**  
 * 郵便番号  
 */  
private String zip;  
  
//-----||  
// アクセッサ / ミューテータ -----||  
//-----||  
  
/**  
 * @return street  
 */  
public String getStreet()  
{  
    return street;  
}  
  
/**  
 * @param street 設定する street  
 */  
public void setStreet(final String street)  
{  
    this.street = street;  
}  
  
/**  
 * @return city  
 */  
public String getCity()
```

```
{
    return city;
}

/**
 * @param city 設定する city
 */
public void setCity(final String city)
{
    this.city = city;
}

/**
 * @return state
 */
public String getState()
{
    return state;
}

/**
 * @param state 設定する state
 */
public void setState(final String state)
{
    this.state = state;
}

/**
 * @return zip
 */
public String getZip()
{
    return zip;
}

/**
 * @param zip 設定する zip
 */
public void setZip(final String zip)
{
    this.zip = zip;
}
}
```

F.3.1.21 Employee.java

```
package org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.singleclass;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

/**
 * 従業員
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity(name = "SINGLECLASS_EMPLOYEE")
@DiscriminatorValue("EMPLOYEE")
public class Employee extends Customer
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 従業員の ID
     */
    private Integer employeeId;

    //-----||
    // アクセッサ / ミューテータ -----||
    //-----||

    /**
     * @return employeeId
     */
    public Integer getEmployeeId()
    {
        return employeeId;
    }

    /**
     * @param employeeId 設定する employeeId
     */
    public void setEmployeeId(final Integer employeeId)
    {
        this.employeeId = employeeId;
    }
}
```

F.3.1.22 Person.java

```

package org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.singleclass;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

/**
 * 人物を表すエンティティのためのベースクラス
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity(name = "SINGLECLASS_PERSON")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "DISCRIMINATOR", discriminatorType = Discriminator
Type.STRING)
@DiscriminatorValue("PERSON")
public class Person
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 主キー
     */
    @Id
    @GeneratedValue
    private Long id;

    /**
     * 人物の姓
     */
    private String firstName;

    /**
     * 人物の名
     */
    private String lastName;

```

```
//-----||  
// アクセッサ / ミューテータ -----||  
//-----||  
  
/**  
 * @return id  
 */  
public Long getId()  
{  
    return id;  
}  
  
/**  
 * @param id 設定する id  
 */  
public void setId(final Long id)  
{  
    this.id = id;  
}  
  
/**  
 * @return firstName  
 */  
public String getFirstName()  
{  
    return firstName;  
}  
  
/**  
 * @param firstName 設定する firstName  
 */  
public void setFirstName(final String firstName)  
{  
    this.firstName = firstName;  
}  
  
/**  
 * @return lastName  
 */  
public String getLastName()  
{  
    return lastName;  
}  
  
/**  
 * @param lastName 設定する lastName
```

```
    */
    public void setLastName(final String lastName)
    {
        this.lastName = lastName;
    }

    //-----||
    // オーバーライドされた実装 -----||
    //-----||

    /**
     * 値の等価性は ID と種類だけに基づく
     */

    /**
     * {@inheritDoc}
     * @see java.lang.Object#hashCode()
     */
    @Override
    public int hashCode()
    {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        return result;
    }

    /**
     * {@inheritDoc}
     * @see java.lang.Object#equals(java.lang.Object)
     */
    @Override
    public boolean equals(final Object obj)
    {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Person other = (Person) obj;
        if (id == null)
        {
            if (other.id != null)
                return false;
        }
    }
}
```

```

        else if (!id.equals(other.id))
            return false;
        return true;
    }

    /**
     * {@inheritDoc}
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString()
    {
        return this.getClass().getSimpleName() + " [firstName=" + firstName + ", id = " + id + ",
        lastName=" + lastName + "];
    }
}

```

F.3.1.23 Customer.java

```
package org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.tableperclass;
```

```
import javax.persistence.Entity;
```

```

/**
 * 会社に関連する {@link Person} である顧客を表す。
 * 継承階層の中間に位置し、
 * {@link Customer} の特殊な種類である
 * 従業員タイプによって拡張される。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity(name = "TABLEPERCLASS_CUSTOMER")
public class Customer extends Person
{
    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * ストリートレベルの住所
     */
    private String street;

    /**
     * 都市

```

```
    */
private String city;

/**
 * 州
 */
private String state;

/**
 * 郵便番号
 */
private String zip;

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return street
 */
public String getStreet()
{
    return street;
}

/**
 * @param street 設定する street
 */
public void setStreet(final String street)
{
    this.street = street;
}

/**
 * @return city
 */
public String getCity()
{
    return city;
}

/**
 * @param city 設定する city
 */
public void setCity(final String city)
{
```

```
        this.city = city;
    }

    /**
     * @return state
     */
    public String getState()
    {
        return state;
    }

    /**
     * @param state 設定する state
     */
    public void setState(final String state)
    {
        this.state = state;
    }

    /**
     * @return zip
     */
    public String getZip()
    {
        return zip;
    }

    /**
     * @param zip 設定する zip
     */
    public void setZip(final String zip)
    {
        this.zip = zip;
    }
}
```

F.3.1.24 Employee.java

```
package org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.tableperclass;

import javax.persistence.Entity;

/**
 * 従業員
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $

```

```

*/
@Entity(name = "TABLEPERCLASS_EMPLOYEE")
public class Employee extends Customer
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 従業員の ID
     */
    private Integer employeeId;

    //-----||
    // アクセッサ / ミューテータ -----||
    //-----||

    /**
     * @return employeeId
     */
    public Integer getEmployeeId()
    {
        return employeeId;
    }

    /**
     * @param employeeId 設定する employeeId
     */
    public void setEmployeeId(final Integer employeeId)
    {
        this.employeeId = employeeId;
    }
}

```

F.3.1.25 Person.java

```

package org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.tableperclass;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;

/**

```

```
* 人物を表すエンティティのためのベースクラス
*
* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
* @version $Revision: $
*/
@Entity(name = "TABLEPERCLASS_PERSON")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Person
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 主キー
     */
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE)
    // クラスごとのテーブル方式ではデフォルト生成方針は受け入れられない
    private Long id;

    /**
     * 人物の姓
     */
    private String firstName;

    /**
     * 人物の名
     */
    private String lastName;

    //-----||
    // アクセッサ / ミューテータ -----||
    //-----||

    /**
     * @return id
     */
    public Long getId()
    {
        return id;
    }

    /**
     * @param id 設定する id

```

```
    */
    public void setId(final Long id)
    {
        this.id = id;
    }

    /**
     * @return firstName
     */
    public String getFirstName()
    {
        return firstName;
    }

    /**
     * @param firstName 設定する firstName
     */
    public void setFirstName(final String firstName)
    {
        this.firstName = firstName;
    }

    /**
     * @return lastName
     */
    public String getLastName()
    {
        return lastName;
    }

    /**
     * @param lastName 設定する lastName
     */
    public void setLastName(final String lastName)
    {
        this.lastName = lastName;
    }

    //-----||
    // オーバーライドされた実装 -----||
    //-----||

    /*
     * 値の等価性は ID と種類だけに基づく
     */
```

```
/**
 * {@inheritDoc}
 * @see java.lang.Object#hashCode()
 */
@Override
public int hashCode()
{
    final int prime = 31;
    int result = 1;
    result = prime * result + ((id == null) ? 0 : id.hashCode());
    return result;
}

/**
 * {@inheritDoc}
 * @see java.lang.Object#equals(java.lang.Object)
 */
@Override
public boolean equals(final Object obj)
{
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Person other = (Person) obj;
    if (id == null)
    {
        if (other.id != null)
            return false;
    }
    else if (!id.equals(other.id))
        return false;
    return true;
}

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    return this.getClass().getSimpleName() + " [firstName=" + firstName + ", id =" + id + ",
lastName=" + lastName + "];";
}
```

```

    }
}

```

14 章を参照して下さい。

F.3.1.26 EntityListenerEmployee.java

```

package org.jboss.ejb3.examples.employeeregistry.ch14.listener;

import java.util.logging.Logger;

import javax.persistence.Entity;
import javax.persistence.PostLoad;
import javax.persistence.PostPersist;
import javax.persistence.PostRemove;
import javax.persistence.PostUpdate;
import javax.persistence.PrePersist;
import javax.persistence.PreRemove;
import javax.persistence.PreUpdate;

import org.jboss.ejb3.examples.testsupport.entity.AutogenIdentityBase;

/**
 * JPA イベントを受け取る
 * 従業員を表す。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
public class EntityListenerEmployee extends AutogenIdentityBase
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログ
     */
    private static final Logger log = Logger.getLogger(EntityListenerEmployee.class.getName());

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**

```

```
* 従業員の名前
*/
private String name;

//-----||
// コンストラクタ -----||
//-----||

/**
 * JPA に必要な引数なしのコンストラクタ
 */
public EntityListenerEmployee()
{
}

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return name
 */
public String getName()
{
    return name;
}

/**
 * @param name 設定する name
 */
public void setName(final String name)
{
    this.name = name;
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
```

```
    {
        return EntityListenerEmployee.class.getSimpleName() + " [name=" + name + ", getId()=" + getId()
+ "]"";
    }

    //-----||
    // イベントリスナ -----||
    //-----||
    /*
     * イベントリスナ：JPA で発行され、EventTracker で状態を追跡
     */

    @PrePersist
    @SuppressWarnings("unused")
    private void prePersist()
    {
        EventTracker.prePersist = true;
        log.info("prePersist: " + this);
    }

    @PostPersist
    @SuppressWarnings("unused")
    private void postPersist()
    {
        EventTracker.postPersist = true;
        log.info("postPersist: " + this);
    }

    @PostLoad
    @SuppressWarnings("unused")
    private void postLoad()
    {
        EventTracker.postLoad = true;
        log.info("postLoad: " + this);
    }

    @PreUpdate
    @SuppressWarnings("unused")
    private void preUpdate()
    {
        EventTracker.preUpdate = true;
        log.info("preUpdate: " + this);
    }

    @PostUpdate
    @SuppressWarnings("unused")
    private void postUpdate()
```

```
{
    EventTracker.postUpdate = true;
    log.info("postUpdate: " + this);
}

@PreRemove
@SuppressWarnings("unused")
private void preRemove()
{
    EventTracker.preRemove = true;
    log.info("preRemove: " + this);
}

@PostRemove
@SuppressWarnings("unused")
private void postRemove()
{
    EventTracker.postRemove = true;
    log.info("postRemove: " + this);
}
}
```

F.3.1.27 EventTracker.java

```
package org.jboss.ejb3.examples.employeeregistry.ch14.listener;
```

```
/**
 * {@link EntityListenerEmployee} で発行されたイベントを追跡
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public class EventTracker
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * イベントが発行されたかどうかを示すフラグ
     */

    public static boolean prePersist;

    public static boolean postPersist;
```

```

public static boolean postLoad;

public static boolean preUpdate;

public static boolean postUpdate;

public static boolean preRemove;

public static boolean postRemove;

//-----||
// 機能メソッド -----||
//-----||

/**
 * すべてのイベントを false にリセット
 */
public static void reset()
{
    prePersist = false;
    postPersist = false;
    postLoad = false;
    preUpdate = false;
    postUpdate = false;
    preRemove = false;
    postRemove = false;
}
}

```

F.3.1.28 persistence.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns=http://java.sun.com/xml/ns/persistence
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
version="2.0">
  <persistence-unit name="tempdb">
    <jta-data-source>java:/DefaultDS</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="create"/>
      <!--
        Hibernate ではこれを有効にして SQL 出力を標準出力に出力できる
      <property name="hibernate.show_sql" value="true"/>
    -->
  </persistence-unit>

```

```
</properties>
</persistence-unit>
</persistence>
```

F.3.2 テストリソース

F.3.2.1 EmployeeIntegrationTest.java

```
package org.jboss.ejb3.examples.employeeregistry;

import java.util.Calendar;
import java.util.Collection;
import java.util.Date;
import java.util.concurrent.Callable;
import java.util.logging.Logger;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.persistence.EmbeddedId;
import javax.persistence.EntityManager;
import javax.persistence.GeneratedValue;
import javax.persistence.IdClass;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

import org.jboss.arquillian.api.Deployment;
import org.jboss.arquillian.api.Run;
import org.jboss.arquillian.api.RunModeType;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.ejb3.examples.employeeregistry.ch09.entitymanager.SimpleEmployee;
import org.jboss.ejb3.examples.employeeregistry.ch10.mapping.EmbeddedEmployeePK;
import org.jboss.ejb3.examples.employeeregistry.ch10.mapping.EmployeeType;
import org.jboss.ejb3.examples.employeeregistry.ch10.mapping.EmployeeWithEmbeddedPK;
import org.jboss.ejb3.examples.employeeregistry.ch10.mapping.EmployeeWithExternalCompositePK;
import org.jboss.ejb3.examples.employeeregistry.ch10.mapping.EmployeeWithMappedSuperClassId;
import org.jboss.ejb3.examples.employeeregistry.ch10.mapping.EmployeeWithProperties;
import org.jboss.ejb3.examples.employeeregistry.ch10.mapping.ExternalEmployeePK;
import org.jboss.ejb3.examples.employeeregistry.ch11.relationships.Address;
import org.jboss.ejb3.examples.employeeregistry.ch11.relationships.Computer;
import org.jboss.ejb3.examples.employeeregistry.ch11.relationships.Customer;
import org.jboss.ejb3.examples.employeeregistry.ch11.relationships.Employee;
import org.jboss.ejb3.examples.employeeregistry.ch11.relationships.Phone;
import org.jboss.ejb3.examples.employeeregistry.ch11.relationships.PhoneType;
import org.jboss.ejb3.examples.employeeregistry.ch11.relationships.Task;
import org.jboss.ejb3.examples.employeeregistry.ch11.relationships.Team;
import org.jboss.ejb3.examples.employeeregistry.ch14.listener.EntityListenerEmployee;
import org.jboss.ejb3.examples.employeeregistry.ch14.listener.EventTracker;
```

```

import org.jboss.ejb3.examples.testsupport.dbquery.EntityManagerExposingBean;
import org.jboss.ejb3.examples.testsupport.dbquery.EntityManagerExposingLocalBusiness;
import org.jboss.ejb3.examples.testsupport.entity.IdentityBase;
import org.jboss.ejb3.examples.testsupport.txwrap.TaskExecutionException;
import org.jboss.ejb3.examples.testsupport.txwrap.TxWrappingBean;
import org.jboss.ejb3.examples.testsupport.txwrap.TxWrappingLocalBusiness;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.runner.RunWith;

/**
 * オブジェクトビュー（エンティティ Bean）に対する簡単な
 * CRUD 操作を行うことができ、変更がトランザクションをまたいで
 * 永続化することを確認するテスト。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@RunWith(Arquillian.class)
@Run(RunModeType.AS_CLIENT)
public class EmployeeIntegrationTest
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(EmployeeIntegrationTest.class.getName());

    /**
     * ネーミングコンテキスト
     * Arquillian が EJB プロキシを注入するようになったら @deprecated を取り除く
     */
    @Deprecated
    private static Context jndiContext;

    /**
     * EJB コンテナへのデプロイ
     */

```

```

@Deployment
public static JavaArchive getDeployment()
{
    final JavaArchive archive = ShrinkWrap.create("entities.jar", JavaArchive.class).
addPackages(false,
    SimpleEmployee.class.getPackage(), EmployeeWithMappedSuperClassId.class.getPackage(),
    Employee.class.getPackage(), TxWrappingLocalBusiness.class.getPackage(),
    EntityListenerEmployee.class.getPackage(), EntityManagerExposingBean.class.getPackage(),
    org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.singleclass.Employee.class.
getPackage(),
    org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.tableperclass.Employee.class.
getPackage(),
    org.jboss.ejb3.examples.employeeregistry.ch12.inheritance.joined.Employee.class.
getPackage())
    .addManifestResource("persistence.xml");
    log.info(archive.toString(true));
    return archive;
}

/*
 * テスト用データ
 */

private static final long ID_DAVE = 1L;

private static final long ID_JOSH = 2L;

private static final long ID_RICK = 3L;

private static final String NAME_DAVE = "Dave";

private static final String NAME_DAVE_NEW = "Dave - The Good Doctor";

private static final String NAME_JOSH = "Josh";

private static final String NAME_RICK = "Rick, Jr.";

//-----||
// インスタンスメンバー -----||
//-----||
/**
 * 提供された {@link Callable} インスタンスを新しいトランザクション内に含める EJB
 */

// 実行メモ：埋め込み JBossAS 用の Arquillian でサポートされたら、ここで @EJB の注入をサポートする
private TxWrappingLocalBusiness txWrapper;

```

```

/**
 * テストで使うために {@link EntityManager} のメソッドへ直接アクセスできるようにする EJB。
 * 返されたエンティティがデタッチされないように既存のトランザクション内で呼び出さなければいけない
 */
// 実行メモ：埋め込み JBossAS 用の Arquillian でサポートされたら、ここで @EJB の注入をサポートする
private EntityManagerExposingLocalBusiness emHook;

//-----||
// ライフサイクル -----||
//-----||

/**
 * スイート全体の初期化を行う
 */
@BeforeClass
public static void init() throws Exception
{
    // サーバの起動後に明示的なプロパティを渡す必要はない
    jndiContext = new InitialContext();
}

/**
 * JNDI で手動で検索して割り当てる
 */
@Before
public void injectEjbsAndClearDB() throws Throwable
{
    // 差し当たりは手動で検索を行って注入をモック
    // 推奨されない部分を実行
    txWrapper = (TxWrappingLocalBusiness) jndiContext.lookup(TxWrappingBean.class.getSimpleName() +
"/local");
    emHook = (EntityManagerExposingLocalBusiness) jndiContext.lookup(EntityManagerExposingBean.class.
getSimpleName() + "/local");

    // 念のため、実行前にすべての従業員を削除する
    this.clearAllEmployees();
}

/**
 * すべてのエンティティコールバックをリセットする
 */
@Before
public void clearEntityCallbacks()
{
    {
        EventTracker.reset();
    }
}

```

```

/**
 * 永続ストレージからすべての従業員を削除する
 * @throws Throwable
 */
@After
public void clearAllEmployees() throws Throwable
{
    // すべての従業員の DB を削除する
    Try
    {
        txWrapper.wrapInTx(new Callable<Void>()
        {
            @Override
            public void call() throws Exception
            {
                final EntityManager em = emHook.getEntityManager();
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(SimpleEmployee.class, em);
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(EmployeeWithMappedSuperClassId.
class, em);
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(EmployeeWithExternalCompositePK.
class, em);
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(EmployeeWithProperties.class, em);
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(Computer.class, em);
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(Phone.class, em);
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(Customer.class, em);
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(Task.class, em);
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(Team.class, em);
                EmployeeIntegrationTest.this.deleteAllEntitiesOfType(Employee.class, em);

                return null;
            }
        });
    }
    catch (final TaskExecutionException tee)
    {
        // アンラップ
        throw tee.getCause();
    }
}

//-----||
// テスト -----||
//-----||

```

```
/**
 * {@link EntityManager} を使ってオブジェクトビューに対して簡単な
 * CRUD（作成、移動、更新、削除）操作を実行でき、
 * 変更が予想どおりに永続化されることを確認する
 */
@Test
public void persistAndModifyEmployees() throws Throwable
{
    Try
    {
        // トランザクションコンテキスト内で従業員の追加と条件チェックを行う
        txWrapper.wrapInTx(new Callable<Void>()
        {
            @Override
            public void call() throws Exception
            {
                // 普通のインスタンスをいくつか作成
                final SimpleEmployee josh = new SimpleEmployee(ID_DAVE, NAME_DAVE);
                final SimpleEmployee dave = new SimpleEmployee(ID_JOSH, NAME_JOSH);
                final SimpleEmployee rick = new SimpleEmployee(ID_RICK, NAME_RICK);

                // テストフックからエンティティマネージャを取得
                final EntityManager em = emHook.getEntityManager();

                // 対応する永続ストレージに従業員が見つかるかどうかを
                // まず調べる（見つからないべき）
                Assert.assertNull("Employees should not have been added to the EM yet",
                    em.find(SimpleEmployee.class, ID_DAVE));

                // オブジェクトが管理されているかどうかを調べる（管理されていないべき）
                Assert.assertFalse("従業員は管理されていないべきです", em.contains(josh));

                // 従業員を永続化
                em.persist(dave);
                em.persist(josh);
                em.persist(rick);
                log.info("追加：" + rick + dave + josh);

                // これで従業員は管理されているはず
                Assert.assertTrue("永続化する呼び出しの後で、従業員は管理されているべきです",
                    em.contains(josh));

                // 戻す
            }
        });
    }
}
```

```

        return null;
    }
});

// トランザクション内で従業員デブの名前を変更。後で変更が DB に反映されていることを確認
txWrapper.wrapInTx(new Callable<Void>()
{
    @Override
    public void call() throws Exception
    {
        // エンティティマネージャを取得
        final EntityManager em = emHook.getEntityManager();

        // エンティティマネージャから ID で「デブ」を検索
        final SimpleEmployee dave = em.find(SimpleEmployee.class, ID_DAVE);

        // デブの名前を変更する
        dave.setName(NAME_DAVE_NEW);
        log.info(" デブの名前を変更 : " + dave);

        // これで完了—トランザクションが完了すると、新しい名前が DB に書き出されるべきである
        return null;
    }
});

// 先ほどのトランザクションでデブの名前を変更したため、変更が
// 書き出されており、新しいトランザクションから見えることを確認
txWrapper.wrapInTx(new Callable<Void>()
{
    @Override
    public void call() throws Exception
    {
        // エンティティマネージャを取得
        final EntityManager em = emHook.getEntityManager();

        // 同じ主キーを持つデタッチされたオブジェクトとして新しい「デブ」を作成
        final SimpleEmployee dave = new SimpleEmployee(ID_DAVE, NAME_DAVE_NEW);

        // デタッチされたインスタンス上の変更を DB にマージ
        em.merge(dave);

        // 名前の変更を確認
        Assert.assertEquals(" 従業員デブの名前は変更されているべきです ", NAME_DAVE_NEW, dave.
getName());
    }
});

```

```

// エンティティマネージャからデーブをデタッチするので、オブジェクトは管理されなくなる
em.detach(dave);

// デーブの名前を再びダミー値に変更する。このオブジェクトは
// デタッチされ、もはや管理されていないので、この新しい値は
// DBとは同期されないはずである
dave.setName(" 永続先に書き出されるべきではない名前 ");
log.info(" デタッチ後にデーブの名前を変更: " + dave);

// Return
return null;
}
});

// 再度調べる。エンティティが管理されておらず EMからデタッチされた状態で
// デーブの名前を変更したので、行った変更が反映されていないことを
// 確認
txWrapper.wrapInTx(new Callable<Void>()
{

@Override
public void call() throws Exception
{
// エンティティマネージャを取得
final EntityManager em = emHook.getEntityManager();

// 新しい「デーブ」インスタンスを生成
final SimpleEmployee dave = em.find(SimpleEmployee.class, ID_DAVE);
log.info("Lookup of Dave after we changed his name on a detached instance: " + dave);

// デーブに行った最後の名前変更が反映されていないことを確認
Assert.assertEquals("Detached object values should not have been flushed", NAME_DAVE_
NEW, dave.getName());

// 戻す
return null;

}
});

// リックが退職を決めた。彼の記録を削除
txWrapper.wrapInTx(new Callable<Void>()
{

@Override

```

```
public void call() throws Exception
{
    // エンティティマネージャを取得
    final EntityManager em = emHook.getEntityManager();

    // リックを検索
    final SimpleEmployee rick = em.find(SimpleEmployee.class, ID_RICK);

    // 削除
    em.remove(rick);
    log.info("削除: " + rick);

    // 戻す
    return null;
}
});

// DB にリックが見つからないことを確認
txWrapper.wrapInTx(new Callable<Void>()
{

    @Override
    public void call() throws Exception
    {
        // エンティティマネージャを取得
        final EntityManager em = emHook.getEntityManager();

        // Look up Rick を検索
        final SimpleEmployee rick = em.find(SimpleEmployee.class, ID_RICK);

        // アサート
        Assert.assertNull("Rick should have been removed from the DB", rick);

        // 戻す
        return null;
    }
});

}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
```

```

}

/**
 * JPA の主キー自動生成の使い方を示す。{@link IdentityBase#getId()} から
 * 主キーサポートを継承する {@link EmployeeWithMappedSuperClassId}
 * を使う。
 * @throws Throwable
 */
@Test
public void autogenPrimaryKeyFromMappedSuperClass() throws Throwable
{
    Try
    {
        // 新しい従業員を作成し、JPA に主キー値を提供してもらう。
        final Long id = txWrapper.wrapInTx(new Callable<Long>()
        {
            @Override
            public Long call() throws Exception
            {
                // 新しい従業員を作成する
                final EmployeeWithMappedSuperClassId alrubinger
                = new EmployeeWithMappedSuperClassId("Andrew Lee Rubinger");

                // 現在は ID がないことを確認する
                Assert.assertNull("Primary key should not be set yet", alrubinger.getId());

                // 永続化
                emHook.getEntityManager().persist(alrubinger);

                // JPA が主キーを生成してくれることを示す
                final Long id = alrubinger.getId();
                Assert.assertNotNull("Persisting an entity with PK " + GeneratedValue.class.getName() +
                " should be created", id);
                log.info("Persisted: " + alrubinger);

                // 戻す
                return id;
            }
        });

        // 与えられた主キーでこの新しいエンティティを検索できることを確認する
        txWrapper.wrapInTx(new Callable<Void>()
        {

```

```
@Override
public void call() throws Exception
{
    // 与えられた ID で従業員を検索する
    final EmployeeWithMappedSuperClassId employee
    = emHook.getEntityManager().find(EmployeeWithMappedSuperClassId.class, id);

    // 見つかることを確認する
    Assert.assertNotNull("Employee should be able to be looked up by PK", employee);

    // 戻す
    return null;
}

});
}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**
 * {@link IdClass} - {@link ExternalEmployeePK} を使って
 * ID を取得するエンティティの使い方を示す
 * @throws Throwable
 */
@Test
public void externalCompositePrimaryKey() throws Throwable
{
    try
    {
        txWrapper.wrapInTx(new Callable<Void>()
        {
            @Override
            public void call() throws Exception
            {
                // 主キー ID を構成する値を定める
                final String lastName = "Rubinger";
                final Long ssn = 100L; // 本物ではない

                // カスタム @IdClass を使った新しい従業員を作成
                final EmployeeWithExternalCompositePK employee = new EmployeeWithExternalCompositePK();
                employee.setLastName(lastName);
            }
        });
    }
}
```

```

        employee.setSsn(ssn);

        // 永続化
        final EntityManager em = emHook.getEntityManager();
        em.persist(employee);
        log.info("Persisted: " + employee);

        // カスタム複合主キー値クラスを使って検索
        final ExternalEmployeePK pk = new ExternalEmployeePK();
        pk.setLastName(lastName);
        pk.setSsn(ssn);
        final EmployeeWithExternalCompositePK roundtrip
        = em.find(EmployeeWithExternalCompositePK.class, pk);

        // 見つかることを確認
        Assert.assertNotNull("Should have been able to look up record via a custom PK composite
        class", roundtrip);

        // 戻す
        return null;
    }

    });
}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**
 * {@link EmbeddedId} - {@link EmployeeWithEmbeddedPK} を使って
 * ID を取得するエンティティの使い方を示す
 * @throws Throwable
 */
@Test
public void embeddedCompositePrimaryKey() throws Throwable
{
    try
    {
        txWrapper.wrapInTx(new Callable<Void>()
        {
            @Override
            public void call() throws Exception

```

```
{
    // 主キー ID を構成する値を定める
    final String lastName = "Rubinger";
    final Long ssn = 100L; // 本物ではない

    // 埋め込み主キークラスを使った新しい従業員を作成
    final EmployeeWithEmbeddedPK employee = new EmployeeWithEmbeddedPK();
    final EmbeddedEmployeePK pk = new EmbeddedEmployeePK();
    pk.setLastName(lastName);
    pk.setSsn(ssn);
    employee.setId(pk);

    // 永続化
    final EntityManager em = emHook.getEntityManager();
    em.persist(employee);
    log.info("Persisted: " + employee);

    // 埋め込み主キー値クラスを使って検索
    final EmployeeWithEmbeddedPK roundtrip = em.find(EmployeeWithEmbeddedPK.class, pk);

    // 見つかることを確認
    Assert.assertNotNull("Should have been able to look up record via a custom embedded PK
class", roundtrip);

    // 戻す
    return null;
}

});
}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**
 * ORM レイヤに DB での表し方を示す追加の JPA メタデータを必要とする
 * 一連の非標準マッピングを使った
 * エンティティの使い方を示す
 */
@Test
public void propertyMappings() throws Throwable
{
    // 従業員のための値を定める
```

```

final byte[] image = new byte[]
{0x00};
final Date since = new Date(0L); // 当初から雇われている
final EmployeeType type = EmployeeType.PEON;
final String currentAssignment = "Learn JPA and EJB!";

try
{
    final Long id = txWrapper.wrapInTx(new Callable<Long>()
    {

        @Override
        public Long call() throws Exception
        {

            // 新しい従業員を作成
            final EmployeeWithProperties employee = new EmployeeWithProperties();
            employee.setImage(image);
            employee.setSince(since);
            employee.setType(type);
            employee.setCurrentAssignment(currentAssignment);

            // 永続化
            final EntityManager em = emHook.getEntityManager();
            em.persist(employee);
            log.info("Persisted: " + employee);

            // ID が割り当てられているので ID を取得
            final Long id = employee.getId();

            // 戻す
            return id;
        }
    });

    // 別のトランザクションで実行し、エンティティマネージャから永続化した
    // オブジェクトへの直接の参照を得るのではなく、実際に DB のロードを行うことを確認
    txWrapper.wrapInTx(new Callable<Void>()
    {

        @Override
        public void call() throws Exception
        {
            // ラウンドトリップルックアップ
            final EmployeeWithProperties roundtrip = emHook.getEntityManager().

```

```
find(EmployeeWithProperties.class, id);
    log.info("Roundtrip: " + roundtrip);

    final Calendar suppliedSince = Calendar.getInstance();
    suppliedSince.setTime(since);
    final Calendar obtainedSince = Calendar.getInstance();
    obtainedSince.setTime(roundtrip.getSince());

    // すべての値が予想どおりであることを確認
    Assert.assertEquals("Binary object was not mapped properly", image[0],
        roundtrip.getImage()[0]);
    Assert.assertEquals("Temporal value was not mapped properly", s
        uppliedSince.get(Calendar.YEAR), obtainedSince.get(Calendar.YEAR));
    Assert.assertEquals("Temporal value was not mapped properly",
        suppliedSince.get(Calendar.MONTH), obtainedSince.get(Calendar.MONTH));
    Assert.assertEquals("Temporal value was not mapped properly",
        suppliedSince.get(Calendar.DATE), obtainedSince.get(Calendar.DATE));
    Assert.assertEquals("Enumerated value was not as expected", type, roundtrip.getType());
    Assert.assertNull("Transient property should not have been persisted",
        roundtrip.getCurrentAssignment());

    // 戻す
    return null;
    }
});
}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**
 * {@link Employee}と{@link Address}間の 1:1 の
 * 一方向マッピングの使い方を示す
 * @throws Throwable
 */
@Test
public void oneToOneUnidirectionalMapping() throws Throwable
{
    // 新しい従業員を作成
    final Employee alrubinger = new Employee("Andrew Lee Rubinger");

    // 新しい住所を作成
    final Address address = new Address("1 JBoss Way", "Boston", "MA");
```

```
try
{
    // 従業員と住所を永続化して関連付ける
    final Long employeeId = txWrapper.wrapInTx(new Callable<Long>()
    {
        @Override
        public Long call() throws Exception
        {
            // エンティティマネージャを取得
            final EntityManager em = emHook.getEntityManager();

            // 永続化
            em.persist(alrubinger);
            em.persist(address);

            // 関連付け
            alrubinger.setAddress(address);

            // 戻す
            return alrubinger.getId();
        }
    });

    // トランザクション完了後に従業員で住所を検索したときに
    // すべてが予想どおりになることを確認
    txWrapper.wrapInTx(new Callable<Void>()
    {
        @Override
        public void call() throws Exception
        {
            // エンティティマネージャを取得
            final EntityManager em = emHook.getEntityManager();

            // 従業員を検索
            final Employee roundtripEmployee = em.find(Employee.class, employeeId);

            // 住所を取得
            final Address persistedAddress = roundtripEmployee.getAddress();

            // 等しいことを確認
            Assert.assertEquals("Persisted address association was not as expected", address,
                persistedAddress);
        }
    });
}
```

```
        // 削除できるように関連付けを解除
        roundtripEmployee.setAddress(null);

        // 戻す
        return null;
    }

});
}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**
 * {@link Employee} and {@link Computer} 間の 1:1 の
 * 双方向マッピングの使い方を示す
 * @throws Throwable
 */
@Test
public void oneToOneBidirectionalMapping() throws Throwable
{
    // 新しいコンピュータを作成
    final Computer computer = new Computer();
    computer.setMake("Computicorp");
    computer.setModel("ZoomFast 100");

    // 新しい従業員を作成
    final Employee carloDeWolf = new Employee("Carlo de Wolf");

    try
    {
        /*
         * まだ関連付けない。カスケードポリシーではまだ永続化されていない
         * 関係を持ったエンティティを永続化することを
         * 禁止している。
         */

        // 永続化して関連付ける
        final Long employeeId = txWrapper.wrapInTx(new Callable<Long>()
        {
```

```

@Override
public Long call() throws Exception
{
    // エンティティマネージャを取得
    final EntityManager em = emHook.getEntityManager();

    // 永続化
    em.persist(carloDeWolf);
    em.persist(computer);

    // 双方向関係の「両側」を関連付ける
    carloDeWolf.setComputer(computer);
    computer.setOwner(carloDeWolf);

    // 戻す
    return carloDeWolf.getId();
}
});

// すべてが正しく関連付けられていることを確認
txWrapper.wrapInTx(new Callable<Void>()
{
    @Override
    public void call() throws Exception
    {
        // エンティティマネージャを取得
        final EntityManager em = emHook.getEntityManager();

        // 従業員を取得
        final Employee carloRoundtrip = em.find(Employee.class, employeeId);

        // 従業員からコンピュータを取得
        final Computer computerRoundtrip = carloRoundtrip.getComputer();

        // コンピュータから従業員を取得
        final Employee ownerOfComputer = computer.getOwner();
        log.info("Employee " + carloRoundtrip + " has computer " + computerRoundtrip);
        log.info("Computer " + computerRoundtrip + " has owner " + ownerOfComputer);

        // すべてが予想どおりであることを確認
        Assert.assertEquals("Computer of employee was not as expected ", computer,
            computerRoundtrip);
        Assert.assertEquals("Owner of computer was not as expected ", carloDeWolf,
            ownerOfComputer);
    }
});

```

```
        // 削除できるように関連付けを解除
        ownerOfComputer.setComputer(null);
        computerRoundtrip.setOwner(null);

        // 戻す
        return null;
    }

});
}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**
 * {@link Employee} と {@link Phone} 間の 1:N の
 * 一方向マッピングの使い方を示す
 * @throws Throwable
 */
@Test
public void oneToManyUnidirectionalMapping() throws Throwable
{
    // 従業員を作成
    final Employee jaikiranPai = new Employee("Jaikiran Pai");

    // 2つの電話を作成
    final Phone phone1 = new Phone();
    phone1.setNumber("800-USE-JBOSS");
    phone1.setType(PhoneType.WORK);
    final Phone phone2 = new Phone();
    phone2.setNumber("800-EJB-TIME");
    phone2.setType(PhoneType.MOBILE);

    try
    {
        // 永続化して関連付ける
        final Long employeeId = txWrapper.wrapInTx(new Callable<Long>()
        {
            @Override
            public Long call() throws Exception
            {
                // エンティティマネージャを取得
```

```

        final EntityManager em = emHook.getEntityManager();

        // 永続化
        em.persist(jaikiranPai);
        em.persist(phone1);
        em.persist(phone2);

        // 関連付け
        jaikiranPai.getPhones().add(phone1);
        jaikiranPai.getPhones().add(phone2);

        // 戻す
        return jaikiranPai.getId();
    }
});

// すべてが正しく関連付けられていることを確認
txWrapper.wrapInTx(new Callable<Void>()
{
    @Override
    public void call() throws Exception
    {
        // エンティティマネージャを取得
        final EntityManager em = emHook.getEntityManager();

        // 従業員を取得
        final Employee jaikiranRoundtrip = em.find(Employee.class, employeeId);

        // 従業員から電話を取得
        final Collection<Phone> phones = jaikiranRoundtrip.getPhones();
        log.info("Phones for " + jaikiranRoundtrip + ": " + phones);

        // すべてが予想どおりであることを確認
        final String assertionError = "Phones were not associated with the employee as
expected";
        Assert.assertEquals(assertionError, 2, phones.size());
        Assert.assertTrue(assertionError, phones.contains(phone1));
        Assert.assertTrue(assertionError, phones.contains(phone2));

        // 削除できるように関連付けを解除
        jaikiranRoundtrip.getPhones().clear();

        // 戻す
        return null;
    }
});

```

```
    });
}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**
 * {@link Employee} と報告者 {@link Employee} 間の 1:N の
 * 双方向マッピングの使い方を示す。
 * また、{@link Employee} のマネージャも示す。
 * @throws Throwable
 */
@Test
public void oneToManyBidirectionalMapping() throws Throwable
{
    // 数名の従業員を作成
    final Employee alrubinger = new Employee("Andrew Lee Rubinger");
    final Employee carloDeWolf = new Employee("Carlo de Wolf");
    final Employee jaikiranPai = new Employee("Jaikiran Pai");
    final Employee bigD = new Employee("Big D");

    try
    {
        // 永続化して関連付ける
        final Long managerId = txWrapper.wrapInTx(new Callable<Long>()
        {
            @Override
            public Long call() throws Exception
            {
                // エンティティマネージャを取得
                final EntityManager em = emHook.getEntityManager();

                // 永続化
                em.persist(jaikiranPai);
                em.persist(alrubinger);
                em.persist(carloDeWolf);
                em.persist(bigD);

                // 双方向関係の「両側」を関連付ける
                final Collection<Employee> peonsOfD = bigD.getPeons();
                peonsOfD.add(alrubinger);
            }
        });
    }
}
```

```

        peonsOfD.add(carloDeWolf);
        peonsOfD.add(jaikiranPai);
        alrubinger.setManager(bigD);
        carloDeWolf.setManager(bigD);
        jaikiranPai.setManager(bigD);

        // 戻す
        return bigD.getId();
    }
});

// 最後のトランザクションですべてを反映させるので、
// 再び検索してアサーションを実行
txWrapper.wrapInTx(new Callable<Void>()
{
    @Override
    public void call() throws Exception
    {
        // エンティティマネージャを取得
        final EntityManager em = emHook.getEntityManager();

        // 従業員 / マネージャを取得
        final Employee managerRoundtrip = em.find(Employee.class, managerId);

        // マネージャへの報告者を取得
        final Collection<Employee> peonsForManager = managerRoundtrip.getPeons();
        log.info("Reports of " + managerRoundtrip + ": " + peonsForManager);

        // すべてが予想どおりであることを確認
        final String assertionMessage = "The Employee Manager/Reports relationship was not as
        expected";
        Assert.assertEquals(assertionMessage, 3, peonsForManager.size());
        Assert.assertTrue(assertionMessage, peonsForManager.contains(alrubinger));
        Assert.assertTrue(assertionMessage, peonsForManager.contains(carloDeWolf));
        Assert.assertTrue(assertionMessage, peonsForManager.contains(jaikiranPai));
        Assert.assertEquals(assertionMessage, bigD, alrubinger.getManager());
        Assert.assertEquals(assertionMessage, bigD, carloDeWolf.getManager());
        Assert.assertEquals(assertionMessage, bigD, jaikiranPai.getManager());

        // 削除できるように関連付けを解除
        for (final Employee peon : peonsForManager)
        {
            peon.setManager(null);
        }
        peonsForManager.clear();
    }
});

```

```
        // 戻す
        return null;
    }

    });

}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**
 * {@link Customer} と主担当 {@link Employee} 間の N:1 の
 * 一方向関係の使い方を示す
 * @throws Throwable
 */
@Test
public void manyToOneUnidirectionalMapping() throws Throwable
{
    // 従業員を作成
    final Employee bstansberry = new Employee("Brian Stansberry");

    // 2人の顧客を作成
    final Customer jgreene = new Customer("Jason T. Greene");
    final Customer bobmcw = new Customer("Bob McWhirter");

    try
    {
        // 永続化して関連付ける
        txWrapper.wrapInTx(new Callable<Void>()
        {
            @Override
            public void call() throws Exception
            {
                // エンティティマネージャを取得
                final EntityManager em = emHook.getEntityManager();

                // 永続化
                em.persist(bstansberry);
                em.persist(jgreene);
                em.persist(bobmcw);
            }
        });
    }
}
```

```

        // 関連付け
        jgreene.setPrimaryContact(bstansberry);
        bobmcw.setPrimaryContact(bstansberry);

        // 戻す
        return null;
    }
});

// 検索してアサーションを実行
txWrapper.wrapInTx(new Callable<Void>()
{
    @Override
    public void call() throws Exception
    {
        // エンティティマネージャを取得
        final EntityManager em = emHook.getEntityManager();

        // 顧客を取得
        final Customer jgreeneRoundtrip = em.find(Customer.class, jgreene.getId());
        final Customer bobmcwRoundtrip = em.find(Customer.class, bobmcw.getId());

        // すべてが予想どおりであることを確認
        final String assertionMessage = "Primary contact was not assigned as expected";
        Assert.assertEquals(assertionMessage, bstansberry, jgreeneRoundtrip.
            getPrimaryContact());
        Assert.assertEquals(assertionMessage, bstansberry, bobmcwRoundtrip.getPrimaryContact());

        // 削除できるように関連付けを解除
        jgreeneRoundtrip.setPrimaryContact(null);
        bobmcwRoundtrip.setPrimaryContact(null);

        // 戻す
        return null;
    }
});
}
catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**

```

```
* {@link Employee} と割り当てられた {@link Task} 間の N:N の
* 一方向マッピングの使い方を示す
* @throws Throwable
*/
@Test
public void manyToManyUnidirectionalMapping() throws Throwable
{
    // 2人の従業員を作成
    final Employee smarlow = new Employee("Scott Marlow");
    final Employee jpederse = new Employee("Jesper Pedersen");

    // 2つのタスクを作成
    final Task task1 = new Task("Go to the JBoss User's Group - Boston");
    final Task task2 = new Task("Pick up flowers for Shelly McGowan");

    try
    {
        // 永続化して関連付ける
        txWrapper.wrapInTx(new Callable<Void>()
        {
            @Override
            public void call() throws Exception
            {
                // エンティティマネージャを取得
                final EntityManager em = emHook.getEntityManager();

                // 永続化
                em.persist(smarlow);
                em.persist(jpederse);
                em.persist(task1);
                em.persist(task2);

                // 関連付け
                task1.getOwners().add(smarlow);
                task1.getOwners().add(jpederse);
                task2.getOwners().add(smarlow);
                task2.getOwners().add(jpederse);

                // 戻す
                return null;
            }
        });

        // 検索してアサーションを実行
        txWrapper.wrapInTx(new Callable<Void>()

```

```

    {

        @Override
        public void call() throws Exception
        {
            // エンティティマネージャを取得する
            final EntityManager em = emHook.getEntityManager();

            // タスクを取得
            final Task task1Roundtrip = em.find(Task.class, task1.getId());
            final Task task2Roundtrip = em.find(Task.class, task2.getId());

            // すべてが予想どおりであることを確認
            final String assertionMessage = "Task owners were not assigned as expected";
            Assert.assertTrue(assertionMessage, task1Roundtrip.getOwners().contains(smarlow));
            Assert.assertTrue(assertionMessage, task1Roundtrip.getOwners().contains(jpederse));
            Assert.assertTrue(assertionMessage, task2Roundtrip.getOwners().contains(smarlow));
            Assert.assertTrue(assertionMessage, task2Roundtrip.getOwners().contains(jpederse));

            // 削除できるように関連付けを解除
            task1Roundtrip.getOwners().clear();
            task2Roundtrip.getOwners().clear();

            // 戻す
            return null;
        }
    });

}

catch (final TaskExecutionException tee)
{
    // アンラップ
    throw tee.getCause();
}
}

/**
 * {@link Employee} とチームメンバー間の N:N の
 * 双方向関係マッピングの使い方を示す
 * @throws Throwable
 */
@Test
public void manyToManyBidirectionalMapping() throws Throwable
{
    // 数名の従業員を作成
    final Employee pmuir = new Employee("Pete Muir");

```

```
final Employee dallen = new Employee("Dan Allen");
final Employee aslak = new Employee("Aslak Knutsen");

// いくつかのチームを作成
final Team seam = new Team("Seam");
final Team arquillian = new Team("Arquillian");

try
{
    // 永続化して関連付ける
    txWrapper.wrapInTx(new Callable<Void>()
    {
        @Override
        public void call() throws Exception
        {
            // エンティティマネージャを取得
            final EntityManager em = emHook.getEntityManager();

            // 永続化
            em.persist(pmuir);
            em.persist(dallen);
            em.persist(aslak);
            em.persist(seam);
            em.persist(arquillian);

            // 「両方向」を関連付ける
            seam.getMembers().add(dallen);
            seam.getMembers().add(pmuir);
            seam.getMembers().add(aslak);
            arquillian.getMembers().add(dallen);
            arquillian.getMembers().add(pmuir);
            arquillian.getMembers().add(aslak);
            aslak.getTeams().add(seam);
            aslak.getTeams().add(arquillian);
            dallen.getTeams().add(seam);
            dallen.getTeams().add(arquillian);
            pmuir.getTeams().add(seam);
            pmuir.getTeams().add(arquillian);

            // Return
            return null;
        }
    });

    // 検索してアサーションを実行
```

```

txWrapper.wrapInTx(new Callable<Void>()
{
    @Override
    public void call() throws Exception
    {
        // エンティティマネージャを取得
        final EntityManager em = emHook.getEntityManager();

        // チームと従業員を管理対象オブジェクトとして取得
        final Team seamRoundtrip = em.find(Team.class, seam.getId());
        final Team arquillianRoundtrip = em.find(Team.class, arquillian.getId());
        final Employee dallenRoundtrip = em.find(Employee.class, dallen.getId());
        final Employee pmuirRoundtrip = em.find(Employee.class, pmuir.getId());
        final Employee aslakRoundtrip = em.find(Employee.class, aslak.getId());

        // すべてが予想どおりであることを確認
        final String assertionMessage = "Team members were not assigned as expected";
        Assert.assertTrue(assertionMessage, seamRoundtrip.getMembers().contains(pmuir));
        Assert.assertTrue(assertionMessage, seamRoundtrip.getMembers().contains(aslak));
        Assert.assertTrue(assertionMessage, seamRoundtrip.getMembers().contains(dallen));
        Assert.assertTrue(assertionMessage, arquillianRoundtrip.getMembers().contains(pmuir));
        Assert.assertTrue(assertionMessage, arquillianRoundtrip.getMembers().contains(aslak));
        Assert.assertTrue(assertionMessage, arquillianRoundtrip.getMembers().contains(dallen));
        Assert.assertTrue(assertionMessage, dallenRoundtrip.getTeams().contains(seamRoundtrip));
        Assert.assertTrue(assertionMessage, dallenRoundtrip.getTeams().
contains(arquillianRoundtrip));
        Assert.assertTrue(assertionMessage, pmuirRoundtrip.getTeams().contains(seamRoundtrip));
        Assert.assertTrue(assertionMessage, pmuirRoundtrip.getTeams().
contains(arquillianRoundtrip));
        Assert.assertTrue(assertionMessage, aslakRoundtrip.getTeams().contains(seamRoundtrip));
        Assert.assertTrue(assertionMessage, aslakRoundtrip.getTeams().
contains(arquillianRoundtrip));

        // 削除できるように関連付けを解除
        aslakRoundtrip.getTeams().clear();
        dallenRoundtrip.getTeams().clear();
        pmuirRoundtrip.getTeams().clear();
        seamRoundtrip.getMembers().clear();
        arquillianRoundtrip.getMembers().clear();

        // 戻す
        return null;
    }
});
}

```

```
        catch (final TaskExecutionException tee)
        {
            // アンラップ
            throw tee.getCause();
        }
    }

/**
 * JPA エンティティコールバックを受信することを確認
 * @throws Exception
 */
@Test
public void entityCallbacks() throws Exception
{
    // 前提条件を確認
    final String preconditionMessage = "Test setup is in error";
    Assert.assertFalse(preconditionMessage, EventTracker.postLoad);
    Assert.assertFalse(preconditionMessage, EventTracker.postPersist);
    Assert.assertFalse(preconditionMessage, EventTracker.postRemove);
    Assert.assertFalse(preconditionMessage, EventTracker.postUpdate);
    Assert.assertFalse(preconditionMessage, EventTracker.prePersist);
    Assert.assertFalse(preconditionMessage, EventTracker.preRemove);
    Assert.assertFalse(preconditionMessage, EventTracker.preUpdate);

    // 新しい従業員を作成
    final EntityListenerEmployee employee = new EntityListenerEmployee();

    // ライフサイクル全般を実行
    txWrapper.wrapInTx(new Callable<Void>()
    {

        @Override
        public void call() throws Exception
        {
            // エンティティマネージャを取得
            final EntityManager em = emHook.getEntityManager();

            // 永続化
            em.persist(employee);

            // リフレッシュ
            em.refresh(employee);

            // 更新
            employee.setName("New Name");
            em.flush();
        }
    });
}
```

```
// 検索
em.find(EntityListenerEmployee.class, employee.getId());

// 削除
em.remove(employee);

// Return
return null;
}
});

// イベントの発行を確認
final String postconditionMessage = "Missing event fired";
Assert.assertTrue(postconditionMessage, EventTracker.postLoad);
Assert.assertTrue(postconditionMessage, EventTracker.postPersist);
Assert.assertTrue(postconditionMessage, EventTracker.postRemove);
Assert.assertTrue(postconditionMessage, EventTracker.postUpdate);
Assert.assertTrue(postconditionMessage, EventTracker.prePersist);
Assert.assertTrue(postconditionMessage, EventTracker.preRemove);
Assert.assertTrue(postconditionMessage, EventTracker.preUpdate);
}

/**
 * JPQL クエリでエンティティを検索できることを確認
 * @throws Exception
 */
@Test
public void jpaQlFind() throws Exception
{
    // 従業員を作成する
    final SimpleEmployee employee = new SimpleEmployee(ID_DAVE, NAME_DAVE);

    // 永続化してから検索する
    txWrapper.wrapInTx(new Callable<Void>()
    {
        @Override
        public void call() throws Exception
        {
            // エンティティマネージャを取得
            final EntityManager em = emHook.getEntityManager();

            // 永続化
            em.persist(employee);

            // 検索
```

```
        final String jpaQuery = "FROM " + SimpleEmployee.class.getSimpleName() + " e WHERE
e.name=?1";
        final SimpleEmployee roundtrip = (SimpleEmployee) em.createQuery(jpaQuery).
setParameter(1, NAME_DAVE).getSingleResult();

        // 予想どおりに取得できることを確認
        Assert.assertEquals("Employee from JPQL Query should equal the record added", employee,
roundtrip);

        // 戻す
        return null;
    }
});
}

/**
 * Criteria API クエリでエンティティを検索できることを確認
 * @throws Exception
 */
@Test
public void criertiaAPIFind() throws Exception
{
    // 従業員を作成
    final SimpleEmployee employee = new SimpleEmployee(ID_DAVE, NAME_DAVE);

    // 永続化してから検索
    txWrapper.wrapInTx(new Callable<Void>()
    {

        @Override
        public void call() throws Exception
        {
            // エンティティマネージャを取得
            final EntityManager em = emHook.getEntityManager();

            // 永続化
            em.persist(employee);

            // 検索
            final CriteriaBuilder builder = em.getCriteriaBuilder();
            final CriteriaQuery<SimpleEmployee> query = builder.createQuery(SimpleEmployee.class);
            Root<SimpleEmployee> root = query.from(SimpleEmployee.class);
            query.select(root).where(builder.equal(root.get("name"), NAME_DAVE));
            final SimpleEmployee roundtrip = (SimpleEmployee) em.createQuery(query).getSingleResult();

            // 予想どおりに取得できることを確認
```

```
        Assert.assertEquals("Employee from Criteria API Query should equal the record added",
employee, roundtrip);

        // 戻す
        return null;
    }
});
}

//-----||
// 内部ヘルパーメソッド -----||
//-----||

/**
 * JPQL Update を発行し、指定された型のすべてのエンティティを削除
 * @param type
 * @param em
 */
private void deleteAllEntitiesOfType(final Class<?> type, final EntityManager em)
{
    assert em != null : EntityManager.class.getSimpleName() + " must be specified";
    assert type != null : "type to be removed must be specified";
    // 指定されたすべての型を削除するための JPQL 文字列
    log.info("Removed: " + em.createQuery("DELETE FROM " + type.getSimpleName() + " entities of type
" + type);
}
}
```


付録 G

セキュリティ：安全な学校の例

G.1 説明

マルチユーザアプリケーションをセキュアにするには、ユーザの種類の違いに配慮しなければいけません。例えば、おそらくシステム管理者は一般ユーザには見えないレコードを変更できるようにすべきでしょう。しかし、アプリケーション内にセキュリティロジックをコーディングすると関心事が混ざり合い、コードが管理しにくくなります。したがって、EJBはロールベースのセキュリティモデルを（メタデータを使った）宣言型および（APIを使った）プログラム型の両方でサービスとして提供します。

本章の例では、誰がいつドアを開けられるかに関する厳密な方針を持つ学校をモデル化します。ここでは、@RolesAllowed、@DeclareRoles、@RunAs、@PermitAllの使い方を紹介します。

G.2 オンライン上の関連情報

ウィキ記事：<http://community.jboss.org/docs/DOC-15571>

ソースの場所：<http://github.com/jbosseb3/oreilly-ejb-6thedition-book-examples/tree/master/ch15-secureschool/>

G.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

G.3.1 実装リソース

G.3.1.1 FireDepartmentLocalBusiness.java

```
package org.jboss.ejb3.examples.ch15.secureschool.api;
```

```
/**
```

- * 緊急事態を宣言できる消防署を表す。誰もがこのサポートを
- * 呼び出すことができ、警報が発せられたら地元の学校を閉鎖する。
- *
- * @author ALR
- * @version \$Revision: \$

```

*/
public interface FireDepartmentLocalBusiness
{
    // -----||
    // 規約 -----||
    // -----||

    /**
     * 緊急事態を宣言するので、地元の学校を閉鎖しなければならない。
     */
    void declareEmergency();
}

```

G.3.1.2 SchoolClosedException.java

```

package org.jboss.ejb3.examples.ch15.secureschool.api;

import javax.ejb.ApplicationException;
import javax.ejb.EJBAccessException;

import org.jboss.ejb3.examples.ch15.secureschool.impl.Roles;

/**
 * {@link Roles#ADMIN} 以外のロールのユーザが、学校が閉まっているときに
 * 正面玄関を開けようとしたときに発行する
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@ApplicationException(rollback = true)
// つまり、EJBException でラップしない
public class SchoolClosedException extends EJBAccessException
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * serialVersionUID
     */
    private static final long serialVersionUID = 1L;

    //-----||
    // コンストラクタ -----||
    //-----||

```

```

/**
 * 新しい例外を作成する
 */
private SchoolClosedException(final String message)
{
    super(message);
}

//-----||
// ファクトリ -----||
//-----||

/**
 * 指定された必要なメッセージで新しい例外を作成する
 * @param message
 * @throws IllegalArgumentException メッセージが指定されていない場合
 */
public static SchoolClosedException newInstance(final String message) throws IllegalArgumentException
{
    // 前提条件を確認
    if (message == null)
    {
        throw new IllegalArgumentException("message must be specified");
    }

    // 戻す
    return new SchoolClosedException(message);
}
}

```

G.3.1.3 SecureSchoolLocalBusiness.java

```

package org.jboss.ejb3.examples.ch15.secureschool.api;

import org.jboss.ejb3.examples.ch15.secureschool.impl.Roles;

/**
 * さまざまなユーザが開ける可能性のあるドアを持つ学校を表す。
 * EJB セキュリティモデルを使うと、特定のドアを開けるための
 * アクセス権を特定のユーザに限定できる。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public interface SecureSchoolLocalBusiness
{

```

```
// -----||  
// 規約 -----||  
// -----||  
  
/**  
 * 業務のために学校を開ける。この時点で、正面玄関は全員に  
 * 対して開かれる。このメソッドはロール {@link Roles#ADMIN} の  
 * ユーザだけが呼び出せる。  
 */  
void open();  
  
/**  
 * 業務としての学校を閉める。この時点で、正面玄関はロール  
 * {@link Roles#ADMIN} のユーザ以外の全員に対して閉じられる。  
 * このメソッドは管理者だけが呼び出せる。  
 */  
void close();  
  
/**  
 * 正面玄関を開ける。学校が開いている間は認証されたすべての  
 * ユーザが正面玄関を開けることができ、学校が開いていないときは  
 * {@link Roles#ADMIN} だけが開けることができる。  
 *  
 * @throws SchoolClosedException 現在のユーザが {@link Roles#ADMIN}  
 * ではなく、{@link SecureSchoolLocalBusiness#isOpen()} が false の  
 * ときに正面玄関を開けようとした場合  
 */  
void openFrontDoor() throws SchoolClosedException;  
  
/**  
 * 通用口を開ける。{@link Roles#STUDENT} ロールのユーザは  
 * このドアを開けられないが、{@link Roles#ADMIN} と  
 * {@link Roles#JANITOR} ロールのユーザは開けられる。  
 */  
void openServiceDoor();  
  
/**  
 * 学校が開いているかどうかを返す。学校が開いているときには  
 * {@link Roles#ADMIN} だけがすべてのドアにアクセスできる。認証されて  
 * いないユーザも含め誰でも学校が開いているかどうかを調べられる。  
 * @return 学校が開いているかどうか  
 */  
boolean isOpen();  
  
}
```

G.3.1.4 FileDepartmentBean.java

```

package org.jboss.ejb3.examples.ch15.secureschool.impl;

import java.util.logging.Logger;

import javax.annotation.security.PermitAll;
import javax.annotation.security.RunAs;
import javax.ejb.EJB;
import javax.ejb.Singleton;

import org.jboss.ejb3.examples.ch15.secureschool.api.FireDepartmentLocalBusiness;
import org.jboss.ejb3.examples.ch15.secureschool.api.SecureSchoolLocalBusiness;

/**
 * 消防署の Bean 実装クラス
 * 緊急事態の場合には地元の学校を閉鎖する。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Singleton
@RunAs(Roles.ADMIN)
@PermitAll
// 暗黙的に適用されるが、アクセスポリシーを明示するためにここに含めた
public class FireDepartmentBean implements FireDepartmentLocalBusiness
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * Logger
     */
    private static final Logger log = Logger.getLogger(FireDepartmentBean.class.getName());

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 緊急事態の際に閉鎖する学校
     */
    @EJB
    private SecureSchoolLocalBusiness school;

```

```

//-----||
// 必要な実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch15.secureschool.api.FireDepartmentLocalBusiness#declareEmergency()
 */
@Override
public void declareEmergency()
{
    log.info("Dispatching emergency support from the Fire Department, closing local school");
    school.close();
}
}

```

G.3.1.5 Roles.java

```

package org.jboss.ejb3.examples.ch15.secureschool.impl;

/**
 * 学校のユーザと関連付けるロールのリストを保持する。
 * EJB セキュリティはロールベースなので、このようにして
 * アクセス権を決定する。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public interface Roles
{
    //-----||
    // 定数 -----||
    //-----||

    /**
     * システムに対する呼び出し側のロール
     */

    /**
     * ユーザが学校管理者であることを示すロール
     */
    String ADMIN = "Administrator";

    /**
     * ユーザが生徒であることを示すロール
     */
}

```

```

String STUDENT = "Student";

/**
 * ユーザが用務員であることを示すロール
 */
String JANITOR = "Janitor";

}

```

G.3.1.6 SecureSchoolBean.java

```

package org.jboss.ejb3.examples.ch15.secureschool.impl;

import java.util.logging.Logger;
import javax.annotation.PostConstruct;
import javax.annotation.Resource;
import javax.annotation.security.DeclareRoles;
import javax.annotation.security.PermitAll;
import javax.annotation.security.RolesAllowed;
import javax.ejb.Local;
import javax.ejb.SessionContext;
import javax.ejb.Singleton;
import javax.ejb.Startup;

import org.jboss.ejb3.examples.ch15.secureschool.api.SchoolClosedException;
import org.jboss.ejb3.examples.ch15.secureschool.api.SecureSchoolLocalBusiness;

/**
 * EJB セキュリティモデルの構成に基づいて、ドアを開く要求を
 * 拒否できる安全な学校
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Singleton
@Local(SecureSchoolLocalBusiness.class)
// システムのロールを宣言する
@DeclareRoles(
{Roles.ADMIN, Roles.STUDENT, Roles.JANITOR})
// デフォルトでは誰にもアクセス権を与えず、より細かい粒度でアクセスを許可
@RolesAllowed(
{})
@Startup
public class SecureSchoolBean implements SecureSchoolLocalBusiness
{
    //-----||

```

```

// クラスメンバー -----||
//-----||

/**
 * ログガー
 */
private static final Logger log = Logger.getLogger(SecureSchoolBean.class.getName());

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * 学校が開いているかどうか
 */
private boolean open;

/**
 * コンテナがセキュリティ情報を取得するためのフック
 */
@Resource
private SessionContext context;

//-----||
// 必要な実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch15.secureschool.api.SecureSchoolLocal
 * Business#openFrontDoor()
 */
// 全員にこのメソッドへのアクセス権を与え、後で制限する
@RolesAllowed(
{Roles.ADMIN, Roles.STUDENT, Roles.JANITOR})
@Override
public void openFrontDoor()
{
// この地点に到達したら、EJB セキュリティをパスしている。しかし、
// 状況に応じたルールを適用してもよい。EJB セキュリティはメソッド
// レベルで宣言されているので、API を使って特定のロジックを実行する。

// 呼び出し側を取得
final String callerName = context.getCallerPrincipal().getName();

// 学校が開いていることを確認

```

```
    if (!open)
    {
        // 学校が閉じているので、管理者だけがドアを開けられる
        if (!context.isCallerInRole(Roles.ADMIN))
        {
            // 追い出す
            throw SchoolClosedException.newInstance("業務時間後に正面玄関を開けようとすることは、" +
                "管理者以外は禁止されています。次のユーザーのアクセスを拒否：" + callerName);
        }
    }

    // ログに記録
    log.info("次のユーザーが正面玄関を開けました：" + callerName);
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch15.secureschool.api.SecureSchoolLocalBusiness#openServiceDoor()
 */
@RolesAllowed(
    {Roles.ADMIN, Roles.JANITOR})
// 生徒はこのドアを開けられない
@Override
public void openServiceDoor()
{
    log.info("Opening service door for: " + context.getCallerPrincipal().getName());
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch15.secureschool.api.SecureSchoolLocalBusiness#close()
 */
@RolesAllowed(Roles.ADMIN)
// 管理者だけに学校の開閉を行わせる
@Override
public void close()
{
    this.open = false;
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch15.secureschool.api.SecureSchoolLocalBusiness#open()
 */
@Override
@PostConstruct
```

```
// 学校は作成時に開かれる
@RolesAllowed(Roles.ADMIN)
// 管理者だけに学校の開閉を行わせる
public void open()
{
    this.open = true;
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch15.secureschool.api.SecureSchoolLocalBusiness#isOpen()
 */
@Override
@PermitAll
// 学校が開いているかどうかは誰でも調べられる
public boolean isOpen()
{
    return open;
}
}
```

G.3.2 テストリソース

G.3.2.1 SecureSchoolIntegrationTest.java

```
package org.jboss.ejb3.examples.ch15.secureschool;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.logging.Logger;

import javax.ejb.EJB;
import javax.ejb.EJBAccessException;
import javax.ejb.SessionContext;
import javax.inject.Inject;
import javax.naming.Context;
import javax.naming.NamingException;

import org.jboss.arquillian.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.arquillian.prototyping.context.api.ArquillianContext;
import org.jboss.ejb3.examples.ch15.secureschool.api.FireDepartmentLocalBusiness;
```

```

import org.jboss.ejb3.examples.ch15.secureschool.api.SchoolClosedException;
import org.jboss.ejb3.examples.ch15.secureschool.api.SecureSchoolLocalBusiness;
import org.jboss.ejb3.examples.ch15.secureschool.impl.SecureSchoolBean;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;

/**
 * SecureSchoolEJB がセキュリティモデルに関する
 * 規約どおりに機能していることを
 * 確認するためのテストクラス
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@RunWith(Arquillian.class)
public class SecureSchoolIntegrationTest
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ロガー
     */
    private static final Logger log = Logger.getLogger(SecureSchoolIntegrationTest.class.getName());

    /**
     * サーバにデプロイする EJB JAR
     * @return
     */
    @Deployment
    public static JavaArchive getDeployment()
    {
        final JavaArchive archive = ShrinkWrap.create("secureSchool.jar", JavaArchive.class).
addPackages(false, SecureSchoolLocalBusiness.class.getPackage(), SecureSchoolBean.class.getPackage());
        log.info(archive.toString(true));
        return archive;
    }

    /**
     * 「管理者」の役割を持つロールの名前
     */

```

```
private static String USER_NAME_ADMIN = "admin";

/**
 * 「admin」ユーザのパスワード
 */
private static String PASSWORD_ADMIN = "adminPassword";

/**
 * 「生徒」の役割を持つロールの名前
 */
private static String USER_NAME_STUDENT = "student";

/**
 * 「student」ユーザのパスワード
 */
private static String PASSWORD_STUDENT = "studentPassword";

/**
 * 「用務員」の役割を持つロールの名前
 */
private static String USER_NAME_JANITOR = "janitor";

/**
 * 「janitor」ユーザのパスワード
 */
private static String PASSWORD_JANITOR = "janitorPassword";

/**
 * EJB を検索する JNDI 名
 */
// Arquillian と接続し、(ログインプロパティと) 提供された JNDI コンテキストを使って EJB を注入するとよ
い
(with login properties) to inject the EJB
private static final String JNDI_NAME_EJB = "SecureSchoolBeanLocal";

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * Arquillian へのフック。提供されたプロパティを使って新しい JNDI コンテキストを作成できるようにする。
 */
@Inject
private ArquillianContext arquillianContext;

/**
```

```

* 明示的なログインや認証 / 承認なしに注入された EJB プロキシ。
* 背後では Arquillian がデフォルト JNDI コンテキストをログインプロパティなし
* で使い、このターゲットにプロキシを注入
*/
@EJB
private SecureSchoolLocalBusiness unauthenticatedSchool;

/**
 * 認証されていないユーザから消防署への参照
 * この EJB を使って緊急事態を宣言したら、
 * 誰でも学校を閉鎖できる。
 */
@EJB
private FireDepartmentLocalBusiness fireDepartment;

//-----||
// テスト -----||
//-----||

/**
 * 認証されていないユーザは正面玄関を開けられないことを確認
 */
@Test(expected = EJBAccessException.class)
public void unauthenticatedUserCannotOpenFrontDoor() throws NamingException
{
    // 認証前に正面玄関を開けようとする。失敗するはずである。
    unauthenticatedSchool.openFrontDoor();
}

/**
 * 「生徒」ユーザが正面玄関を開けられることを確認
 */
@Test
public void studentCanOpenFrontDoor() throws NamingException
{
    /**
     * このログインコードとロックアップコードは OpenEJB コンテナに特化している
     */

    // JNDI を使って「生徒」ユーザとしてログイン
    final Context context = this.login(USER_NAME_STUDENT, PASSWORD_STUDENT);

    try
    {

```

```
// 取得
final SecureSchoolLocalBusiness school = this.getEjb(context);

// 呼び出し（成功するはずである。未承認エラーで失敗しない）
school.openFrontDoor();
}
finally
{
    // クリーンアップ。コンテキストを閉じてログアウト
    context.close();
}
}

/**
 * 「用務員」ユーザが通用口を開けられることを確認
 */
@Test
public void janitorCanOpenServiceDoor() throws NamingException
{
    /**
     * このログインコードとロックアップコードは OpenEJB コンテナに特化している
     */

    // JNDI を使って「用務員」ユーザとしてログイン
    final Context context = this.login(USER_NAME_JANITOR, PASSWORD_JANITOR);

    try
    {
        // 取得
        final SecureSchoolLocalBusiness school = this.getEjb(context);

        // 呼び出し（成功するはずである。未承認エラーで失敗しない）
        school.openServiceDoor();
    }
    finally
    {
        // クリーンアップ。コンテキストを閉じてログアウト
        context.close();
    }
}

/**
 * 「生徒」ユーザが通用口を開けられないことを確認
 */
@Test(expected = EJBAccessException.class)
```

```
public void studentCannotOpenServiceDoor() throws NamingException
{
    /*
     * このログインコードとルックアップコードは OpenEJB コンテナに特化している
     */

    // JNDI を使って「生徒」ユーザとしてログイン
    final Context context = this.login(USER_NAME_STUDENT, PASSWORD_STUDENT);

    try
    {
        // 取得
        final SecureSchoolLocalBusiness school = this.getEjb(context);

        // 呼び出し（失敗するはずである）
        school.openServiceDoor();
    }
    finally
    {
        // クリーンアップ。コンテキストを閉じてログアウト
        context.close();
    }
}

/**
 * 「生徒」ユーザが学校を閉められない（そして、早く家に帰る）ことを確認
 */
@Test(expected = EJBAccessException.class)
public void studentCannotCloseSchool() throws NamingException
{
    /*
     * このログインコードとルックアップコードは OpenEJB コンテナに特化している
     */

    // JNDI を使って「生徒」ユーザとしてログイン
    final Context context = this.login(USER_NAME_STUDENT, PASSWORD_STUDENT);

    try
    {
        // 取得
        final SecureSchoolLocalBusiness school = this.getEjb(context);

        // 呼び出し（失敗するはずである）
        school.close();
    }
}
```

```
    }
    finally
    {
        // クリーンアップ。コンテキストを閉じてログアウト
        context.close();
    }
}

/**
 * 「管理者」ユーザが学校を閉められることを確認する
 */
@Test
public void adminCanCloseSchool() throws NamingException
{
    /**
     * このログインコードとロックアップコードは OpenEJB コンテナに特化している
     */

    // JNDI を使って「管理者」ユーザとしてログイン
    final Context context = this.login(USER_NAME_ADMIN, PASSWORD_ADMIN);

    try
    {
        // 取得
        final SecureSchoolLocalBusiness school = this.getEjb(context);

        // 呼び出し（成功するはずである）
        school.close();

        // テスト
        Assert.assertFalse("School should now be closed", school.isOpen());

        // 次のテストのために学校を開く
        school.open();

        // テスト
        Assert.assertTrue("学校が空いている必要があります", school.isOpen());
    }
    finally
    {
        // クリーンアップ。コンテキストを閉じてログアウトする。
        context.close();
    }
}
```

```
/**
 * 学校が開いているかどうかを未認証ユーザが調べられることを確認
 */
@Test
public void unauthenticatedUserCanCheckIfSchoolIsOpen()
{
    // 学校が開いているかどうかを確認
    Assert.assertTrue("Unauthenticated user should see that school is open",
        unauthenticatedSchool.isOpen());
}

/**
 * 学校が閉まっているときには生徒は正面玄関を
 * 開けられないことを確認する。実装クラスで {@link SessionContext} を使って
 * プログラムでセキュリティをテストする。
 */
@Test(expected = SchoolClosedException.class)
public void studentCannotOpenFrontDoorsWhenSchoolIsClosed() throws Throwable
{
    /*
     * このログインコードとルックアップコードは OpenEJB コンテナに特化している
     */

    try
    {
        // JNDI を使って「管理者」ユーザとしてログインする
        final Context context = this.login(USER_NAME_ADMIN, PASSWORD_ADMIN);

        // 取得
        final SecureSchoolLocalBusiness school = this.getEjb(context);

        // 学校を閉じる
        school.close();

        // ログアウト
        context.close();

        // 閉じていることを確認する
        Assert.assertFalse("School should now be closed", school.isOpen());

        // 生徒として正面玄関を開けようとする。OpenEJB は
        // このスレッドのセキュリティコンテキストを「管理者」に関連付けるので、
        // これは別のスレッドで行う。
        final Callable<Void> studentOpenDoorTask = new Callable<Void>()
        {

```

```
@Override
public void call() throws Exception
{
    // JNDI を使って「生徒」ユーザとしてログイン
    final Context context = SecureSchoolIntegrationTest.this.login(USER_NAME_STUDENT,
        PASSWORD_STUDENT);

    try
    {
        // 取得
        final SecureSchoolLocalBusiness school = SecureSchoolIntegrationTest.this.
            getEjb(context);

        // 正面玄関を開けようとする（失敗するはずである）
        school.openFrontDoor();

        // 戻す
        return null;
    }
    finally
    {
        context.close();
    }
}

};
final ExecutorService service = Executors.newSingleThreadExecutor();
final Future<Void> future = service.submit(studentOpenDoorTask);
try
{
    future.get();// ここでは失敗するはずである
}
catch (final ExecutionException ee)
{
    // アンラップ。SchoolClosedException を発行するはずである
    throw ee.getCause();
}

}
finally
{
    // クリーンアップし、別のテストのために学校を開く
    final Context context = this.login(USER_NAME_ADMIN, PASSWORD_ADMIN);
    final SecureSchoolLocalBusiness school = this.getEjb(context);

    // 次のテストのために学校を開く
```

```
school.open();

// テスト
Assert.assertTrue(" 学校が開いている必要があります ", school.isOpen());

// クリーンアップ。コンテキストを閉じてログアウトする。
context.close();

}
}

/**
 * 認証されていない任意のユーザが緊急事態を宣言でき、学校を閉鎖できることを確認する
 */
@Test
public void anyoneCanDeclareEmergencyAndCloseSchool() throws NamingException
{
    // まず学校が開かれていることを確認する
    Assert.assertTrue(" テストを開始するためには、学校が開いている必要があります ",
unauthenticatedSchool.isOpen());

    // 学校を直接閉鎖できないことを確認 (アクセス権がない)
    boolean gotAccessException = false;
    try
    {
        unauthenticatedSchool.close();
    }
    catch (final EJBAccessException e)
    {
        // 予想される結果
        log.info(" 自分で学校を閉鎖できません。緊急事態を宣言します。");
        gotAccessException = true;
    }
    Assert.assertTrue(" 学校を直接閉鎖できてはいけません ", gotAccessException);

    // 消防署を使って緊急事態を宣言する
    fireDepartment.declareEmergency();

    // 自分で直接閉鎖する権限がなくても、学校が閉鎖されるはず
    Assert.assertFalse(" 緊急事態宣言後、学校が閉鎖されるはずです ", unauthenticatedSchool.isOpen());

    // 学校を開く
    // クリーンアップし、別のテストのために学校を開く
    final Context context = this.login(USER_NAME_ADMIN, PASSWORD_ADMIN);
    try
```

```

    {
        final SecureSchoolLocalBusiness school = this.getEjb(context);

        // 次のテストのために学校を開く
        school.open();

        // テスト
        Assert.assertTrue("School should now be open", school.isOpen());
    }
    finally
    {
        // クリーンアップ。コンテキストを閉じてログアウトする。
        context.close();
    }
}

//-----||
// 内部ヘルパーメソッド -----||
//-----||

/**
 * 指定されたユーザ名とパスワードで JNDI（ひいては EJB セキュリティシステム）
 * にログインする。このメカニズムは OpenEJB コンテナ
 * に特化している。
 */
private Context login(final String username, final String password)
{
    // 前提条件を確認
    assert username != null : "username must be supplied";
    assert password != null : "password must be supplied";

    // ログインしてコンテキストを作成する
    final Map<String, Object> namingContextProps = new HashMap<String, Object>();
    namingContextProps.put(Context.SECURITY_PRINCIPAL, username);
    namingContextProps.put(Context.SECURITY_CREDENTIALS, password);
    final Context context = arquillianContext.get(Context.class, namingContextProps);

    // 戻す
    return context;
}

/**
 * (ユーザ認証に使われた) 指定された JNDI コンテキストを使って
 * EJB へのプロキシを取得する
 * @param context
 * @return
 */

```

```
* @throws NamingException
*/
private SecureSchoolLocalBusiness getEjb(final Context context) throws NamingException
{
    // OpenEJB に特化した JNDI で検索する
    // グローバル JNDI を使う
    return (SecureSchoolLocalBusiness) context.lookup(JNDI_NAME_EJB);
}
}
```

G.3.2.2 groups.properties

```
# OpenEJB Roles Configuration
# Format: Role=Username
Administrator=admin
Janitor=janitor
Student=student
```

G.3.2.3 users.properties

```
# OpenEJB Users Configuration
# Format: Username=Password
admin=adminPassword
student=studentPassword
janitor=janitorPassword
```


付録 H

トランザクション： ブラックジャックゲームの例

H.1 説明

最も簡単なアプリケーションでも、ほとんどの場合、複合的な処理（例えば「ユーザ登録」など）が完全に成功するか失敗するかのどちらかでなければいけないという要件があります。このような場合、1つの要求が実は小さな処理の集合なのです。要求の処理中にこれらの小さな処理の一部が途中で失敗すると、アプリケーションは一貫性のない状態（エラー状態）になる可能性があります。トランザクションを使うと、すべての複合処理がACID特性（Atomicity：原子性、Consistency：一貫性、Isolation：独立性 / 分離性、Durability：持続性）を満たすようにすることができます。つまり、複合処理は1つの単位として完全に成功するか失敗するかのどちらかになるのです。

EJBは、Java EEのTransactionManagerを使って宣言的な方法とプログラム的な方法でトランザクションをサポートしており、トランザクション境界を管理するためのアノテーションやXMLが充実しています。

本章の例はトランザクション対応のポーカーサービスであり、@TransactionAttributeとさまざまなTransactionAttributeTypeの使い方を示しています。

H.2 オンライン上の関連情報

ウィキ記事：<http://community.jboss.org/docs/DOC-15573>

ソースの場所：<http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples/tree/master/ch17-transactions/>

H.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

H.3.1 実装リソース

H.3.1.1 BankLocalBusiness.java

```
package org.jboss.ejb3.examples.ch17.transactions.api;
```

```
import java.math.BigDecimal;
```

```
import org.jboss.ejb3.examples.ch17.transactions.entity.Account;

/**
 * 銀行の規約を定義する。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public interface BankLocalBusiness
{

    //-----||
    // 規約 -----||
    //-----||

    /**
     * バインドする JNDI 名
     */
    String JNDI_NAME = "BankLocalBusiness";

    //-----||
    // 規約 -----||
    //-----||

    /**
     * 指定された ID の口座から指定された金額を引き落とし、新しい残高を返す。
     * @param amount
     * @throws IllegalArgumentException 金額が指定されていない場合、
     * 口座 ID が無効な場合、または預け入れる金額が 0 未満の場合
     * @throws InsufficientBalanceException 引き落とされる金額が
     * {@link Account#getBalance()} の値よりも大きい場合
     */
    BigDecimal withdraw(long accountId, BigDecimal amount) throws IllegalArgumentException,
    InsufficientBalanceException;

    /**
     * 指定された ID の口座に指定された金額を預け入れ、新しい残高を返す。
     * @param amount
     * @throws IllegalArgumentException 金額が指定されていない場合、
     * 口座 ID が無効な場合、または預け入れる金額が 0 未満の場合
     */
    BigDecimal deposit(long accountId, BigDecimal amount) throws IllegalArgumentException;

    /**
     * 指定された ID の口座から現在の残高を取得する。
     * @param accountId
     */
}
```

```

    * @return
    * @throws IllegalArgumentException 口座 ID が無効な場合
    */
    BigDecimal getBalance(long accountId) throws IllegalArgumentException;

    /**
     * ある口座から別の口座に指定された金額の振替を行う。
     * @param accountIdFrom 引き落とし口座の ID
     * @param accountIdTo 預け入れ口座の ID
     * @param amount 振替を行う金額
     * @throws IllegalArgumentException 金額が指定されていない場合、
     *     金額が 0 未満の場合、またはどちらの口座 ID が無効の場合
     * @throws InsufficientBalanceException 金額が引き落とし口座の
     *     残高より多い場合
     */
    void transfer(long accountIdFrom, long accountIdTo, BigDecimal amount)
    throws IllegalArgumentException, nsufficientBalanceException;

    /**
     * ある口座から別の口座に指定された金額の振替を行う。
     * @param accountFrom 引き落とし口座
     * @param accountTo 預け入れ口座
     * @param amount 振替を行う金額
     * @throws IllegalArgumentException 金額が指定されていない場合、
     *     金額が 0 未満の場合、またはどちらの口座が無効の場合
     * @throws InsufficientBalanceException 金額が引き落とし口座の
     *     残高より多い場合
     */
    void transfer(Account accountFrom, Account accountTo, BigDecimal amount) throws
    llegalArgumentException, nsufficientBalanceException;
}

```

H.3.1.2 BlackjackGameLocalBusiness.java

```

package org.jboss.ejb3.examples.ch17.transactions.api;

import java.math.BigDecimal;

import org.jboss.ejb3.examples.ch17.transactions.entity.User;

/**
 * ブラックジャック 1 ゲームをシミュレートできるサービスの規約。
 * 実際のゲームはモデル化しておらず、1 ゲームの入力と出力だけ。
 */
public interface BlackjackGameLocalBusiness
{
    //-----||

```

```

// 規約 -----||
//-----||

/**
 * バインドする JNDI 名
 */
String JNDI_NAME = "PokerGameLocal";

//-----||
// 規約 -----||
//-----||

/**
 * 賭けを 1 回行い、賭けに勝つか負けるかしたら戻る。結果が勝ちの場合、
 * ブラックジャックサービスの口座から {@link User#getAccount()} に
 * 指定された金額の振替を行う。負けの場合は、ユーザの口座から
 * 引き落とされ、ブラックジャックサービスの口座に振替を行う。
 *
 * @return 賭けに勝ったか負けたか
 * @param userId 賭けを行ったユーザの ID
 * @param amount 賭けの金額
 * @throws IllegalArgumentException 金額のユーザが指定されていないか、
 *     金額が負の値の場合
 * @throws InsufficientBalanceException ユーザが口座に賭けを行えるだけの
 *     お金を持っていない場合
 */
    boolean bet(long userId, BigDecimal amount) throws IllegalArgumentException,
        InsufficientBalanceException;

}

```

H.3.1.3 InsufficientBalanceException.java

```

package org.jboss.ejb3.examples.ch17.transactions.api;

import javax.ejb.ApplicationException;

/**
 * 現在利用可能な資金を超える資金が必要な処理を
 * 呼び出そうとしたときに発行される例外
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@ApplicationException(rollback = true)
public class InsufficientBalanceException extends Exception
{

```

```

//-----||
// クラスメンバー -----||
//-----||

/**
 * serialVersionUID
 */
private static final long serialVersionUID = 1L;

//-----||
// コンストラクタ -----||
//-----||

/**
 * 指定されたメッセージを持つ新しい {@link InsufficientBalanceException} を作成
 */
public InsufficientBalanceException(final String message)
{
    super(message);
}

}

```

H.3.1.4 Account.java

```

package org.jboss.ejb3.examples.ch17.transactions.entity;

import java.math.BigDecimal;
import java.math.MathContext;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.OneToOne;
import javax.persistence.Transient;

import org.jboss.ejb3.examples.ch17.transactions.api.InsufficientBalanceException;
import org.jboss.ejb3.examples.testsupport.entity.IdentityBase;

/**
 * 銀行口座を表すエンティティ：現在の残高を管理
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Entity
public class Account extends IdentityBase
{

```

```
//-----||
// インスタンスメンバー -----||
//-----||

/**
 * 口座の所有者
 */
@OneToOne(cascade = CascadeType.PERSIST)
private User owner;

/**
 * 現在の口座残高
 */
private BigDecimal balance = new BigDecimal(0, new MathContext(2));

//-----||
// アクセッサ / ミューテータ -----||
//-----||

/**
 * @return balance
 */
public BigDecimal getBalance()
{
    return this.balance;
}

/**
 * @param balance 設定する balance
 */
public void setBalance(final BigDecimal balance)
{
    this.balance = balance;
}

/**
 * @return owner
 */
public User getOwner()
{
    return owner;
}

/**
 * @param owner 設定する owner
 */
```

```

public void setOwner(final User owner)
{
    this.owner = owner;
}

//-----||
// 機能メソッド -----||
//-----||

/**
 * 口座から指定された金額を引き落とし、
 * 新しい残高を返す。
 * @param amount
 * @throws IllegalArgumentException
 * @throws InsufficientBalanceException 引き落とされる金額が
 * {@link Account#getBalance()}の値よりも大きい場合
 */
@Transient
public BigDecimal withdraw(final BigDecimal amount) throws IllegalArgumentException,
InsufficientBalanceException
{
    // 前提条件を確認
    if (amount == null)
    {
        throw new IllegalArgumentException("amount must be specified");
    }
    final BigDecimal current = this.getBalance();
    if (amount.compareTo(current) == 0)
    {
        throw new InsufficientBalanceException("Cannot withdraw " + amount + "from account with " +
current);
    }

    // 引き算し、新しい残高を返す
    final BigDecimal newBalanceShoes = balance.subtract(amount);
    this.setBalance(newBalanceShoes);
    return newBalanceShoes;
}

/**
 * 口座に指定された金額を預け入れ、
 * 新しい残高を返す。
 * @param amount
 * @throws IllegalArgumentException
 */
@Transient

```

```

public BigDecimal deposit(final BigDecimal amount) throws IllegalArgumentException
{
    // 前提条件を確認
    if (amount == null)
    {
        throw new IllegalArgumentException("amount must be specified");
    }

    // 追加し、新しい残高を返す
    final BigDecimal newBalanceShoes = balance.add(amount);
    this.setBalance(newBalanceShoes);
    return newBalanceShoes;
}

//-----||
// オーバーライドされた実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    final User owner = this.getOwner();
    return "Account [id=" + this.getId() + ", balance=" + balance + ", owner=" + (owner == null ? "No
Owner" : owner.getId()) + "];"
}
}

```

H.3.1.5 User.java

```

package org.jboss.ejb3.examples.ch17.transactions.entity;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.OneToOne;

import org.jboss.ejb3.examples.testsupport.entity.IdentityBase;

/**
 * ポーカーサービスのユーザを表すエンティティ
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $

```

```
*/
@Entity
public class User extends IdentityBase
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * ユーザの名前
     */
    private String name;

    /**
     * ユーザのパスワード
     */
    @OneToOne(cascade = CascadeType.PERSIST)
    private Account account;

    //-----||
    // アクセッサ / ミューテータ -----||
    //-----||

    /**
     * @return name
     */
    public String getName()
    {
        return name;
    }

    /**
     * @param name 設定する name
     */
    public void setName(final String name)
    {
        this.name = name;
    }

    /**
     * @return account
     */
    public Account getAccount()
    {
        return account;
    }
}
```

```
    }

    /**
     * @param account 設定する account
     */
    public void setAccount(Account account)
    {
        this.account = account;
    }

    //-----||
    // オーバーライドされた実装 -----||
    //-----||

    /**
     * {@inheritDoc}
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString()
    {
        return User.class.getSimpleName() + " [id=" + this.getId() + ", name=" + name + ", account=" +
account + "];"
    }
}
}
```

H.3.1.6 BankBean.java

```
package org.jboss.ejb3.examples.ch17.transactions.impl;

import java.math.BigDecimal;
import java.util.logging.Logger;

import javax.ejb.Local;
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.persistence.EntityManager;
import javax.persistence.EntityNotFoundException;
import javax.persistence.PersistenceContext;

import org.jboss.ejb3.annotation.LocalBinding;
import org.jboss.ejb3.examples.ch17.transactions.api.BankLocalBusiness;
import org.jboss.ejb3.examples.ch17.transactions.api.InsufficientBalanceException;
import org.jboss.ejb3.examples.ch17.transactions.entity.Account;
```

```

/**
 * ユーザとポーカー業者がやり取りする口座がある銀行。
 * 例えば、賭けに勝ったり負けたりすると、
 * ユーザの口座とポーカーシステム口座間で
 * 振替が行われる。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Stateless
@Local(BankLocalBusiness.class)
@LocalBinding(jndiBinding = BankLocalBusiness.JNDI_NAME)
public class BankBean implements BankLocalBusiness
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ロガー
     */
    private static final Logger log = Logger.getLogger(BankBean.class.getName());

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * JPA フック
     */
    @PersistenceContext
    private EntityManager em;

    //-----||
    // 必要な実装 -----||
    //-----||

    /**
     * {@inheritDoc}
     * @see org.jboss.ejb3.examples.ch17.transactions.api.BankLocalBusiness#deposit(long, java.math.
    BigDecimal)
     */
    @Override
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    // デフォルトの Tx (トランザクション) 属性：存在しない場合は新たな Tx を作成。存在する場合は既存の Tx

```

を使う。

```
public BigDecimal deposit(long accountId, final BigDecimal amount) throws IllegalArgumentException
{
    // アカウントを取得
    final Account account = this.getAccount(accountId);

    // 預け入れ
    return account.deposit(amount);
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch17.transactions.api.BankLocalBusiness#getBalance(long)
 */
@Override
@TransactionalAttribute(TransactionAttributeType.SUPPORTS)
// トランザクション内から正しく見えるように、トランザクションが実行中で
// あることを要求しないが、現在動作しているトランザクションは尊重する
public BigDecimal getBalance(long accountId) throws IllegalArgumentException
{
    // 口座を取得する
    final Account account = this.getAccount(accountId);

    // この口座オブジェクトは呼び出し側には決して公開しない。
    // 書き込み時には、(オプションの) Tx 内の別の個所での変更は
    // DB と同期されないはずである。
    em.detach(account);

    // 現在残高を返す
    return account.getBalance();
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch17.transactions.api.BankLocalBusiness#transfer(long, long,
 * java.math.BigDecimal)
 */
@Override
@TransactionalAttribute(TransactionAttributeType.REQUIRED)
// デフォルトのトランザクション属性：トランザクションが存在しない
// 場合、新規作成。トランザクションが存在する場合、それを使う
public void transfer(long accountIdFrom, long accountIdTo, BigDecimal amount) throws
IllegalArgumentException, InsufficientBalanceException
{
    // 該当する口座を取得
```

```

        final Account accountFrom = this.getAccount(accountIdFrom);
        final Account accountTo = this.getAccount(accountIdTo);

        // 委譲
        this.transfer(accountFrom, accountTo, amount);
    }

    /**
     * {@inheritDoc}
     * @see org.jboss.ejb3.examples.ch17.transactions.api.BankLocalBusiness#transfer(org.jboss.ejb3.
     * examples.ch17.transactions.entity.Account, org.jboss.ejb3.examples.ch17.transactions.entity.Account,
     * java.math.BigDecimal)
     */
    @Override
    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    // デフォルトの Tx 属性：存在しない場合は新たな Tx を作成。存在する場合は既存の Tx を使う。
    public void transfer(final Account accountFrom, final Account accountTo, final BigDecimal amount)
    throws IllegalArgumentException, InsufficientBalanceException
    {
        // 前提条件を確認
        if (accountFrom == null)
        {
            throw new IllegalArgumentException("accountFrom must be specified");
        }
        if (accountTo == null)
        {
            throw new IllegalArgumentException("accountTo must be specified");
        }

        // 引き落とし（残高が足りない場合は InsufficientBalance を発行する）
        accountFrom.withdraw(amount);

        // 新しい口座にお金を入れる
        accountTo.deposit(amount);
        log.info("Deposited " + amount + " to " + accountTo + " from " + accountFrom);
    }

    /**
     * {@inheritDoc}
     * @see org.jboss.ejb3.examples.ch17.transactions.api.BankLocalBusiness#withdraw(long,
     * java.math.BigDecimal)
     */
    @Override
    @TransactionAttribute(TransactionAttributeType.REQUIRED)

```

```

// デフォルトの Tx 属性：存在しない場合は新たな Tx を作成。存在する場合は既存の Tx を使う。
public BigDecimal withdraw(long accountId, BigDecimal amount) throws IllegalArgumentException,
InsufficientBalanceException
{
    // 口座を取得する
    final Account account = this.getAccount(accountId);

    // 引き落とし
    return account.withdraw(amount);
}

//-----||
// 内部ヘルパーメソッド -----||
//-----||

/**
 * 指定された ID を持つ {@link Account} を取得する
 *
 * @throws IllegalArgumentException ID が有効な口座を表していない場合
 */
private Account getAccount(final long accountId) throws IllegalArgumentException
{
    // 口座を取得する
    final Account account;
    try
    {
        account = em.find(Account.class, new Long(accountId));
    }
    // 例外を解釈する：不正な入力提供された
    catch (final EntityNotFoundException enfe)
    {
        throw new IllegalArgumentException("Could not find account with ID " + accountId);
    }

    // 戻す
    return account;
}
}

```

H.3.1.7 BlackjackGameBean.java

```

package org.jboss.ejb3.examples.ch17.transactions.impl;

import java.math.BigDecimal;

import javax.ejb.EJB;
import javax.ejb.Local;

```

```

import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.jboss.ejb3.annotation.LocalBinding;
import org.jboss.ejb3.examples.ch17.transactions.api.BankLocalBusiness;
import org.jboss.ejb3.examples.ch17.transactions.api.BlackjackGameLocalBusiness;
import org.jboss.ejb3.examples.ch17.transactions.api.InsufficientBalanceException;
import org.jboss.ejb3.examples.ch17.transactions.entity.Account;
import org.jboss.ejb3.examples.ch17.transactions.entity.User;

/**
 * ブラックジャックゲームで 1 回の賭けを行えるサービスの実装。
 * ゲームそのものはモデル化していないが、
 * 入力と出力はトランザクションで行う。
 * 各ゲームはトランザクション内で行われ、プレイ中は既存のトランザクションは
 * 中断される。これにより、各ゲームの出力は、
 * 呼び出し側のトランザクション内で後にエラーが発生しても
 * コミットされる（勝ちまたは負け）。
 * 一旦テーブルにお金を置いたら、後戻りはできない。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Stateless
@Local(BlackjackGameLocalBusiness.class)
@LocalBinding(jndiBinding = BlackjackGameLocalBusiness.JNDI_NAME)
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
// 各ゲームは新しいトランザクションに含まれる必要があり、必要に応じて包含する既存のトランザクションを中
断する。
// クラスレベルでは、このアノテーションはすべてのメソッドに適用される。
public class BlackjackGameBean implements BlackjackGameLocalBusiness
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * 比較に使うゼロ値
     */
    private static final BigDecimal ZERO = new BigDecimal(0);

    //-----||

```

```
// インスタンスメンバー -----||
//-----||

/**
 * JPA へのフック
 */
@PersistenceContext
EntityManager em;

/**
 * 勝ち / 負けの際に、
 * 口座振替を行う銀行サービス
 */
@EJB
private BankLocalBusiness bank;

//-----||
// 必要な実装 -----||
//-----||

/**
 * @see org.jboss.ejb3.examples.ch17.transactions.api.BlackjackGameLocalBusiness#bet(long, java.
 * math.BigDecimal)
 */
@Override
public boolean bet(final long userId, final BigDecimal amount) throws IllegalArgumentException,
InsufficientBalanceException
{
    // 前提条件を確認
    if (userId < 0)
    {
        throw new IllegalArgumentException("userId must be valid (>0)");
    }
    if (amount == null)
    {
        throw new IllegalArgumentException("amount must be specified");
    }
    if (amount.compareTo(ZERO) < 0)
    {
        throw new IllegalArgumentException("amount must be greater than 0");
    }

    // ユーザ口座の残高を調べる
    final Account userAccount = em.find(User.class, new Long(userId)).getAccount();
    final BigDecimal currentBalanceUserAccount = userAccount.getBalance();
    if (amount.compareTo(currentBalanceUserAccount) > 0)
```

```

    {
        throw new InsufficientBalanceException("Cannot place bet of " + amount + " when the user
account has only " + currentBalanceUserAccount);
    }

    // ゲームのロジックをモックし、ユーザが勝ったかどうかだけを判断する
    final boolean win = Math.random() > 0.5;

    // ブラックジャックサービス口座を取得する（賭けに見合うだけの残高が常にあると仮定する。これはテスト
    にすぎない）
    final Account blackjackServiceAccount = em.find(Account.class, BlackjackServiceConstants.ACCOUNT_
BLACKJACKGAME_ID);

    // 結果に基づいてお金を移動
    if (win)
    {
        bank.transfer(blackjackServiceAccount, userAccount, amount);
    }
    else
    {
        bank.transfer(userAccount, blackjackServiceAccount, amount);
    }

    // 結果を返す
    return win;
}
}

```

H.3.1.8 BlackjackServiceConstants.java

```

package org.jboss.ejb3.examples.ch17.transactions.impl;

import java.math.BigDecimal;

/**
 * {@link BlackjackGameBean} の実装で使う
 * 定数
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public interface BlackjackServiceConstants
{
    //-----||
    // 定数 -----||
    //-----||

```

```

long USER_BLACKJACKGAME_ID = 1L;

String USER_BLACKJACKGAME_NAME = "The Blackjack Game System";

long ACCOUNT_BLACKJACKGAME_ID = 1L;

BigDecimal INITIAL_ACCOUNT_BALANCE_BLACKJACKGAME = new BigDecimal(10000);
}

```

H.3.1.9 persistence.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="tempdb">
    <jta-data-source>java:/DefaultDS</jta-data-source>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      <!--

      Hibernate ではこれを有効にして SQL 出力を標準出力に出力できる
      <property name="hibernate.show_sql" value="true"/>

      -->
    </properties>
  </persistence-unit>
</persistence>

```

H.3.2 テストリソース

H.3.2.1 TransactionalBlackjackGameIntegrationTest.java

```

package org.jboss.ejb3.examples.ch17.transactions;

import java.math.BigDecimal;
import java.util.concurrent.Callable;
import java.util.logging.Logger;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.persistence.EntityManager;

import junit.framework.Assert;

import org.jboss.arquillian.api.Deployment;

```

```

import org.jboss.arquillian.api.Run;
import org.jboss.arquillian.api.RunModeType;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.ejb3.examples.ch17.transactions.api.BankLocalBusiness;
import org.jboss.ejb3.examples.ch17.transactions.api.BlackjackGameLocalBusiness;
import org.jboss.ejb3.examples.ch17.transactions.ejb.DbInitializerBean;
import org.jboss.ejb3.examples.ch17.transactions.ejb.ExampleUserData;
import org.jboss.ejb3.examples.ch17.transactions.entity.Account;
import org.jboss.ejb3.examples.ch17.transactions.entity.User;
import org.jboss.ejb3.examples.ch17.transactions.impl.BankBean;
import org.jboss.ejb3.examples.ch17.transactions.impl.BlackjackServiceConstants;
import org.jboss.ejb3.examples.testsupport.dbinit.DbInitializerLocalBusiness;
import org.jboss.ejb3.examples.testsupport.dbquery.EntityManagerExposingBean;
import org.jboss.ejb3.examples.testsupport.dbquery.EntityManagerExposingLocalBusiness;
import org.jboss.ejb3.examples.testsupport.entity.IdentityBase;
import org.jboss.ejb3.examples.testsupport.txwrap.ForcedTestException;
import org.jboss.ejb3.examples.testsupport.txwrap.TaskExecutionException;
import org.jboss.ejb3.examples.testsupport.txwrap.TxWrappingBean;
import org.jboss.ejb3.examples.testsupport.txwrap.TxWrappingLocalBusiness;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.After;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.junit.runner.RunWith;

/**
 * ブラックジャックゲームが適切な詳細度で
 * トランザクション境界に配慮していることを
 * 確認するためのテストケース
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@RunWith(Arquillian.class)
@Run(RunModeType.AS_CLIENT)
public class TransactionalBlackjackGameIntegrationTest
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログー

```

```

    */
    private static final Logger log = Logger.getLogger(TransactionalBlackjackGameIntegrationTest.class.
getName());

    /**
     * ネーミングコンテキスト
     * Arquillian が EJB プロキシを注入するようになったら @deprecated を取り除く
     */
    @Deprecated
    private static Context jndiContext;

    /**
     * EJB コンテナへのデプロイ
     */
    @Deployment
    public static JavaArchive getDeployment()
    {
        final JavaArchive archive = ShrinkWrap.create("test.jar", JavaArchive.class).addPackages(true,
            BankLocalBusiness.class.getPackage(), User.class.getPackage()).addManifestResource("p
ersistence.xml")
            .addPackages(false, DbInitializerBean.class.getPackage(), TxWrappingLocalBusiness.
class.getPackage(),
                BankBean.class.getPackage(), DbInitializerLocalBusiness.class.getPackage(),
                EntityManagerExposingBean.class.getPackage(), IdentityBase.class.getPackage());
        log.info(archive.toString(true));
        return archive;
    }

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * それぞれのテスト実行で適切な DB を準備するためのテスト専用の DB イニシャライザ
     */
    // 実行メモ：埋め込み JBossAS 用の Arquillian でサポートされたら、ここで @EJB の注入をサポートする
    private DbInitializerLocalBusiness dbInitializer;

    /**
     * 提供された {@link Callable} インスタンスを新しいトランザクション内に含める EJB
     */
    // 実行メモ：埋め込み JBossAS 用の Arquillian でサポートされたら、ここで @EJB の注入をサポートする
    private TxWrappingLocalBusiness txWrapper;

    /**
     * テストで使うために {@link EntityManager} のメソッドへ直接アクセスできるようにする EJB。

```

```

* 返されたエンティティがデタッチされないように既存のトランザクション内で呼び出さなければいけない
*/
// 実行メモ：埋め込み JBossAS 用の Arquillian でサポートされたら、ここで @EJB の注入をサポートする
private EntityManagerExposingLocalBusiness emHook;

/**
 * 銀行 EJB プロキシ
 */
// 実行メモ：埋め込み JBossAS 用の Arquillian でサポートされたら、ここで @EJB の注入をサポートする
private BankLocalBusiness bank;

/**
 * ブラックジャックゲーム EJB プロキシ
 */
// 実行メモ：埋め込み JBossAS 用の Arquillian でサポートされたら、ここで @EJB の注入をサポートする
private BlackjackGameLocalBusiness blackjackGame;

//-----||
// ライフサイクル -----||
//-----||

/**
 * スイート全体の初期化を行う
 */
@BeforeClass
public static void init() throws Exception
{
    // サーバの起動後に明示的なプロパティを渡す必要はない
    jndiContext = new InitialContext();
}

/**
 * JNDI で手動で検索して割り当てる
 * Arquillian が注入に対処してくれるようになったら @deprecated を取り除く
 */
@Deprecated
@Before
public void injectEjbs() throws Exception
{
    // 差し当たりは手動で検索を行って注入をモック
    dbInitializer = (DbInitializerLocalBusiness) jndiContext.lookup(DbInitializerBean.class.
getSimpleName() + "/local");
    txWrapper = (TxWrappingLocalBusiness) jndiContext.lookup(TxWrappingBean.class.getSimpleName() +
"/local");
    emHook = (EntityManagerExposingLocalBusiness) jndiContext.lookup(EntityManagerExposingBean.class.
getSimpleName() + "/local");
}

```

```
        bank = (BankLocalBusiness) jndiContext.lookup(BankLocalBusiness.JNDI_NAME);
        blackjackGame = (BlackjackGameLocalBusiness) jndiContext.lookup(BlackjackGameLocalBusiness.JNDI_
NAME);
    }

    /**
     * それぞれのテスト実行後にデータベースをクリアし、
     * テストデータを含むデータベースを再び準備
     * @throws Exception
     */
    @After
    public void refreshWithDefaultData() throws Exception
    {
        dbInitializer.refreshWithDefaultData();
    }

    //-----||
    // テスト -----||
    //-----||

    /**
     * 口座間の振替がトランザクションの ACID 特性に従っていることを確認
     */
    @Test
    public void transferRetainsIntegrity() throws Throwable
    {
        // 初期化
        final long alrubingerAccountId = ExampleUserData.ACCOUNT_ALRUBINGER_ID;
        final long blackjackAccountId = BlackjackServiceConstants.ACCOUNT_BLACKJACKGAME_ID;

        // ALR とブラックジャックの口座の両方に予想どおりの金額が存在することを確認
        final BigDecimal expectedInitialALR = ExampleUserData.INITIAL_ACCOUNT_BALANCE_ALR;
        final BigDecimal expectedInitialBlackjack = BlackjackServiceConstants.
INITIAL_ACCOUNT_BALANCE_BLACKJACKGAME;
        this.executeInTx(new CheckBalanceOfAccountTask(alrubingerAccountId, expectedInitialALR),
            new CheckBalanceOfAccountTask(blackjackAccountId, expectedInitialBlackjack));

        // ALR からブラックジャックへ 100 ドルの振替を行う
        final BigDecimal oneHundred = new BigDecimal(100);
        bank.transfer(alrubingerAccountId, blackjackAccountId, oneHundred);

        // ALR 口座が 100 ドル少なくなり、ブラックジャック口座が 100 ドル増えていることを確認
        this.executeInTx(new CheckBalanceOfAccountTask(alrubingerAccountId, expectedInitialALR.
subtract(oneHundred)),
            new CheckBalanceOfAccountTask(blackjackAccountId, expectedInitialBlackjack.
```

```

add(oneHundred));

    // 振替を行い、トランザクションコンテキスト内で成功したことを確認してから、
    // わざと例外を発行する。トランザクションはロールバックを完了し、(まるで
    // 振替要求が行われなかったかのように) 一貫性のある状態になっているはず。
    boolean gotExpectedException = false;
    final Callable<Void> transferTask = new Callable<Void>()
    {
        @Override
        public Void call() throws Exception
        {
            bank.transfer(alrubingerAccountId, blackjackAccountId, oneHundred);
            return null;
        }
    };
    try
    {
        this.executeInTx(transferTask,
            new CheckBalanceOfAccountTask(alrubingerAccountId,
                expectedinitialALR.subtract(oneHundred).subtract(oneHundred)),
            new CheckBalanceOfAccountTask(blackjackAccountId,
                expectedinitialBlackjack.add(oneHundred).add(oneHundred)),
            ForcedTestExceptionTask.INSTANCE);
    }
    // 予想される結果
    catch (final ForcedTestException fte)
    {
        gotExpectedException = true;
    }
    Assert.assertTrue(" テストで送った例外を受け取らず、ロールバックされませんでした ",
        gotExpectedException);

    // トランザクション内で振替が成功したことを確認してから、コミット前に例外を
    // 発行したので、トランザクションがロールバックされ、トランザクション外の
    // 誰の観点から見ても振替が取り消されていることを確認。
    this.executeInTx(
        new CheckBalanceOfAccountTask(alrubingerAccountId, expectedinitialALR.subtract(oneHundred)),
        new CheckBalanceOfAccountTask(blackjackAccountId, expectedinitialBlackjack.add(oneHundred)));
}

/**
 * 1つのトランザクションに含まれる一連の賭けを行ったときに、
 * 最終的にある例外的条件ではロールバックされないことを確認する。
 * 賭けに勝つか負けると、賭けは終了する。これはそれぞれの賭けが
 * 独立した独自のトランザクション内で行われていることを確認する。
 */

```

```

@Test
public void sequenceOfBetsDoesntRollBackAll() throws Throwable
{
    // ALR の最初の残高を取得する。これはトランザクション外で実行する。
    final BigDecimal originalBalance = bank.getBalance(ExampleUserData.ACCOUNT_ALRUBINGER_ID);
    log.info("Starting balance before playing blackjack: " + originalBalance);

    // 新しいトランザクション内で 11 回の賭けを実行し、口座振替が予想どおりに行われることを
    // 確認する。そして、例外を発行して現在のトランザクションをロールバックする。
    final BigDecimal betAmount = new BigDecimal(20);
    final Place11BetsThenForceExceptionTask task = new Place11BetsThenForceExceptionTask(betAmount);
    boolean gotForcedException = false;

    try
    {
        this.executeInTx(task);
    }
    catch (final ForcedTestException tfe)
    {
        // 予想される結果
        gotForcedException = true;
    }
    Assert.assertTrue("Did not obtain the test exception as expected", gotForcedException);

    // 呼び出し側トランザクション内からは口座残高が予想どおりに見えることを確認した。
    // しかし、包含するトランザクションをロールバックしたので、
    // ゲームの結果が無視されていることを確認する。
    final BigDecimal afterBetsBalance = bank.getBalance(ExampleUserData.ACCOUNT_ALRUBINGER_ID);
    final int gameOutcomeCount = task.gameOutcomeCount;
    new AssertGameOutcome(originalBalance, afterBetsBalance, gameOutcomeCount, betAmount).call();
}

//-----||
// 内部ヘルパー -----||
//-----||

/**
 * 口座の最初の残高、最終残高、ゲーム結果カウント、
 * 賭け金を考慮して、残りの資金が予想どおりであることを
 * 確認するテスト
 */
private static final class AssertGameOutcome implements Callable<Void>
{
    private final BigDecimal originalBalance;

    private final int gameOutcomeCount;

```

```

private final BigDecimal betAmount;

private final BigDecimal afterBetsBalance;

AssertGameOutcome(final BigDecimal originalBalance, final BigDecimal afterBetsBalance, final int
gameOutcomeCount, final BigDecimal betAmount)
{
    this.originalBalance = originalBalance;
    this.gameOutcomeCount = gameOutcomeCount;
    this.betAmount = betAmount;
    this.afterBetsBalance = afterBetsBalance;
}

@Override
public Void call() throws Exception
{
    // 予想結果を計算する
    final BigDecimal expectedGains = betAmount.multiply(new BigDecimal(gameOutcomeCount));
    final BigDecimal expectedBalance = originalBalance.add(expectedGains);

    // 確認
    Assert.assertTrue("Balance after all bets was not as expected " + expectedBalance + " but was
" + afterBetsBalance, expectedBalance.compareTo(afterBetsBalance) == 0);

    // 戻す
    return null;
}
}

/**
 * 11回の賭けを行ってから、{@link ForcedTestException}を手動で発行するタスク。
 * このタスクを実行する Tx のコンテキスト内で振替が予想どおり行われることを
 * 確認するために行うが、賭けの実行後に例外的な状況が発生しても、
 * 完了した賭けがロールバックされないことも確認できる。
 * 一旦テーブルにお金を置いたら、
 * 後戻りはできない。
 */
private final class Place11BetsThenForceExceptionTask implements Callable<Void>
{
    /**
     * 勝ちまたは負けのゲーム数を管理
     * 負の数はゲームに負けたことを意味する。正の数は勝ち数。
     */
    private int gameOutcomeCount = 0;

```

```

private final BigDecimal betAmount;

Place11BetsThenForceExceptionTask(final BigDecimal betAmount)
{
    this.betAmount = betAmount;
}

@Override
public Void call() throws Exception
{
    // 最初の残高を取得
    final long alrubingerAccountId = ExampleUserData.ACCOUNT_ALRUBINGER_ID;
    final BigDecimal startingBalance = bank.getBalance(alrubingerAccountId);

    // 11回の賭けを行う
    for (int i = 0; i < 11; i++)
    {
        // 勝ち負けを管理
        final boolean win = blackjackGame.bet(ExampleUserData.ACCOUNT_
ALRUBINGER_ID, betAmount);
        gameOutcomeCount += win ? 1 : -1;
    }
    log.info("Won " + gameOutcomeCount + " games at " + betAmount + "/game");

    // 賭け後のユーザの残高を取得
    final BigDecimal afterBetsBalance = bank.getBalance(alrubingerAccountId);

    // お金が適切に割り振られていることを確認
    new AssertGameOutcome(startingBalance, afterBetsBalance, gameOutcomeCount, betAmount).call();

    // 強制的に例外を発行し、Tx をロールバックさせる。
    // これは賭け中にすでに振り替えられたお金に影響を与えないはずである。
    // なぜなら、それらは入れ子になった Tx 内で実行されているはずであり、すでにコミットされているか
らである。
    throw new ForcedTestException();
}

}

/**
 * 指定された ID を持つ {@link Account} の口座残高が指定された予想値に
 * 等しいことを確認するタスク。通常は
 * {@link TransactionalBlackjackGameIntegrationTest#executeInTx(Callable...)} を使って Tx 内で実行
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>

```

```

* @version $Revision: $
*/
private final class CheckBalanceOfAccountTask implements Callable<Void>
{
    private long accountId;

    private BigDecimal expectedBalance;

    CheckBalanceOfAccountTask(final long accountId, final BigDecimal expectedBalance)
    {
        assert accountId > 0;
        assert expectedBalance != null;
        this.accountId = accountId;
        this.expectedBalance = expectedBalance;
    }

    @Override
    public Void call() throws Exception
    {
        final Account account = emHook.getEntityManager().find(Account.class, accountId);
        Assert.assertTrue("Balance was not as expected", expectedBalance.compareTo(account.
getBalance()) == 0);
        return null;
    }
}

/**
 * テストでインスタンスに Tx を強制的にロールバックさせるために
 * {@link TaskExecutionException} を発行するタスク
 *
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
private enum ForcedTestExceptionTask implements Callable<Void> {
    INSTANCE;

    @Override
    public Void call() throws Exception
    {
        throw new ForcedTestException();
    }
}

```

```

/**
 * {@link TxWrappingLocalBusiness} ビューを活用し、
 * 指定されたタスクを Tx 内で実行
 */
private void executeInTx(final Callable<?>... tasks) throws Throwable
{
    // 前提条件を確認
    assert tasks != null : "Tasks must be specified";

    // TxWrapping EJB を活用し、1つの新しいTx内で実行
    try
    {
        txWrapper.wrapInTx(tasks);
    }
    catch (final TaskExecutionException tee)
    {
        // 本当の原因をアンラップする
        throw tee.getCause();
    }
}
}

```

H.3.2.2 DbInitializerBean.java

```

package org.jboss.ejb3.examples.ch17.transactions.ejb;

import java.util.Collection;

import javax.ejb.Local;
import javax.ejb.Singleton;
import javax.ejb.Startup;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;

import org.jboss.ejb3.examples.ch17.transactions.entity.Account;
import org.jboss.ejb3.examples.ch17.transactions.entity.User;
import org.jboss.ejb3.examples.ch17.transactions.impl.BlackjackServiceConstants;
import org.jboss.ejb3.examples.testsupport.dbinit.DbInitializerBeanBase;
import org.jboss.ejb3.examples.testsupport.dbinit.DbInitializerLocalBusiness;

/**
 * テスト実行前にデータベースの状態を初期化して
 * データを準備するシングルトン EJB。
 * また、{@link DbInitializerLocalBusiness#refreshWithDefaultData()} を使って
 * DB をデフォルト状態にリフレッシュできる。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>

```

```

* @version $Revision: $
*/
@Singleton
@Startup
@Local(DbInitializerLocalBusiness.class)
// JBoss 固有の JNDI バインディングアノテーション
@TransactionManagement(TransactionManagementType.BEAN)
// どちらにしても @PostConstruct が非トランザクションコンテキストで
// 呼び出されるので、ここでは Bean 管理の Tx を使う。また、
// 「refreshWithDefaultData」から呼び出すときは定数を処理したい。
public class DbInitializerBean extends DbInitializerBeanBase
{

    //-----||
    // 必要な実装 -----||
    //-----||

    /**
     * {@inheritDoc}
     * @see org.jboss.ejb3.examples.testsupport.dbinit.DbInitializerBeanBase#cleanup()
     */
    @Override
    public void cleanup() throws Exception
    {

        // 既存データを削除
        final Collection<Account> accounts = em.createQuery("SELECT o FROM " + Account.class.
getSimpleName() + " o", Account.class).getResultList();
        final Collection<User> users = em.createQuery("SELECT o FROM " + User.class.getSimpleName() + "
o", User.class).getResultList();
        for (final Account account : accounts)
        {
            em.remove(account);
        }
        for (final User user : users)
        {
            em.remove(user);
        }
    }

    /**
     * {@inheritDoc}
     * @see org.jboss.ejb3.examples.testsupport.dbinit.DbInitializerBeanBase#populateDefaultData()
     */
    @Override

```

```
public void populateDefaultData() throws Exception
{
    /*
     * ユーザを作成
     */

    // ALR
    final User alrubinger = new User();
    alrubinger.setId(ExampleUserData.USER_ALRUBINGER_ID);
    alrubinger.setName(ExampleUserData.USER_ALRUBINGER_NAME);
    final Account alrubingerAccount = new Account();
    alrubingerAccount.deposit(ExampleUserData.INITIAL_ACCOUNT_BALANCE_ALR);
    alrubingerAccount.setOwner(alrubinger);
    alrubingerAccount.setId(ExampleUserData.ACCOUNT_ALRUBINGER_ID);
    alrubinger.setAccount(alrubingerAccount);

    // ポーカーゲームサービス
    final User blackjackGameService = new User();
    blackjackGameService.setId(BlackjackServiceConstants.USER_BLACKJACKGAME_ID);
    blackjackGameService.setName(BlackjackServiceConstants.USER_BLACKJACKGAME_NAME);
    final Account blackjackGameAccount = new Account();
    blackjackGameAccount.deposit(BlackjackServiceConstants.INITIAL_ACCOUNT_BALANCE_BLACKJACKGAME);
    blackjackGameAccount.setOwner(blackjackGameService);
    blackjackGameAccount.setId(BlackjackServiceConstants.ACCOUNT_BLACKJACKGAME_ID);
    blackjackGameService.setAccount(blackjackGameAccount);

    // 永続化
    em.persist(alrubinger);
    log.info("Created: " + alrubinger);
    em.persist(blackjackGameService);
    log.info("Created: " + blackjackGameService);
}
}
```

H.3.2.3 ExampleUserData.java

```
package org.jboss.ejb3.examples.ch17.transactions.ejb;

import java.math.BigDecimal;

/**
 * テストで使うサンプルユーザデータを含む
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
```

```
*/
public interface ExampleUserData
{
    /*
     * テストデータ
     */

    long USER_ALRUBINGER_ID = 2L;

    String USER_ALRUBINGER_NAME = "Andrew Lee Rubinger";

    long ACCOUNT_ALRUBINGER_ID = 2L;

    BigDecimal INITIAL_ACCOUNT_BALANCE_ALR = new BigDecimal(500);
}
```


付録 I

インターセプタ： テレビチャンネルサービスの例

I.1 説明

セキュリティとトランザクションで見てきたように、中心的な関心事に含まれないビジネスロジックが存在することが少なくありません。対照的に、おそらくアプリケーションにはさまざまなモジュールや EJB に対して横断的に使う必要があるルールがあり、このようなロジックを実装に含めると関心事が混ざり合い、時間がたつにつれ保守が困難なシステムになっていきます。

EJB インターセプタモデルは、サーブレットフィルタや AOP アスペクトと同様にアプリケーション開発者が受信した要求にロジックを適用するためのメカニズムを提供します。`@Interceptors` アノテーションは、所定の呼び出しに特定のインターセプタを適用すべきことを EJB コンテナに知らせるためのマッピングマーカースとしての機能を果たします。

本章の例は、クライアントがチャンネル番号を要求したときにチャンネルストリームを返すテレビサーバを表します。ここでは、管理者がチャンネル 2 を閉鎖できる独自のセキュリティポリシーを適用します。アクセスが許可されていないときにクライアントがチャンネル 2 を要求すると、その呼び出しは「`Channel2ClosedException`」を返します。このロジックのすべては、要求されたチャンネルを返す主要コードとは分離されているので、他のモジュールに適用することもできれば、既存のアプリケーションを全く変更することなくシステムから取り除くこともできます。

I.2 オンライン上の関連情報

ウィキ記事：<http://community.jboss.org/docs/DOC-15574>

ソースの場所：<http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples/tree/master/ch18-interceptors/>

I.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

I.3.1 実装リソース

I.3.1.1 AuditedInvocation.java

```
package org.jboss.ejb3.examples.ch18.tuner;
```

```
import java.security.Principal;

import javax.interceptor.InvocationContext;

/**
 * 呼び出しの際の検査可能なプロパティをカプセル化するデータオブジェクト
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public class AuditedInvocation
{

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 呼び出しコンテキスト
     */
    private final InvocationContext context;

    /**
     * 呼び出し側
     */
    private final Principal caller;

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * 新しいインスタンスを生成する
     */
    AuditedInvocation(final InvocationContext context, final Principal caller)
    {
        // 前提条件を確認
        assert context != null : "context must be specified";
        assert caller != null : "caller must be specified";

        // 設定
        this.context = context;
        this.caller = caller;
    }

    //-----||

```

```

// 機能メソッド -----||
//-----||

/**
 * @return context
 */
public InvocationContext getContext()
{
    return context;
}

/**
 * @return caller
 */
public Principal getCaller()
{
    return caller;
}
}

```

I.3.1.2 CachingAuditor.java

```

package org.jboss.ejb3.examples.ch18.tuner;

import java.security.Principal;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.logging.Logger;

import javax.annotation.Resource;
import javax.ejb.SessionContext;
import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;

/**
 * インターセプトしたすべての呼び出しのキャッシュをグローバルにアクセスできる
 * キャッシュに保持するアスペクト
 *
 * テストと学習の目的のためにわかりやすくしているが、
 * これは実際の検査メカニズムとしては非常にお粗末な例である。運用環境では、
 * キャッシュのコピーオンライトの性質は時間とともに幾何学的に悪化し、
 * さらに {@link CachingAuditor#getInvocations()} の呼び出し側に
 * 可変ビュー ( {@link InvocationContext#setParameters(Object[])} ) を
 * 公開する。
 *
 */

```

```

* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
* @version $Revision: $
*/
public class CachingAuditor
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(CachingAuditor.class.getName());

    /**
     * キャッシュされた呼び出し：このメンバーはすべてのインターセプタインスタンス
     * で共有され、それらは Bean インスタンスにリンクされるので、スレッドセーフな
     * 実装でなければいけない。各 Bean インスタンスは一度に1つだけのスレッドで
     * 使われることが保証されているが、複数の Bean インスタンスが同時に実行される可能性がある。
     */
    private static final List<AuditedInvocation> invocations = new CopyOnWriteArrayList<AuditedInvocati
on>();

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 現在の EJB コンテキスト：EJB コンテナで注入するか、または単体テストで
     * 手動で設定
     */
    @Resource
    SessionContext beanContext;

    //-----||
    // 必要な実装 -----||
    //-----||

    /**
     * 後で取得できるように、インターセプトした呼び出しを検査可能なビューに
     * キャッシュ
     */
    @AroundInvoke
    public Object audit(final InvocationContext invocationContext) throws Exception
    {

```

```
// 前提条件を確認
assert invocationContext != null : "コンテキストが指定されていません";

// 呼び出し側を取得
Principal caller;
try
{
    caller = beanContext.getCallerPrincipal();
}
catch (final NullPointerException npe)
{
    caller = new Principal()
    {
        @Override
        public String getName()
        {
            return "Unauthenticated Caller";
        }
    };
}

// 新しいビューを作成
final AuditedInvocation audit = new AuditedInvocation(invocationContext, caller);

// キャッシュに呼び出しを追加
invocations.add(audit);

// 呼び出しを実行する。呼び出しの前後（周辺）でインターセプトした場所を記録
try
{
    // ロギング
    log.info("インターセプト: " + invocationContext);

    // 戻す
    return invocationContext.proceed();
}
finally
{
    // ロギング
    log.info("完了: " + invocationContext);
}

}

//-----||
```

```

// 機能メソッド -----||
//-----||

/**
 * インターセプタがキャッシュした {@link InvocationContext}
 * の読み取り専用ビューを返す
 */
public static List<AuditedInvocation> getInvocations()
{
    // 公開するためにコピーする
    return Collections.unmodifiableList(invocations);
}

/**
 * 呼び出しを削除するためのテスト専用のフック
 */
static void clearInTesting()
{
    invocations.clear();
}
}

```

I.3.1.3 Channel2AccessPolicy.java

```

package org.jboss.ejb3.examples.ch18.tuner;

/**
 * チャンネル 2 が現在アクセス可能かどうかを決める
 * 許可方針を定める
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public class Channel2AccessPolicy
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * チャンネル 2 を表示すべきかどうかを示すフラグ
     */
    private static boolean channel2Permitted = false;

    //-----||
    // コンストラクタ -----||

```

```

//-----||

/**
 * インスタンス化しない
 */
private Channel2AccessPolicy()
{
    throw new UnsupportedOperationException("No instances permitted");
}

//-----||
// 機能メソッド -----||
//-----||

/**
 * チャンネル 2 を視聴する要求が許可されているかどうかを返す
 */
public static boolean isChannel2Permitted()
{
    return channel2Permitted;
}

/**
 * チャンネル 2 を視聴する要求が許可されているかどうかを返す
 */
public static void setChannel2Permitted(final boolean channel2Permitted)
{
    Channel2AccessPolicy.channel2Permitted = channel2Permitted;
}
}

```

I.3.1.4 Channel2ClosedException.java

```

package org.jboss.ejb3.examples.ch18.tuner;

import javax.ejb.ApplicationException;

/**
 * チャンネル 2 が現在視聴できないことを示す
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@ApplicationException
// この例外型はラップせずにそのままの状態クライアントに返すべきであることを示す
public class Channel2ClosedException extends Exception
{

```

```
//-----||
// クラスメンバー -----||
//-----||

/**
 * serialVersionUID
 */
private static final long serialVersionUID = 1L;

/**
 * 唯一のインスタンス。この型は状態を持たない。
 */
public static final Channel2ClosedException INSTANCE;
static
{
    INSTANCE = new Channel2ClosedException();
}

/**
 * 受信したすべての例外に対するメッセージ
 */
private static final String MSG = "Channel 2 is not currently available for viewing";

//-----||
// コンストラクタ -----||
//-----||

/**
 * 新しいインスタンスを生成する
 */
private Channel2ClosedException()
{
    super(MSG);
}
}
```

I.3.1.5 Channel2Restrictor.java

```
package org.jboss.ejb3.examples.ch18.tuner;

import java.lang.reflect.Method;
import java.util.logging.Logger;

import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;

/**
```

```

* ネットワークが放送を許可していない限り
* チャンネル 2 へのアクセスを制限するアスペクト
*
* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
* @version $Revision: $
*/
public class Channel2Restrictor
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ロガー
     */
    private static final Logger log = Logger.getLogger(Channel2Restrictor.class.getName());

    /**
     * チャンネルのコンテンツを要求するメソッドの名前
     */
    private static final String METHOD_NAME_GET_CHANNEL;
    static
    {
        METHOD_NAME_GET_CHANNEL = TunerLocalBusiness.class.getMethods()[0].getName();
    }

    //-----||
    // 必要な実装 -----||
    //-----||

    /**
     * 指定された要求を調べ、呼び出し側がチャンネル 2 のコンテンツを取得しようと
     * しているかどうかを判断する。取得しようとしており、チャンネル 2 が現在非公開の
     * 場合、代わりに {@link Channel2ClosedException} を発行して要求をブロックする。
     */
    @AroundInvoke
    public Object checkAccessibility(final InvocationContext context) throws Exception
    {
        // 前提条件を確認
        assert context != null : "コンテキストが指定されていません";

        // チャンネル 2 を要求しているかどうかを確認する
        if (isRequestForChannel2(context))
        {
            // チャンネル 2 が公開されているかを確認する

```

```
        if (!Channel2AccessPolicy.isChannel2Permitted())
        {
            // アクセスをブロックする
            throw Channel2ClosedException.INSTANCE;
        }
    }

    // それ以外の場合は続ける
    return context.proceed();
}

//-----||
// 機能メソッド -----||
//-----||

/**
 * 指定されたコンテキストがチャンネル2への要求を表しているかどうかを判断
 */
private static boolean isRequestForChannel2(final InvocationContext context)
{
    // 前提条件を確認
    assert context != null : "Context was not specified";

    // ターゲットメソッドを取得
    final Method targetMethod = context.getMethod();

    // 新しいチャンネルを要求している場合
    final String targetMethodName = targetMethod.getName();
    if (targetMethodName.equals(METHOD_NAME_GET_CHANNEL))
    {
        log.info("This is a request for channel content: " + context);
        // Get the requested channel
        final int channel = ((Integer) context.getParameters()[0]).intValue();
        if (channel == 2)
        {
            // チャンネル2を望んでいる
            return true;
        }
    }
}

// 戻す
return false;
}
}
```

I.3.1.6 TunerBean.java

```

package org.jboss.ejb3.examples.ch18.tuner;

import java.io.IOException;
import java.io.InputStream;
import java.util.logging.Logger;

import javax.ejb.Local;
import javax.ejb.Stateless;
import javax.interceptor.Interceptors;
import javax.interceptor.InvocationContext;

/**
 * クライアントに参照を返す簡単な EJB。
 * インターセプタの設定を表すのに使う。
 * ここでは EJB で以前に作成したすべての {@link InvocationContext}
 * を記憶するように {@link CachingAuditor} を設定している。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Stateless
// クラスレベルのインターセプタは、この EJB のすべてのメソッドへの要求に対して実行される
@Interceptors(CachingAuditor.class)
@Local(TunerLocalBusiness.class)
public class TunerBean implements TunerLocalBusiness
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(TunerBean.class.getName());

    //-----||
    // 必要な実装 -----||
    //-----||

    /**
     * {@inheritDoc}
     * @see org.jboss.ejb3.examples.ch18.tuner.TunerLocalBusiness#getChannel(int)
     */
    // このメソッドだけで実行される、メソッドレベルのインターセプタを宣言

```

```
@Interceptors(Channel2Restrictor.class)
@Override
public InputStream getChannel(final int channel) throws IllegalArgumentException
{
    // 使用するストリームを宣言する
    final InputStream stream;
    switch (channel)
    {
        // チャンネル1を希望
        case 1 :
            stream = new InputStream()
            {
                @Override
                public int read() throws IOException
                {
                    return 1;
                }
            };
            break;
        // チャンネル2を希望
        case 2 :
            stream = new InputStream()
            {
                @Override
                public int read() throws IOException
                {
                    return 2;
                }
            };
            break;

        // 不適切なチャンネルを要求
        default :
            throw new IllegalArgumentException("Not a valid channel: " + channel);
    }

    // 戻す
    log.info("Returning stream for Channel " + channel + ": " + stream);
    return stream;
}
}
```

I.3.1.7 TunerLocalBusiness.java

```
package org.jboss.ejb3.examples.ch18.tuner;
```

```

import java.io.InputStream;

/**
 * テレビストリームへアクセスするための
 * EJB のローカルビジネスインタフェース
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public interface TunerLocalBusiness
{
    //-----||
    // 規約 -----||
    //-----||

    /**
     * 指定されたテレビチャンネルの視聴可能なコンテンツを含む
     * ストリームを取得する。サポートするチャンネルは 1 と 2 である。
     *
     * @param channel
     * @return
     * @throws IllegalArgumentException チャンネルが有効でない場合
     */
    InputStream getChannel(int channel) throws IllegalArgumentException;
}

```

I.3.2 テストリソース

I.3.2.1 CachingInterceptorUnitTestCase.java

```

package org.jboss.ejb3.examples.ch18.tuner;

import java.security.Identity;
import java.security.Principal;
import java.util.Properties;
import java.util.logging.Logger;

import javax.ejb.EJBHome;
import javax.ejb.EJBLocalHome;
import javax.ejb.EJBLocalObject;
import javax.ejb.EJBObject;
import javax.ejb.SessionContext;
import javax.ejb.TimerService;
import javax.interceptor.InvocationContext;
import javax.transaction.UserTransaction;
import javax.xml.rpc.handler.MessageContext;

```

```
import junit.framework.TestCase;

import org.jboss.ejb3.examples.ch18.tuner.AuditedInvocation;
import org.jboss.ejb3.examples.ch18.tuner.CachingAuditor;
import org.jboss.ejb3.examples.ch18.tuner.TunerLocalBusiness;
import org.junit.Before;
import org.junit.Test;

/**
 * 完全なコンテナのコンテキスト外で
 * {@link CachingAuditor} インターセプタが予想どおりに
 * 機能していることを確認するテスト
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public class CachingInterceptorUnitTestCase
{
    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(CachingInterceptorUnitTestCase.class.getName());

    /**
     * モックユーザの名前
     */
    private static String NAME_PRINCIPAL = "Mock User";

    /**
     * 返すべきプリンシパル
     */
    private Principal PRINCIPAL = new Principal()
    {
        @Override
        public String getName()
        {
            return NAME_PRINCIPAL;
        }
    };
};
```

```

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * テストするインターセプタインスタンス
 */
private CachingAuditor interceptor;

//-----||
// ライフサイクル -----||
//-----||

/**
 * テストで使うインターセプタインスタンスを生成
 */
@Before
public void createInterceptor()
{
    interceptor = new CachingAuditor();
    // プリンシパルを返すだけのモックビューを EJBContext に手動で設定
    interceptor.beanContext = new SessionContext()
    {

        /**
         * getCallerPrincipal 以外のメソッドを呼び出した場合に発行する例外
         */
        private UnsupportedOperationException UNSUPPORTED
        = new UnsupportedOperationException("Not supported in mock implementation");

        @Override
        public void setRollbackOnly() throws IllegalStateException
        {
            throw UNSUPPORTED;
        }

        @Override
        public Object lookup(String arg0) throws IllegalArgumentException
        {
            throw UNSUPPORTED;
        }

        @Override
        public boolean isCallerInRole(String arg0)
        {
            throw UNSUPPORTED;
        }
    };
}

```

```
    }

    @Override
    @SuppressWarnings("deprecation")
    public boolean isCallerInRole(Identity arg0)
    {
        throw UNSUPPORTED;
    }

    @Override
    public UserTransaction getUserTransaction() throws IllegalStateException
    {
        throw UNSUPPORTED;
    }

    @Override
    public TimerService getTimerService() throws IllegalStateException
    {
        throw UNSUPPORTED;
    }

    @Override
    public boolean getRollbackOnly() throws IllegalStateException
    {
        throw UNSUPPORTED;
    }

    @Override
    public Properties getEnvironment()
    {
        throw UNSUPPORTED;
    }

    @Override
    public EJBLocalHome getEJBLocalHome()
    {
        throw UNSUPPORTED;
    }

    @Override
    public EJBHome getEJBHome()
    {
        throw UNSUPPORTED;
    }

    @Override
```

```
public Principal getCallerPrincipal()
{
    return PRINCIPAL;
}

@Override
@SuppressWarnings("deprecation")
public Identity getCallerIdentity()
{
    throw UNSUPPORTED;
}

@Override
public <T> T getBusinessObject(Class<T> businessInterface) throws IllegalStateException
{
    throw UNSUPPORTED;
}

@Override
public EJBLocalObject getEJBLocalObject() throws IllegalStateException
{
    throw UNSUPPORTED;
}

@Override
public EJBObject getEJBObject() throws IllegalStateException
{
    throw UNSUPPORTED;
}

@Override
public Class<?> getInvokedBusinessInterface() throws IllegalStateException
{
    throw UNSUPPORTED;
}

@Override
public MessageContext getMessageContext() throws IllegalStateException
{
    throw UNSUPPORTED;
}

@Override
public boolean isCancelled() throws IllegalStateException
{
    throw UNSUPPORTED;
}
```

```

    }
};
}

//-----||
// テスト -----||
//-----||

/**
 * インターセプタを通過したコンテキストがキャッシュされることを確認
 */
@Test
public void testCache() throws Exception
{
    // キャッシュが最初は空であることを確認
    TestCase.assertEquals("Cache should start empty", 0, CachingAuditor.getInvocations().size());

    // 呼び出し
    final InvocationContext invocation
    = new MockInvocationContext(TunerLocalBusiness.class.getMethods()[0], new Object[] {1});
    interceptor.audit(invocation);

    // 呼び出しが適切にキャッシュされたことを確認
    TestCase.assertEquals("Cache should have the first invocation", 1,
        CachingAuditor.getInvocations().size());
    final AuditedInvocation audit = CachingAuditor.getInvocations().get(0);
    TestCase.assertEquals("Invocation cached was not the one that was invoked", invocation,
        audit.getContext());
    TestCase.assertEquals("Invocation did not store the caller as expected", PRINCIPAL,
        audit.getCaller());
}
}
}

```

I.3.2.2 Channel2RestrictorUnitTestCase.java

```

package org.jboss.ejb3.examples.ch18.tuner;

import java.lang.reflect.Method;
import java.util.logging.Logger;

import javax.interceptor.InvocationContext;

import junit.framework.TestCase;

import org.jboss.ejb3.examples.ch18.tuner.Channel2AccessPolicy;
import org.jboss.ejb3.examples.ch18.tuner.Channel2ClosedException;

```

```

import org.jboss.ejb3.examples.ch18.tuner.Channel2Restrictor;
import org.jboss.ejb3.examples.ch18.tuner.TunerLocalBusiness;
import org.junit.Before;
import org.junit.Test;

/**
 * 完全なコンテナのコンテキスト外で
 * {@link Channel2Restrictor} インターセプタが予想どおりに
 * 機能していることを確認するテスト
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
public class Channel2RestrictorUnitTestCase
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(Channel2RestrictorUnitTestCase.class.getName());

    /**
     * チャンネルコンテキストを取得するメソッド
     */
    private static final Method METHOD_GET_CHANNEL = TunerLocalBusiness.class.getMethods()[0];

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * テストするインターセプタインスタンス
     */
    private Channel2Restrictor interceptor;

    //-----||
    // ライフサイクル -----||
    //-----||

    /**
     * テストで使うインターセプタインスタンスを生成
     */

```

```

@Before
public void createInterceptor()
{
    interceptor = new Channel2Restrictor();
}

//-----||
// テスト -----||
//-----||

/**
 * チャンネル 2 へのアクセスが許可されていないときには、チャンネル 2 への要求がブロックされることを確
認
 */
@Test(expected = Channel2ClosedException.class)
public void requestsToChannel2Blocked() throws Exception
{
    // アクセスポリシーをブロックに設定
    Channel2AccessPolicy.setChannel2Permitted(false);

    // 呼び出し
    final InvocationContext invocation = new MockInvocationContext(METHOD_GET_CHANNEL, new Object[]
{2});
    interceptor.checkAccessibility(invocation);
}

/**
 * チャンネル 2 へのアクセスが許可されているときには、チャンネル 2 への要求がブロックされないことを確
認
 */
@Test
public void requestsToChannel2NotBlocked() throws Exception
{
    // アクセスポリシーを許可に設定
    Channel2AccessPolicy.setChannel2Permitted(true);

    // 呼び出し
    final InvocationContext invocation = new MockInvocationContext(METHOD_GET_CHANNEL, new Object[]
{2});
    try
    {
        interceptor.checkAccessibility(invocation);
    }
    catch (final Channel2ClosedException e)
    {
        TestCase.fail("Should not have been blocked with: " + e);
    }
}

```

```

    }
}

/**
 * チャンネル 2 へのアクセスが許可されていないときに、
 * チャンネル 1 への要求がブロックされないことを確認
 */
@Test
public void requestsToChannel1NeverBlocked() throws Exception
{
    // アクセスポリシーをブロックに設定
    Channel2AccessPolicy.setChannel2Permitted(false);

    // 呼び出し
    final InvocationContext invocation = new MockInvocationContext(METHOD_GET_CHANNEL, new Object[]
{1});
    interceptor.checkAccessibility(invocation);
}
}
}

```

1.3.2.3 InterceptorIntegrationTest.java

```

package org.jboss.ejb3.examples.ch18.tuner;

import java.io.IOException;
import java.io.InputStream;
import java.lang.reflect.UndeclaredThrowableException;
import java.util.logging.Logger;

import javax.ejb.EJB;
import javax.interceptor.Interceptors;
import javax.naming.NamingException;

import junit.framework.TestCase;

import org.jboss.arquillian.api.Deployment;
import org.jboss.arquillian.junit.Arquillian;
import org.jboss.ejb3.examples.ch18.tuner.CachingAuditor;
import org.jboss.ejb3.examples.ch18.tuner.Channel2AccessPolicy;
import org.jboss.ejb3.examples.ch18.tuner.Channel2ClosedException;
import org.jboss.ejb3.examples.ch18.tuner.Channel2Restrictor;
import org.jboss.ejb3.examples.ch18.tuner.TunerBean;
import org.jboss.ejb3.examples.ch18.tuner.TunerLocalBusiness;
import org.jboss.shrinkwrap.api.ShrinkWrap;
import org.jboss.shrinkwrap.api.spec.JavaArchive;
import org.junit.After;

```

```

import org.junit.Test;
import org.junit.runner.RunWith;

/**
 * {@link Interceptors} を宣言した EJB が呼び出し時に
 * 割り込まれることを確認する統合テスト
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@RunWith(Arquillian.class)
public class InterceptionIntegrationTest
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * ログガー
     */
    private static final Logger log = Logger.getLogger(InterceptionIntegrationTest.class.getName());

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * デプロイを表すアーカイブ
     */
    @Deployment
    public static JavaArchive createDeployment()
    {
        final JavaArchive deployment = ShrinkWrap.create("echo.jar", JavaArchive.class).
addClasses(TunerLocalBusiness.class, TunerBean.class, CachingAuditor.class, Channel2Restrictor.class);
        log.info(deployment.toString(true));
        return deployment;
    }

    /**
     * 呼び出しを行う Bean
     */
    @EJB
    private TunerLocalBusiness bean;

    //-----||

```

```

// ライフサイクル -----||
//-----||

/**
 * クリーンアップ
 */
@After
public void clearInvocationsAfterTest()
{
    // クリーンアップ
    CachingAuditor.clearInTesting();
}

//-----||
// テスト -----||
//-----||

/**
 * {@link CachingAuditor} を宣言した EJB での呼び出しにより、
 * 対象となるメソッドが割り込まれることを確認
 */
@Test
public void testCachingInterception() throws NamingException, IOException
{
    // 呼び出しがまだ割り込まれていないことを確認
    TestCase.assertEquals("No invocations should have yet been intercepted", 0, CachingAuditor.
getInvocations().size());

    // 呼び出し
    final int channel = 1;
    final InputStream content = bean.getChannel(channel);

    // 応答が予想どおりであることを確認
    TestCase.assertEquals("Did not obtain expected response", channel, content.read());

    // 呼び出しが割り込まれたことを確認
    TestCase.assertEquals("The invocation should have been intercepted", 1, CachingAuditor.
getInvocations().size());
}

/**
 * チャンネル 2 が制限されているときには、チャンネル 2 を取得する要求が {@link Channel2ClosedException}
でブロックされることを確認
 */
@Test(expected = Channel2ClosedException.class)
public void testChannel2Restricted() throws Throwable

```

```
{
    // チャンネル2をブロックするようにポリシーを設定
    Channel2AccessPolicy.setChannel2Permitted(false);

    // 呼び出し
    try
    {
        bean.getChannel(2);
    }
    // 予想される結果
    catch (final UndeclaredThrowableException ute)
    {
        throw ute.getCause();
    }

    // ここに到達した場合は失敗
    TestCase.fail("Request should have been blocked");
}

/**
 * チャンネル2が制限されていないときには、チャンネル2を取得する要求が成功することを確認
 */
@Test
public void testChannel2Allowed() throws NamingException, IOException
{
    // チャンネル2を許可するようにポリシーを設定
    Channel2AccessPolicy.setChannel2Permitted(true);

    // 呼び出し
    final int channel = 2;
    final InputStream stream = bean.getChannel(channel);

    // テスト
    TestCase.assertEquals("Unexpected content obtained from channel " + channel, channel, stream.
read());
}
}
```

1.3.2.4 MockInvocationContext.java

```
package org.jboss.ejb3.examples.ch18.tuner;

import java.lang.reflect.Method;
import java.util.Map;

import javax.interceptor.InvocationContext;
```

```

/**
 * {@link InvocationContext#proceed()} (常に null を返す)、
 * {@link InvocationContext#getMethod()}、{@link InvocationContext#getParameters()}
 * 以外の要求されたすべてのメソッドに対して {@link UnsupportedOperationException} を発行する {@link
InvocationContext}
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
class MockInvocationContext implements InvocationContext
{

    //-----||
    // クラスメンバー -----||
    //-----||

    /**
     * 処理がサポートされていないことを示すために使うメッセージ
     */
    private static final String MSG_UNSUPPORTED = "Not supported in mock implementation";

    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * 呼び出されるメソッド
     */
    private final Method method;

    /**
     * 要求内のパラメータ
     */
    private final Object[] params;

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * 指定された必要な引数を使って新しいインスタンスを生成
     * @param method
     * @param params
     */
    MockInvocationContext(final Method method, final Object[] params)
    {

```

```
    assert method != null : "method must be specified";
    assert params != null : "params must be specified";
    this.method = method;
    this.params = params;
}

//-----||
// 必要な実装 -----||
//-----||

@Override
public Map<String, Object> getContextData()
{
    throw new UnsupportedOperationException(MSG_UNUPPORTED);
}

@Override
public Method getMethod()
{
    return method;
}

@Override
public Object[] getParameters()
{
    return params;
}

@Override
public Object getTarget()
{
    throw new UnsupportedOperationException(MSG_UNUPPORTED);
}

@Override
public Object proceed() throws Exception
{
    return null;
}

@Override
public void setParameters(final Object[] arg0)
{
    throw new UnsupportedOperationException(MSG_UNUPPORTED);
}
}
```

I.3.2.5 SecurityActions.java

```

package org.jboss.ejb3.examples.ch18.tuner;

import java.security.AccessController;
import java.security.PrivilegedAction;

/**
 * セキュリティ動作がこのパッケージ外に漏れないように保護
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
class SecurityActions
{

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * 外部からはインスタンス化できない
     */
    private SecurityActions()
    {

    }

    //-----||
    // ユーティリティメソッド -----||
    //-----||

    /**
     * スレッドコンテキストクラスローダを取得
     */
    static ClassLoader getThreadContextClassLoader()
    {
        return AccessController.doPrivileged(GetTcclAction.INSTANCE);
    }

    /**
     * 現在のスレッドコンテキストに指定された CL を設定
     *
     * @param cl
     * @throws IllegalArgumentException CL が null の場合
     */
    static void setThreadContextClassLoader(final ClassLoader cl) throws IllegalArgumentException

```

```

{
    if (cl == null)
    {
        throw new IllegalArgumentException("ClassLoader was null");
    }

    AccessController.doPrivileged(new PrivilegedAction<Void>()
    {
        public Void run()
        {
            Thread.currentThread().setContextClassLoader(cl);
            return null;
        }
    });
}

/**
 * 指定されたキーを持つシステムプロパティを取得
 *
 * @param key
 * @return
 * @throws IllegalArgumentException If the key is null
 */
static String getSystemProperty(final String key) throws IllegalArgumentException
{
    // 前提条件を確認
    if (key == null)
    {
        throw new IllegalArgumentException("key was null");
    }

    // システムプロパティを取得
    return AccessController.doPrivileged(new GetSystemPropertyAction(key));
}

//-----||
// 内部クラス -----||
//-----||

/**
 * TCCL を取得するための {@link PrivilegedAction} 動作
 */
private enum GetTcclAction implements PrivilegedAction<ClassLoader> {
    INSTANCE;

    @Override

```

```
public ClassLoader run()
{
    return Thread.currentThread().getContextClassLoader();
}
}

/**
 * システムプロパティにアクセスするための {@link PrivilegedAction}
 *
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
private static class GetSystemPropertyAction implements PrivilegedAction<String>
{
    /**
     * 取得するシステムプロパティの名前
     */
    private String sysPropName;

    /**
     * 指定されたシステムプロパティを名前を取得できる新しいインスタンスを生成
     * @param sysPropName
     */
    public GetSystemPropertyAction(final String sysPropName)
    {
        this.sysPropName = sysPropName;
    }

    /**
     * {@inheritDoc}
     * @see java.security.PrivilegedAction#run()
     */
    @Override
    public String run()
    {
        return System.getProperty(sysPropName);
    }
}
}
```


付録 J

タイマサービス： クレジットカード処理の例

J.1 説明

メッセージ駆動型 Bean で見えてきたように、ビジネスタスクフローを開始する方法はクライアントの要求が唯一の手段ではありません。MDB のように受信メッセージをリスンすることに加え、時間的な基準に基づいてイベントを発行することもできます。EJB はタイマサービスでこれに対処します。

EJB 3.1 では、全面改良された自然言語構文を使い、タイマサービスに関する Bean プロバイダのビューが大幅に進化しています。ここでの例ではクレジットカード処理をモデル化し、毎正時にキューイングされたすべてのトランザクションを処理します。ジョブのスケジューリングは `javax.ejb.TimerService` API を使ってプログラムで行うか、`@Timeout` と `@Schedule` 使って宣言的に行います。

J.2 オンライン上の関連情報

ウィキ記事：<http://community.jboss.org/docs/DOC-15575>

ソースの場所：<http://github.com/jbossejb3/oreilly-ejb-6thedition-book-examples/tree/master/ch19-timer/>

J.3 ソースリスト

以下は、実行可能な例で使うすべてのソースコードの完全なリストです。

J.3.1 実装リソース

J.3.1.1 CreditCardTransaction.java

```
package org.jboss.ejb3.examples.ch19.timer.api;

import java.math.BigDecimal;

/**
 * 1つのクレジットカード取引を表す値オブジェクト
 * 不変
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
```

```
* @version $Revision: $
*/
public class CreditCardTransaction
{
    //-----||
    // インスタンスメンバー -----||
    //-----||

    /**
     * カード番号
     */
    private final String cardNumber;

    /**
     * 課金する金額
     */
    private final BigDecimal amount;

    //-----||
    // コンストラクタ -----||
    //-----||

    /**
     * 指定されたクレジットカード番号と金額で新しいインスタンスを生成
     * @param cardNumber
     * @param amount
     * @throws IllegalArgumentException どちらかの引数が null の場合
     */
    public CreditCardTransaction(final String cardNumber, final BigDecimal amount) throws
    IllegalArgumentException
    {
        // 前提条件を調べる
        if (cardNumber == null || cardNumber.length() == 0)
        {
            throw new IllegalArgumentException("card number must be specified");
        }
        if (amount == null)
        {
            throw new IllegalArgumentException("amount must be specified");
        }

        // 設定
        this.amount = amount;
        this.cardNumber = cardNumber;
    }

    //-----||
}
```

```

// 必要な実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see java.lang.Object#toString()
 */
@Override
public String toString()
{
    return "CreditCardTransaction [amount=" + amount + ", cardNumber=" + cardNumber + "];"
}

//-----||
// 機能メソッド -----||
//-----||

/**
 * @return cardNumber
 */
public String getCardNumber()
{
    return cardNumber;
}

/**
 * @return amount
 */
public BigDecimal getAmount()
{
    return amount;
}
}

```

J.3.1.2 CreditCardTransactionProcessingLocalBusiness.java

```
package org.jboss.ejb3.examples.ch19.timer.api;
```

```
import java.util.Date;
import java.util.List;
```

```
import javax.ejb.ScheduleExpression;
import javax.ejb.Timer;
```

```
/**
 * 処理すべき一連の {@link CreditCardTransaction} の格納、処理の
 * スケジューリング、保留中のすべての取引の支払いを実行できる

```

```

* サービスの規約
*
* @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
* @version $Revision: $
*/
public interface CreditCardTransactionProcessingLocalBusiness
{
    //-----||
    // 規約 -----||
    //-----||

    /**
     * 処理保留中のすべての取引の不変のビューを返す
     * @return
     */
    List<CreditCardTransaction> getPendingTransactions();

    /**
     * 保留中のすべての {@link CreditCardTransaction} を処理し、
     * 完了したら保留中リストから削除する
     */
    void process();

    /**
     * 処理すべき指定された {@link CreditCardTransaction} を追加する
     * @param transaction
     * @throws IllegalArgumentException 取引が null の場合
     */
    void add(CreditCardTransaction transaction) throws IllegalArgumentException;

    /**
     * 提供された {@link ScheduleExpression} に従って、保留中の支払いを
     * 処理するための新しい {@link Timer} をスケジューリングする。
     * 次の処理がいつ開始されるかを表す {@link Date} を返す。
     * @param expression
     * @return
     * @throws IllegalArgumentException 式が null の場合
     */
    Date scheduleProcessing(ScheduleExpression expression) throws IllegalArgumentException;
}

```

J.3.1.3 CreditCardTransactionProcessingBean.java

```

package org.jboss.ejb3.examples.ch19.timer.impl;

import java.util.ArrayList;

```

```

import java.util.Collections;
import java.util.Date;
import java.util.List;
import java.util.logging.Logger;

import javax.annotation.Resource;
import javax.ejb.ConcurrencyManagement;
import javax.ejb.ConcurrencyManagementType;
import javax.ejb.Local;
import javax.ejb.Lock;
import javax.ejb.LockType;
import javax.ejb.Schedule;
import javax.ejb.ScheduleExpression;
import javax.ejb.SessionContext;
import javax.ejb.Singleton;
import javax.ejb.Timeout;
import javax.ejb.Timer;
import javax.ejb.TimerService;

import org.jboss.ejb3.examples.ch19.timer.api.CreditCardTransaction;
import org.jboss.ejb3.examples.ch19.timer.api.CreditCardTransactionProcessingLocalBusiness;

/**
 * 保留中の {@link CreditCardTransaction} を後で処理するために格納できる
 * サービスの実装。
 * これは {@link CreditCardTransactionProcessingLocalBusiness#process()} を
 * 呼び出すか、EJB タイマサービスを使って
 * 設定された任意の数のタイマで処理できる。
 * デプロイ時には、毎正時に実行されるようにデフォルトタイマが設定されている
 * ({@link CreditCardTransactionProcessingBean#processViaTimeout(Timer)}) の
 * {@link Schedule} で設定する)。
 * {@link CreditCardTransactionProcessingBean#scheduleProcessing(ScheduleExpression)}
 * メソッドは、{@link ScheduleExpression} が指定された場合のプログラムによる
 * タイマの作成方法を表す。
 *
 * @author <a href="mailto:andrew.rubinger@jboss.org">ALR</a>
 * @version $Revision: $
 */
@Singleton
@Local(CreditCardTransactionProcessingLocalBusiness.class)
@ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)
public class CreditCardTransactionProcessingBean implements CreditCardTransaction
ProcessingLocalBusiness
{

    //-----||

```

```

// クラスメンバー -----||
//-----||

/**
 * ログガー
 */
private static final Logger log = Logger.getLogger(CreditCardTransactionProcessingBean.class.
getName());

/**
 * タイマ式で「すべて」を表すワイルドカード
 */
private static final String EVERY = "*";

/**
 * 0 を表すタイマ値
 */
private static final String ZERO = "0";

//-----||
// インスタンスメンバー -----||
//-----||

/**
 * EJB コンテナへのフックとなる {@link SessionContext}。
 * ここから {@link SessionContext#getTimerService()} を使って {@link TimerService} を取得。
 */
@Resource
private SessionContext context;

/**
 * {@link TimerService} を直接注入することもできる
 */
@Resource
@SuppressWarnings("unused")
// 単なる例として
private TimerService timerService;

/**
 * 保留中のすべての取引の {@link List}
 * この EJB の同時実行性ポリシーで保護される
 */
private final List<CreditCardTransaction> pendingTransactions = new ArrayList<CreditCardTransacti
on>();

//-----||

```

```

// 機能メソッド -----||
//-----||

@Timeout
// このメソッドを、プログラマ的に作成したタイマに対する EJB タイムアウトメソッドにする
// @Schedule を使ってタイマを作成している場合には、@Timer は不要
@Schedule(dayOfMonth = EVERY, month = EVERY, year = EVERY, second = ZERO, minute = ZERO, hour =
EVERY)
// このタイムアウトはデプロイ時に作成され、毎正時に起動される：宣言的な作成
@Lock(LockType.WRITE)
public void processViaTimeout(final Timer timer)
{
    // 単にビジネスメソッドに委譲する
    this.process();
}

//-----||
// 必要な実装 -----||
//-----||

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch19.timer.api.CreditCardTransactionProcessingLocalBusiness#add(org.
jboss.ejb3.examples.ch19.timer.api.CreditCardTransaction)
 */
@Override
@Lock(LockType.WRITE)
public void add(final CreditCardTransaction transaction) throws IllegalArgumentException
{
    // 前提条件を調べる
    if (transaction == null)
    {
        throw new IllegalArgumentException("transaction must be specified");
    }

    // 追加
    this.pendingTransactions.add(transaction);
    log.info("Added transaction pending to be processed: " + transaction);
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch19.timer.api.CreditCardTransactionProcessingLocalBusiness#getPend
ingTransactions()
 */
@Override

```

```
@Lock(LockType.READ)
public List<CreditCardTransaction> getPendingTransactions()
{
    // 呼び出し側が内部状態を変更できないように不変リストを返す
    return Collections.unmodifiableList(pendingTransactions);
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch19.timer.api.CreditCardTransactionProcessingLocalBusiness#process()
 */
@Override
@Lock(LockType.WRITE)
public void process()
{
    // 保留中のすべての取引を処理
    for (final CreditCardTransaction transaction : pendingTransactions)
    {
        // 疑似的な処理。EJB 書籍の例で
        // 本当に課金することはない。
        log.info("Processed transaction: " + transaction);
    }

    // すべてを課金したので、保留中の支払いを削除
    pendingTransactions.clear();
}

/**
 * {@inheritDoc}
 * @see org.jboss.ejb3.examples.ch19.timer.api.CreditCardTransactionProcessingLocalBusiness#scheduleProcessing(javax.ejb.ScheduleExpression)
 */
@Override
public Date scheduleProcessing(final ScheduleExpression expression) throws IllegalArgumentException
{
    // 前提条件を調べる
    if (expression == null)
    {
        throw new IllegalArgumentException(" タイマ式を指定しなくてはなりません ");
    }

    // SessionContext から TimerService を使って、指定された式から
    // 新しいTimer をプログラムの作成
    final TimerService timerService = context.getTimerService();
    final Timer timer = timerService.createCalendarTimer(expression);
}
```

```
    final Date next = timer.getNextTimeout();
    log.info("Created " + timer + " to process transactions; next fire is at: "
+ timer.getNextTimeout());
    return next;
}
}
```

