js-ctypesとXPConnectの違いを理解する

JavaScriptからネイティブなコンポーネントを呼び出すための新しい仕組みについて解説します。

js-ctypes、はFirefox 3.6（Gecko 1.9.2）から導入された、JavaScriptからバイナリ形式のコンポーネントの機能を呼び出すための新しい仕組みです。Firefox 3.6では非常に限定的な機能しかありませんでしたが、Firefox 4以降では配列や構造体などが利用できるようになり、十分に実用に耐える物となっています。これはXPConnectといったい何が違うのでしょうか。それを語る前に、まずはXPCOMとXPConnectがなんなのかを改めて解説したいと思います。

XPCOM

Firefoxの祖先にあたるNetscape NavigatorというWebブラウザは、Windows、Unix系OS、Mac OS向けのそれぞれのバージョンが別々に開発されていました。この開発手法には、各プラットフォームごとに最適なアプリケーションを提供できるというメリットがありますが、他方で、ほとんど同じ振る舞いをするソフトウェアを3つも作らなければならず、単純計算で開発の工数が3倍になってしまうというデメリットがあります。

これに対して当時のMozilla（そしてNetscape）が選んだのは、XPCOM（Cross Platform Component Object Model）というプラットフォーム非依存のコンポーネント呼び出しのための仕組みを設けることでした。

例えばファイルの読み書き1つを取っても、パスの表記の仕方からファイルの開き方、閉じ方など、WindowsとUnix系OSではさまざまな点で様式ややり方が異なります。これについてXPCOMでは、「nsIFile」という1つのインタフェース（メソッドの名前、引数の取り方、戻り値の型）に沿った形でWindows用のファイル読み書き用XPCOMコンポーネント、Linux用のファイル読み書き用XPCOMコンポーネント、Mac OS X用のファイル読み書き用コンポーネントのそれぞれを開発しておきます。これによって、他のコンポーネントでは「nsIFileのインタフェースでファイルの操作を行う」というただ1種類のコードだけを書いておけばすべてのプラットフォームで同じ結果を得られることになります（**図60-1**）。

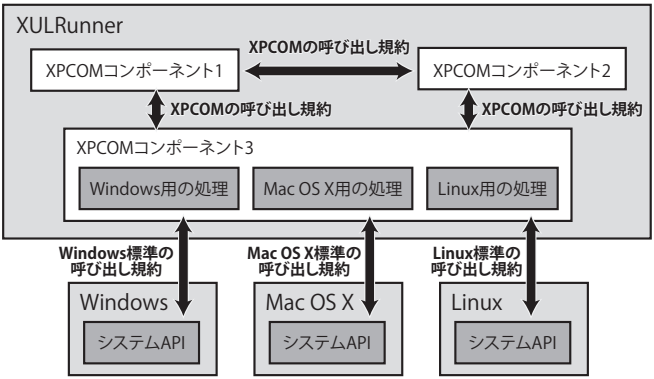

図60-1　XPCOM

このようなXPCOMコンポーネント群によって構成されたクロスプラットフォームなアプリケーション実行環境は、かつてはGRE（Gecko Runtime Environment）と呼ばれ、現在ではXULRunnerと呼ばれています。

XPConnect

C言語で書かれたXPCOMコンポーネントとJavaScriptのコードの間を橋渡しするのが、XPConnectという技術です。JavaScriptからXPCOMのコンポーネントを呼び出す際に使う`Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces.nsILocalFile)`といった定型文がありますが、これはXPConnectの最もわかりやすい例でしょう。またこれ以外にも、DOMノードのメソッドを呼び出すときやプロパティにアクセスするときなど、JavaScriptからネイティブな実装に触れることになるあらゆる場面でXPConnectは作用しています。

XPConnectはJavaScriptからネイティブなコンポーネントの機能にアクセスする機能だけでなく、その逆の機能も提供します。**[Hack #31]**ではJavaScriptでXPCOMのインタフェースを備えたコンポーネントを登録していますが、このとき登録されたコンポーネントには、Firefox自身に組み込まれたネイティブなコンポーネントがXPConnectを経由してアクセスしてくることになります（**図60-2**）。

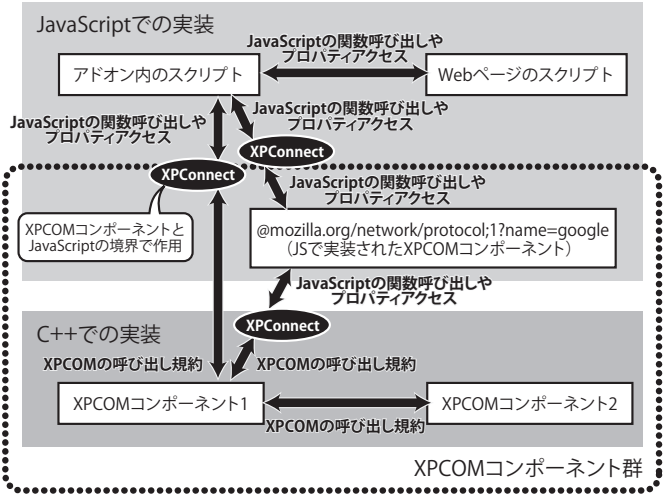

図60-2 XPConnect

XPCOMの欠点とjs-ctypes

このようにXPCOMとXPConnectはFirefoxを支える重要な技術なのですが、「C言語で書かれたコンポーネントの機能をJavaScriptから呼び出す仕組み」としてだけ見ると、新しいコンポーネントを利用できるようにするまでの障壁が非常に高いという欠点があります。

C言語で書かれたコンポーネントをXPCOMコンポーネントとして利用できるようにするためには、XPIDLというインタフェース定義用の言語でnsIFileのようなインタフェースを定義しなくてはなりませんし、XPCOMコンポーネントの様式に則った形でメソッドを定義する必要もあります。XULRunner SDK（Gecko SDK）を使用するとXPIDLのインタフェース定義から自動的にC言語用のヘッダファイルやテンプレート的なソースコードが出力されるので、ある程度の労力の削減はできますが、しかし肝心のメソッドの内容を書くためには、XPCOMコンポーネント開発のためのさまざまな定型文を使いこなさなくてはなりません。

他のXPCOMコンポーネントと密に連携して動作するようなコンポーネントを開発するのであれば、このような手間がかかるのは仕方がないでしょう。しかし、単に文字列を1つ受け取ってそれを加工した結果の文字列を返すというだけのコンポーネントであっても、XPCOMコンポーネントにするためには必ず上記のステップを踏む必要があります。また、XPCOMコンポーネントになっていないネイティブなコンポーネント、

例えばC/Migemo[†]を利用したいと思った場合にも、JavaScriptの間のブリッジとしてのXPCOMコンポーネントを開発しなければなりません。これはいかにも冗長です。

js-ctypesは、「C言語で書かれたコンポーネント[††]内で定義されている関数をJavaScriptから呼べるようにする」という目的に特化した仕組みです。先にあげたC/Migemoのようなライブラリに対する、JavaScriptとの間の汎用のブリッジと考えると理解しやすいでしょう。

js-ctypesの利用形態としては、大別して以下の2通りが考えられます。

- プラットフォームにインストールされた共有ライブラリをjs-ctypes経由で呼び出す
- 自分でC言語で共有ライブラリを開発してアドオンに同梱し、それをjs-ctypes経由で呼び出す

前者は、従来のアドオン開発における、プラットフォームの共有ライブラリを呼び出すためだけにブリッジとしてのXPCOMコンポーネントを開発していた場合の置き換えにあたります。この場合、そもそもバイナリ形式のコンポーネントを用意する必要すらなくなるため、開発の手間を大きく削減できるでしょう。

後者は、従来のアドオン開発における、プラットフォームごとのネイティブな実装をXPCOMコンポーネントとして開発していた場合の置き換えにあたります。この場合、ビルド環境を整える手間などは従来とあまり変わりませんが、XULRunner SDKが必要なかったり、XPCOMの様式に合わせてコンポーネントを開発する必要がなかったり（js-ctypes経由で呼び出したい関数のインタフェースさえ揃えておけばよい）と、従来の開発に比べて開発の手間はある程度少なくなるでしょう。

後者の場合のjs-ctypesの利用方法も、呼び出す共有ライブラリの場所が異なるだけで、手続き自体は前者の場合と変わりません。次節では前者の利用形態の具体的な例をあげて、js-ctypesの基本的な使い方を解説します。

---

† ローマ字表記の入力を受け取って、相当する日本語のかな漢字交じりの文字列にマッチする正規表現を生成するライブラリ。
参照：http://www.kaoriya.net/software/cmigemo

†† C++で開発されたライブラリでも、関数が`extern "C"`宣言を伴って（＝修飾名がC言語で記述されたライブラリと同様になって）いればjs-ctypes経由で呼び出すことができます。

js-ctypesの由来

js-ctypesは、Python 2.5以降で利用できる標準ライブラリの1つ「ctypes」を参考に開発されています†。ctypesも、Pythonとプラットフォームネイティブなコンポーネントの間の汎用のブリッジとして機能します。また、Rubyにも同様の技術としてRuby-FFIというライブラリがあります††。

MozillaのJavaScriptに対するXPCOMとXPConnectや、C/Rubyに対する拡張ライブラリのように、ネイティブなコンポーネントを利用するための機能をLL（Lightweight Language：軽量言語）の実装系が提供しているケースは珍しくありません。これらの仕組みは基本的には、ネイティブなコンポーネント側がスクリプト言語のインタフェースに合わせた入出力を備えるという形を取ります。言語間の差異やプラットフォームごとの事情を吸収するための手厚い配慮をライブラリ側が負担することによって、LLの流儀で簡単にコードを書けるように保ちながら、LLの標準的な機能だけではできないことを実現するというのが、XPConnectや拡張ライブラリの基本的な考え方といえるでしょう。

js-ctypesやPythonのctypes、RubyのFFIなどはそれとは逆のアプローチで、スクリプト言語の側がC言語に合わせた入出力を行います。例えばjs-ctypesであれば、JavaScriptではざっくりと「数値」という型で表現されている物をその都度16ビット整数や32ビット整数という風なC言語の型で表現し直したり、そのプラットフォームにおける共有ライブラリの標準的な置き場所を探したり、という風な諸々の面倒事は基本的にJavaScriptの側で負担しなければなりません。よって、これらの技術を使う場合にはC言語についてのある程度の知識が必須となります。

— Hiroshi Shimoda

† http://starkravingfinkle.org/blog/2007/09/hello-js-ctypes-goodbye-binary-components/

†† https://github.com/ffi/ffi