Vol.2010-ARC-189 No.15
Vol.2010-OS-114 No.15
2010/4/22

ブラウザ内 Java エンジン CABER の動作性能

新田善久[†1]　中山健[†1]　小川貴英[†1]

CABER (Connected Applet as a Back-End serveR for client-side JavaScript) は，JavaScript の処理分担のために同一 Web ブラウザ内の Java Applet をエンジンまたはサーバとして利用するフレームワークである．これに基づく CABER システムを，Java Applet と JavaScript との間の通信には LiveConnect を，jar ファイルの動的なダウンロードには JNLP を利用して実装した．CABER システムの実行性能は，JavaScript と Java の間での (1) 双方向のメソッド呼び出しの性能と，(2) 仮想マシン間のデータ変換の性能に大きく影響される．本報告は，これらの性能を 3 種類の Web ブラウザにおいて測定するとともに，AJAX を利用した場合とも比較して評価した．その結果，CABER は Web フレームワークとして十分な性能があることが示された．

Performance Evaluation of CABER, the In-Browser Java Engine for JavaScript

Yoshihisa Nitta,[†1] Ken Nakayama[†1] and Takahide Ogawa[†1]

CABER (Connected Applet as a Back-End serveR for client-side JavaScript) is a Web framework to use Java Applet as a back-end server/engine for client-side JavaScript in a Web browser. We implemented a CABER system based on the framework using LiveConnect for the communication between JavaScript and Java, and JNLP for the dynamic loading of jar files from the Web server. The performance of a CABER system is primarily affected by (1) the method call performace between JavaScript and Java, and (2) the data transfer performance between JavaScript virtual machine and Java virtual machine. We measured and compared these kinds of performance on three major Web browsers. Furthermore, we compared the CABER's performance with AJAX's. These results show that the CABER system has a sufficient performance as a Web framework.

1. はじめに

Web ブラウザ内で動作する代表的な言語は JavaScript と Java Applet であるが，AJAX 技術の普及により JavaScript 言語の重要性が高まっている．JavaScript には，(1) 他の Web サービス API の利用の容易さ，(2) HTML ドキュメントの内部構造である DOM の操作の容易さ，という利点があり，一方，Java Applet には (1)multi thread，(2) 豊富なクラスライブラリ，(3) 言語としての堅牢さ，という利点がある．Web アプリケーション上で両者を容易に協調動作できれば，双方の利点を享受できてその有用性は非常に高い．

我々は，Java Applet をブラウザ内で JavaScript のバックエンド・サーバとして動作させ，JavaScript の実行をサポートするためのフレームワークを CABER （Connected Applet as a Back-End serveR for client-side JavaScript）として提案し，システムを実装した[1)]．本稿では，CABER システムの主として通信に関する性能を測定し評価を行う．

2. CABER フレームワーク

2.1 Applet を JavaScript と連携させる際の要件

JavaScript と Java Applet を連携させる際には，(1) JavaScript が single thread である事，(2) Applet の起動時間，(3) 仮想マシン間でのデータ変換，(4) Java 側でのインスタンスの管理，という 4 点に注意が必要である．

2.1.1 JavaScript が single thread である事

ブラウザ内の JavaScript は single thread で動作しているため，JavaScript が Java Applet のメソッド呼び出しを行うと，そのメソッドが値を返すまで JavaScript の動作はブロックされる．これでは複数の要求を Java Applet に出すことができず，Java が multi thread である利点が利用できない．したがって，JavaScript からの Java Applet のメソッド呼び出しは，要求の登録だけを行い即座に終了する必要がある．Java メソッドの実行は別の Java thread が行い，計算結果は Java の thread が JavaScript のコールバック関数を呼び出すことによって JavaScript 側に返すべきである．

2.1.2 Applet の起動時間

Java Applet のコードを含んだ jar ファイルや class ファイルをダウンロードする時間が

†1 津田塾大学
Tsuda College

Vol.2010-ARC-189 No.15
Vol.2010-OS-114 No.15
2010/4/22

長いと，Web アプリケーションが起動するまでの待ち時間が長くなってしまい，快適な利用ができない．したがって，Applet は必要最小限のクラス定義だけで実行を開始し，後から必要になった Java クラスは Web サーバから動的にダウンロードすべきである．

2.1.3 仮想マシン間でのデータ変換

Java Applet 内に組み込まれた Java のメソッドは，JavaScript から引数を指定して呼び出される．また，メソッドの実行結果は Java から JavaScript に返される．その際に，データは JavaScript 仮想マシンと Java 仮想マシンの間を移動するので，オブジェクトの変換が必要となる．

後述する LiveConnect[2] はデータ変換機能を持っているが，JavaScript は強い型付けを持たない言語なので型付けに厳密な Java 言語に適用する際に問題になる場合がある．たとえば JavaScript 内では “1.0” という文字列と 1.0 という浮動小数点数は状況に応じて型変換されて適切に処理されるが，このデータを LiveConnect を使って Java に渡す場合はその時点の型で Java の型に変換されるので Java のメソッドのシグニチャと適合せず，メソッド呼び出しが失敗する場合がありえる．よって，JavaScript から Java メソッドへのデータの受け渡しは，データの変換で型が予想外になったとしてもメソッド呼び出し自体が失敗しないような方法で行うべきである．

2.1.4 Java 側でのインスタンスの管理

JavaScript に機能を提供する Java のクラスを動的にダウンロードして Java 側でインスタンスを生成したときに，データ変換に伴うオーバーヘッドを削減するために，JavaScript と Java の間で仮想マシンを跨いだオブジェクトの交換が必要以上に行われない仕組みが必要である．すなわち，Java のインスタンスは Java 側で `HashMap` などに登録し，識別子のみを JavaScript に渡すようにすべきである．

2.2 CABER フレームワークの提案

以下の方針に沿った Web アプリケーションの作成方法を CABER フレームワークとして提案した[1]**(図 1)**.

- JavaScript と同じ Web ブラウザ内で Java Applet をサーバとして動作させ，JavaScript にサービスを提供する．
- JavaScript から Java Applet への要求の送出はただちに終了し，要求に対する Java からの結果はコールバック関数によって JavaScript に非同期的に返される．
- JavaScript と Java Applet の間で渡されるデータは JSON[3] 形式で表現する．
- JavaScript からの要求を実行する Java のクラスを機能モジュールと定義する．機能モジュールは Web サーバから必要に応じて動的に Java の仮想マシンにダウンロードされる．機能モジュールは Java 側で管理し，識別子のみを JavaScript に渡す．

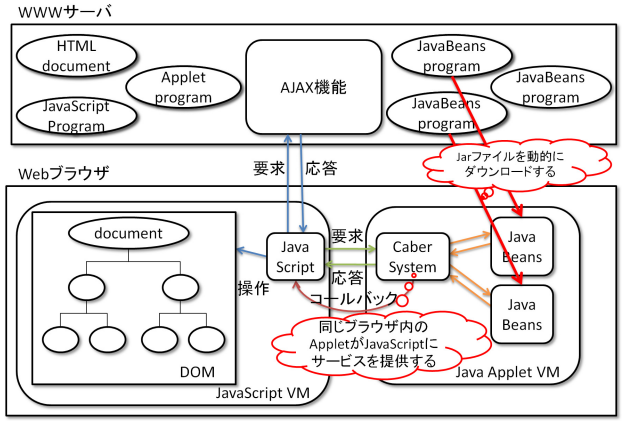

図 1 CABER フレームワーク
Fig. 1 CABER Framework

表 1 JDK6u10 以降で強化された機能
Table 1 Enhanced Features of JDK6u10

項目	説明
LiveConnect	JavaScript と Java Applet の通信．お互いに変数の値を参照したり，関数やメソッドをそれぞれ呼び出せる．
Java 仮想マシンの強化	Java 仮想マシンをブラウザとは別プロセスとして起動したり，実行時のヒープ量などを指定したりできる．
JNLP の改善	Java Network Launching Protocol の機能が強化され，Java 仮想マシンが jar ファイルを動的にロードすることが可能になった．
DOM へのアクセス	HTML 文書の Document Object Model 構造にアクセスして，内容を変更できる．

3. Java において強化された機能

Sun が提供する Java の実行環境は，2008 年秋にリリースされた JDK6u10 以降では **表 1** に示すような機能強化が図られている[4]．

3.1 LiveConnect とその threading model

LiveConnect は，JavaScript と Java が通信するための API である．LiveConnect を用

Vol.2010-ARC-189 No.15
Vol.2010-OS-114 No.15
2010/4/22

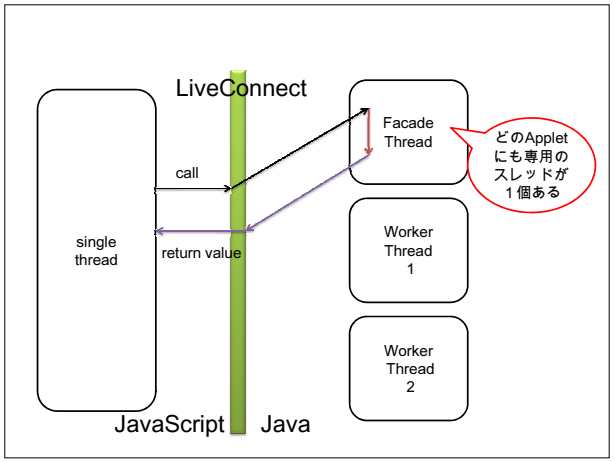

図 2 JavaScript から Java の呼び出し
Fig. 2 JavaScript to Java Call

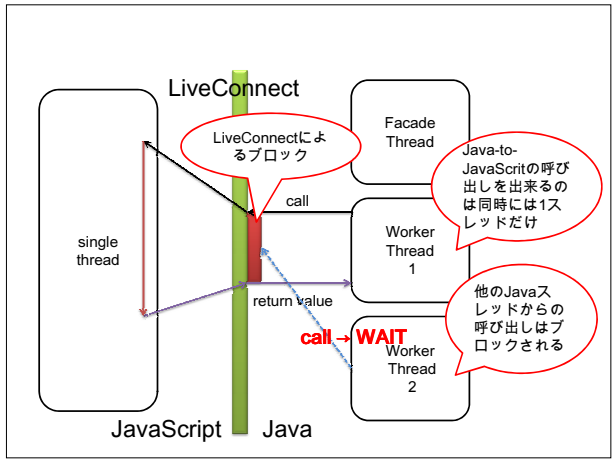

図 3 Java から JavaScript の呼び出し
Fig. 3 Java to JavaScript Call

いることで，JavaScript と Java Applet の間で互いに相手の関数やメソッドを呼び出したり，変数にアクセスしたりできる．Netscape のブラウザに最初に導入されたが，その後 Sun の JDK においてもサポートされた[5]．JDK 6u10 以降では Java Plug-In を利用して Java 仮想マシンを動作させているので，いろいろなブラウザで LiveConnect が利用できるようになった．

Single thread で動作する JavaScript と，multi thread で動作する Java Applet が，LiveConnect を用いてそれぞれお互いを呼び出すときのそれぞれの thread の動作は次の通りである[2]．

(1) どの Java Applet にも，JavaScript-to-Java の呼び出しを処理する専用の thread がそれぞれ 1 個ずつ存在する．デフォルトでは，JavaScript-to-Java の呼び出しはこの thread が処理する (**図 2**) (CABER システムではこの thread を Facade thread 呼ぶ)．

(2) Java の thread が Java-to-JavaScript の呼び出しをしようとした時，JavaScript が IDLE 状態でなければ thread は WAIT 状態へと移行し，JavaScript が IDLE 状態になるまで Java-to-JavaScrit 呼び出しは待たされる．**図 3** の例では，Java の thread 1 が Java-to-JavaScript の呼び出しを行っている最中に，他の thread 2 が Java-to-JavaScript の呼び出しを行おうとすると，thread 2 は待たされることになる．

(3) Java の thread 1 が Java-to-JavaScript の呼び出しを行っている処理の途中で JavaScript-to-Java の呼び出しが起きると，その JavaScript-to-Java の実行は thread 1 が行う (**図 4**)．

(4) JavaScript-to-Java の呼び出しを行ってその結果を待っている JavaScript は IDLE 状態なので，Java の別の thread 2 は Java-to-JavaScript の呼び出しを実行できる (図 4)．

3.2 JNLP (Java Network Launching Protocol) を用いた Applet の実行

Java Web Start は，Web サーバから Java アプリケーションをダウンロードして実行する仕組みである．JNLP は，「Java Web Start で用いる，Java アプリケーションの所在などを記述したファイル」または「Java Web Start の別称」である．Java Applet を JNLP 経由で起動した場合には，jar ファイルを Web サーバから動的にダウンロードして Java 仮想マシンに Java クラスを追加できる[6]．jar ファイルを動的にダウンロードする Applet は JNLP を用いて起動しなくてはならない．`applet` タグを用いて JNLP ファイルを指定した例を**図 5** に，指定した JNLP ファイルの内容を**図 6** に示す．`applet` タグで指定した `archive` プロパティと `code` プロパティは無視されて，`param` タグの `jnlp_href` の値とし

Vol.2010-ARC-189 No.15
Vol.2010-OS-114 No.15
2010/4/22

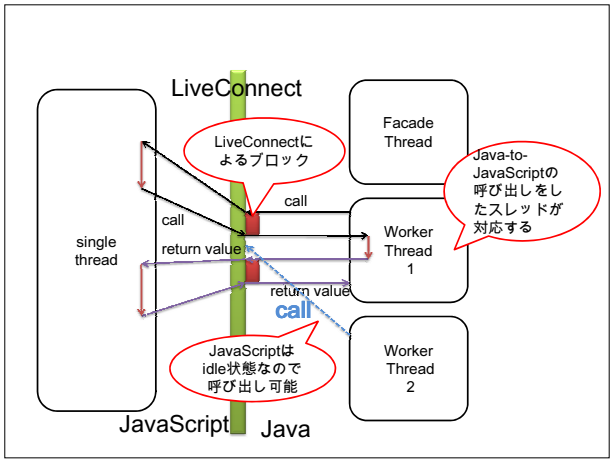

図 4 Java から JavaScript のラウンドロビン呼び出し
Fig. 4 Round-robin Java to JavaScript Call

```
<applet id="cabernet" archive="caber.jar"
  code="CaberFacade.class"
 width="10" height="10">
 <param name="jnlp_href" value="caber.jnlp" />
</applet>
```

図 5 `applet` タグでの JNLP ファイルの指定
Fig. 5 JNLP File Specification with `applet` Tag

て指定したファイル内の記述が用いられる．ただし，Applet の `width` と `height` に関しては `applet` タグの指定が JNLP ファイル内の記述よりも優先される．

4. CABER システム

4.1 CABER システムの実装方針

CABER フレームワークに基づく CABER システムを以下の方針に沿って実装した．

- JavaScript と Java Applet との間の通信には JDK6u10 以降で提供されている LiveConnect の機能を用いる．
- JavaScript から Java Applet への要求の送出は，LiveConnect を用いて JavaScript が Java Applet のメソッドを呼び出すことによって実現する．ただし，JavaScript から直接呼び出される Java Applet のメソッドは，JavaScript からの要求を queue に積む操作だけを行い，実際に要求を実行する役目は別の Java thread が担う．
- Java Applet から JavaScript への応答は，LiveConnect の機能を用いて Java が JavaScript の関数を呼び出すことによって行う．呼び出すコールバック関数は，要求送出時に JavaScript によって指定される．
- JavaScript から Java Applet へのデータの受け渡しは JSON 形式を用い，仮想マシン間のデータの変換は LiveConnect に任せる．
- Java Applet から JavaScript へのデータの受け渡しは Java オブジェクトの配列を用い，データの変換は LiveConnect に任せる．
- Java Applet が Web サーバから jar ファイルを動的にダウンロードする仕組みには JNLP の機能を用いる．
- JavaScript がメソッドを呼び出す Java クラスのインスタンスは Java の仮想マシン内で管理し，JavaScript にはその識別子だけを渡す．
- JavaScript が呼び出せる Java のメソッドは，インスタンスの生成時に reflection を用いて Java 側の `ConcurrentHashMap` オブジェクトに登録し，以降の呼び出しの高速化を図る．

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.5+" href="caber.jnlp">
 <information>
  <title>Caber Core Server</title>
  <vendor>Yoshihisa Nitta
   (nitta@tsuda.ac.jp)</vendor>
  <homepage href="http://nw.tsuda.ac.jp/" />
  <description>Caber Core Server</description>
 </information>
 <resources>
  <java version="1.6+"
  href="http://java.sun.com/products/autodl/j2se" />
  <jar href="caber.jar" main="true" />
 </resources>
 <applet-desc name="jp.ac.tsuda.caber.CaberFacade"
           main-class="jp.ac.tsuda.caber.CaberFacade"
                width="10" height="10">
 </applet-desc>
</jnlp>
```

図 6 JNLP ファイルの例
Fig. 6 An Example of JNLP File

Vol.2010-ARC-189 No.15
Vol.2010-OS-114 No.15
2010/4/22

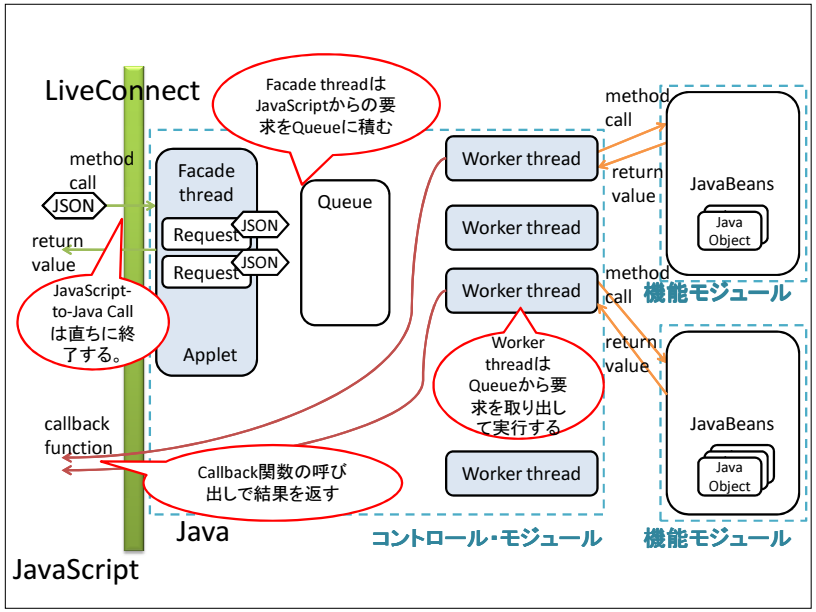

図 7 CABER システムの内部処理
Fig. 7 The Processing Flow of the CABER System

表 2 CABER システムの構成
Table 2 Components of CABER System

モジュール名称	種類	説明
コントロール・モジュール	Java Applet	JavaScript との通信やスケジューリングを行う
機能モジュール	JavaBeans	JavaScript に機能を提供するメソッドを備えている

4.2 CABER システムの構成

CABER システム（**図 7**）は 1 個のコントロール・モジュールと，複数の機能モジュールから構成される (**表 2**)．CABER システムは，最初はコントロール・モジュールだけが Web サーバからダウンロードされて Applet として動作を開始し，JavaScript からの要求に応じて必要な機能モジュールを動的にダウンロードする．

4.3 コントロール・モジュール

コントロール・モジュールは Java Applet を継承しており，その内部は 1 個の Facade thread と，1 個以上の Worker thread から構成される．

4.3.1 Facade thread

Facade thread は JavaScript からの要求を受け付けるための窓口である．JavaScript からの要求を受け付けて，その要求を `SynchronizedBlockingQueue` に積んで直ちに戻る．すなわち，JavaScript は Facade thread を使って Java に要求を渡すと，その要求が処理されるのを待つことなしに直ちに続きの処理に取り掛かれる．

JavaScript が Java Applet へ渡す要求は，(1) クラス名を指定して機能モジュールのインスタンスを生成する，(2) 機能モジュールのインスタンスを指定してメソッドを呼び出す (引数は JSON 形式の JavaScript オブジェクト)，(3) 機能モジュールのインスタンスを指定してメソッドを呼び出す (引数は機能モジュールのインスタンスを表す識別子)，の 3 種類である．(1) の要求の場合は，「jar ファイルの Web サーバ上における Path」，「クラス名」，「JavaScript のコールバック関数名」，「JavaScript のエラーハンドラ関数名」という 4 要素から構成される．また，(2) と (3) の要求は「機能モジュールのインスタンスの識別子」，「メソッド名」，「引数」，「JavaScript のコールバック関数名」，「JavaScript のエラーハンドラ関数名」という 5 要素から構成される．この場合の「引数」は (2) の場合は 1 個の JSON オブジェクトで，(3) の場合は 1 個の「機能モジュールのインスタンスの識別子」である．

4.3.2 Worker thread

Worker thread は，JavaScript から送られてきた要求を `SynchoronizedBlockingQueue` から取り出して，実際の処理を実行する．Worker thread は要求を queue から取り出すと，その中の「機能モジュールの識別子」と「メソッド名」を用いてメソッド・オブジェクトを求める（**図 8**）．引数が JSON の場合は `netscape.javascript.JSObject` として扱えるので，そのまま用いる．引数が機能モジュールの識別子の場合は，インスタンスに変換してからメソッド呼び出しに用いる．

メソッドの実行が終わったら，計算結果を引数として JavaScript のコールバック関数を呼び出して JavaScript に結果を伝える．該当するメソッドが存在しない場合や，メソッド呼び出しの途中でエラーが起きた場合は，Java の `Exception` オブジェクトを引数として JavaScript のエラーハンドラ関数を呼び出す．

複数の Worker thread が動作していて，それぞれの thread が要求を処理しているので，それぞれの要求に対するコールバック関数が呼ばれる順序は規定されない．そのため，逐次的に行いたい複数の要求を JavaScript が送る場合は，JavaScript 側のコールバック関数の中で次の要求を送出するなどの方法を用いて JavaScript 側が逐次処理を指定する必要が

Vol.2010-ARC-189 No.15
Vol.2010-OS-114 No.15
2010/4/22

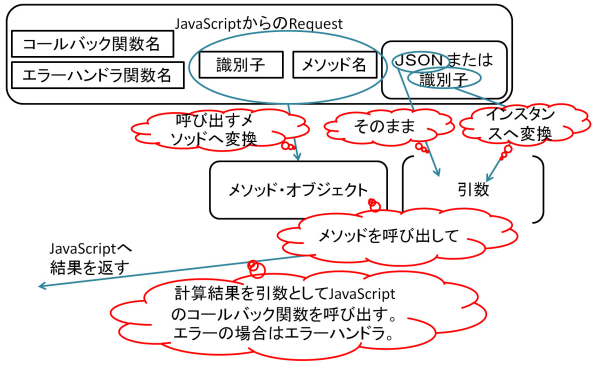

図 8 Worker thread のメソッド呼び出し
Fig. 8 Method Call by Worker Thread

ある．

4.4 機能モジュール

機能モジュールは，JavaScript に機能を提供するためのメソッドを備えた Java のクラスである．jar 形式で保存されていて，必要に応じて Web サーバからダウンロードされる．機能モジュールの実体は，`CaberBean` interface を implements した JavaBeans である．JavaBeans であるので次のような特徴を持つ．

- 引数の無いコンストラクタを持つ
- `package` 宣言を含む
- 内部フィールドに対する setter/getter メソッドを持つ．

`CaberBean` interface の定義は以下の通りである．

```
public interface CaberBean {
    public String getCertificate(String s);
    public Object putAuthInfo(CaberAuthInfo info);
}
```

機能モジュールの新しいインスタンスを生成した時には，コントロール・モジュールが認証のために `String getCertificate(String)` メソッドと `Object putAuthInfo(CaberAuthInfo)` メソッドをこの順に呼び出す．`getCertificate` メソッドでモジュールの作者やバージョンを取り出し，`putAuthInfo` メソッドでコントロール・モジュールの情報を与える．`CaberAuthInfo` クラスはコントロール・モジュールの情報を含むクラスである．

機能モジュールにコントロール・モジュールの情報を与えるのは，機能モジュールが Applet クラスのメソッドを利用しなければならない場合があるからである．たとえば，機能モジュールが `java.awt.Graphics` クラスを使って画像データを作り出すためには，たとえ Applet の画面に画像を描画をしないとしても `java.applet.Applet.getGraphics()` を用いて `java.awt.Graphics` クラスのインスタンスを取得しなければならない．

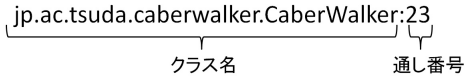

図 9 機能モジュールのインスタンスを表す識別子
Fig. 9 Instance Identifier of Function Module

4.5 機能モジュールのインスタンスの管理

JavaScript からの要求によって新しい機能モジュールのインスタンスを生成すると，一意の識別子をキーとしてそのインスタンスを `ConcurrentHashMap` に登録し，JavaScript には識別子を返す．インスタンスを表す識別子は,「クラス名」と「生成順の通し番号」を':' でつないだ文字列である (**図 9**)．

JavaScript から呼び出せるインスタンスメソッドは `java.lang.Class.newInstance()` 以外では，次のシグニチャを持つ 2 種類である．

`public Object[] メソッド名 (netscape.javascript.JSObject)`
`public Object[] メソッド名 (jp.ac.tsuda.caber.CaberBean)`

新しいインスタンスを生成した際に，Java の reflection 機能を用いてこれらのシグニチャを持つメソッドを探し，識別子とメソッド名をキーとしてメソッド・オブジェクトを `ConcurrentHashMap` に登録しておく．

4.6 CABER システムの Worker thread の同期

CABER システムでは，JavaScript から Java へ要求を送出するために JavaScript-to-Java の呼び出しを行い，要求を queue に積む．その後，Java の別 thread が要求を処理し，その結果を返すために Java-to-JavaScript の呼び出しで JavaScript のコールバック・ハンドラを実行する．この結果を返すための Java-to-JavaScript の呼び出しは `synchronized` で行っているので，JavaScript のコールバック・ハンドラの動作中は他の thread が Java-to-JavaScript の呼び出しをすることはない (**図 10**)．

5. CABER システムの実行性能

CABER システムの基本的な性能を評価するため，いくつかの速度計測を行った．シス

Vol.2010-ARC-189 No.15
Vol.2010-OS-114 No.15
2010/4/22

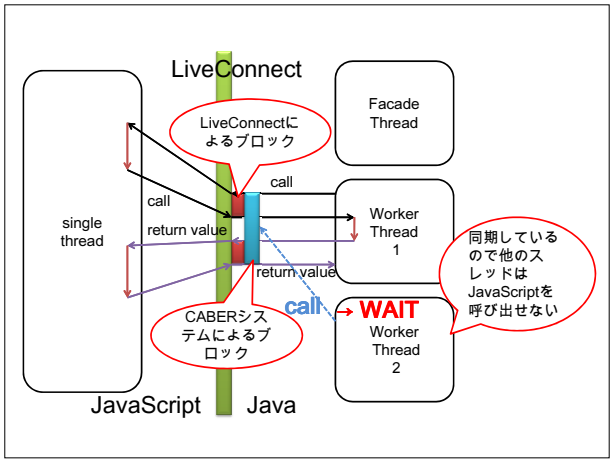

図 10 CABER における Java から JavaScript の呼び出し
Fig. 10 Java to JavaScript Call in CABER

テム全体の性能は，

- 単体での実行速度
  V_{JS}: JavaScript の実行速度
  V_J: Java の実行速度
  V_R: ブラウザのレンダリング速度
- 仮想マシン間の通信速度
  - JavaScript-to-Java-Call
    $C_{JS\to J}$: JavaScript からの Java メソッド呼び出し 1 回あたりの時間
    $T_{JS\to J}(x)$: JavaScript オブジェクトの Java オブジェクトへの変換速度 (データ量 x に依存)
  - Java-to-JavaScript-Call
    $C_{J\to JS}$: Java からの JavaScript 関数呼び出し 1 回あたりの時間
    $T_{J\to JS}(x)$: Java オブジェクトの JavaScript へのオブジェクトの変換速度 (データ量 x に依存)

に大きく影響を受けると考えられる．

実験環境として，Web サーバには，VMware ESXi4 をホスト OS とした計算機 (CPU: AMD Opteron 2376 2.3GHz × 2 個, メモリ 16GByte) 上で動作しているゲスト OS (CentOS5.3，仮想メモリ 2GByte) 上の Tomcat5.5.23 を用いた．ブラウザは，Windows XP SP3 の計算機 (CPU: Core2 6600 2.40GHz, メモリ 2GByte) 上で動作する

- FireFox 3.6
- Internet Explorer (IE) 8.0.6001.18702
- Chrome 4.1.249.1036(41514)

を用いた．Java Plug-In は JDK1.6.0_17 である．測定時間は全て ms 単位で表している．

表 3 実験 1
Table 3 Experiment 1

ブラウザ	(a) (ms)	(b) (ms)	(c) (ms)	(d) (回数)	(d) (ms)	(e) (回数)	(e) (ms)
FireFox	2854	2784	70	50	26	0	0
IE	2578	1759	819	45	38	1	45
Chrome	5551	5549	2	50	27	0	0

5.1 実　験　1

CABER の基本的な呼び出し速度である $C_{JS\to J} + C_{J\to JS}$ を計測した．この実験では，JavaScript から Java に送るデータも Java が JavaScript に返すデータもどちらも小さくなければいけない．JavaScript から Java へは小さな JSON 形式のデータ`{ value: 文字列 }`を，Java から結果が返されるのを持つことなく 10000 個連続して送り，最初の要求送出から全ての結果が返されるまでの時間を計測した (**表 3**)．Java 側では送られてきた文字列をそのまま返す．全要求を連続して送るので，Java 側に大量の要求が溜まる可能性がある．

JSON 形式のオブジェクトは JavaScript から Java に渡される際にコピーされることなく実体が渡される．そのため JavaScript と Java の仮想マシン間で競合が発生する可能性があるので，JavaScript が送出する JSON データは毎回新しく生成した．

計測したのは次の項目である．

(a) 最初の要求送出開始から全てのコールバック関数が呼ばれるまでの時間
(b) 要求送出開始から 10000 個送出し終わるまでの時間
(c) 要求送出終了から，最後のコールバック関数が呼ばれるまでの時間
(d) (a) の間に Java 側で起きた Copy GC の回数と時間
(e) (a) の間に Java 側で起きた Sweep GC の回数と時間

表 3 の (a) のデータを呼び出し回数の 10000 で割ると，CABER の 1 回の呼び出しにか

Vol.2010-ARC-189 No.15
Vol.2010-OS-114 No.15
2010/4/22

表 4 実験 2
Table 4 Experiment 2

文字列の長さ	FireFox		IE		Chrome	
(KByte)	CABER	AJAX	CABER	AJAX	CABER	AJAX
1	15.68	16.17	15.63	15.64	3.00	4.00
2	15.64	16.24	15.64	15.63	3.01	4.19
4	15.65	16.32	15.63	15.66	3.27	4.92
8	15.71	16.32	15.64	15.63	3.93	4.96
16	15.65	16.77	15.63	15.63	4.36	8.07
32	16.39	22.92	16.78	15.63	6.93	9.26
64	17.97	47.47	20.93	15.63	14.52	11.33
128	27.17	76.52	41.49	15.66	34.51	16.14
256	110.18	180.53	96.57	15.66	98.75	27.55
512	172.67	566.50	156.88	30.91	162.74	50.72
1024	256.15	1727.71	238.45	130.63	191.42	105.98
2048	320.79	-	309.22	569.69	346.69	212.30
4096	481.10	-	463.13	2415.7	533.24	396.26

(ms)

表 5 実験 3
Table 5 Experiment 3

文字列の長さ	FireFox	IE	Chrome
1	15.61	15.63	2.40
2	15.61	15.63	2.37
4	15.60	15.63	2.97
8	15.59	15.63	3.00
16	15.57	15.63	3.15
32	15.46	15.67	4.64
64	15.63	15.66	6.42
128	16.83	16.78	11.67
256	27.60	27.97	27.60
512	66.46	68.05	72.84
1024	113.55	119.83	124.26
2048	154.85	167.11	184.32
4096	254.81	265.86	314.16

(ms)

かる時間は，ブラウザ毎に異なるが 0.25〜0.56 ms であることがわかる．IE においては，Java 側で Copy GC だけではなく Sweep GC まで起きている．最後の要求を送出した後から全ての結果を受け取るまでの時間が長いことから，JavaScript からの大量の要求が Java の queue に溜まる現象が起きたと推測される．IE では JavaScript と Java のスケジューリングに関して他のブラウザと違いがあるといえる．Chrome では，CABER の呼び出しは他のブラウザと比較すると遅い．GoogleMap と CABER を用いた「道案内 Web アプリケーション CaberWalker」[1] を作成した我々の経験からは「FireFox と Chrome が実行速度が速く IE は遅い」という印象であったが，この実験では逆の結果が出た．

5.2 実　験　2

この実験で，Java から JavaScript にデータを返す場合の性能である $C_{JS\to J} + C_{J\to JS} + T_{J\to JS}(x)$ を測定し，さらに，CABER と AJAX の性能についても比較する．上述の CaberWalker では JavaScript から「実数値 6 個」と「長さ 11 の文字列」を Java に送り，Java で生成した画像を BASE64 エンコーディングされた文字列として JavaScript が受け取る．この状況での性能を評価するため，上記のデータを

CABERブラウザ内の CaberBean

AJAXWeb サーバ (Tomcat) 上の Servlet

それぞれに対して送り，JavaScript に返す文字列の長さを変えて時間を計測した．JavaScript から Java への要求送出は，前の要求に対する結果を Java が返すのを待って行う．要求送出はプログラムで複数 (10〜1000) 回繰り返し，最後の結果を受け取るまでの全経過時間を回数で割って 1 回あたりの処理時間を求めた (**表 4**)．CABER と AJAX のどちらにおいても Chrome の性能が高い．FireFox においてデータが大きくなると AJAX での速度低下が大きかったが，JavaScript のデバッグ環境が影響を与えている可能性があるので，実験結果からは省略した．Java の `String` を JavaScript に渡す場合は JavaScript の文字列に変換されるので，CABER のように同一計算機内でのデータ移動であってもコストが発生し，CABER にとっては有利ではない．また，本実験はネットワークの遅延が小さく，Web サーバ側の負荷が低い環境で行った．このような条件下でも，CABER は AJAX とほぼ同等の性能を持つといえる．サーバの負荷が高かったり，ネットワークの速度が遅かったり遅延が大きかったりする場合では，CABER が優位となるだろう．

5.3 実　験　3

この実験では JavaScript から Java へデータを送出する場合の性能である $C_{JS\to J} + C_{J\to JS} + T_{JS\to J}(x)$ を測定する．JavaScript から Java へ文字列を送り，その文字列の長さを Java が返すまで時間を測定した．この実験は，JavaScript から Java へ送るデータは大きく，Java から JavaScript に送るデータは小さいという条件を満たす．JavaScript から

Vol.2010-ARC-189 No.15
Vol.2010-OS-114 No.15
2010/4/22

Java への要求送出は，前の要求に対する結果を Java が返すのを待って行う．最初の要求送出から全ての結果を得るまでの時間を要求送出回数で割って 1 回の要求にかかる時間を求めた (**表 5**)．JavaScript の文字列オブジェクトは Java へ渡しても，データ変換は必要ないのでデータが大きくなっても性能の低下は大きくはない．

6. 結　　論

JavaScript の処理分担相手として同一ブラウザ内の Java Applet を使うというのが CABER フレームワークのアイディアである．CABER システムは通信面では AJAX と同等以上の性能があることが示された．

AJAX における使い勝手は，ネットワークの速度や遅延状況，サーバの速度や負荷状況に大きく依存するが，CABER では手元の計算機だけで処理できる．CABER と AJAX を比較すると，サーバ側への負荷の集中や通信のコストの面では CABER が有利であり，携帯端末の処理能力やバッテリーの持ち時間の面では AJAX が有利であるといえる．CABER と AJAX は対立するものではないので，双方のよい点を利用して Web アプリケーションを作成していくことが重要である．

今後の課題として，CABER のいろいろな機能モジュールを開発していく必要がある．

参 考 文 献

1) 新田, 中山, 小川: JavaScript のための Java Applet によるブラウザ内サーバ, 信学技報 IEICE Techincal Report IN2009-130(2010-2)pp.23–28, 2010.
2) Sun, LiveConnect Support in the Next Generation Java^{TM} Plug-In Technology Introduced in Java SE 6 update 10, http://java.sun.com/javase/6/webnotes/6u10/plugin2/liveconnect/index.html
3) D. Crockford, The application/json Media Type for JavaScript Object Notation (JSON), http://tools.ietf.org/html/rfc4627
4) Sun, Next-Generation Java^{TM} Plug-In Technology Introduced in Java SE 6 update 10, http://java.sun.com/javase/6/webnotes/6u10/plugin2/
5) Sun, Java-to-Javascript Communication, http://java.sun.com/j2se/1.5.0/docs/guide/plugin/developer_guide/java_js.html
6) Sun, Java Network Launch Protocol (JNLP) Support, http://java.sun.com/javase/6/webnotes/6u10/plugin2/jnlp/