Fermi User Guide and Reference

Version 1.0 / January 1, 2014

Igor Urisman

1 Introduction

1.1 Summary

Fermi is a Server-to-Client remote method invocation (RMI) framework. (In fact, the name "Fermi" loosely stands for Flip Ended Remote Method Invocation.) It enables remote invocation of client-side JavaScript code from server-side Java. It is not a Java wrapper around server-side JavaScript: the Java code runs under a Web Servlet container such as Tomcat, and the JavaScript runs in a Web browser, such as Firefox, fig. 1.

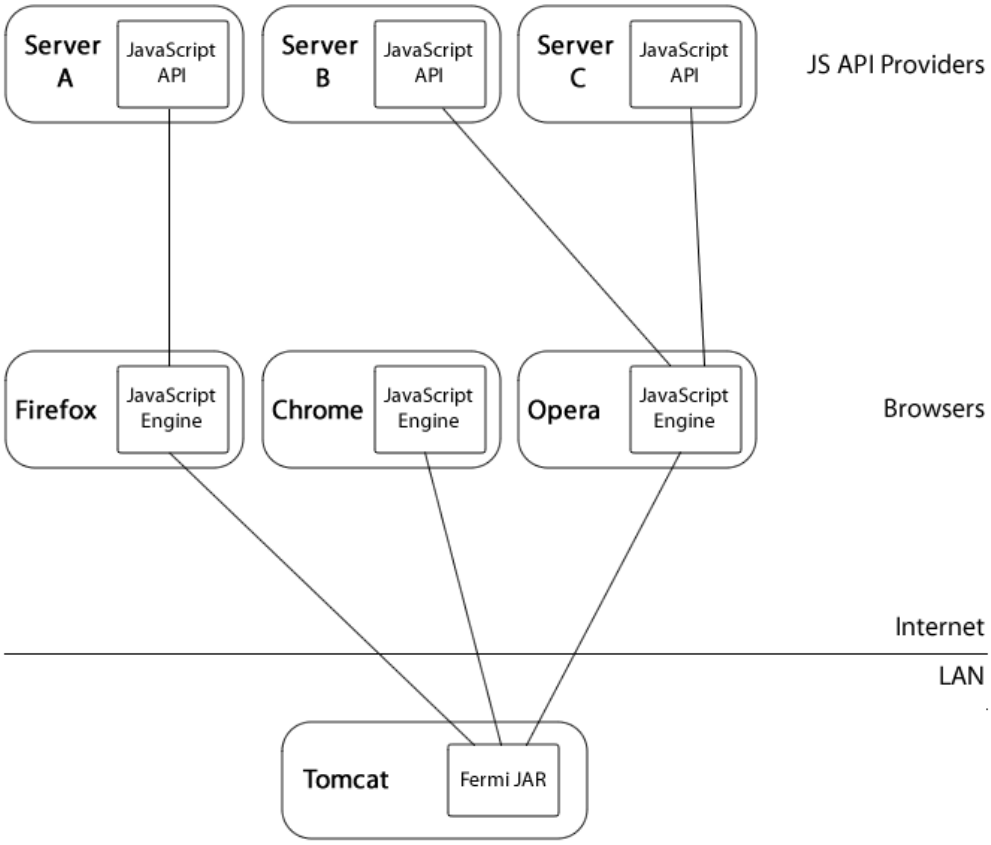

Fig. 1. There are two primary motivations behind this "flip-ended" RMI. One is to enable an elegant way to push data to the browser, as the Chrome browser in the middle. But perpahs even more importantly, it enables the server side Java to directly consume JavaScript APIs from third parties, such as the Google Maps or Google Charts APIs.

The Fermi framework is build on top of WebSockets API, which provides the low-level full-duplex message transport between the remote JavaScript client and the Java server. Just like WebSockets, it has a Java API and a JavaScript API, but unlike WebSockets, it exposes a higher-level abstraction, where the Java code can directly call JavaScript functions in the remote browser, pass arguments to them, and receive their return objects.

1.2 Example

Below is the minimal code that will get things off the ground.

Server:

```
FermiEndpoint ep = FermiContainer.getInstance().newEndpoint("/example");
FermiSession session = ep.getSessionBlocking(request.getSession());
session.invoke(new Method("myJsFunction"));
```

Client:

```
var session = new fermi.Session('/example');
function myJsFunction() {
	// do something
}
```

The code above hardly needs explaining. In the Java section, we create a FermiEndpoint, i.e. the communication mechanism between browsers and this server. On the second line we acquire the connection to the particular browser, represented by the current HTTP Session. And finally, we invoke a JavaScript method called *myJavaScriptFunction*.

For this to work, on the JavaScript side of this connection, the client must create the client's side of the session by instantiating the fermi.Session object, and define the function *myJavaScriptFunction*.

In actuality, things can get much more complex. Fermi supports

- Asynchronous or blocking session acquisition.
- Method parameters.
- Asynchronous or blocking method invocation.
- Return object inspection.
- Support for marshalling and un-marshalling of simple types.
- All asynchronous operations are exposed via the `java.util.concurrent.Future` mechanism.

2 User Guide

2.1 Distribution

The Fermi framework is published to client code via the server-side Java API and client-side JavaScript API. The distribution contains the following failes:

`fermi-1.0.jar`	The compiled Java classes. Must be present on the classpath at compilation and run time.
`fermi-1.0-src.jar`	Java sources.
`fermi-user-guide-1.0.pdf`	This document.
`javadoc/`	Javadoc. Point your browser to javadoc/index.html
`fermi-demo/`	The demo application. Copy to the webapps directory and it should come up on the /fermi-demo context.
`lib/`	3rd party Java libraries that Fermi depends on. Must be present on the classpath at run time.

2.2 Supported Platforms

The following Server and Client side platforms have been tested and are supported.

Client		**Server**
Operating System	**Browser**	Tomcat 8
OS X	Firefox	yes

2.3 Concepts

The Fermi framework enables Java to JavaScript remote method invocation (RMI). It is a thin layer of code on top of the Websocket API that adds functional structure to Websocket's low-level string oriented communication mechanism. The typical use of Websocket is to push data from server to client. The Fermi framework replaces that with calling a JavaScript method from Java and passing it an arbitrary argument or arguments, and even receiving the response back in the Java domain.

The following are the steps a Java application takes to take advantage of the Fermi framework.

2.3.1 Establish A Connection

Before the server is able to send RMI requests to the client, it has to obtain a FermiSession object that abstracts a connection to a particular HTTP client as identified by its HTTPSession. To do so, these steps are required:

- Server creates a FermiEndpoint at a particular URL, e.g. `/fermi`.
- Client connects to the FermiEndpoint.
- Server gets hold of the client connection request and creates the FermiSession object.

Server:

```
FermiEndpoint ep = FermiContainer.getInstance().newEndpoint("/fermi");
FermiSession session = ep.getSessionBlocking(request.getSession());
```

```
session.invoke(new Method("myJsFunction"));
```

Client:

```
var session = new fermi.Session('/fermi);
function myJsFunction() {
       // do something
}
```

2.3.2 Define A Callable Remote Function

In order to be callable by the Fermi framework, a JavaScript function must be named and referenceable from the document context. It is not possible to remotely call an anonymous or an inner function.

2.3.3 Returning Result From A Remote Function

The result object of a Fermi remote function is *not* its return object. The reason for this is that JavaScript is inherently asynchronous; a function may return well before the code path, triggered by it, completes. If a remote function wishes to return a result of its computation back to the server, it must instead use the fermi.Result object. It can be passed down the callback chain and held on to beyond the original function's stack life.

The fermi.Rsult object has two methods to communicate back to the server the outcome of its execution: `send()` and `throw()`. `Send()` indicates to the server a normal completion of the remote call and relays the result object, passed to it as the argument. `Throw()` relays an exception.

The fermi.Result object is passed by the framework to any remote function it invokes as the first argument. In the following example, the server gets the client's geo location information by having the client ping (by way of jQuery, incidentally) the popular free IP-to-geo location service at freegeoip.net. The JavaScript support for AJAX is inherently asynchronous, i.e. the outer function getGeoLocation() will not wait for the completion of the inner $.ajax() function.

JavaScript's way to manage asynchrony is via callback functions and closures. The second and third arguments to the $.ajax() function are functions themselves and they are the code that will be executed once the response data is received from freegeoip.org. Thanks to JavaScript's closure mechanism we can simply reference the result object inside the success anonymous function and set the result of the overall remote invocation there.

Server:

```
FermiEndpoint ep = FermiContainer.getInstance().newEndpoint("/fermi");                (1)
```

```
FermiSession session = ep.getSessionBlocking(request.getSession());                    (2)
Result result = session.invokeBlocking(new Method("getGeoLocation"));                  (3)
FreegeoipResponse resp = result.asObject(Freegeoip.class);                             (4)
```

Client:

```
var session = new fermi.Session('/fermi');                                             (1)
function getGeoLocation(result) {                                                      (2)
    $.ajax({                                                                           (3)
       url: "http://freegeoip.net/json/",                                              (4)
       success: function(data) {                                                       (5)
         result.send(eval(data));                                                      (6)
       },                                                                              (7)
       error:function() {                                                              (8)
          result.throw('Badness');                                                     (9)
       }                                                                              (10)
    });
}
```

Code comments:

- Client line 1 will fail if executed before server line 1 is because a Websocket client cannot connect to an endpoint that hasn't yet been created.
- Server line 3 executes the remote call. The invokeBlocking() method will block until the call completes, i.e. when the client calls the fermiResult.send() or fermi.Result.throw() method.
- Client lines 5 and 8 define the callback functions that will be called after the AJAX call returns. On success, we use the result object to return the data back to the server. On error, we want the server code to receive an exception.

2.3.4 Void Remote Function

If a remote function fails to call the send() or throw() method, the server will never know that the remote invocation is completed. The remote calls that fail to communicate their completion are known as runaways and ought to be avoided.

If a remote function calls the send() method without the argument, the server will see this remote call as of type "void."

2.3.5 Passing Parameters To A Remote Function

If a remote function takes parameters that must be sent from the server, they are set by the Method.setParameters() method. It takes an arbitrary number of arguments of type JsonEncoder, i.e. know how to marshal themselves into a JSON representation.

In the following example, the code listing 2.3.3 is expanded by making the URL to the Free geo IP service a parameter to the remote call.

Server:

```
FermiEndpoint ep = FermiContainer.getInstance().newEndpoint("/fermi");                (1)
FermiSession session = ep.getSessionBlocking(request.getSession());                   (2)
Method method = new Method("getGeoLocation");                                         (3)
method.setParameters(new JsonEncoderSimple("http://freegeoip.net/json/"));            (4)
Result result = session.invokeBlocking(method);                                       (5)
FreegeoipResponse resp = result.asObject(Freegeoip.class);                            (6)
```

Client:

```
var session = new fermi.Session('/fermi');                                            (1)
function getGeoLocation(result, serviceUrl) {                                        (2)
    $.ajax({                                                                          (3)
       url: serviceUrl,                                                               (4)
       success: function(data) {                                                      (5)
         result.send(eval(data));                                                     (6)
       },                                                                             (7)
       error:function() {                                                             (8)
          result.throw('Badness');                                                    (9)
       }                                                                             (10)
    });
}
```

Code comments:

- Server line 4 defines the parameter to the remote method. The parameter can be any class that implements the JsonEncoder interface, i.e. knows how to serialize itself into a JSON string. The JsonEocnderSimple class (comes with the framework) implements the JsonEncoder interface for a number of common data types, such as String.

- Client line 2 defines the remote function. This time it takes two parameters: the same result object, passed to any remote function by the framework, and the service URL parameter we bound on the server line 4.

2.3.6 Blocking And Non-blocking Operations

A remote call may take a while to complete. The Fermi framework supports both synchronous and asynchronous invocation. The server has the option of blocking for the entire duration of the remote call by using the FermiSession.invokeBlocking() method. This method will not return until the remote has invoked the fermi.Result.send() method.

Alternatively, the server has the option of calling the FermiSession.invoke() method, which returns right away, leaving the caller with a java.util.Future object:

```
Future<Result> future = session.invoke(method);
```

The future object returned by the invoke() call can be manipulated regularly ([3]).

3 Java API Reference

3.1 Package `org.fermi`

Public interface to the Fermi Framework for Java to JavaScript remote method invocation (RMI).

Since: 1.0

3.1.1 public abstract class FermiContainer

A static singleton class representing the entire Fermi container. Bootstrapped automatically by the Servlet container. Use the getInstance() method to obtain the singleton instance. For example, to create a new FermiEndpoint with the default configuration:

```
FermiEndpoint demoEndpoint = FermiContainer.getInstance().newEndpoint("/demo");
```

Since: 1.0

```
public static FermiContainer getInstance()
```

Get the singleton instance of this class.

```
public FermiEndpoint newEndpoint
    (java.lang.String path, FermiEndpointConfigurator config)
```

Create a new `FermiEndpoint` on a given path with a given configuration. All client connections to this path that came prior to this call are ignored. It is an error to call this twice for the same path

Parameters:

`path` - the path of the FermiEndpoint created, e.g. "/demo"

`config` - endpoint configurator.

Returns:

a `FermiEndpoint` object

```
public FermiEndpoint newEndpoint(java.lang.String path)
```

Create a new FermiEndpoint on a given path with a default configuration.

Parameters:

`path` - the path of the `FermiEndpoint` created, e.g. `"/demo"`

Returns:

a `FermiEndpoint` object

3.1.2 public interface FermiEndpoint

A communication endpoint that is opened by the Fermi container to receive connection requests from HTTP clients. The server must first create a FermiEndpoint before clients can connection to it. All client connections to a path for which no FermiEndpoint has been created will be ignored.

A one-to-one abstraction over the javax.websocket.server.Endpoint class. The underlying low-level WebSocket objects can be obtained via the #getServerEndpointConfig() method.

Since: 1.0

See Also: `FermiContainer`

```
public FermiEndpoint.State  getState()
```

State of this endpoint.

```
public java.lang.String  getPath()
```

This endpoint's (and that of the underlying WebSocket server Endpoint) URL path, as created with FermiContainer's newEndpoint(...) methods.

```
Public FermiEndpointConfigurator getConfigurator()
```

This endpoint's configurator, as created with FermiContainer's newEndpoint(...) methods.

```
public ServerEndpointConfig  getWsEndpointConfig()
```

This endpoint's underlying WebSocket javax.websocket.server.ServerEndpointConfig

```
public java.util.concurrent.Future<FermiSession>

    getSession(HttpSession httpSession)
```

Get the FermiSession for the remote client represented by this HttpSession. This method is asynchronous and returns right away, leaving the caller with the Future<FermiSession> object to manipulate for asynchronous acquisition of a FermiSession.

The first time this is called with a particular HttpSession, the underlying WebSocket session must be created. This requires that the client connect to this FermiEndpoint, which may be hard to synchronize perfectly, hence the asynchronous option.

More broadly:

If a client connection to a path comes before a FermiEndpoint is created on it, the connection is ignored.

If a client connection to a path comes after a FermiEndpoint is created, but before this call is made, it is accepted and parked until this call is made.

If this call comes before the client represented by this HttpSession connect to this FermiEndpoint, the Future object returned by it will remain in the not done state until the client connects.

On a subsequent call to this method, when the underlying FermiSession object for this HttpSession has already been created, this method will return instantaneously and the Future return object will contain reference to it right away. Typically, a caller will use this asynchronous method only on the first invocation, when the state of the connection is uncertain.

Parameters:

`httpSession` - represents a particular remote client.

Returns:

Future<FermiSession>

```
public FermiSession getSessionBlocking(HttpSession httpSession)
```

Get FermiSession for the remote client represented by this HttpSession. This method is synchronous; it blocks until the underlying `FermiSession` is available. It is equivalent to

getSession(httpSession).get()

Parameters:

`httpSession` - represents a particular remote client.

Returns:

`FermiSession`

`public void close()`

Close this FermiEndpoint and release underlying resources. All underlying FermiSessions are closed. Note that there's currently no way to close a WebSocket, so in actuality this method will leave the WebSocket open and accepting of new connection request.

3.1.3 public static enum FermiEndpoint.State

Possible state of a FermiEndpoint.

`ACTIVE`

Active endpoint.

`CLOSED`

Closed endpoint.

3.1.4 public interface FermiEndpointConfigurator

Interface must be implemented by a class whose instance can be passed to FermiContainer.newEndpoint(String, FermiEndpointConfigurator)

Since: 1.0

`public long invokeMaxWaitForResponse()`

The Future<Result>.get() object returned by FermiSession.invoke(org.fermi.method.Method) (and, consequently, the FermiSession.invokeBlocking(org.fermi.method.Method) method) will throw unchecked FermiException after waiting this many milliseconds.

`public long sessionMaxWaitForConnection()`

The Future<FermiSession>.get() object returned by FermiEndpoint.getSession(HttpSession) (and, consequently, the FermiEndpoint.getSessionBlocking(HttpSession) method) will throw unchecked FermiException after waiting this many milliseconds for the remote client to connect.

3.1.5 public interface FermiSession

A representation of the connection to a remote client that persists between requests. It is also bound at once to the `HttpSession` representing the same HTTP client and the WebSocket `Session`.

Since: 1.0

```
public java.lang.String getId()
```

Get this session's unique identifier. Same as `getWsSession().getId()`.

```
public FermiSession.State getState()
```

This session's current state. The only states visible to the caller are `FermiSession.State.ACTIVE` and `FermiSession.State.CLOSED`.

```
public void close()
```

Close this session. After this, calling any method on this session will result in the unchecked FermiException. Closes the underlying WebSocket session with the `javax.websocket.CloseReason.CloseCodes.NORMAL_CLOSURE` close reason.

```
public HttpSession getHttpSession()
```

Get the HTTP session representing the same remote client as this `FermiSession`.

```
public javax.websocket.Session getWsSession()
```

Get the underlying WebSocket Session.

```
public long getTimestampCreated()
```

Session creation timestamp. This is the time of `FermiSession` handshake, not the time when the underlying WebSocket session was created.

```
public java.util.concurrent.Future<Result> invoke(Method method)
```

Invoke a method on the remote client represented by this session. This method is asynchronous and returns right away, leaving the caller with the `Future<Result>` object to manipulate for asynchronous acquisition of a `Result` object.

Parameters:

method - an object of type Method representing a bound remote function, i.e. function's name and parameter values.

Returns:

an object of type java.util.concurrent.Future<Result>. This object has methods for asynchronous acquisition of the a Result object, which represent the result of the remote method invocation.

```
public Result invokeBlocking(Method method)
```

Invoke a method on the remote client represented by this session. This method will block until either of the following occurs:

Remote invocation completes and client returns the result back to the server.

Endpoint-wide maximum wait time (as defined by `FermiEndpointConfigurator.invokeMaxWaitForResponse()`) is reached.

```
public FermiEndpoint getEndpoint()
```

Get the Fermi endpoint on which this session was created.

Returns:

an FermiEndpoint object

3.1.6 public static enum FermiSession.State

Possible state of a `FermiSession`.

`CONNECTED_REMOTE_ONLY`

Session object (and the underlying WebSocket Session) have been created by a remote connection, but no local FermiEndpoint.getSession...(...) method has claimed it yet. These stick around indefinitely, until cleaned with the HTTP Session. Internal state not visible to callers.

`CONNECTED_LOCAL_ONLY`

Session object has been created by a local FermiEndpoint.getSession...(...) call, but the remote client represented by this HTTP Session has not made a connection to this

FermiEndpoint. These are cleaned up after FermiEndpointConfigurator.sessionMaxWaitForConnection() millis. Internal state not visible to client callers.

`TIMEDOUT`

A local session request has timed out. In other words, more than FermiEndpointConfigurator.sessionMaxWaitForConnection() milliseconds has passed since the beginning of a local FermiEndpoint.getSession...(...) call.

`CONNECTED`

Both sides of the rendezvous connected, but not yet returned to the caller. Internal state not visible to callers.

`ACTIVE`

CONNECTED and returned to the caller.

`CLOSED`

Can be de-linked by an asynchronous clean-up thread.

3.2 Package org.fermi.method

Public interface to the Fermi Framework for Java to JavaScript remote method invocation (RMI). This package contains classes that represent remote methods, their parameters and results. It also has a number of built in JSON decoders and encoders for marshalling common Java data types.

Since: 1.0

3.2.1 public interface JsonDecoder<T>

A Java class that wishes to be unmarshalled automatically from a JSON representation must implement this interface.

```
T decode(java.lang.String jsonString)
```

Synthesize an object of type T out of a JSON string. An implementing class must implement this method to enable the framework to unmarshall a JSON string sent by the remote client into the target object of type T.

3.2.2 public class JsonDecoderSimple

A collection of implementations of the JsonDecoder interface for a variety of commonly used Java types: String, String[], Long, Long[], Boolean, Boolean[]. These can be used for automatic instantiation of return objects of these types as in the following example.

```
Result result = session.invokeBlocking(method);  Boolean[] asBooleanArray =
result.asObject(JsonDecoderSimple.BooleanArray.class);
```

3.2.3 public interface JsonEncoder

A Java class that wishes to be marshalled automatically into a JSON representation must implement this interface.

java.lang.String encode()

Convert an instance of self to a JSON string. An implementing class must implement this method to enable the framework to marshall an instance of itself into a JSON string that can be instantiated by the remote as a JavaScript object.

3.2.4 public class JsonEncoderSimple implements JsonEncoder

A collection of implementations of the JsonEncoder interface applicable to a variety of commonly used Java types: String, String[], Long, Long[], Boolean, Boolean[]. These can be used for binding Java objects as method parameters in remote calls as in the following example.

```
Method method = new Method("remoteFunctionName");
method.setParameters(new JsonEncoderSimple(23));
Result result = session.invokeBlocking(method);
```

3.2.5 public class Method

Representation of a remote JavaScript function. The minimal configuration only requires the function's name. The name must be fully qualified so it can be resolved by the JavaScript compiler in the outermost (Document) scope. To pass arguments, use the setParameters(JsonEncoder...) method. It is OK to reuse an object of this type for multiple remote calls.

public Method(java.lang.String functionName)

Construct the remote method representation.

Parameters:

> functionName - name of the remote Java Script function. It must be visible from the Document scope.

public java.lang.String getFunctionName()

The name of the remote JavaScript function.

```
public void setParameters(JsonEncoder... params)
```

Set parameter values for this function. Subsequent invocation replaces currently set parameters.

```
public JsonEncoder[] getParameters()
```

Currently set parameter list.

3.2.6 public interface Result

Representation of the return value of a remote JavaScript function invocation. An object of this type is either returned by FermiSession.invokeBlocking(org.fermi.method.Method) or is obtained from the Future object returned by FermiSession.invoke(org.fermi.method.Method) method.

```
public java.lang.String asRawJson()
```

Raw JSON representation of the return object, as sent by the remote client.

```
public <T> T asObject(java.lang.Class<? extends JsonDecoder<T>> decoderClass)
```

Get representation of the return object marshalled into a Java class. This target Java class to be marshalled must have a default no argument constructor and implement the JsonDecoder interface.

Type Parameters:

T - target class that represents the return value of a remote JavaScript function.

Parameters:

decoderClass - target Java class' Class object.

Returns:

an instance of that represents a marshalled return value of the remote JavaScript function or null if this Result object is of type void.

```
public boolean isVoid()
```

Was remote call of return type void?

Returns:

true if remote method did not set any response or false otherwise, even if the remote client set a null response.

4 JavaScript API Reference

4.1.1 fermi.Session class

This class encapsulates a connection between this remote client and a particular server endpoint.

`Session(path:String)`

Use this constructor to establish a Websocket connection to the server endpoint identified by its URL path. It must match exactly the path on which the server endpoint was created with the FermiContainer.newEndpoint() method. Must start with the forward slash (/) character. It is an error to call this method more than once.

4.1.2 fermi.Result class

A Result object is created by the Fermi framework and is passed to a callable user function as its first argument. It is used by the function body to send the result object that will be sent back to the server.

`Response.send(response:Object)`

Send the response object to the server. It may be any JavaScript object, including null. The framework will use the standard JSON.stringify() method to marshall this object into a JSON representation before sending it to the client. If the argument is omitted, this response object will be rendered in Java domain as a void response.

`Response.throw(message:String)`

Send an exception to the server. This will cause the FutureResponse.get() (and, consequently, FermiSession.invokeBlocking()) methods to throw the run time FermiException.

5 Fermi-Demo Application

To Be Completed

6 References

[1] Websocket JavaScript API

http://www.w3.org/TR/websockets/

[2] Websocket Java API
http://docs.oracle.com/javaee/7/tutorial/doc/websocket.htm

[3] java.util.concurrent.Future interface Javadoc page
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html