Introduction to WebSocket RPC Compiler

ObjectRiver has built a Cloud Compiler that is capable of parsing many metadata languages, and generating complex solutions. WebSocket RPC was developed to facilitate running ObjectRiver Programmable Metadata Compiler in the cloud.

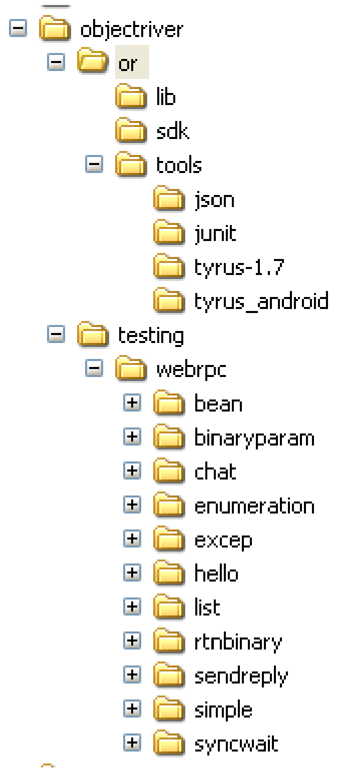
This manual is describing the WebSocket RPC metadata needed to generate a JSR356 compliant application.

Download

ObjectRiver Cloud Compiler can be downloaded from our web site once you join our forum at objectriver/forum

You will need your forum name and password to run the compiler.

Cloud Compiler SDK Directories



```
objectriver
    or
        lib
        sdk
        tools
            json
            junit
            tyrus-1.7
            tyrus_android
    testing
        webrpc
            bean
            binaryparam
            chat
            enumeration
            excep
            hello
            list
            rtnbinary
            sendreply
            simple
            syncwait
```

The cloud compiler tool kit is all jars files.

- or/lib
    - webrpcrt
      WebRPC runtime which includes some XDR and JSON marshalling classes.
- or/sdk
    - cloudcclient.jar
      ObjectRiver Cloud Compiler client.
    - Intellijplugin
      Intellij plugin for the Cloud Compiler
    - Eclipseplugin
      Eclipse plugin for the Cloud Compiler

- or/tools/tyrus1.7
  This is the client/server Tyrus runtime. WebRPC can be run completely standalone during development. Tyrus is the WebSocket reference implementation and is bundled with Glassfish.

- or/tools/tyrus_android
  This the Tyrus client compiled with Java 1.6 for the Android. It is the full client which support callbacks etc…

- testing/webrpc/bean
  Test example for bean definitions.

- testing/webrpc/binaryparam
  Test example which passes a BinaryStream from client to server.

- testing/webrpc/chat
  Chat application example

- testing/webrpc/enumeration
  Enumerated types example

- testing/webrpc/excep
  Exception example

- testing/webrpc/hello
  Hello World example from JavaOne 2014

- testing/webrpc/list
  List example

- testing/webrpc/rtnbinary
  Example which returns a BinaryStream and return argument

- testing/webrpc/sendreply
  Special RPC type where the application sends the reply instead of just returning from the method. See RPC type SendReplyMethod.

- testing/webrpc/simple
  Not so simple sample.

- testing/webrpc/syncwait
  Special RPC where the client application is responsible for calling wait.

Executing Cloud Compiler

The compiler currently ships with a IDE plugin for Intellij. Installing this plugin is the easiest way to execute the compiler. The installation instructions detail how to configure this plug within the Intellij IDE.

The following a instructions for directly calling CloudCompilerClient.class directly from another IDE like Eclipse or NetBeans.

Usage: CloudCompilerClient model.lang -U user pw [-Iinclude -Ddefine -OoutDir -SsrcDir]
Example: CloudCompilerClient model/hello.webrpc –U steveoriver fredfred –O. –S./src

Users will need to specify the name of the model and their ObjectRiver forum name and password. The cloud compiler determines the language based on the model extension. WebSocket model would have an extension of **.webrpc**.

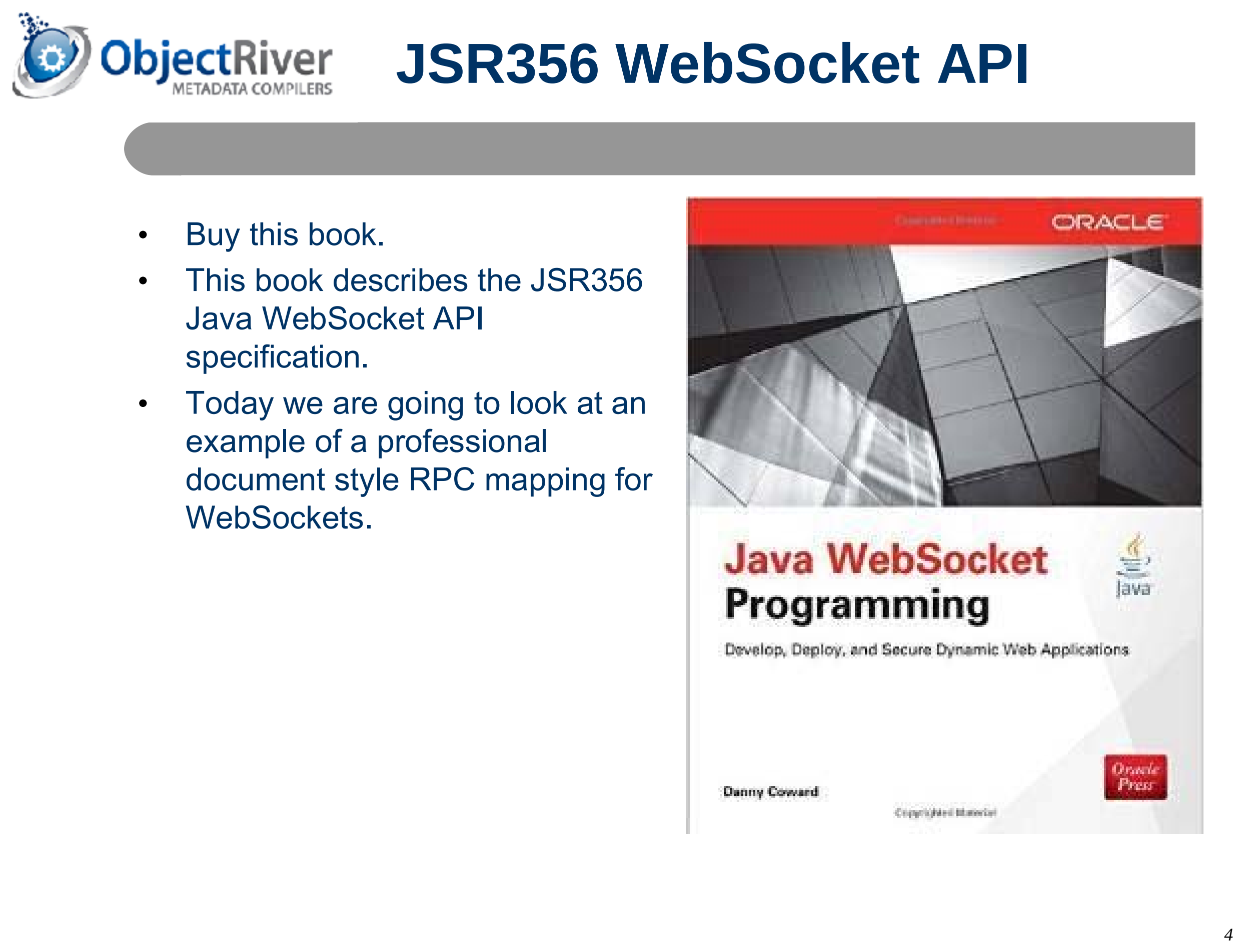
Arguments:

- model is the language the is being processed.
- -U user password
- -Iinclude
- -Ddefine
- -Ooutdir
- -Ssrcdir

There is also –Ddefine variables. The CloudCompiler uses CPP (‘C’ PreProcessor) on all metadata. There are optional defines for the IDE and destination Web Server.
[-Dintellij [-Dtomcat | -Dwildfly | -Dglassfish]]
These defines will generate the appropriate artifacts for building the server WAR file. If you are using the plugin, these options will be automatic.

WebSockets Jsr356

This is an excellent reference book that describes the JSR356 Java interface to WebSockets.



ObjectRiver METADATA COMPILERS

JSR356 WebSocket API

- Buy this book.
- This book describes the JSR356 Java WebSocket API specification.
- Today we are going to look at an example of a professional document style RPC mapping for WebSockets.

ORACLE
Java WebSocket Programming
Develop, Deploy, and Secure Dynamic Web Applications
Java
Danny Coward
Oracle Press

4

WebRpc is a description language describing remote procedure calls (RPC). From the description the ObjectRiver Cloud Compiler generates readable code that complies with the JSR356 specification.

The generated code uses the document style RPC pattern for implementing a request/reply RPC call.

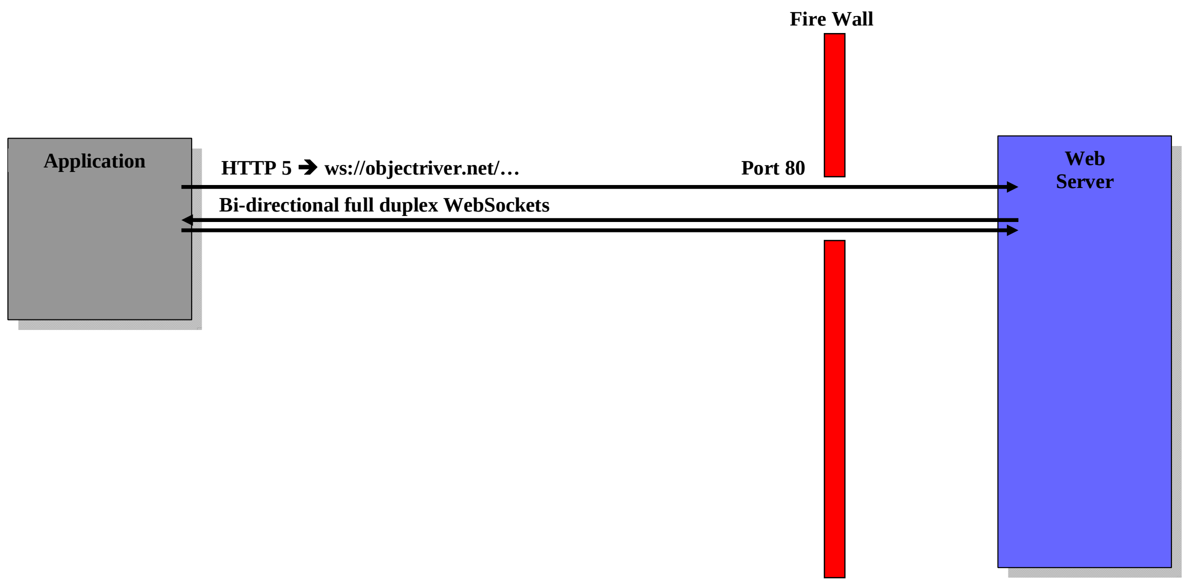
Sockets for the Web

WebSockets is a state-full, full duplex communications platform for the web. It is low latency because it has tiny headers and supports binary transmission of data.

Probably it’s biggest benefit, is that it runs over the same port 80 connection as any other web application.

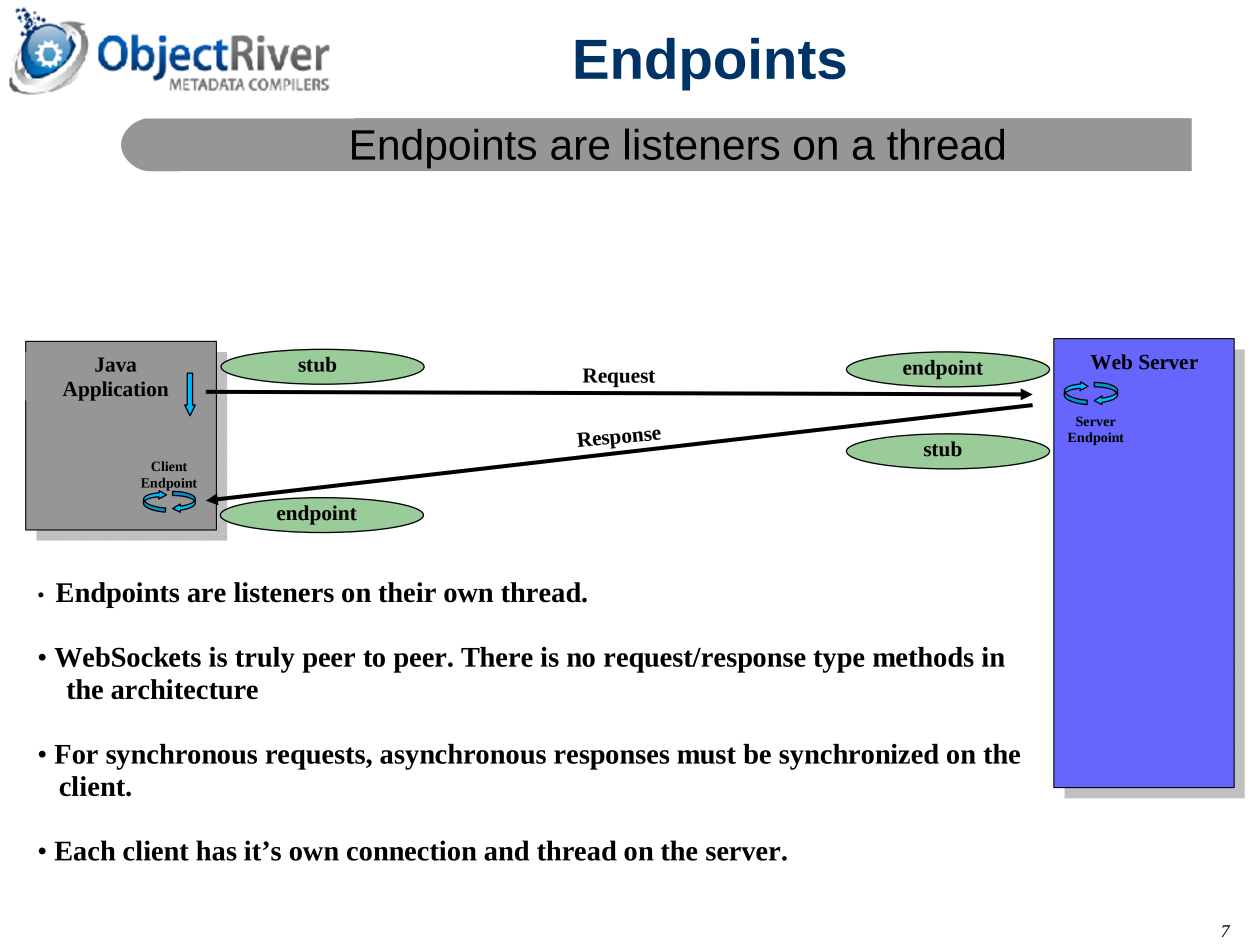
WebSockets into port 80

HTTP 5 request gets promoted to WebSockets


Fire Wall
Application
HTTP 5 ➔ ws://objectriver.net/…
Port 80
Bi-directional full duplex WebSockets
Web Server

Basic WebSockets

If you create a basic WebSocket application it looks like the following slide.



Endpoints

Endpoints are listeners on a thread

Java Application
stub
Request
endpoint
Web Server
Server Endpoint
Response
stub
Client Endpoint
endpoint

- Endpoints are listeners on their own thread.
- WebSockets is truly peer to peer. There is no request/response type methods in the architecture
- For synchronous requests, asynchronous responses must be synchronized on the client.
- Each client has it’s own connection and thread on the server.

7

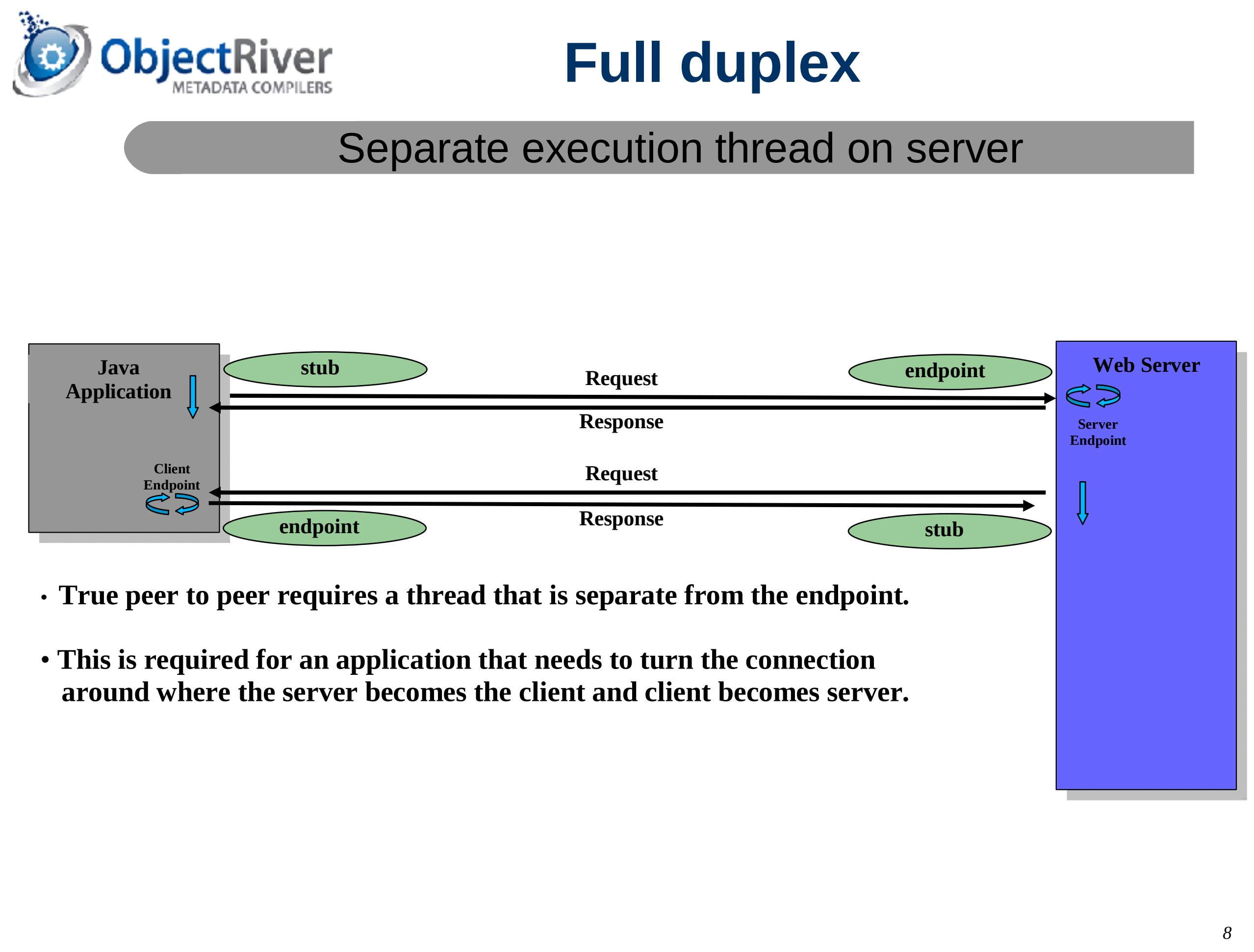
This diagram show two threads on the client, and one thread on the server. Each endpoint runs on its own thread, listening for inbound asynchronous messages. For a each client the server processes requests in serial. The server does not process the next message until it returns from it’s OnMessage() routine.

So, out of the box, the system is not truly peer to peer if you are doing any kind of synchronous requests. The protocol by definition only processes asynchronous messages.

WebRPC breaks synchronous requests into multiple asynchronous messages and synchronizes them together.

Peer to Peer

If you truly wants peer to peer full duplex communication, your app would look like this.



Full duplex

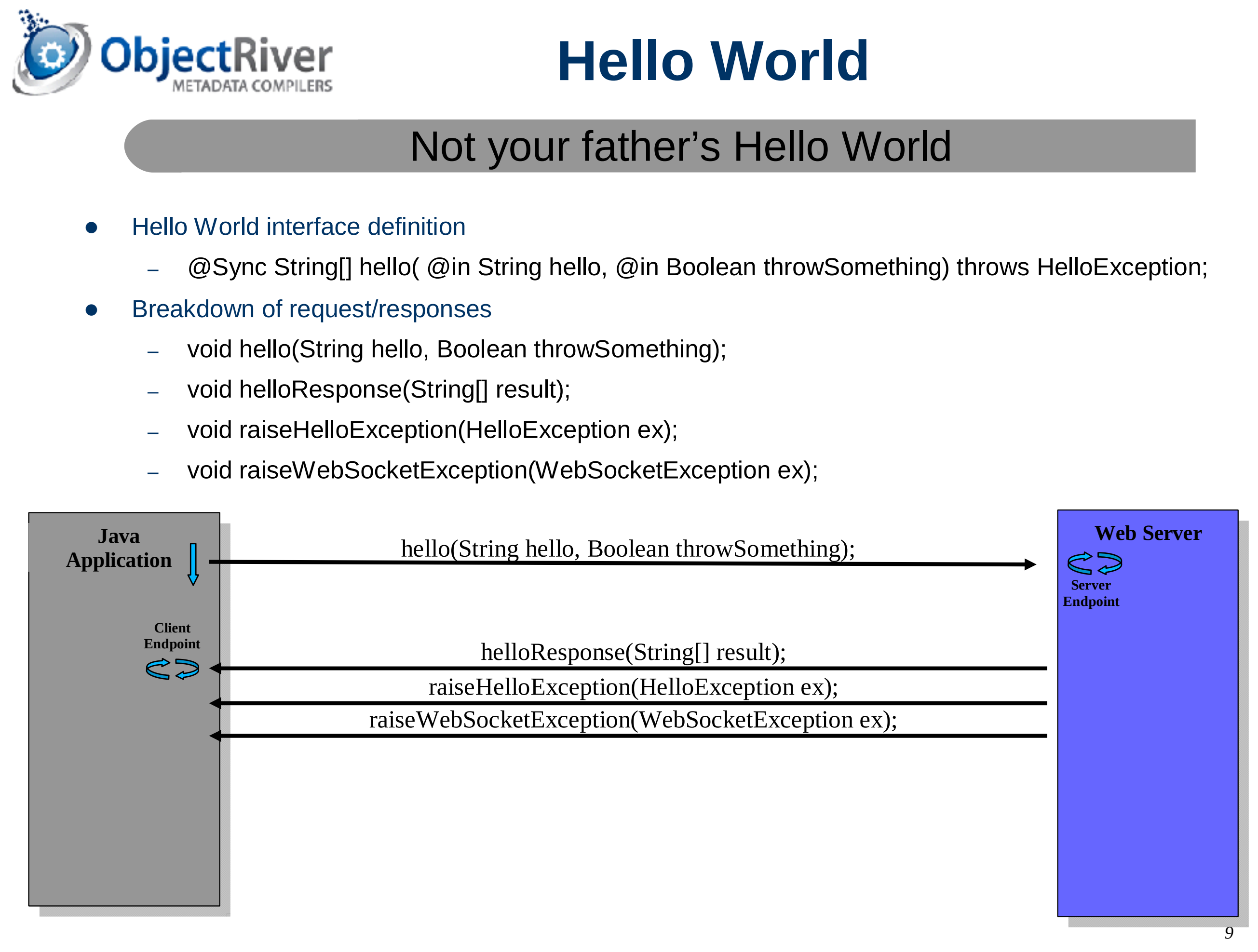
Separate execution thread on server

- True peer to peer requires a thread that is separate from the endpoint.
- This is required for an application that needs to turn the connection around where the server becomes the client and client becomes server.

In this example, the server has an extra thread that behaves like and independent client. You can have synchronous requests because the system is behaving like two separate applications. See **@OnThreadMethod** for more information. Many times we just see the client initiate the communication to the web server, and the session is turned around so the server becoming the client and the client becomes the server.

Hello World

Here is a “Hello World” example showing how the WebRpc Cloud Compiler breaks a simple RPC request into multiple asynchronous messages.



Hello World

Not your father’s Hello World

- Hello World interface definition
  - @Sync String[] hello(@in String hello, @in Boolean throwSomething) throws HelloException;
- Breakdown of request/responses
  - void hello(String hello, Boolean throwSomething);
  - void helloResponse(String[] result);
  - void raiseHelloException(HelloException ex);
  - void raiseWebSocketException(WebSocketException ex);

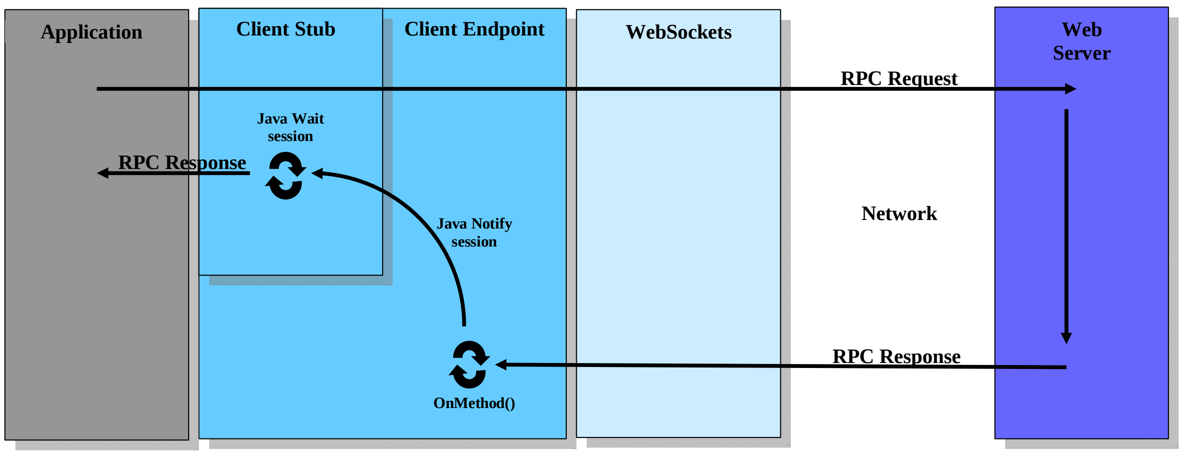
Java Application
Client Endpoint
Web Server
Server Endpoint
hello(String hello, Boolean throwSomething);
helloResponse(String[] result);
raiseHelloException(HelloException ex);
raiseWebSocketException(WebSocketException ex);
9

Notice how the single method hello is broken down into four asynchronous messages. One request named hello() and for responses named helloResponse(), raiseHelloException() and raiseWebSocketException().

The final piece of the puzzle is synchronizing the response to create a synchronous request. The WebRPC mapping generates a Java wait/notify logic which synchronizes the request with the replies.

hello_wait()

Java wait/notify synchronizes request



•Client sends request to server

•Client blocks on session with Java wait() with timeout from client handle

• Client endpoint in onMessage() wakes up client with Java notify synchronising two asynchronous methods

Interface Definition

Definition language for describing the RPC model. The language lets you describe the characteristics of the API that you are trying to distribute.

Application

Syntax: **Application Simple = com.companyname { … }**
Application describes the namespace for the RPC project. Application name will be used as a prefix on much of the generated source code. From the example above the **com.companyname** will be used for the Java package location.

Endpoint

Syntax: **Endpoint Simple= 1 { … }**
Endpoint describes the interface name for the procedures being distributed. This statement also has the version number for the interface. You can have multiple interfaces with the same name with different version numbers. See Versioning.

Data Types

- String
  String/string both define ASCII string type. **String** type can be null, where **string** is defined as not null. Empty strings (“”) are considered null.

- Short/Integer/Long

Short/short, Integer/int, Long/long integer types. Short/Integer/Long can be null, where short/int/long can not be null.

- **Float/Double**
  Float/float, Double/double are decimal types. Float/Double can be null, where float/double can not be null.

- Date
  Date/date represent a calendar date. Date can be null, where date can not be null.

- Timestamp
  Timestamp/timestamp represent date and time. Timestamp can be null, where timestamp can not be null.

- Boolean
  Boolean/boolean represent Boolean value. Boolean can be null, where Boolean can not be null.

- Byte/byte[]
  Byte represents a single binary character and may be null.
  byte[] represents array is bytes that will be marshaled as a group.

- Character
  Character/char represents a single character. Character can be null, where char can not be null.

- BinaryStream
  BinaryStream argument can be passed as an @In argument or returned as a return value.

  If BinaryStream is declared as an @In argument, the method must return void. This is because the application must start streaming to the variable once the method returns from the stub. In addition to the BinaryStream @In argument the method may have other arguments.

  If the method returns a BinaryStream, the server-side behaves like a @SyncReplyMethod method which sends the reply from the application implementation, and then streams the return value.

Client & Server Method Definitions

```
Server Singleton {
    @SyncMethod String hello ( @In Integer input ) raises MyException
};
Client Session {
     @AsyncMethod boolean callback ( @In String output ) raises MyException;
};
```

Server and Client define which the RPC is going. **Server** define a method from client to server, and **Client** specifies from server to client.

This syntax also defines whether the endpoint is a **Session**, or a **Singleton** endpoint.

Method Defintions

- Arguments
  Methods have arguments and a return value. Arguments can be annotated with **@In, @Inout, @Out**, and **@NotNull** modifiers that indicate the direction in which the argument must be passed. **@NotNull** just indicate that the argument can not be null, and a marshalling error will be thrown.
- Exceptions
  Methods can throw application based exceptions. Exceptions can be defined like the following example. **Exception SimpleException {};**
  Exceptions can also contain members that are defined with the curly braces as follows. **Exception SimpleException { Integer code; };**

Data Structures

- Enumeration
```
Enumeration MyMonth { Jan=1, Feb=2, Mar=3, Apr=4, May=5, Jun=6,
        Jul=7, Aug=8, Sept=9, Oct=10, Nov=11, Dec=12 };
Enumeration MyDay { Sun=1, Mon=2, Tues=3, Wed=4, Thu=5, Fri=6, Sat=7 };
```

- Bean
```
Bean MyTime {
   int hours;
   int minutes;
   int seconds;
};
Bean MyDateTime {
   Bean MyDate {
      int year;
      MyMonth month;
      MyDay day;
   } date;
   MyTime time;
};
```
  Beans are aggregated data types used for constructing records or just collections of other data types.

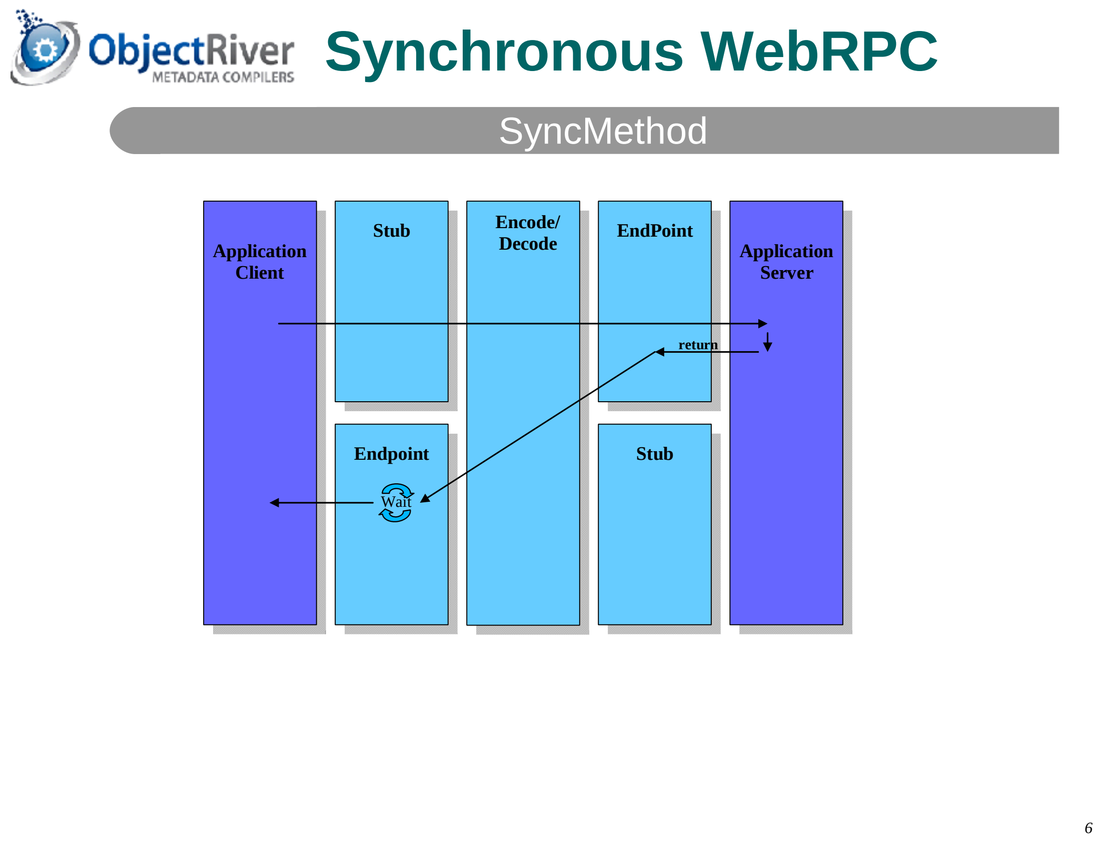
- Array
  String[] args
  Arrays are variable length lists of data types.

- List
  List<String> args
  List are collections of data types.

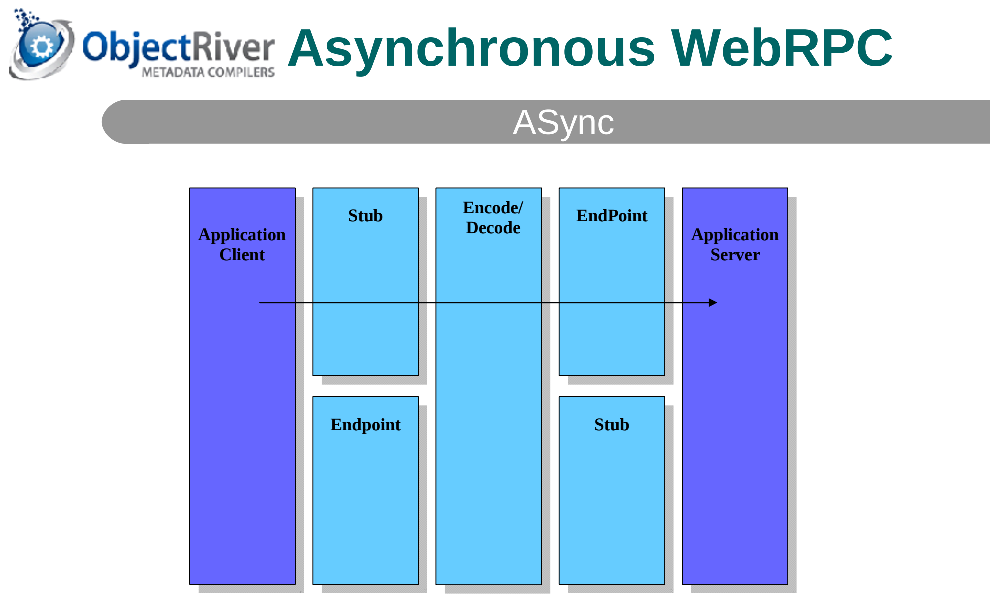
- Exceptions
  Exception MyException { int code;};
  Exceptions are beans which by default contain a String message, and a Throwable cause elements. Exception may contain additional members. Note: Exceptions are not versioned.

Method Types

- Synchronous
  @SyncMethod is a synchronous response/response method call. Client send input arguments to the server, and wait for the return argument response.

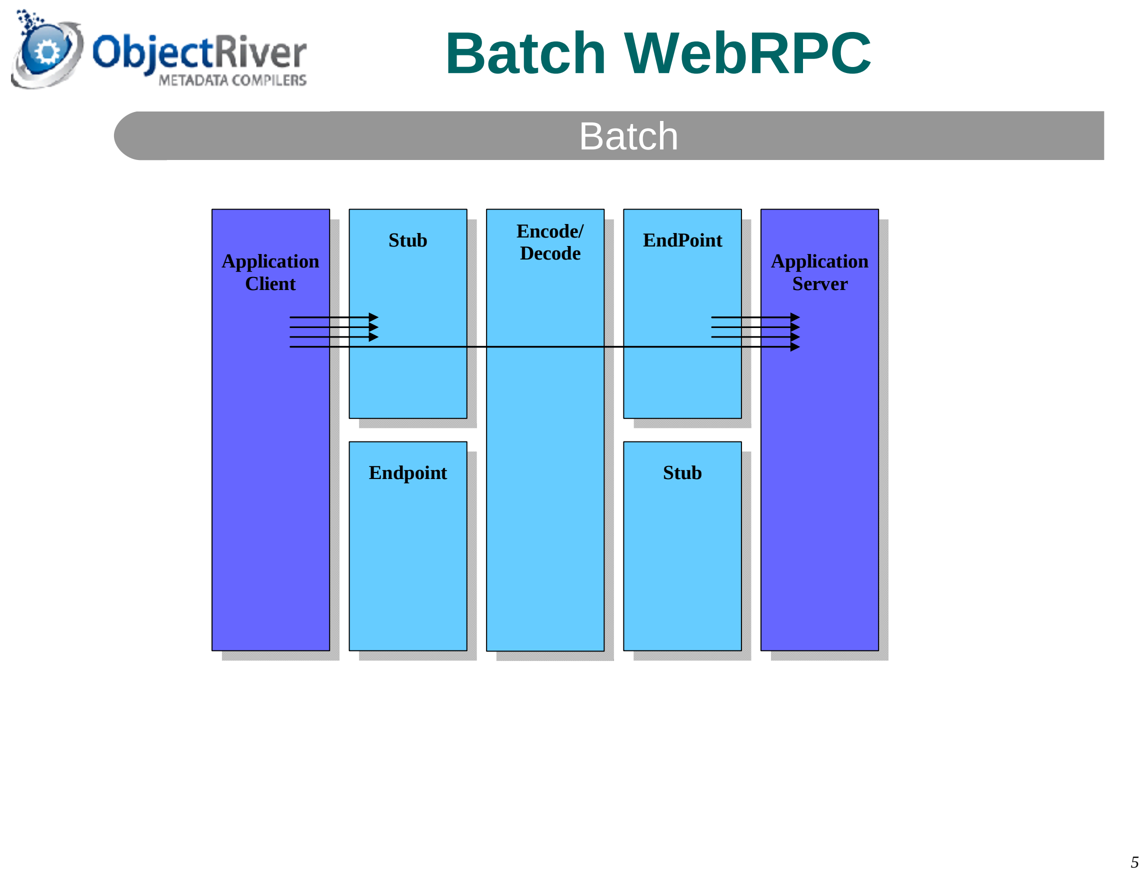


- Asynchronous
  @AsyncMethod is a asynchronous request method call. Client sends input arguments to the server and immediately returns.



- Batch
  @BatchMethod is a asynchronous request method call. Client buffers the method and its input arguments on the client-side. When a synchronous method call is made, and previous batch methods are batched to the server. Batched methods are processed in serial once they arrive on the server.

  Batch methods are typically simple methods like variable setters.



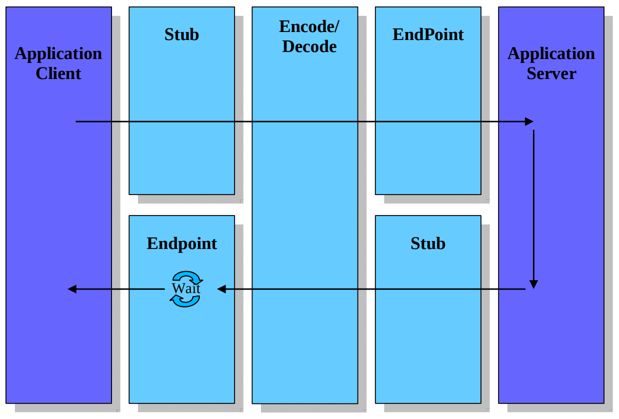
- Synchronous Application Reply
  @SyncReplyMethod is a synchronous request/reply method that has a special generated server stub, that is used by the application to send the reply arguments back to the client. Once the reply has been sent by the server, this action releases the client from waiting. The server now can do some work asyncrously from the client, but will not process any more RPC's until it returns from OnMessage.

  In the case where the RPC method is defined as returning a BinaryStream, the method is automatically promoted from a @SyncMethod to a @SyncSendReply method. This allows the server to start streaming results to the client, just after the call to the sendreply stub.

  @SyncReplyMethod is also used to avoid argument promotions to ensure the defined API is strictly defined for the Java mapping. See argument promotions.

Synch Reply RPC

SyncAppReplyMethod

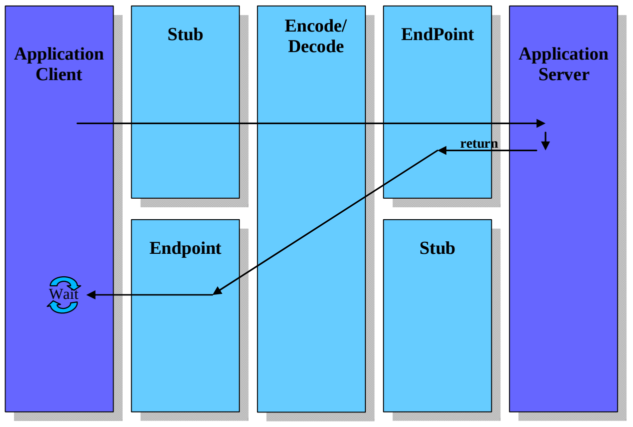


- Synchronous Application Wait
  @SyncWaitMethod is a synchronous request/reply method where the application calls the client stub wait method to wait for the response. This gives the application more control of the client side events.

  In the case where the client defines a method that contains a BinaryStream inout argument, the method automatically promoted from a @SyncMethod to a @SyncWaitMethod. This lets the client start streaming the to the server after the method has been sent.

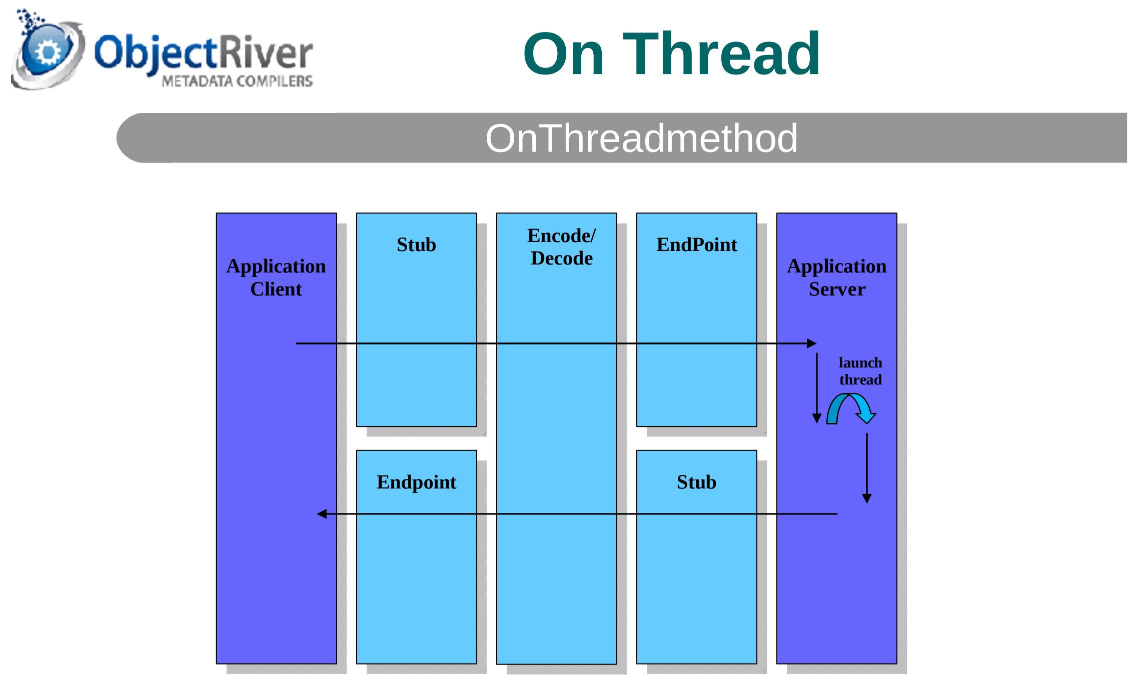
Synch Wait WebRPC

SyncAppWaitMethod



- Asynchronous On Thread
  @OnThreadMethod is typically only used once in the case the application a role reversal or the client and server. This is where the client becomes the server, and server becomes the client.

  It is also used in case where the application wants to implement full duplex communication between the client and
  s

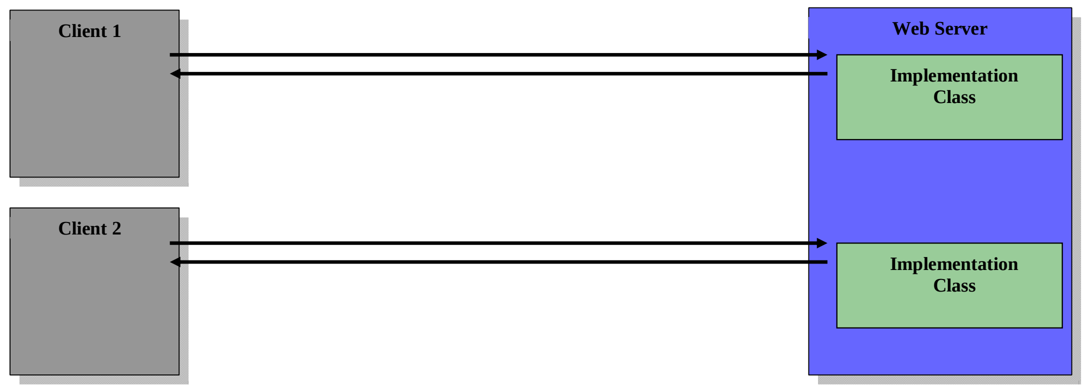


Servers

- Session

Session Server

- Session Server
  - This is where each client has it’s own state-full copy of the server’s implementing class.
  - The state lasts for the duration of the WebSockets session with the client
  - It’s exactly like each client have it’s own copy of the server.

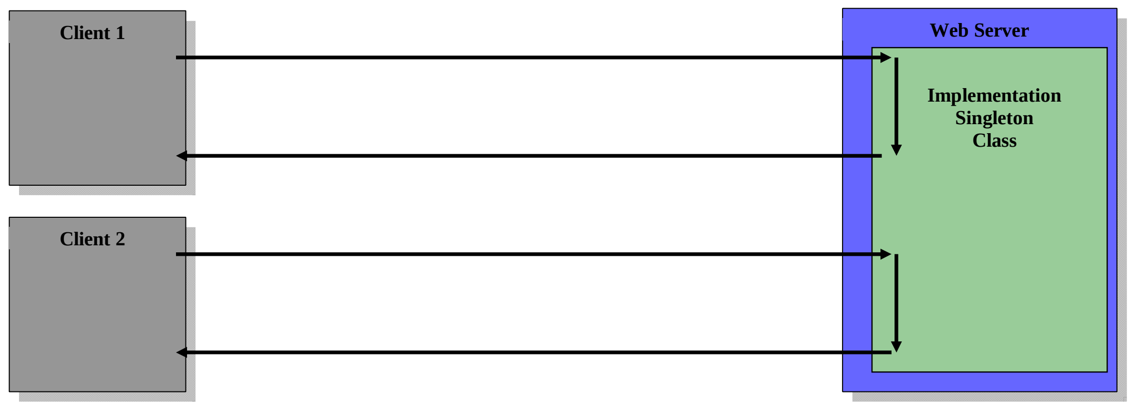


29

- Singleton

Singleton Server

- Singleton Server
  - This is where there is a single instance of the implementing class, and the server’s state is shared.
  - Therefore multiple client threads of execution are executing on the single singleton class.
  - Implementing class must be thread safe.
  - Individual client state can be stored in the sessions user properties.



31