Unleashing the Power of JAX-WS RI: Spring, Stateful Web Services, SMTP, and More

Kohsuke Kawaguchi
Jitendra Kotamraju
Vivek Pandey

Sun Microsystems, Inc.
http://jax-ws.dev.java.net

TS-4948

Goal of the Talk

Java™ APIs for XML Web Services (JAX-WS) RI is so much more than JAX-WS API.

JAX-WS RI and Extensibility

Background

- Extensibility was a key goal
- We are putting our money where our mouth is
  - WSIT
  - http://jax-ws-commons.dev.java.net/
  - Java Business Integration (JBI)
  - Expect more in the future

Agenda

Spring Support

JSON Support

Stateful Web Service

Transports

Additional Headers

Multiple Service Instances

Monitoring and Logging

Server-Side Asynchrony

Agenda

Spring Support

JSON Support

Stateful Web Service

Transports

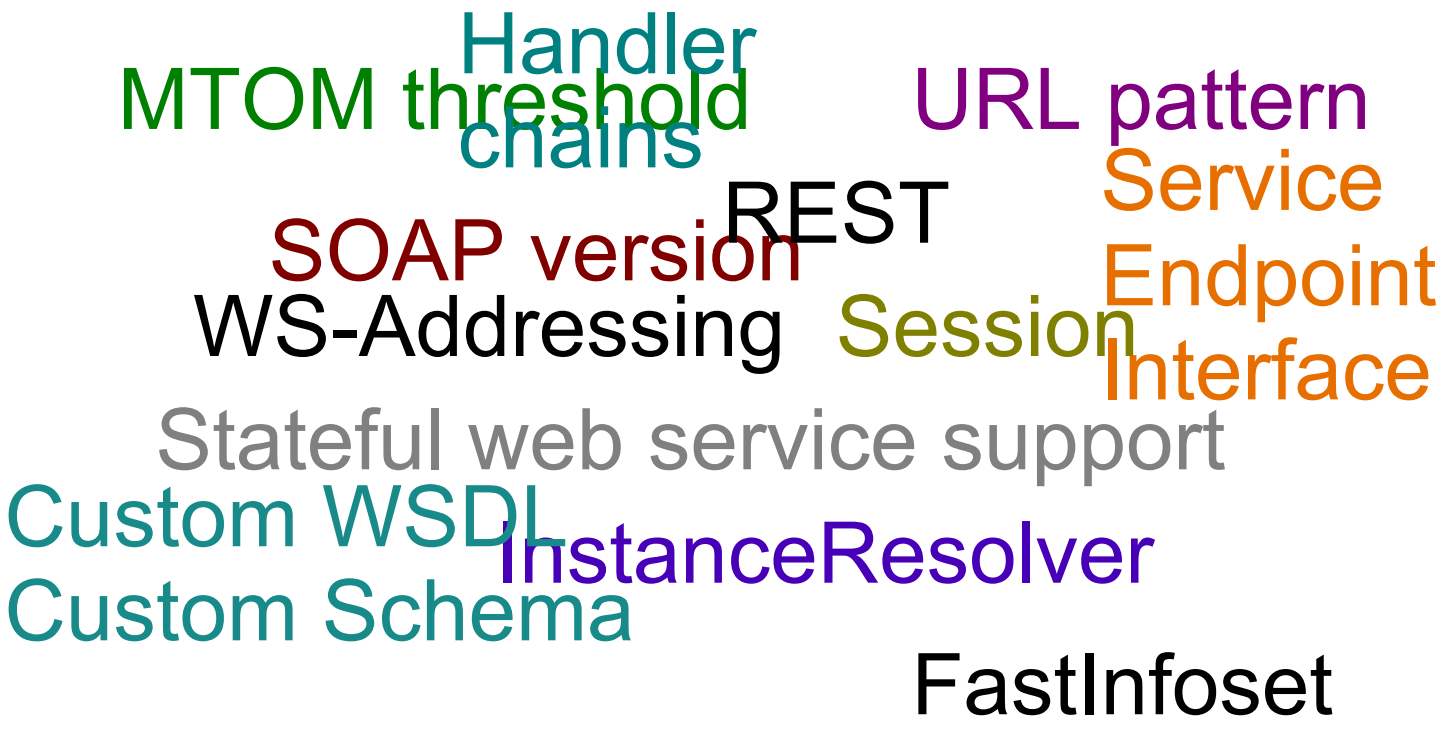
Additional Headers

Multiple Service Instances

Monitoring and Logging

Server-Side Asynchrony

There Are Million Different JAX-WS API Configurations!



…and More Deployment Problems

- Just doing Java Platform, Enterprise Edition (Java EE platform) 5 would have been easier
  - But we needed to do `sun-jaxws.xml` for Tomcat, Jetty, etc.
- Hard to justify the cost of custom syntax
- Syntax not very extensible

Spring Support

What is this?

- Single mechanism that works everywhere
  - …provided that Spring exists
- Many people seem to be already using it
- Extensions will work very seamlessly

Spring Support

What is this?

- If you are already using Spring
  - This should make a lot of sense, almost obvious
  - You'll want to switch from JSR 109/sun-jaxws.xml
- If you are using other IoC containers
  - Let us know what you use
- If you don't believe in IoC
  - Continue to use JSR 109 or sun-jaxws.xml
  - Spring is not for everyone

JSR = Java Specification Request

Spring Support

Why should I care?

- BookService will need to access other objects
  - Data access objects, audit logger, etc.
- How does BookService and MyHandler talk to each other?

```
@WebService
class BookService {
    List<Book> getRecommendedBooks() {
        ...
    }
}
class MyHandler implements Handler<Message> {
    int affiliateId;
    ...
}
```

Spring Support

Sample configuration

```
<beans xmlns=...>
  <bean id="myHandler" class="foo.MyHandler" />
```

```
MyHandler {
    ...
}
```

Spring Support

Sample configuration

```
<beans xmlns=...>
  <bean id="myHandler" class="foo.MyHandler" />

  <bean id="myService" class="foo.BookService">
```

```
MyHandler {
    ...
}

BookService {
    ...
}
```

Spring Support

Sample configuration

```
<beans xmlns=...>
  <bean id="myHandler" class="foo.MyHandler" />

  <bean id="myService" class="foo.BookService">
    <property name="handler" ref="myHandler" />
```

```
MyHandler {
    ...
}

BookService {
    MyHandler handler;
}
```

Spring Support

Sample configuration

```
<beans xmlns=...>
  <bean id="myHandler" class="foo.MyHandler" />

  <bean id="myService" class="foo.BookService">
    <property name="handler" ref="myHandler" />

  <wss:binding url="/stockQuote">
    <wss:service>
      <ws:service bean="myService">
        <ws:handlers>
          <ref bean="myHandler" />
```

Spring Support

Sample configuration

```
<beans xmlns=...>
  <bean id="myHandler" class="foo.MyHandler" />

  <bean id="myService" class="foo.BookService">
    <property name="handler" ref="myHandler" />

  <wss:binding url="/stockQuote">
    <wss:service>
      <ws:service bean="myService">
        <ws:handlers>
          <ref bean="myHandler" />
```

Spring Support

Sample configuration

```
<beans xmlns=...>
  <bean id="myHandler" class="foo.MyHandler" />

  <bean id="myService" class="foo.BookService">
    <property name="handler" ref="myHandler" />

  <wss:binding url="/stockQuote">
    <wss:service>
      <ws:service bean="myService">
        <ws:handlers>
          <ref bean="myHandler" />
```

Spring Support

Sample configuration

```
<beans xmlns=...>
  <bean id="myHandler" class="foo.MyHandler" />

  <bean id="myService" class="foo.BookService">
    <property name="handler" ref="myHandler" />

  <wss:binding url="/stockQuote">
    <wss:service>
      <ws:service bean="myService">
        <ws:handlers>
          <ref bean="myHandler" />
```

Agenda

Spring Support

JSON Support

Stateful Web Service

Transports

Additional Headers

Multiple Service Instances

Monitoring and Logging

Server-Side Asynchrony

One Day It Occurred to Us…

- You already wrote a web service
- Wouldn't it be nice if you can expose the same service to JavaScript™ technology clients?
  - That means sending JSON, not XML

Simple Example: Server

```
@WebService
@BindingType(JSONBindingID.JSON_BINDING)
public class MyService {
    public String sayHelloTo(@WebParam("name") String name) {
        return "Hello, "+name;
    }
}
```

Simple Example: Client

On the client, first you include proxy code

```
<script src="endpoint?js"></script>
```

Simple Example: Client

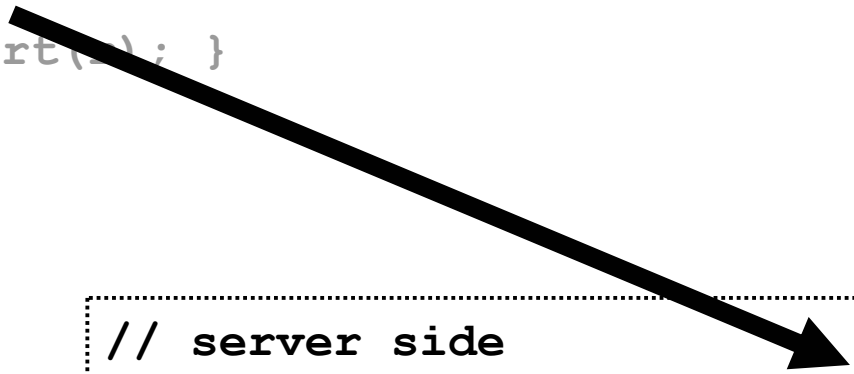
…then make a call

```
<script src="endpoint?js"></script>
<script>
  myService.sayHelloTo(
    { name:“duke” },
    function(r) { alert(r); }
  );
</script>
```

Simple Example: Client

…then make a call

```
<script src="endpoint?js"></script>
<script>
  myService.sayHelloTo(
    { name:"duke" },
    function(r) { alert(r); }
  );
</script>
```

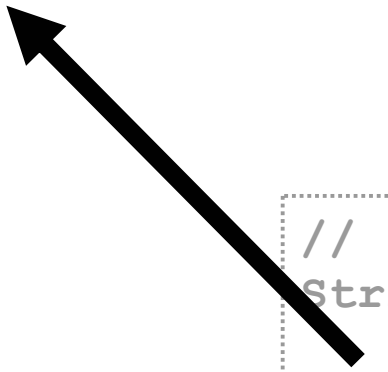


```
// server side
String sayHelloTo(String name) {
    return "Hello,"+name;
}
```

Simple Example: Client

Response triggers callback

```
<script src="endpoint?js"></script>
<script>
  myService.sayHelloTo(
    { name:"duke" },
    function(r) { alert(r); }
  );
</script>
```



```
// server side
String sayHelloTo(String name) {
    return "Hello,"+name;
}
```

Simple Example: Client

Response triggers callback

```
<script sr
<script>
  myServic
    { name
    functi
  );
</script>
```


```
// server side
String sayHelloTo(String name) {
    return "Hello,"+name;
}
```

Simple Example: Client

Let's not pollute the namespace

```
<script src="endpoint1?js&varName=ws.svc1"></script>
<script src="endpoint2?js&varName=ws.svc2"></script>
<script>
  ws.svc1.sayHelloTo(
    { name:"duke" },
    function(r) { alert(r); }
  );
</script>
```

JSON Support

- Any Java Architecture for XML Binding (JAXB) beans are supported

```
class Book {
    public int id = 1;
    public String title = "Java";
}
```

- You'll get

```
{ id:"1", title:"Java" }
```

DEMO

Agenda

Spring Support

JSON Support

Stateful Web Service

Transports

Additional Headers

Multiple Service Instances

Monitoring and Logging

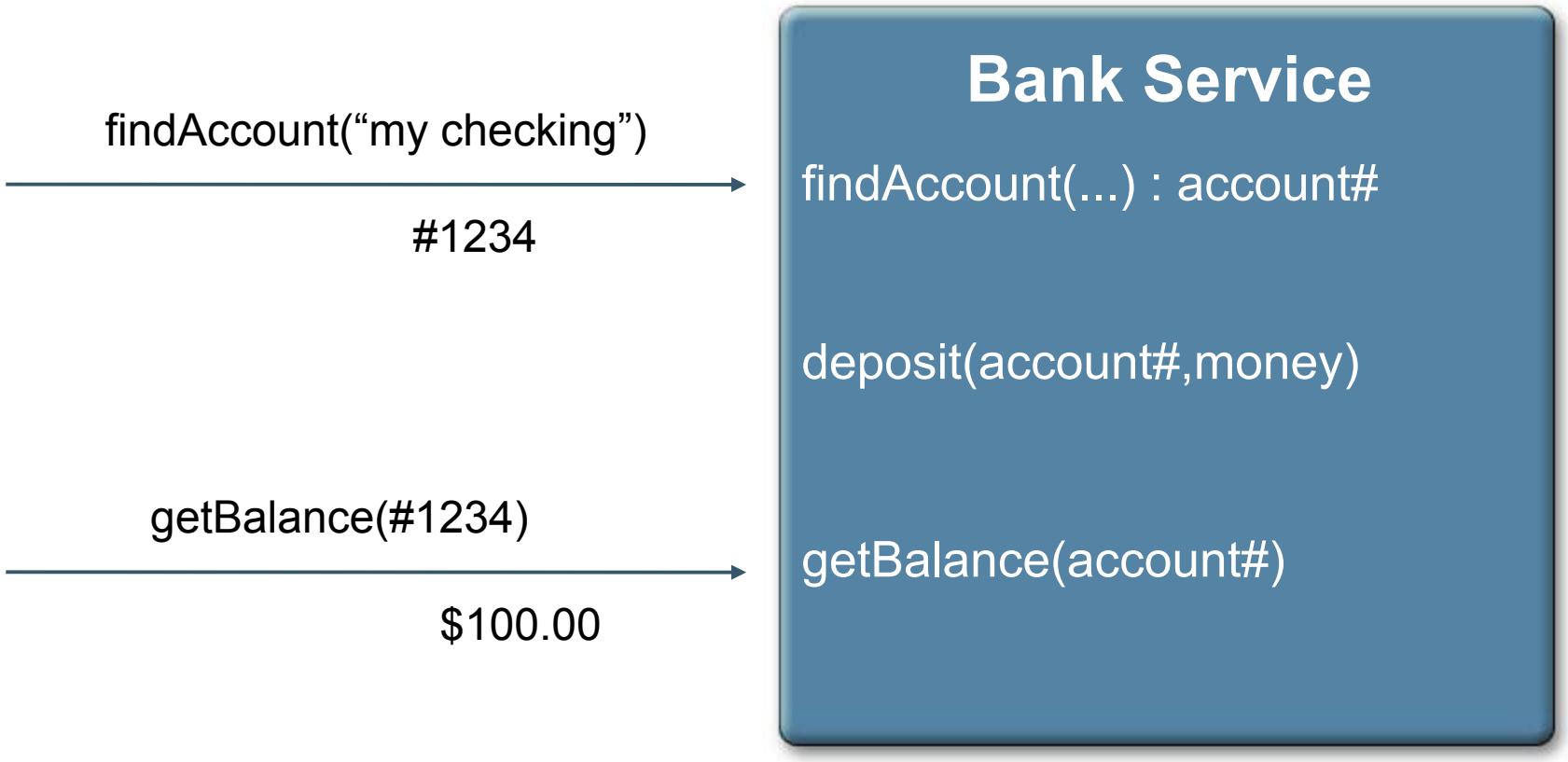
Server-Side Asynchrony

Stateful Web Service

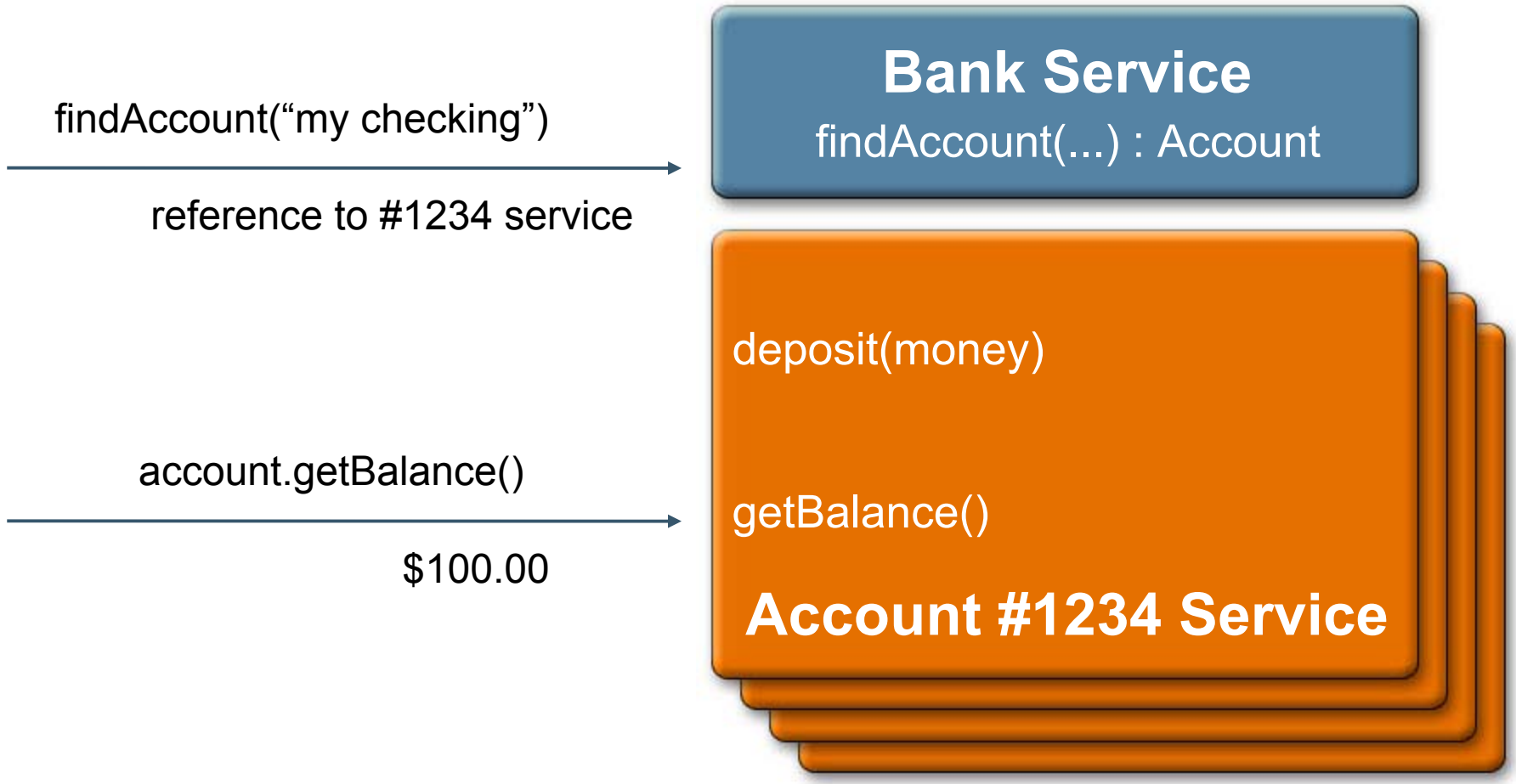
Motivation

- Web service programming model is very primitive
  - Basically a bunch of “global functions” akin to shared libraries
  - We, OTOH, think in terms of objects
- Wouldn’t it be nice if it’s richer?

Hypothetical Bank Service: Before



Hypothetical Bank Service: After



Stateful Web Service

Example

```
@com.sun.xml.ws.developer.Stateful @WebService @Addressing
public class Account {
    private final int id;
    private int amount;

    public Account(int id) { this.id = id; }
    public void deposit(int delta) { amount += delta; }
    public int getBalance() { return amount; }

    public static
    StatefulWebServiceManager<Account> manager;
}
```

Stateful Web Service

Example

```
@com.sun.xml.ws.developer.Stateful @WebService @Addressing
public class Account {
    private final int id;
    private int amount;

    public Account(int id) { this.id = id; }
    public void deposit(int delta) { amount += delta; }
    public int getBalance() { return amount; }

    public static
    StatefulWebServiceManager<Account> manager;
}
```

Stateful Web Service

Example

```
@com.sun.xml.ws.developer.Stateful @WebService @Addressing
public class Account {
    private final int id;
    private int amount;

    public Account(int id) { this.id = id; }
    public void deposit(int delta) { amount += delta; }
    public int getBalance() { return amount; }

    public static
    StatefulWebServiceManager<Account> manager;
}
```

Stateful Web Service

Example

```
@com.sun.xml.ws.developer.Stateful @WebService @Addressing
public class Account {
    private final int id;
    private int amount;

    public Account(int id) { this.id = id; }
    public void deposit(int delta) { amount += delta; }
    public int getBalance() { return amount; }

    public static
    StatefulWebServiceManager<Account> manager;
}
```

Stateful Web Service

Example

```
@WebService
public class BankService {
    Map<Integer,Account> accounts = new HashMap();

    public EndpointReference findAccount(int id) {
        Account a = accounts.get(id);
        if(a==null)
            accounts.put(id, a=new Account(id));

        return Account.manager.export(a);
    }

}
```

Stateful Web Service

Example

```
BankService svc = new BankService();
BankPort bank = svc.createBankPort();
```

Stateful Web Service

Example

```
BankService svc = new BankService();
BankPort bank = svc.createBankPort();

EndpointReference aepr = bank.findAccount(0);
Account a = aepr.getPort(Account.class);
```

Stateful Web Service

Example

```
BankService svc = new BankService();
BankPort bank = svc.createBankPort();

EndpointReference aepr = bank.findAccount(0);
Account a = aepr.getPort(Account.class);
a.deposit(100);
assert a.getBalance()==100;
```

Stateful Web Service

Example

```
BankService svc = new BankService();
BankPort bank = svc.createBankPort();

EndpointReference aepr = bank.findAccount(0);
Account a = aepr.getPort(Account.class);
a.deposit(100);
assert a.getBalance()==100;

EndpointReference bepr = bank.findAccount(1);
Account b = bepr.getPort(Account.class);
b.deposit(500);
assert b.getBalance()==500;
```

Stateful Web Service

Example

```
BankService svc = new BankService();
BankPort bank = svc.createBankPort();

EndpointReference aepr = bank.findAccount(0);
Account a = aepr.getPort(Account.class);
a.deposit(100);
assert a.getBalance()==100;

EndpointReference bepr = bank.findAccount(1);
Account b = bepr.getPort(Account.class);
b.deposit(500);
assert b.getBalance()==500;

assert a.getBalance()==100;
```

Stateful Web Service

Portability

- Proprietary API, but standard on the wire
  - WS-Addressing
  - Other toolkits should be able to understand it
  - Although they might not offer a programming model benefit

Agenda

Spring Support

JSON Support

Stateful Web Service

Transports

Additional Headers

Multiple Service Instances

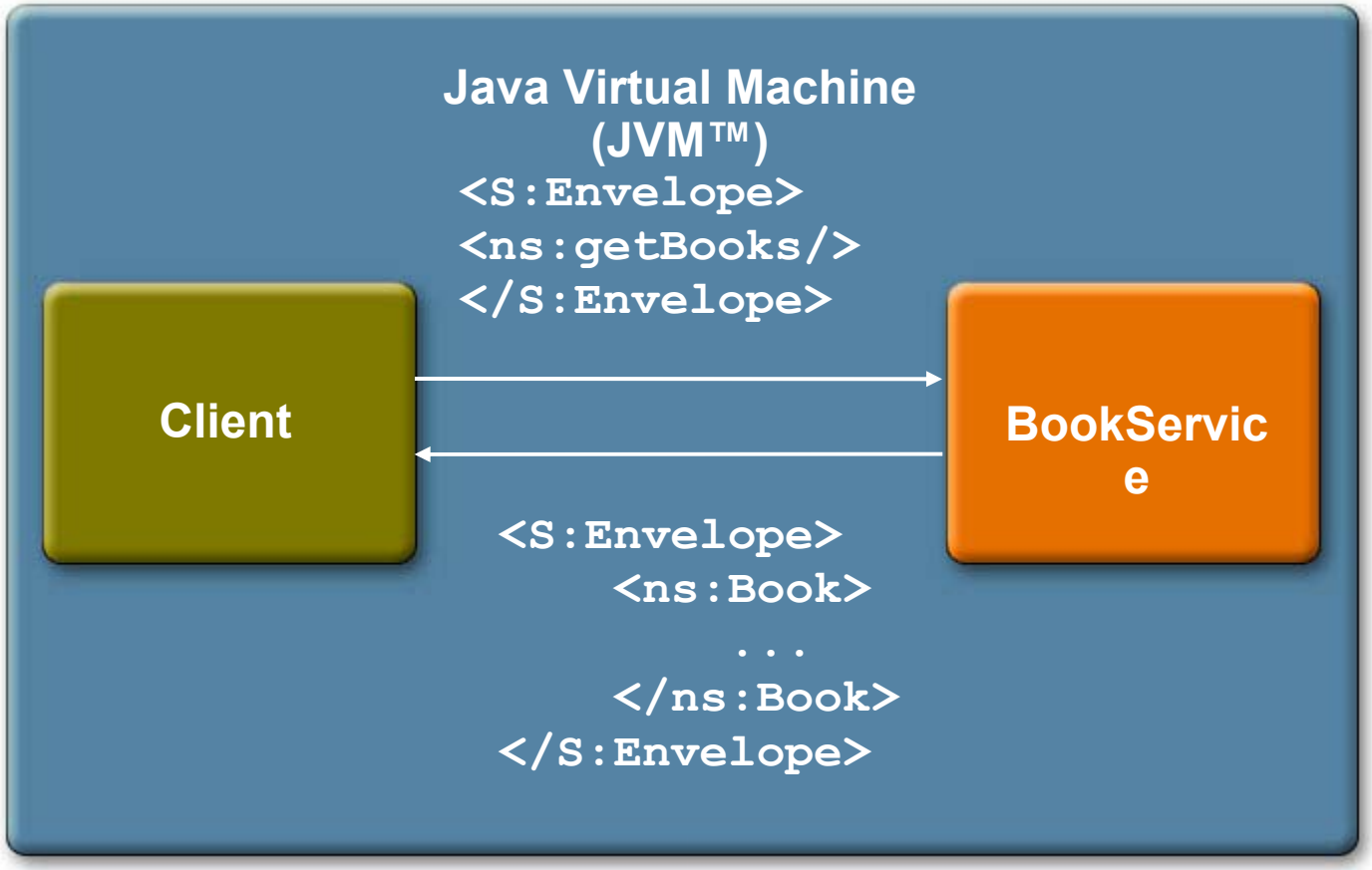
Monitoring and Logging

Server-Side Asynchrony

Transports

- JAX-WS API is not just HTTP
- Several implementations already available
  - In-VM
  - Java Message Service (JMS)
  - SMTP
  - SOAP/TCP
- Selection based on scheme
  - e.g,. **`smtp://myservice@sun.com`**

In-VM



The terms “Java Virtual Machine” and “JVM” mean a Virtual Machine for the Java™ platform.

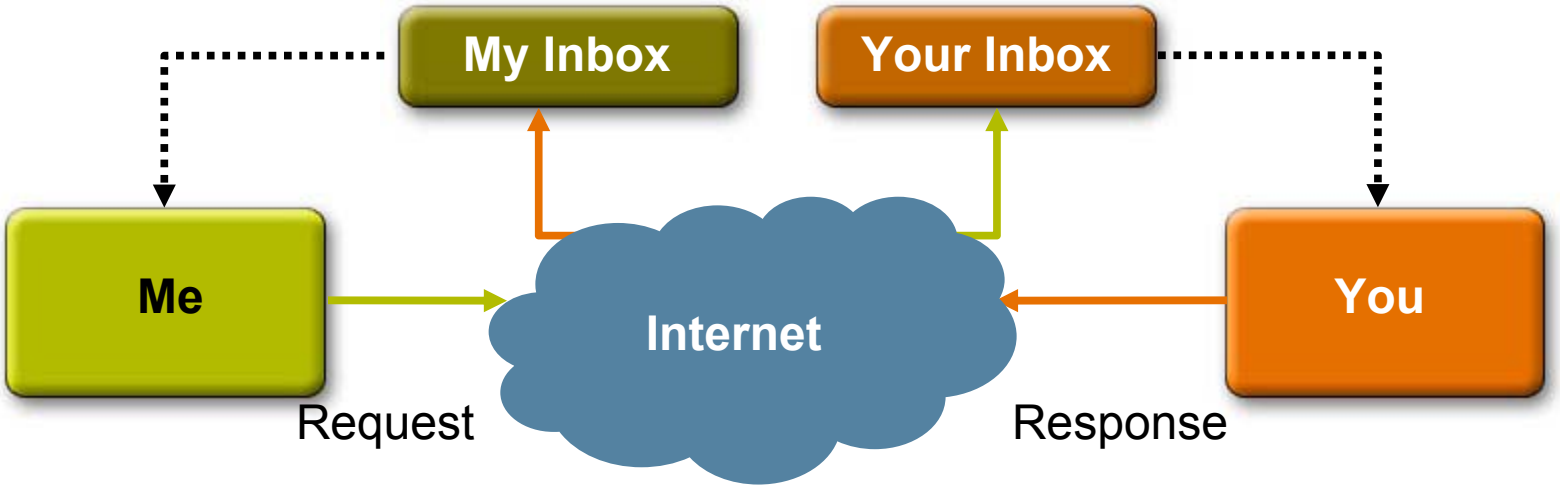
Running on In-VM

- Deploy

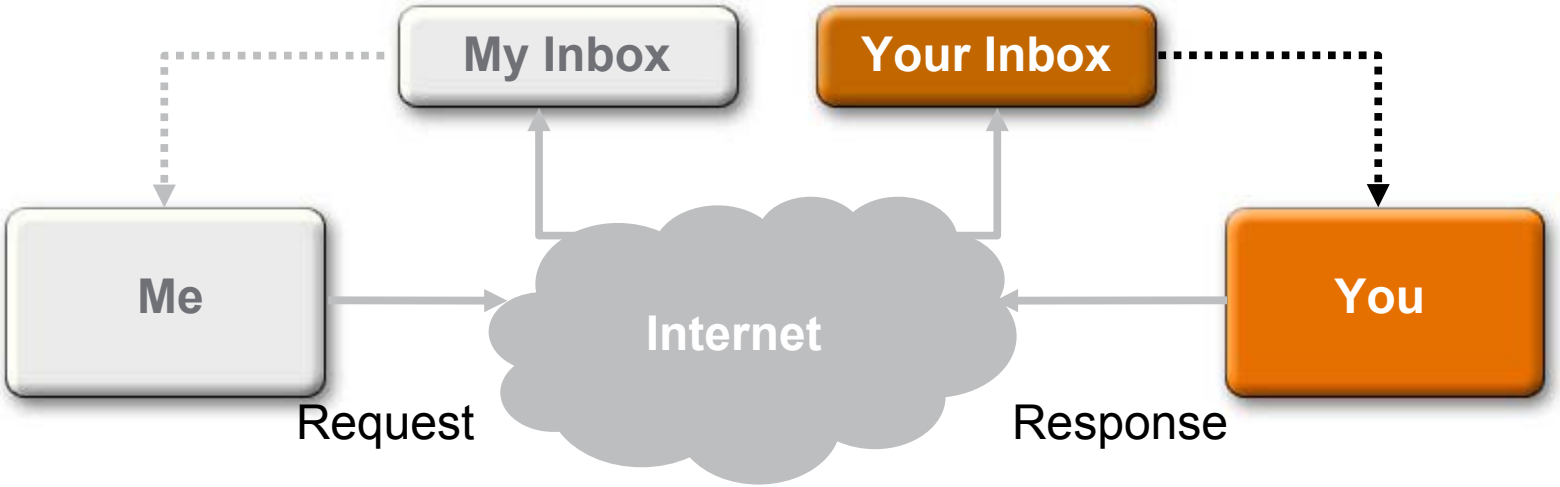
```
<in-vm:bindings id="books">
  <in-vm:endpoints>
    <ws:service id="bookService"
                impl="books.BookService" />
```

- **Accessing the service**
  - **Spring bean ID becomes part of the URL**
    - **in-vm://books/**

SMTP Transport

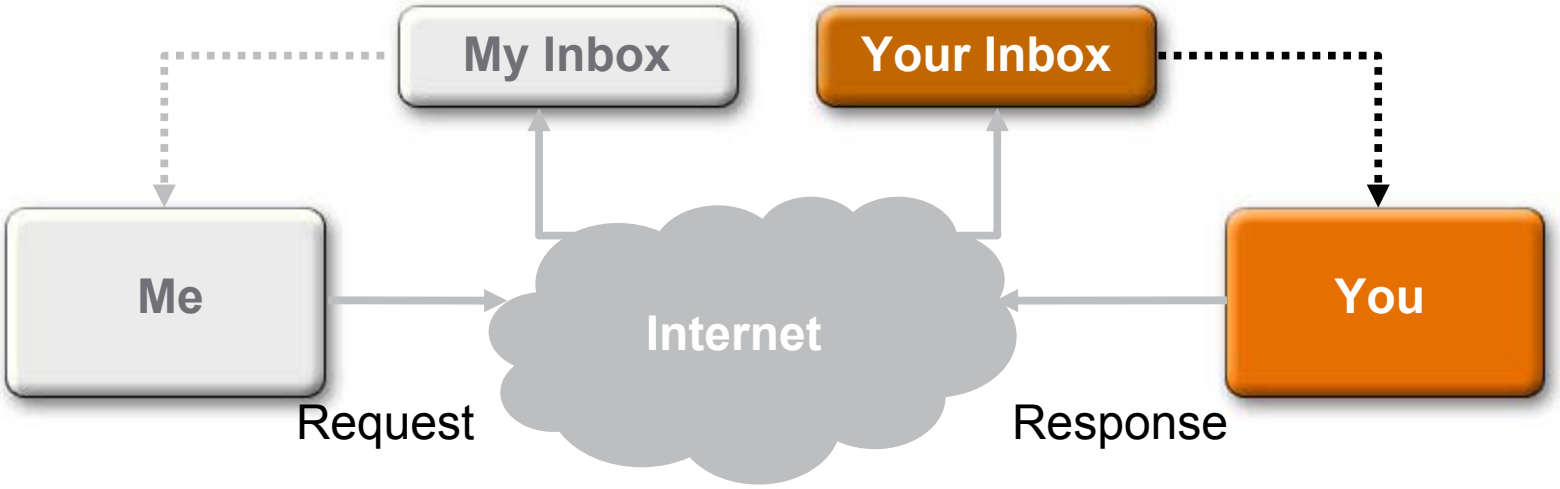


Configure Endpoint



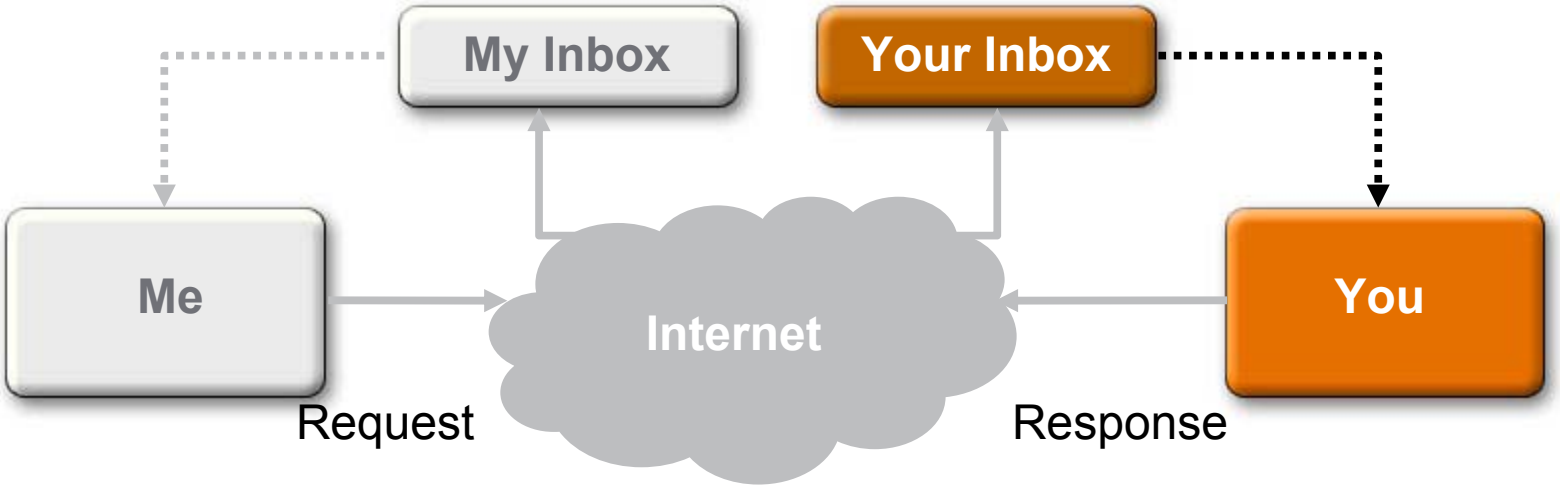

```
<bean id="greeter" class="greeter.GreetingService"/>
```

Configure Endpoint



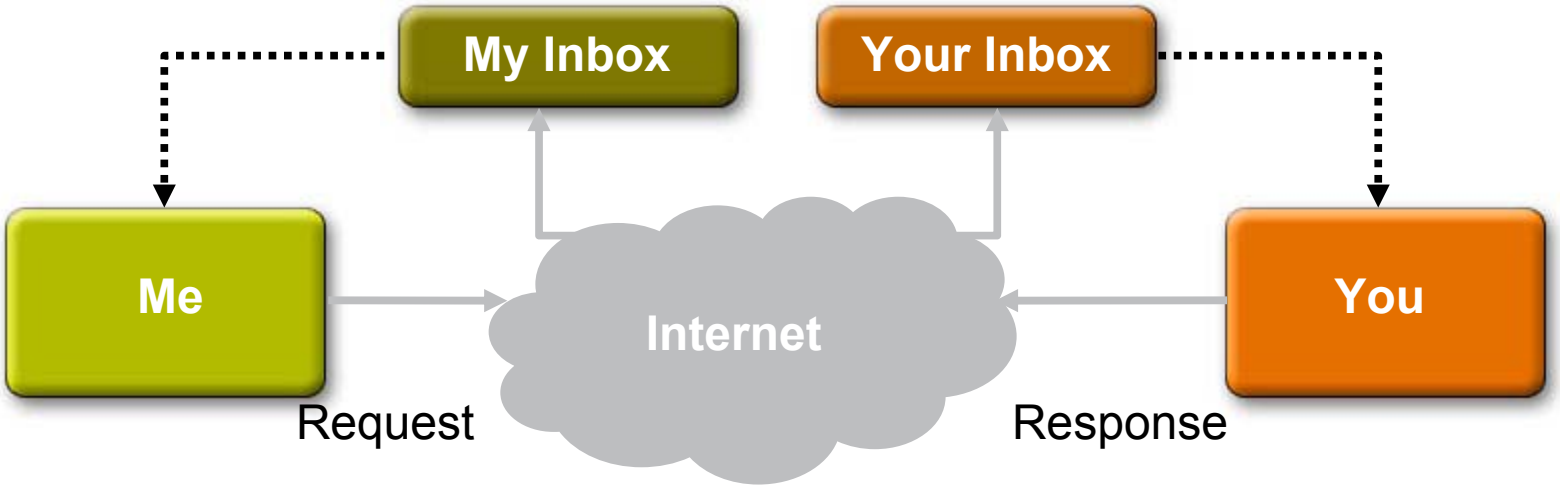

```
<bean id="greeter" class="greeter.GreetingService"/>
<ws:service id="greetingService" bean="greeter"/>
```

Configure Endpoint



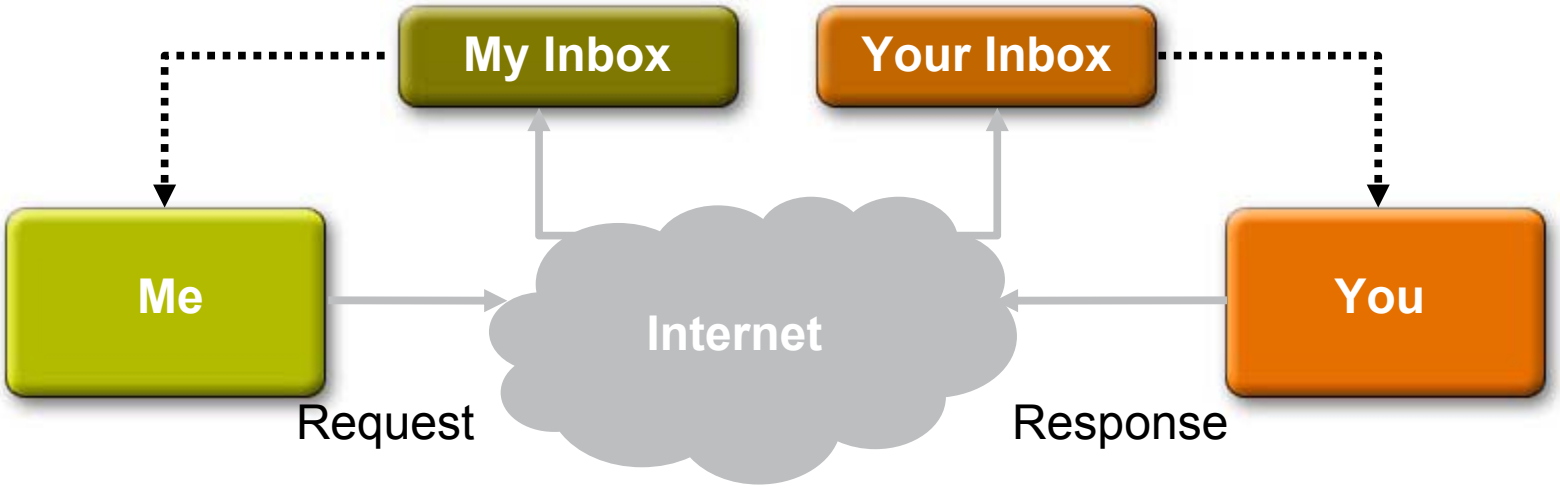

```
<bean id="greeter" class="greeter.GreetingService"/>
<ws:service id="greetingService" bean="greeter"/>

<wsm:smtp service="#greetingService" incoming="#in"
                  outgoing="#out" />
<wsm:sender id="out" host="smtpHost"
              from="server@smtpHost"/>
<wsm:pop3 id="in" host="pop3Host"
              uid="me" password="password" />
```

Configure Client



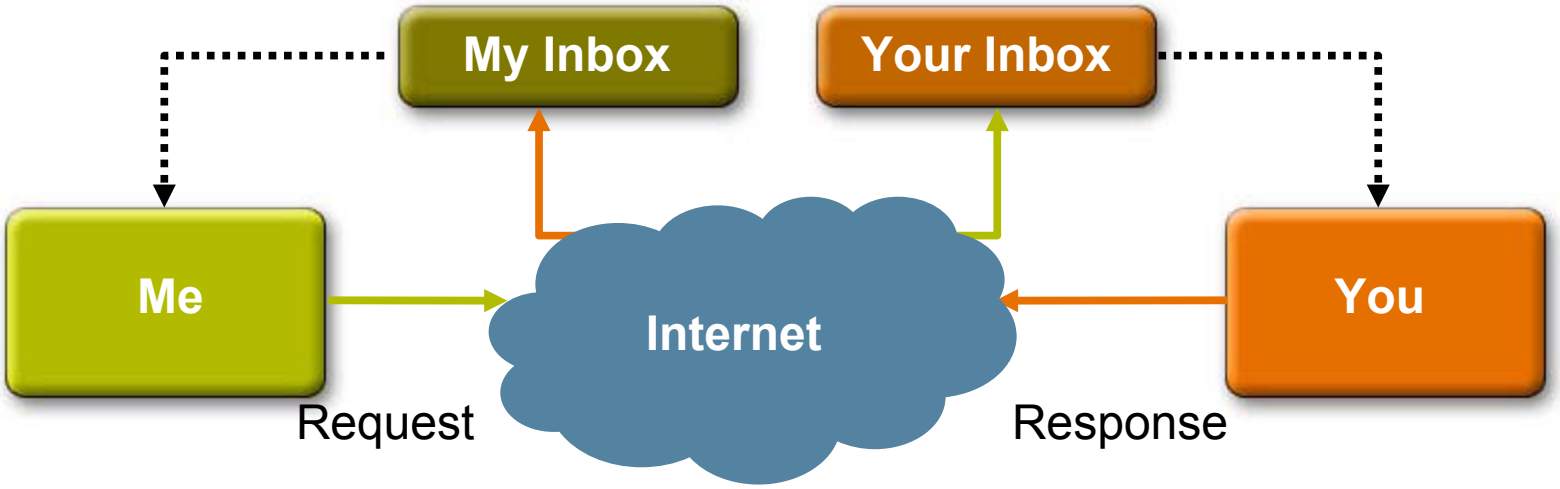

```
feature = new SMTPFeature("smtphost", "me@sun.com");
feature.setPOP3("pop3host","me","password");
```

Configure Client




```
feature = new SMTPFeature("smtphost", "me@sun.com");
feature.setPOP3("pop3host","me","password");
proxy = service.getGreetingServicePort(feature);
bp = (WSBindingProvider)proxy;
bp.setAddress("smtp://you@acme.com");
```

Invoke the Endpoint




```
feature = new SMTPFeature("smtphost", "me@sun.com");
feature.setPOP3("pop3host","me","password");
proxy = service.getGreetingServicePort(feature);
bp = (WSBindingProvider)proxy;
bp.setAddress("smtp://you@acme.com");
proxy.sayHelloTo("jitu");
```

DEMO

JMS Technology

- Works like SMTP
- Uses one queue for request and a temporary queue for the response
- Can be used as standalone as well as with Enterprise JavaBeans™ (EJB™) endpoint

SOAP/TCP

- Efficient transmission of SOAP messages over TCP
- Uses FastInfoset
- Part of WSIT
- Configurable using NetBeans™ software

Agenda

Spring Support

JSON Support

Stateful Web Service

Transports

Additional Headers

Multiple Service Instances

Monitoring and Logging

Server-Side Asynchrony

Additional Headers

- When?
  - SEI doesn't contain additional headers, but you want to send headers
  - Typically authentication headers
- Portable way
  - Use SOAPHandler, but that's hard
- Easier way
  - Use RI's WSBindingProvider

Additional Headers

Example

- Step.1 cast a port to WSBindingProvider

```
HelloPort proxy = ...;
WSBindingProvider bp = (WSBindingProvider)proxy;
```

Additional Headers

Example

- Step.2 create header objects

```
HelloPort proxy = ...;
WSBindingProvider bp = (WSBindingProvider)proxy;

Header h = Headers.create(new QName("hdr"), "mystr");
```

Additional Headers

Example

- Step.3 add them to WSBindingProvider

```
HelloPort proxy = ...;
WSBindingProvider bp = (WSBindingProvider)proxy;

Header h = Headers.create(new QName("hdr"), "mystr");

bp.addOutboundHeaders(h);
```

```
<S:Envelope><S:Header>
  <hdr>mystr</hdr>
```

Additional Headers

Example

- Step.3 add them to WSBindingProvider

```
HelloPort proxy = ...;
WSBindingProvider bp = (WSBindingProvider)proxy;

bp.addOutboundHeaders(
  Headers.create(new QName("hdr1"), "mystr1"),
  Headers.create(new QName("hdr2"), "mystr2")
);
```

```
<S:Envelope><S:Header>
  <hdr1>mystr1</hdr1>
  <hdr2>mystr2</hdr2>
```

Additional Headers

Example

- …there are many different ways to create them

```
HelloPort proxy = ...;
WSBindingProvider bp = (WSBindingProvider)proxy;

@XmlRootElement
class AuthHeader {
  public String username, password;
}
bp.setOutboundHeaders(new AuthHeader("uid","pwd"));
```

```
<S:Envelope><S:Header>
  <authHeader>
    <username>uid</username>
    <password>pwd</password>
  </authHeader>
```

Agenda

Spring Support

JSON Support

Stateful Web Service

Transports

Additional Headers

Multiple Service Instances

Monitoring and Logging

Server-Side Asynchrony

Multiple Service Instances

Traditionally, only one service instance

- Often leads to concurrency problems
- Plus state handling issue

Shopping Cart Service Is Easy!

```
@WebService
public class ShoppingCart {
    List<Item> items = new ArrayList<Item>();

    public void addItem(Item item) {
        items.add(item);
    }
}
```

Doesn't work!

…Or Is It?

```
@WebService
public class ShoppingCart {

    public void addItem(Item item) {

    }
}
```

…Or Is It?

```
@WebService
public class ShoppingCart {
    @Resource
    WebServiceContext wsContext;

    public void addItem(Item item) {
        MessageContext mc = wsContext.getMessageContext();
        HttpServletRequest req = (HttpServletRequest)
            mc.get(MessageContext.SERVLET_REQUEST);
        HttpSession session = req.getSession();




    }
}
```

…Or Is It?

```
@WebService
public class ShoppingCart {
    @Resource
    WebServiceContext wsContext;

    public void addItem(Item item) {
        MessageContext mc = wsContext.getMessageContext();
        HttpServletRequest req = (HttpServletRequest)
            mc.get(MessageContext.SERVLET_REQUEST);
        HttpSession session = req.getSession();

        List<Item> items =
            (List<Item>)session.getAttribute("items");
        if (items == null) {
            items = new ArrayList<Item>();
            session.setAttribute("items", items);
        }
        items.add(item);
    }
}
```

Shouldn't It Be Easy?

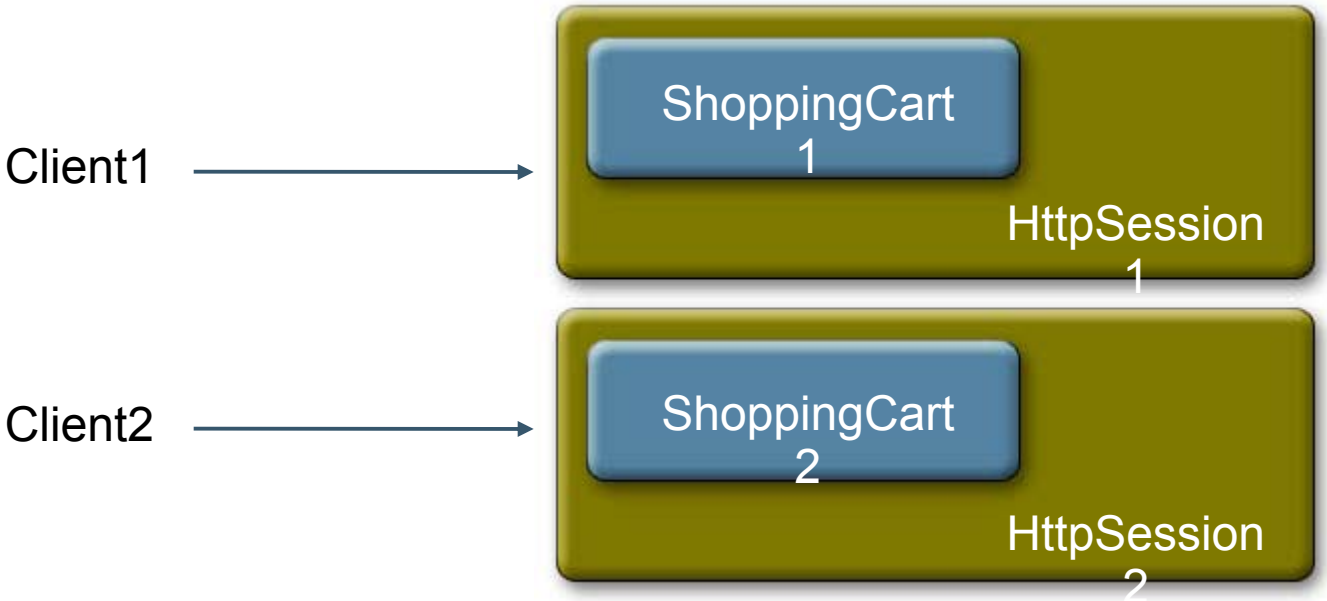
```
@WebService
public class ShoppingCart {
    List<Item> items = new ArrayList<Item>();

    public void addItem(Item item) {
        items.add(item);
    }
}
```

Yes, It Is!

```
@WebService @HttpSessionScope
public class ShoppingCart {
    List<Item> items = new ArrayList<Item>();

    public void addItem(Item item) {
        items.add(item);
    }
}
```

HttpSessionScop

- One service instance per one HttpSession
- No need to store your state to HttpSession
  - Just use instance variables



Resource Contention

```
@WebService
public class DateService {
    DateFormat df = new SimpleDateFormat("MMM d,yyyy");

    public String getDate() {
        return df.format(new Date());
    }
}
```

Doesn't work!

Resource Contention

- ***Multiple requests cause resource contention***

Resource Synchronization

```
@WebService
public class DateService {
    DateFormat df = new SimpleDateFormat("MMM d,yyyy");

    public synchronized String getDate() {
        return df.format(new Date());
    }
}
```

ThreadLocal Resource

```
@WebService
public class DateService {

    private static ThreadLocal<DateFormat> tl =
                        new ThreadLocal<DateFormat>() {
        protected DateFormat initialValue() {
            return new SimpleDateFormat("MMM d,yyyy");
        }
    };

    public String getDate() {
        DateFormat df = tl.get();
        return df.format(new Date());
    }
}
```

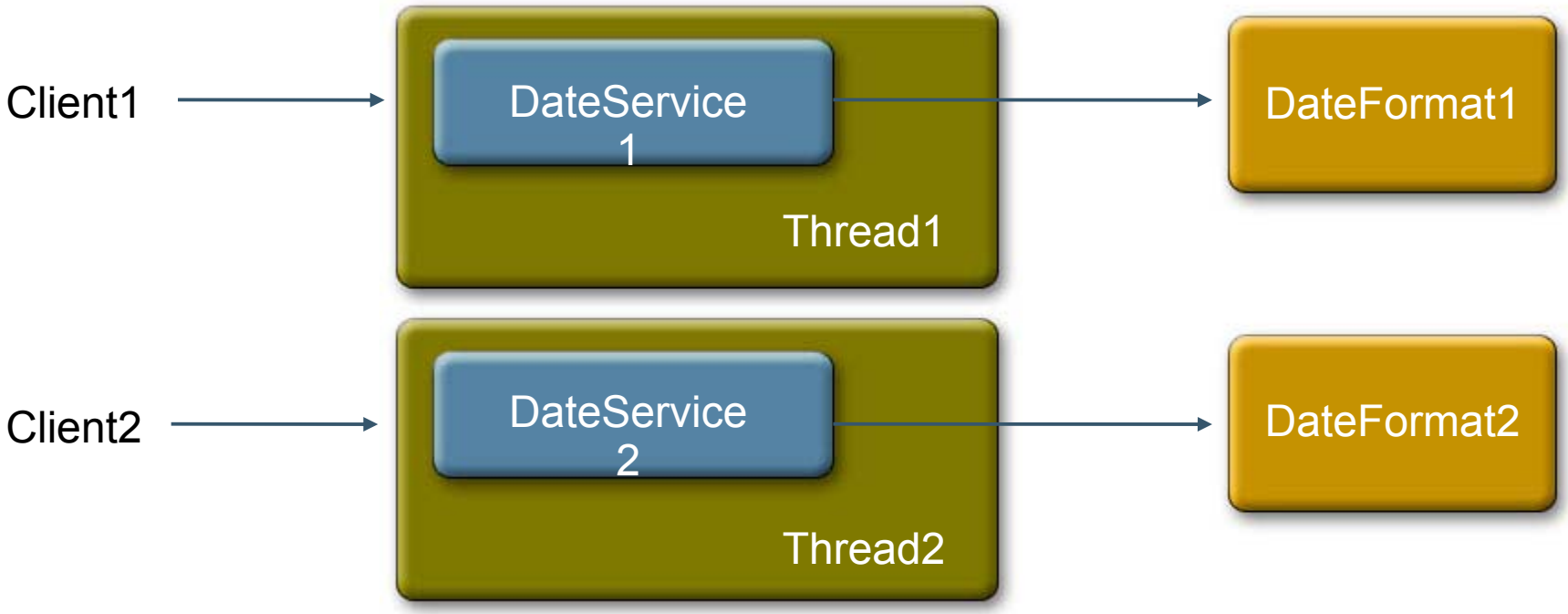
ThreadScope

```
@WebService @ThreadScope
public class DateService {
    DateFormat df = new SimpleDateFormat("MMM d,yyyy");

    public String getDate() {
        return df.format(new Date());
    }
}
```

- No need to worry about synchronization, etc.

ThreadScope

One service instance per thread



Multiple Service Instances

Summary

- RI provides ways to create multiple instances
  - Based on HTTP session, thread, request, etc.
  - Can create instances based on any custom policy

Agenda

Spring Support

JSON Support

Stateful Web Service

Transports

Additional Headers

Multiple Service Instances

Monitoring and Logging

Server-Side Asynchrony

Monitoring and Logging

Transport level logging of SOAP message

- No port forwarding or proxy setup
- Useful for debugging and trouble-shooting

Monitoring and Logging

HTTP traffic

- System properties for client and server transports
  - com.sun.xml.ws.transport.http.client.HttpTransportPipe.dump=true
  - com.sun.xml.ws.transport.http.HttpAdapter.dump=true

```
--[HTTP request]---
SOAPAction: ""
Content-Type: text/xml
<S:Envelope xmlns:S="...">...</S:Envelope>

---[HTTP response 200]---
Date: Thu, 17 Aug 2006 00:35:42 GMT
Content-type: text/xml
<S:Envelope xmlns:S="...">...</S:Envelope>
```

Agenda

Spring Support

JSON Support

Stateful Web Service

Transports

Additional Headers

Multiple Service Instances

Monitoring and Logging

Server-Side Asynchrony

Problem

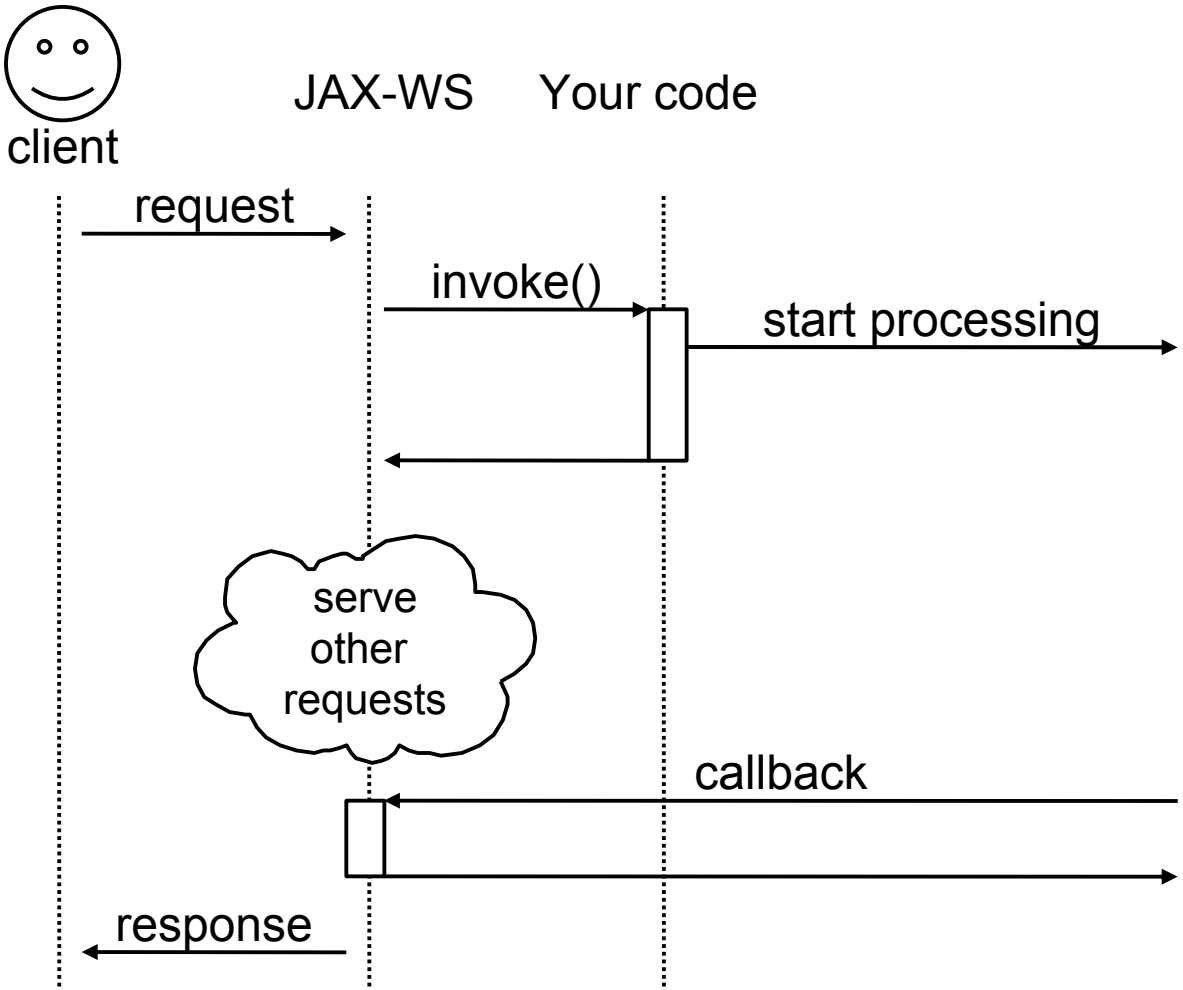

```
@WebService
class BrokeringService {
    public Bean foo(Bean request) {
        sendRequestToBackend(request);
        // Backend takes some time to give response
        return waitForResponse();
    }
}
```

Your service won't scale!

Solution: AsyncProvider

- Don’t return a response
- Instead, call back later with response

AsyncProvider Sequence Diagram



AsyncProvider Details

```
@WebServiceProvider
class MyService implements AsyncProvider<Source> {
    public void invoke(Source request,
        AsyncProviderCallback<Source> callback,
        WebServiceContext context) {

        sendRequestToBackend(request,callback);
    }
}

// later
callback.send(response);
```

So Much to Offer

- JAX-WS API
  - Standard
  - Portable
- JAX-WS RI
  - Compatibility built-in from day 1
  - Battle-tested
    - Project GlassFish™, BEA WebLogic 10, TmaxSoft JEUS6, Java Development Kit (JDK™) 6
  - Lots of productivity features
  - Fast!
- WSIT
- Vibrant community

Summary

- JAX-WS RI is **much** more than JAX-WS API
- Many useful extensions are already there
- Use them!

For More Information

- If you are interested in writing extensions
  - BOF-8034: Extending and Embedding JAX-WS 2.1 RI
- http://jax-ws.dev.java.net
- http://jax-ws-commons.dev.java.net
- http://forums.java.net/jive/forum.jspa?forumID=46
- users@jax-ws.dev.java.net
- http://glassfish.dev.java.net
- http://wsit.dev.java.net

Q&A

http://jax-ws.dev.java.net/

We Can Do Better!

```
public EndpointReference findAccount(int id) {
    Account a = new Account(id);
    return Account.manager.export(a);
}

Account a = bank.findAccount(0).getPort(Account.class);
```

We Can Do Better!

```
public Account findAccount(int id) {
    Account a = new Account(id);
    return a;
}

Account a = bank.findAccount(0).getPort(Account.class);
```

We Can Do Better!

```
public Account findAccount(int id) {
    Account a = new Account(id);
    return a;
}

Account a = bank.findAccount(0).getPort(Account.class);
```

We Can Do Better!

```
public Account findAccount(int id) {
    Account a = new Account(id);
    return a;
}

Account a = bank.findAccount(0);
```

Unleashing the Power of JAX-WS RI: Spring, Stateful Web Services, SMTP, and More

Kohsuke Kawaguchi
Jitendra Kotamraju
Vivek Pandey

Sun Microsystems, Inc.
http://jax-ws.dev.java.net

TS-4948