



# Module 25: Spring Web Services

CS544: Enterprise Architecture

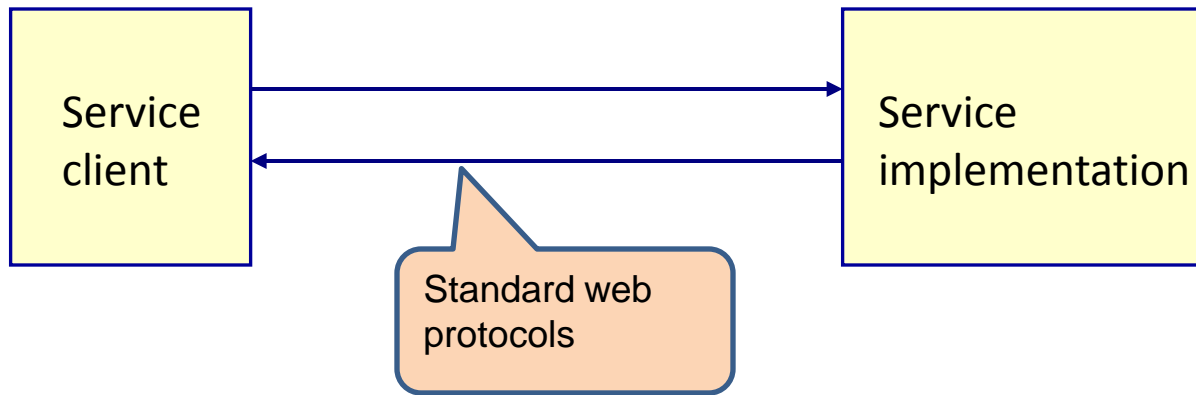


Spring Web Services:

# **BASICS OF WEBSERVICES**



# What is a Web Service?

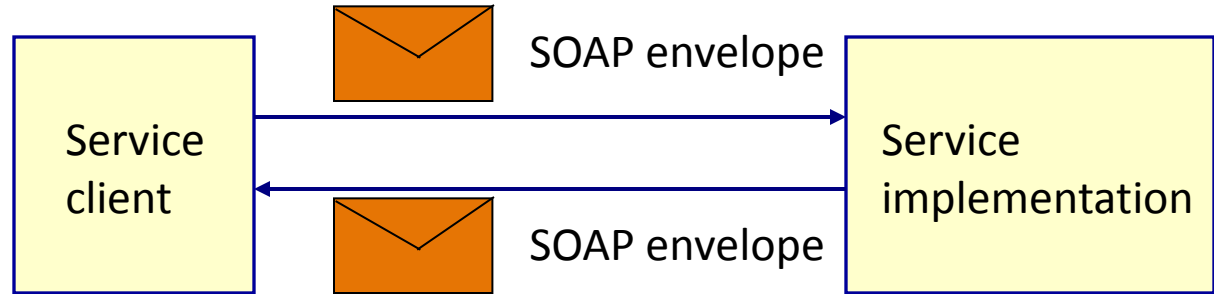


- A web service offers functionality that can be called by other clients using standard web protocols (SOAP, XML, HTTP)

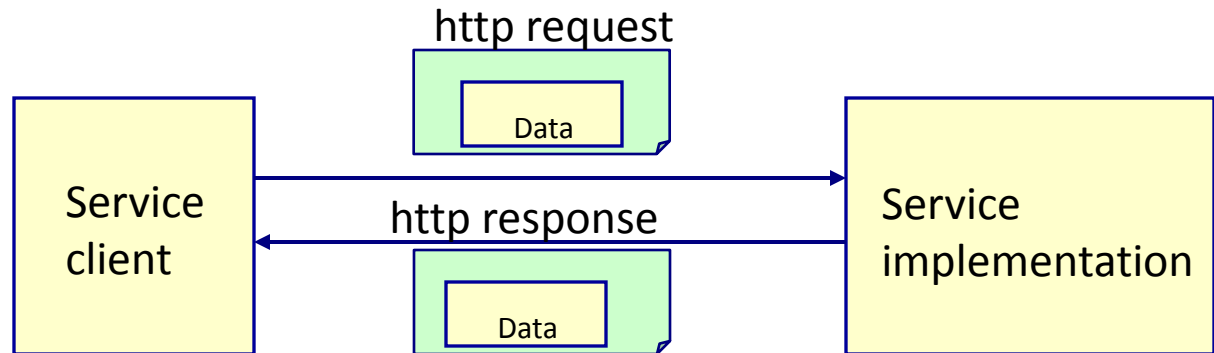


# Types of Web Services

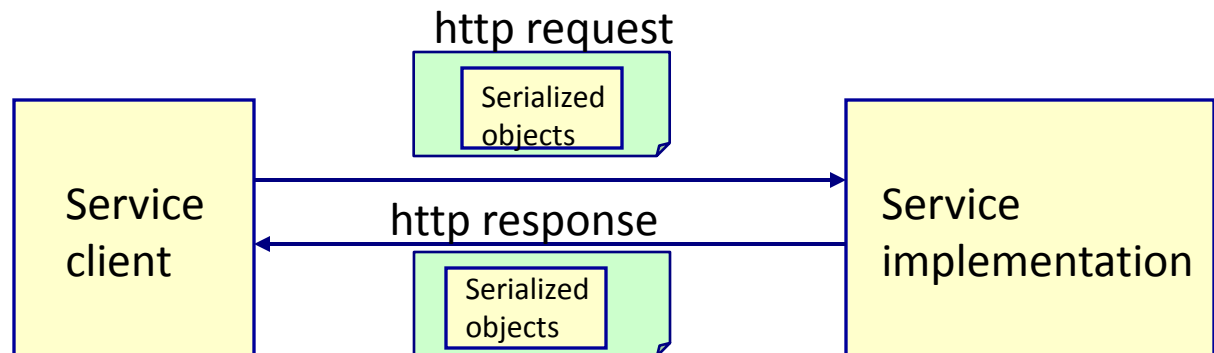
- SOAP



- REST

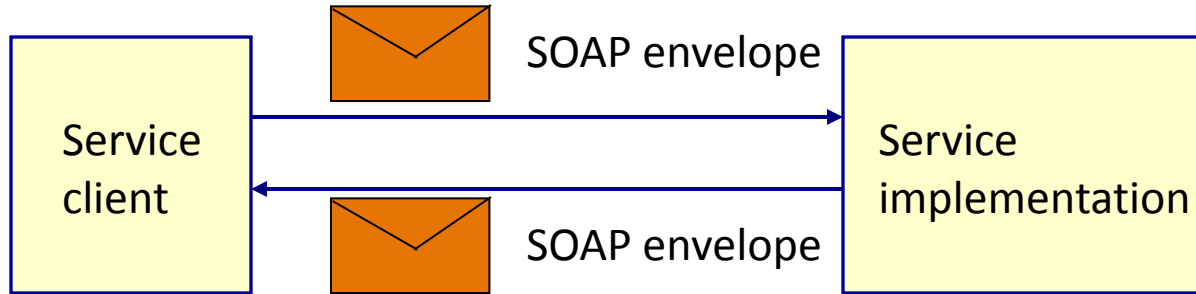


- Serialized objects





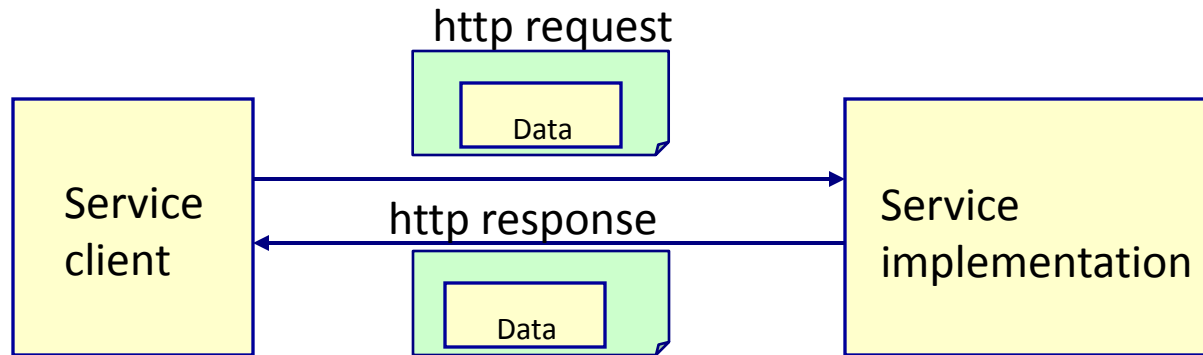
# SOAP Web Services



- WSDL interface description
- Contract first vs contract last
- SOAP frameworks
  - Axis2
  - CXF
  - Spring-WS



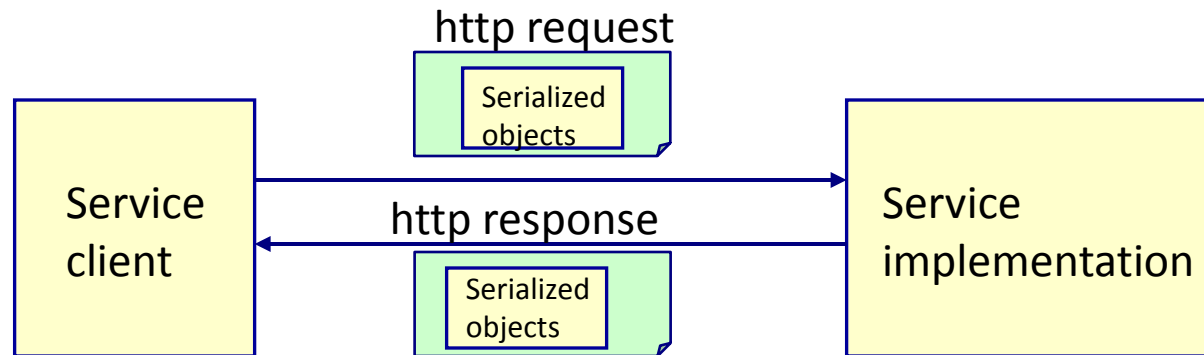
# RESTful Web Services



- Data in HTTP messages
  - GET message for retrieving data
  - POST message for creating data
  - PUT message for updating data
  - DELETE message for deleting data



# Serialized objects



- If the client and server are both Java
- Sending serialized object is faster than sending XML
- Like RMI over HTTP



# Examples of Web Services with Spring

---

- This module is divided into the following sections:
  1. Spring REST support
  2. Spring-WS SOAP webservices
  3. Spring HTTPInvoker





Spring Web Services:

# **SPRING REST WEBSERVICES**



# RESTful Web Services

---

- RESTful Web Services are closely tied to the HTTP protocol
  - Not bound to a specific data format
  - The URL specifies the resource to act on
  - The HTTP method specifies the action type
    - GET method for retrieving data
    - POST method for creating data
    - PUT method for updating data
    - DELETE method for deleting data



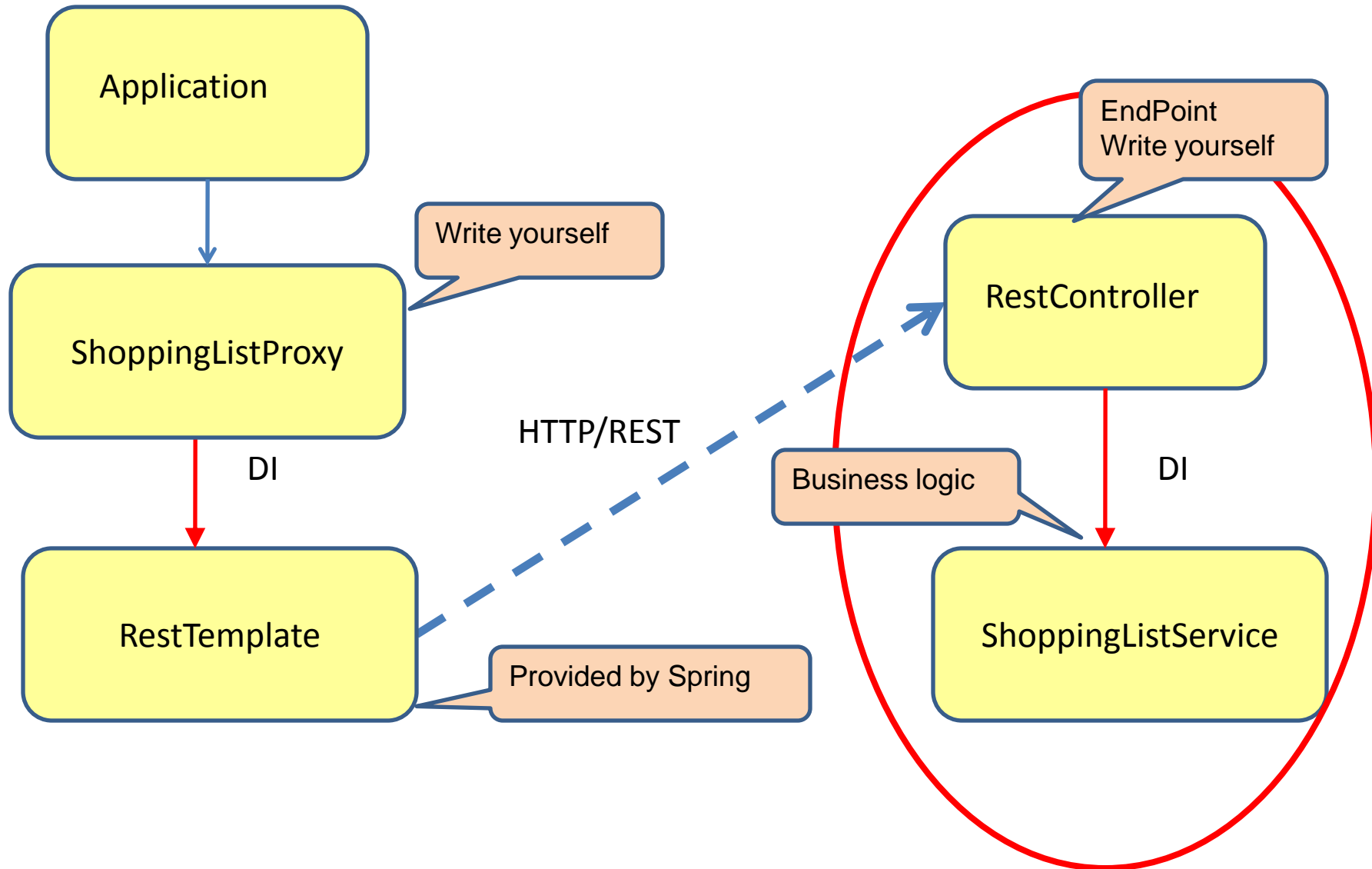
# Shopping List Example

---

- Shopping List RESTful Web Service
  - Chosen to use XML as our data format
  - GET /list returns entire shopping list
  - POST /list to add an item to the list
  - GET /item/{product} returns item details
  - PUT /item/{product} to update an item
  - DELETE /item/{product} to delete an item
- Where {product} is a variable string, e.g. /item/Tomatoes or /item/Avacados



# The Server





# JAXB O/X Mapping

---

- We will use JAXB again for our O/X mapping
- This gives us roughly the same 3 steps as our previous Spring WS example
  1. Write example XML messages
  2. Generate Java based on this XML
  3. Write the Web Service Endpoint, a Service Implementation and a Spring configuration



# 1: Example XML Messages

## ■ Sample ShoppingList:

```
<?xml version="1.0" encoding="UTF-8"?>
<shopping-list xmlns="http://springtraining/shopping-list">
  <item qty="1" notes="Either organic or non-organic">Lemons</item>
  <item qty="3" notes="Organic is better">Tomatoes</item>
  <item qty="2" notes="Both black and green">Olives</item>
  <item qty="2" notes="Not too ripe">Avocados</item>
</shopping-list>
```

## ■ Sample Item:

```
<?xml version="1.0" encoding="UTF-8"?>
<item qty="3" notes="Organic is better">Tomatoes</item>
```



## 2: Generated ShoppingList

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {"item"})
@XmlRootElement(name = "shopping-list")
public class ShoppingList {
```

JAXB annotations

Collection of Items is its only attribute.

Attribute name has to be "item" for JAXB to create the correct tags name

```
    protected Collection<Item> item;
```

```
    public ShoppingList() {
    }
```

```
    public ShoppingList(Collection<Item> item) {
        this.item = item;
    }
```

Added convenience constructor

```
    public Collection<Item> getItem() {
        if (item == null) {
            item = new ArrayList<Item>();
        }
        return this.item;
    }
```

Added a toString method for easy printing of the list

```
    public String toString() {
        String result = "";
        for (Item itm : item) {
            result += String.format("%2d %-20s %-20s\n", itm.getQty(), itm.getProduct(),
                                    itm.getNotes());
        }
        return result;
    }
}
```



# Generated Item Class

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {"product"})
@XmlRootElement(name = "item")
public class Item {
```

JAXB annotations

```
    @XmlValue
    protected String product;
    @XmlAttribute
    protected Integer qty;
    @XmlAttribute
    protected String notes;
```

Item has three attributes: product, qty and notes

```
    public Item() {
    }
```

Default constructor and an added convenience constructor

```
    public Item(String product, Integer qty, String notes) {
        this.product = product;
        this.qty = qty;
        this.notes = notes;
    }
```

```
    ...
```





# Item Class Continued

```
public String getProduct() {  
    return product;  
}  
  
public void setProduct(String product) {  
    this.product = product;  
}  
  
public Integer getQty() {  
    return qty;  
}  
  
public void setQty(Integer value) {  
    this.qty = value;  
}  
  
public String getNotes() {  
    return notes;  
}  
  
public void setNotes(String value) {  
    this.notes = value;  
}  
}
```

Generated Getters and Setter



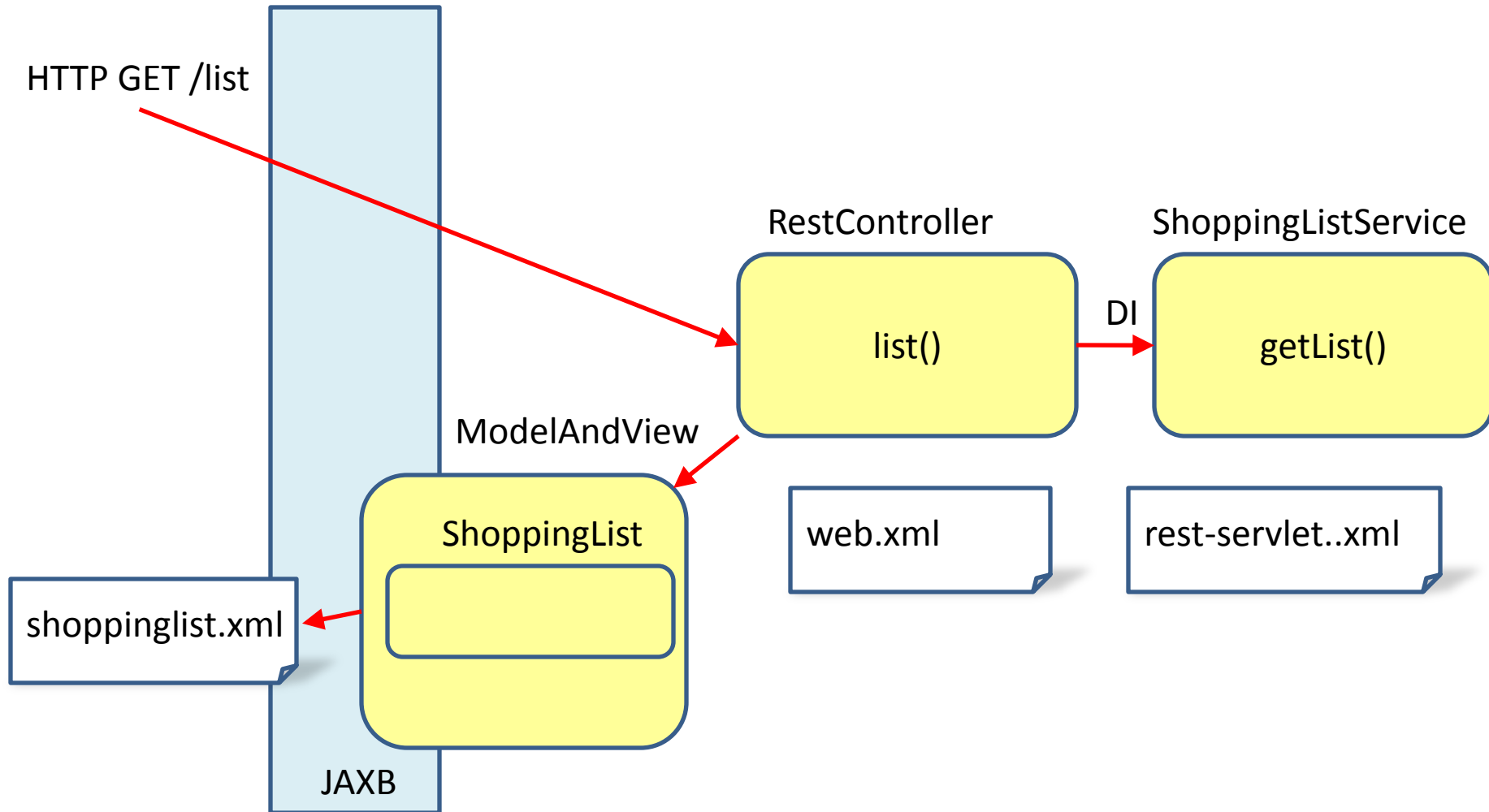
# 3: Implementation

---

- The Service implementation has 3 parts
  1. The shopping list service class
  2. The Web Service endpoint
  3. Spring and web configurations
  
- The Service class implements the business logic
- The Web Service endpoint provides the RESTful interface to this logic
- The configurations bind it all together

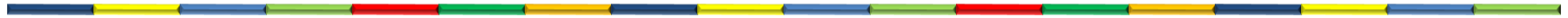


# HTTP GET /list

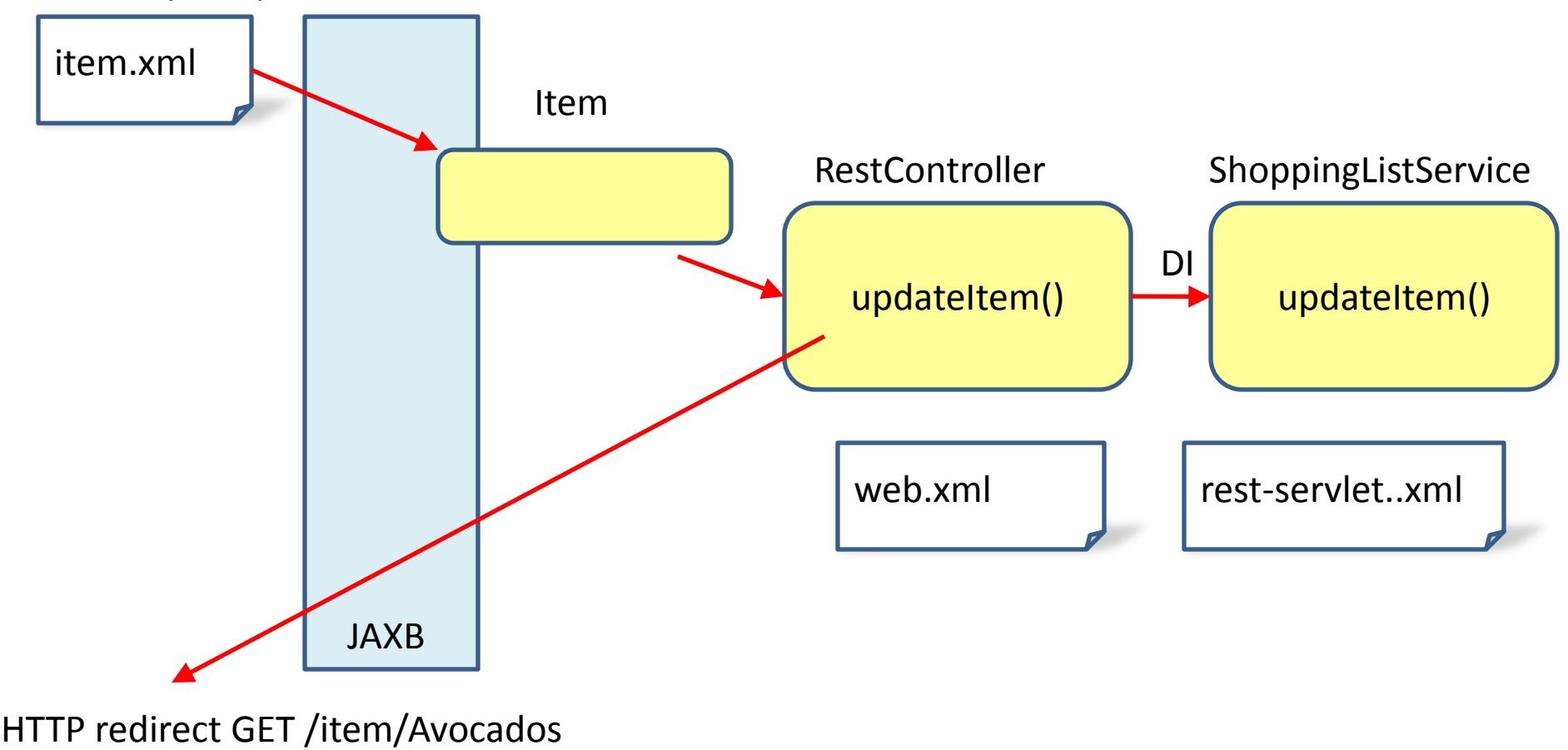




# HTTP PUT /item/Avocados



HTTP PUT /item/Avocados





# Shopping ListService

```
public class ShoppingListService implements IShoppingListService {  
    private Map<String, Item> items = new HashMap<String, Item>();  
  
    public ShoppingList getList() {  
        return new ShoppingList(items.values());  
    }  
  
    public Item getItem(String product) {  
        return items.get(product);  
    }  
  
    public void addToList(Item item) {  
        if (items.containsKey(item.getProduct())) {  
            Item current = items.get(item.getProduct());  
            current.setQty(current.getQty() + item.getQty());  
            if (!current.getNotes().equals(item.getNotes())) {  
                current.setNotes(current.getNotes() + "\n" + item.getNotes());  
            }  
        } else {  
            items.put(item.getProduct(), item);  
        }  
    }  
  
    public void removeFromList(String product) {  
        items.remove(product);  
    }  
  
    public void updateItem(Item item) {  
        items.put(item.getProduct(), item);  
    }  
}
```

Map to hold shoppingList data



# IShoppingListService

```
public interface IShoppingListService {  
  
    public ShoppingList getList();  
    public Item getItem(String product);  
    public void addToList(Item item);  
    public void removeFromList(String product);  
    public void updateItem(Item item);  
  
}
```



# RestController (endpoint)

@Controller

MVC @Controller annotation

```
public class RestController {  
    private ShoppingListService shoppingListService;  
    public void setShoppingService(ShoppingListService shoppingListService) {  
        this.shoppingListService = shoppingListService;  
    }  
}
```

DI ShoppingListService

@RequestMapping for GET

```
@RequestMapping(value = "/list", method = RequestMethod.GET)  
public ModelAndView list() {  
    ModelAndView mav = new ModelAndView();  
    mav.setViewName("marshalview");  
    mav.addObject("list", shoppingListService.getList());  
    return mav;  
}
```

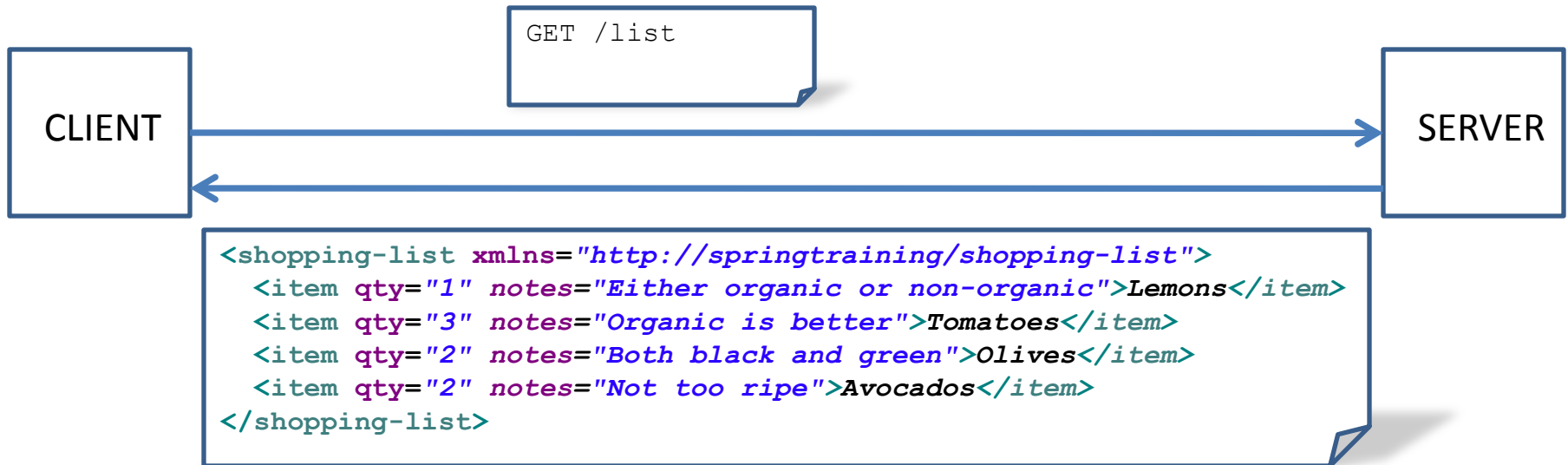
Show view  
"marshalview"

Put ShoppingList in Model

...



# GET method for all items



## @RequestMapping for GET

```
@RequestMapping(value = "/list", method = RequestMethod.GET)
public ModelAndView list() {
    ModelAndView mav = new ModelAndView();
    mav.setViewName("marshalview");
    mav.addObject("list", shoppingListService.getList());
    return mav;
}
```

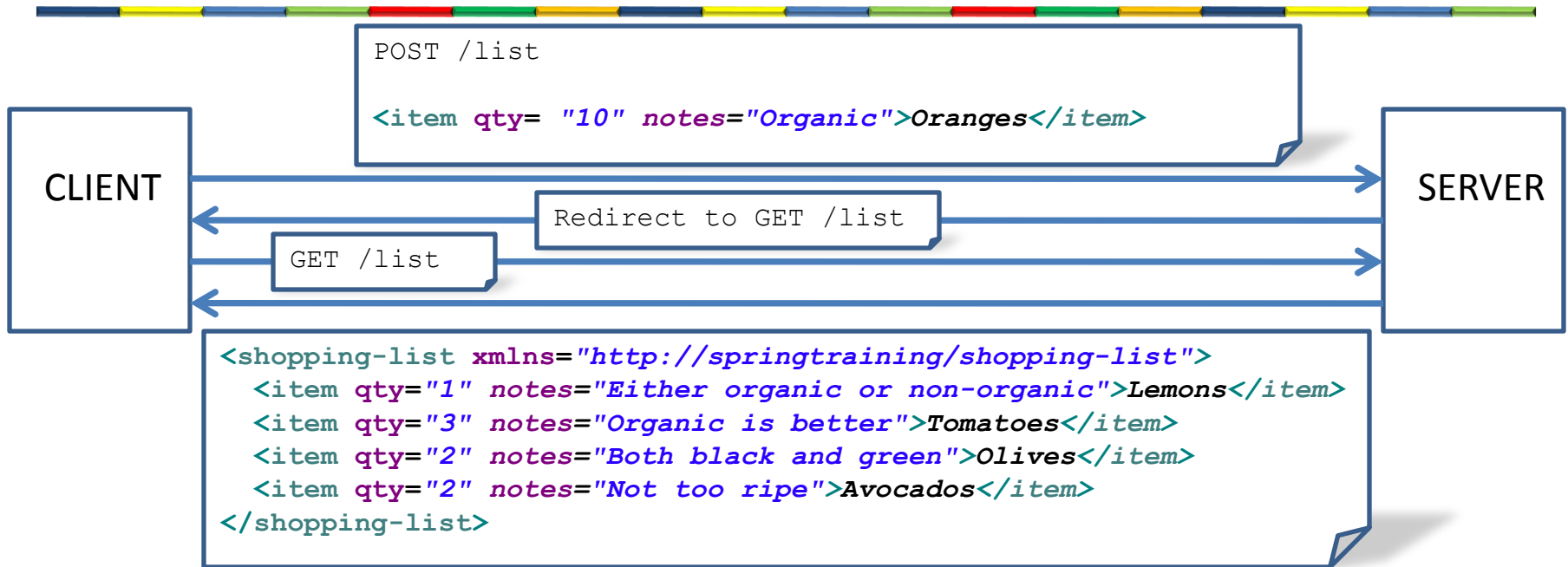
Show view "marshalview"

Put ShoppingList in Model





# POST method



## @RequestMapping for POST

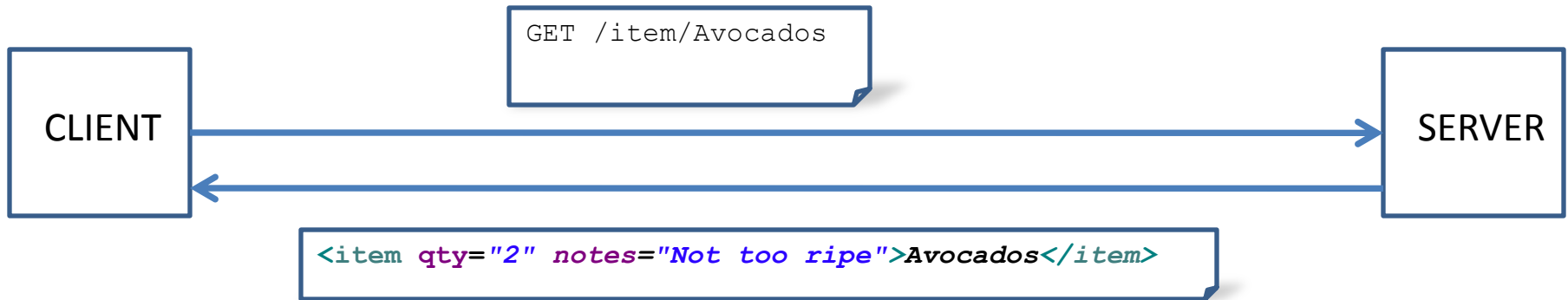
```
@RequestMapping(value = "/list", method = RequestMethod.POST)
public RedirectView addItem(@RequestBody Item item) {
    shoppingListService.addToList(item);
    return new RedirectView("list");
}
```

Add Item to list

Redirect back to GET "list"



# GET method for 1 item



## @RequestMapping for GET

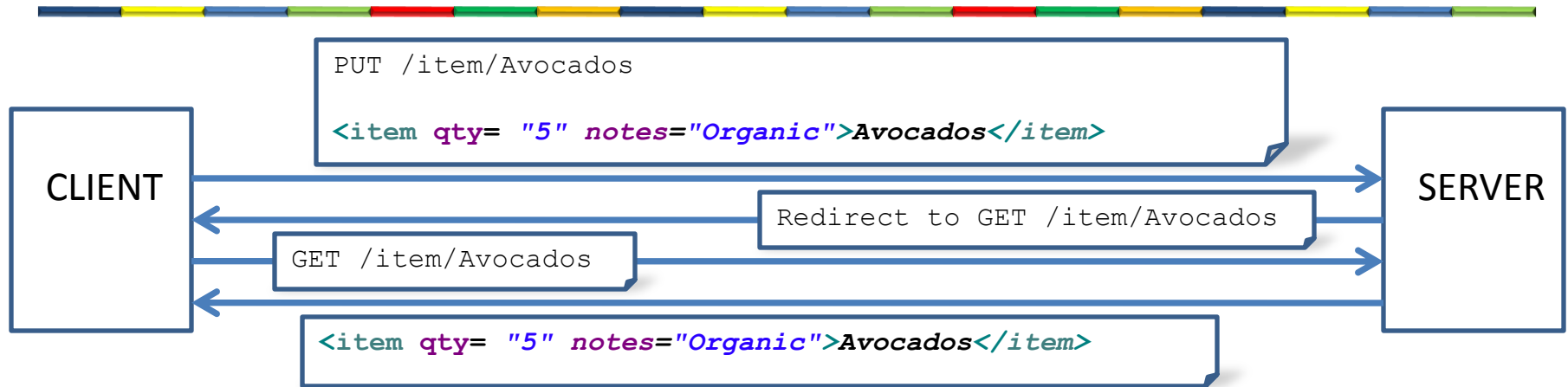
```
@RequestMapping(value = "/item/{product}*", method = RequestMethod.GET)
public ModelAndView item(@PathVariable("product") String product) {
    ModelAndView mav = new ModelAndView();
    mav.setViewName("marshalview");
    Item item = shoppingListService.getItem(product);
    if (item != null) {
        mav.addObject("item", item);
    }
    return mav;
}
```

Set "marshalview" view

Add requested item to the model



# PUT method



## @RequestMapping for PUT

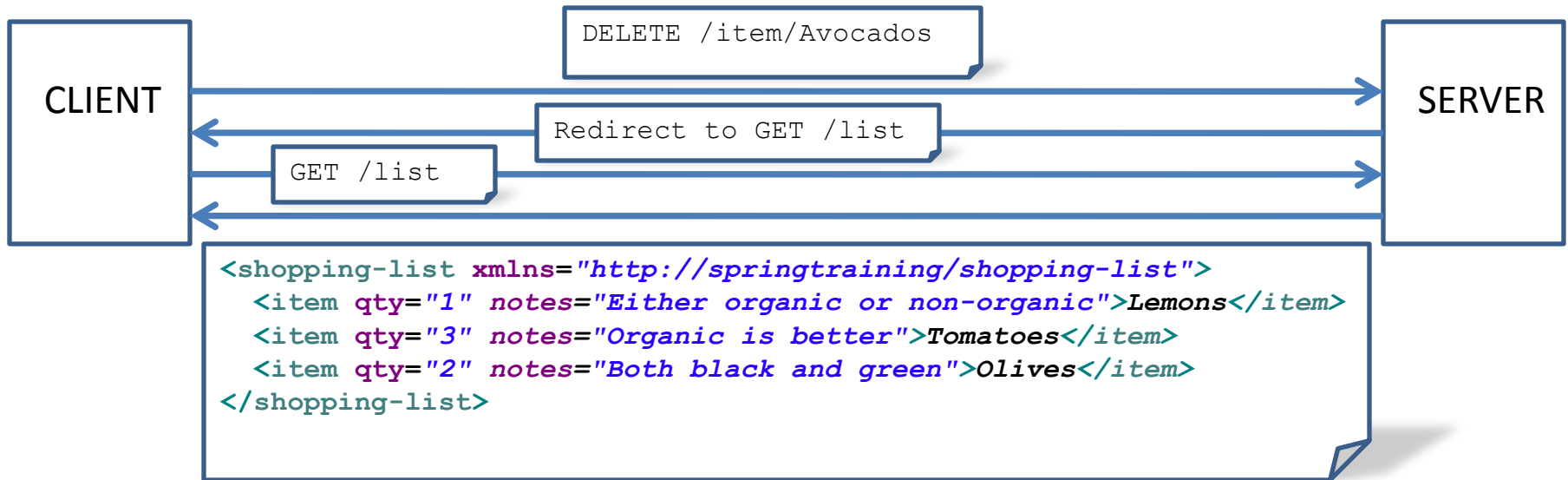
```
@RequestMapping(value = "/item/{product}", method = RequestMethod.PUT)
public RedirectView updateItem(@RequestBody Item item) {
    shoppingListService.updateItem(item);
    return new RedirectView(item.getProduct());
}
```

Update item at this url

Redirect back to GET this item



# DELETE method



## @RequestMapping for DELETE

```
@RequestMapping(value = "/item/{product}", method = RequestMethod.DELETE)
public RedirectView deleteItem(@PathVariable("product") String product) {
    shoppingListService.removeFromList(product);
    return new RedirectView("../list");
}
```

Remove item at this url

Redirect back to GET the list

../list is a relative url, pointing one level lower (away from the item directory) to the list



# web.xml

```
<web-app ... version="2.5">

  <display-name>module13Example-REST</display-name>

  <servlet>
    <servlet-name>rest</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>rest</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>

</web-app>
```



# rest-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:util="http://www.springframework.org/schema/util"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan base-package="demo"/>

    <bean id="shoppingListService" class="demo.ShoppingListService"/>

    <bean id="restController" class="demo.RestController">
        <property name="shoppingListService" ref="shoppingListService"/>
    </bean>

    <bean class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

    <bean id="marshaller" class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
        <property name="contextPath" value="generated"/>
    </bean>

    ...
</beans>
```

Scan all files in the demo package for classes with @Controller annotations and map their @RequestMapping

DI the shoppingListService into our restController

Resolves view names to beans

The JAXB O/X mapping framework



# rest-servlet.xml Continued

```
<bean id="marshallingHttpMessageConverter"  
      class="org.springframework.http.converter.xml.MarshallingHttpMessageConverter">  
  <property name="marshaller" ref="marshaller"/>  
  <property name="unmarshaller" ref="marshaller"/>  
</bean>
```

Can Converts http messages into Objects using the marshaller

```
<bean  
  class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">  
  <property name="messageConverters">  
    <util:list id="beanList">  
      <ref bean="marshallingHttpMessageConverter"/>  
    </util:list>  
  </property>  
</bean>
```

Applies http converter to incoming messages

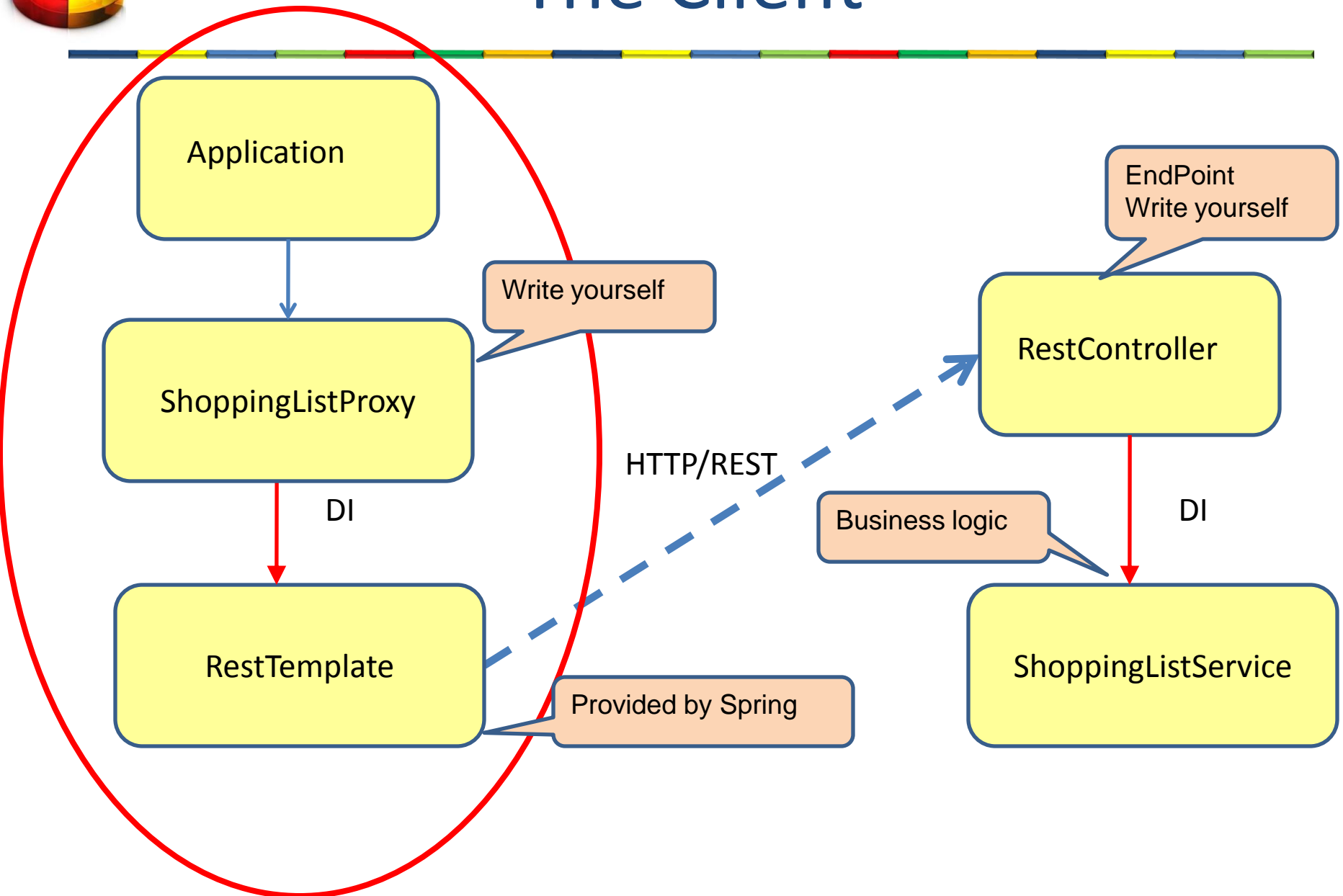
```
<bean id="marshalview" class="org.springframework.web.servlet.view.xml.MarshallingView">  
  <property name="contentType" value="text/xml"/>  
  <property name="marshaller" ref="marshaller"/>  
</bean>
```

The "marchalview" view bean is an object marshaller using JAXB

```
</beans>
```



# The Client





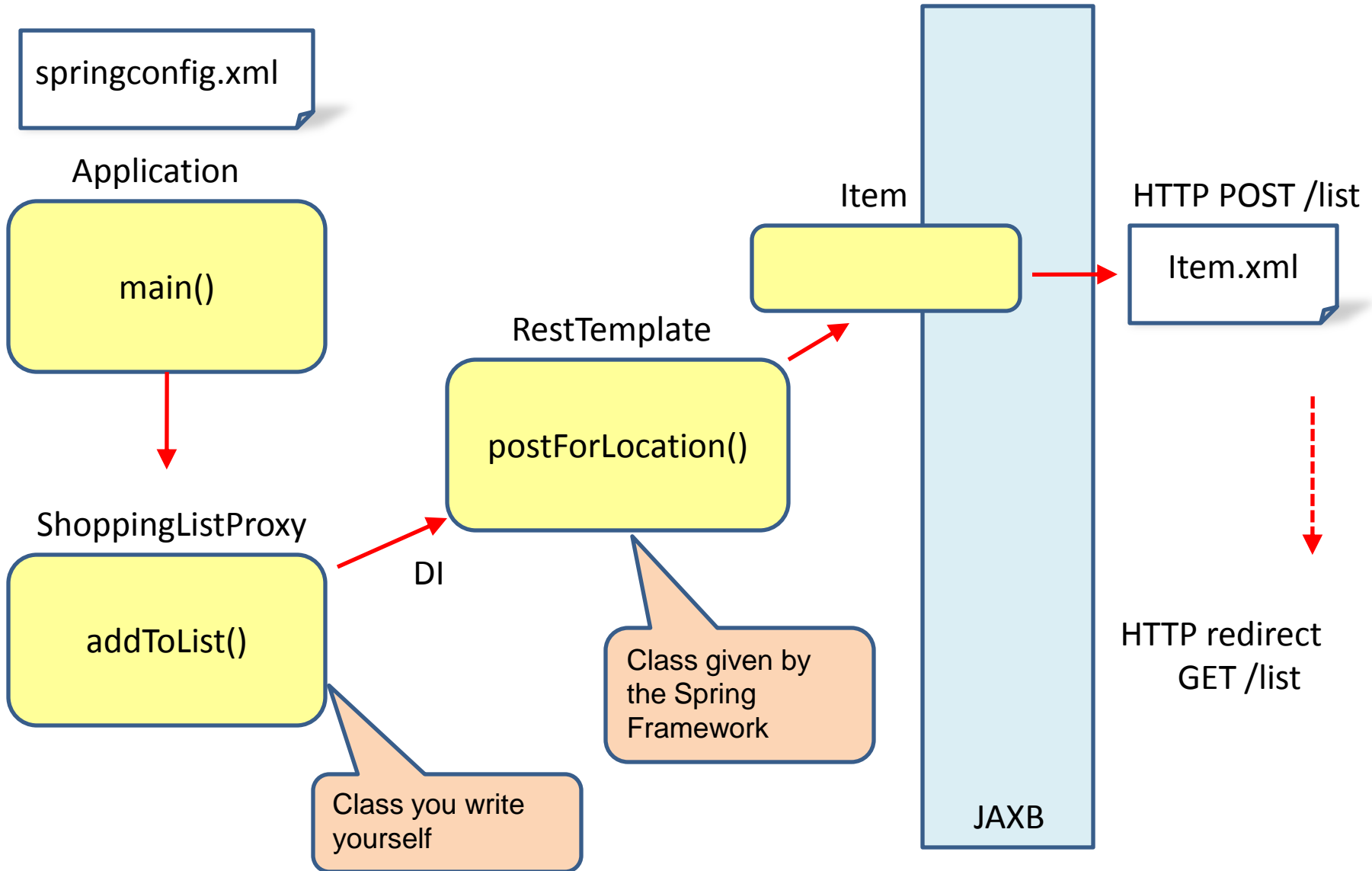


# RestTemplate methods

HTTP	Method	Description
DELETE	<code>delete()</code>	Delete the resources at the specified URI
GET	<code>getForObject()</code>	Retrieve a representation by doing a GET
POST	<code>postForLocation</code>	Create a new resource by POSTing the given object
PUT	<code>put</code>	Creates or updates the resource at the given location by PUTting the given object

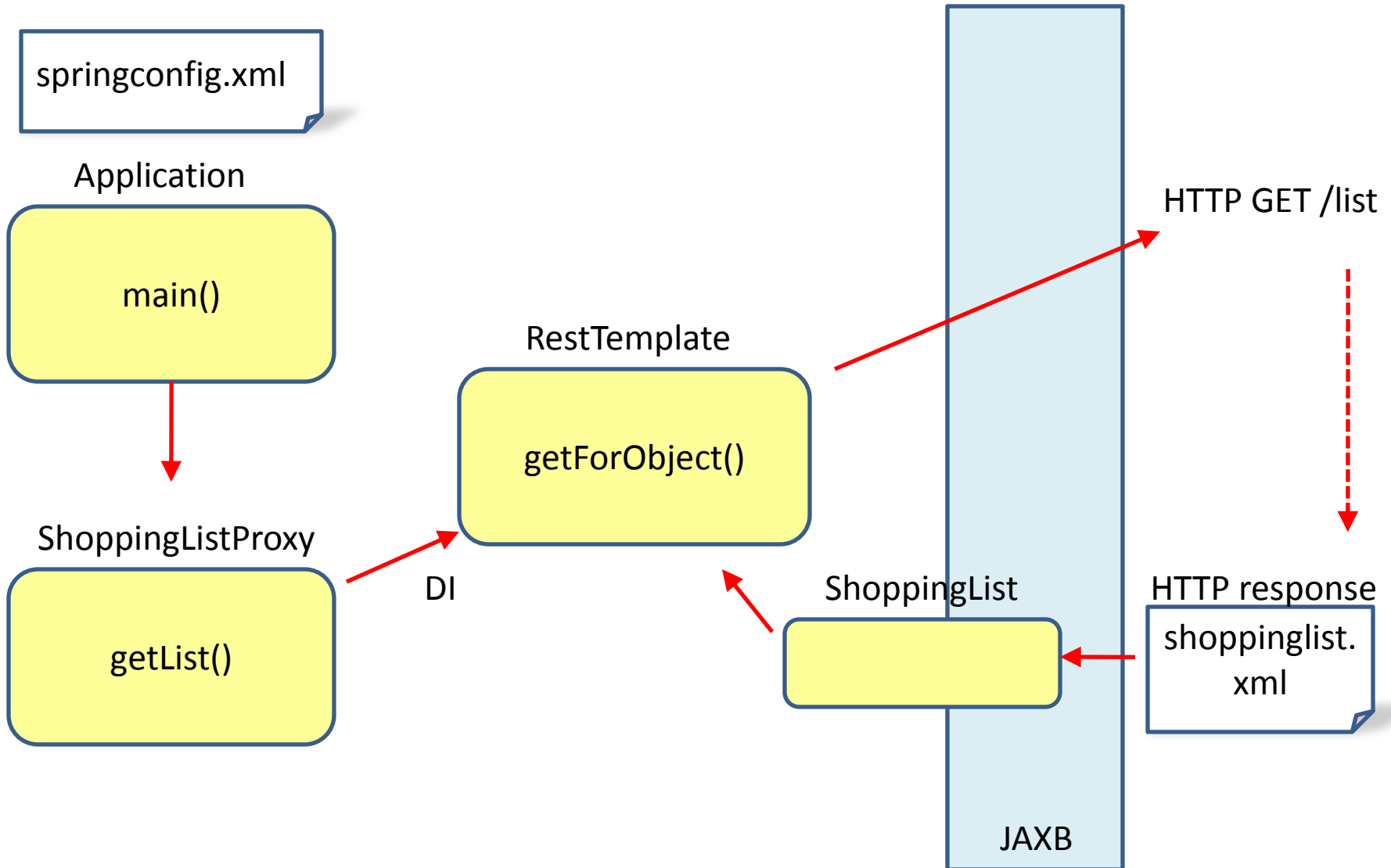


# Client addToList()





# Client getList()





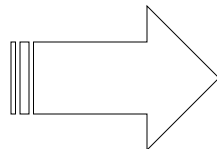
# REST Client

```
public class Application {  
  
    public static void main(String[] args) {  
        ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");  
        IShoppingListService remoteService = context.getBean("shopListProxy",  
                                                             IShoppingListService.class);  
  
        Item tomato = new Item("Tomatoes", 3, "Prefer Organic");  
        Item avocado = new Item("Avocados", 3, "Organic or non-organic");  
        remoteService.addToList(tomato);  
        remoteService.addToList(avocado);  
        System.out.println(remoteService.getList());  
  
        tomato.setQty(5);  
        remoteService.updateItem(tomato);  
        System.out.println(remoteService.getList());  
  
        remoteService.removeFromList("Avocados");  
        System.out.println(remoteService.getList());  
    }  
}
```

Add Tomatoes  
and Avocados

Update Tomatoes

Remove Avocados



3 Avocados	Organic or non-organic
3 Tomatoes	Prefer Organic
3 Avocados	Organic or non-organic
5 Tomatoes	Prefer Organic
5 Tomatoes	Prefer Organic



# ShopListProxy

```
public class ShopListProxy implements IShoppingListService {  
    private static final String listURL = "http://localhost:8080/REST/rest/list";  
    private static final String itemURL = "http://localhost:8080/REST/rest/item/{product}";  
  
    private RestTemplate restTemplate;  
    public void setRestTemplate(RestTemplate restTemplate) {  
        this.restTemplate = restTemplate; DI restTemplate  
    }  
  
    public void addToList(Item item) {  
        restTemplate.postForLocation(listURL, item); POST item to listURL  
    }  
    public Item getItem(String product) {  
        return restTemplate.getForObject(itemURL, Item.class, product); GET item from itemURL  
    }  
    public ShoppingList getList() {  
        return restTemplate.getForObject(listURL, ShoppingList.class); GET list from listURL  
    }  
    public void removeFromList(String product) {  
        restTemplate.delete(itemURL, product); DELETE item from itemURL  
    }  
    public void updateItem(Item item) {  
        restTemplate.put(itemURL, item, item.getProduct()); UPDATE item to itemURL  
    }  
}
```



# springconfig.xml

```
<beans ... >

  <bean id="shopListProxy" class="demo.ShopListProxy">
    <property name="restTemplate" ref="restTemplate"/>
  </bean>

  <bean id="restTemplate" class="org.springframework.web.client.RestTemplate">
    <property name="messageConverters">
      <list>
        <bean
          class="org.springframework.http.converter.xml.MarshallingHttpMessageConverter">
            <property name="marshaller" ref="marshaller"/>
            <property name="unmarshaller" ref="marshaller"/>
          </bean>
        </list>
      </property>
    </bean>

    <bean id="marshaller" class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
      <property name="contextPath" value="generated"/>
    </bean>

  </beans>
```



Module 25: Spring webservices

# **SPRING SOAP WEBSERVICES**



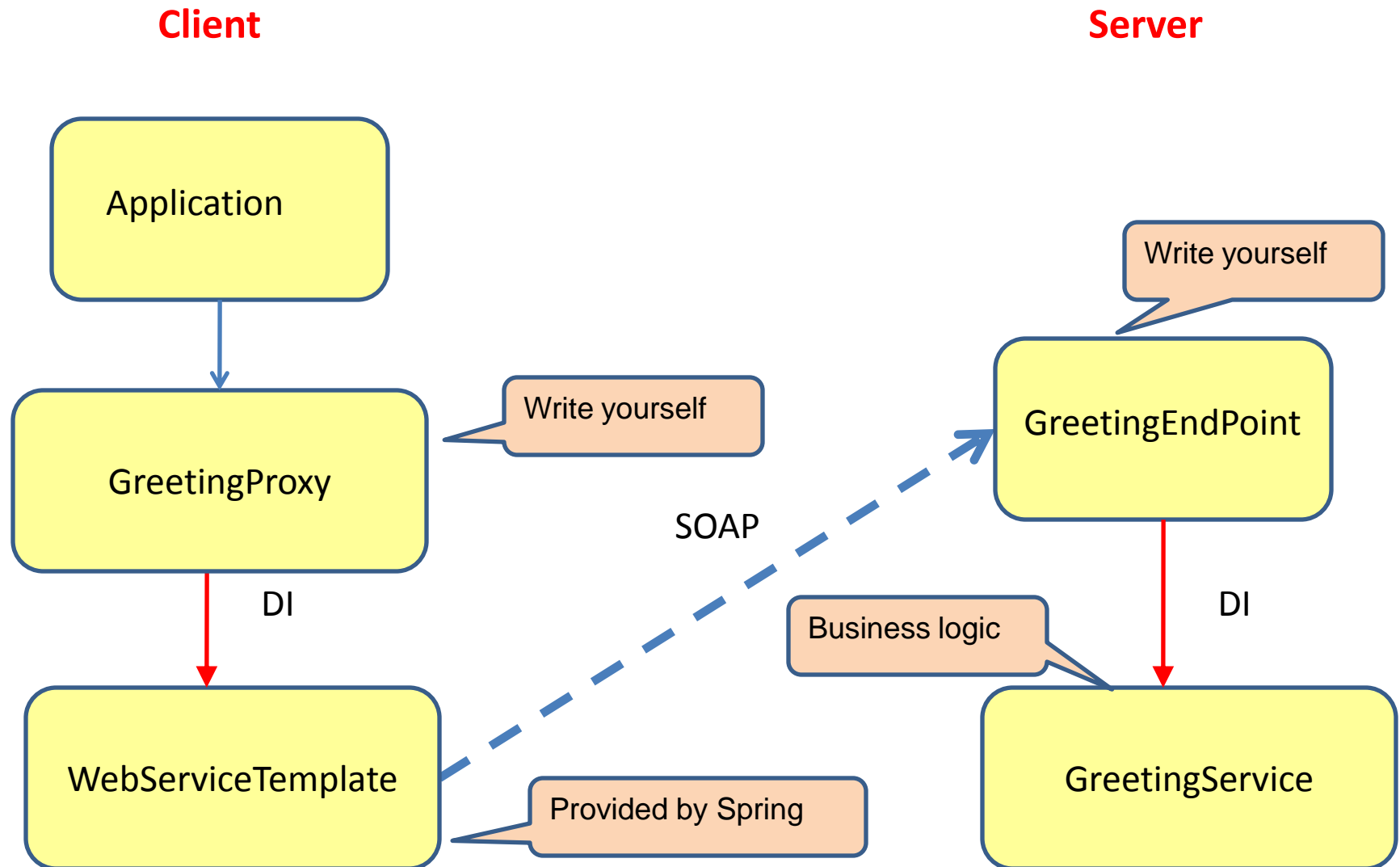
# Spring WS

- In a contract first approach we need to:
  1. Create a WSDL file for our web service
  2. Create implementation based on the WSDL
- Spring WS can automatically generate WSDL from an XML schema definition
- We can generate Java classes from an XML schema for our implementation



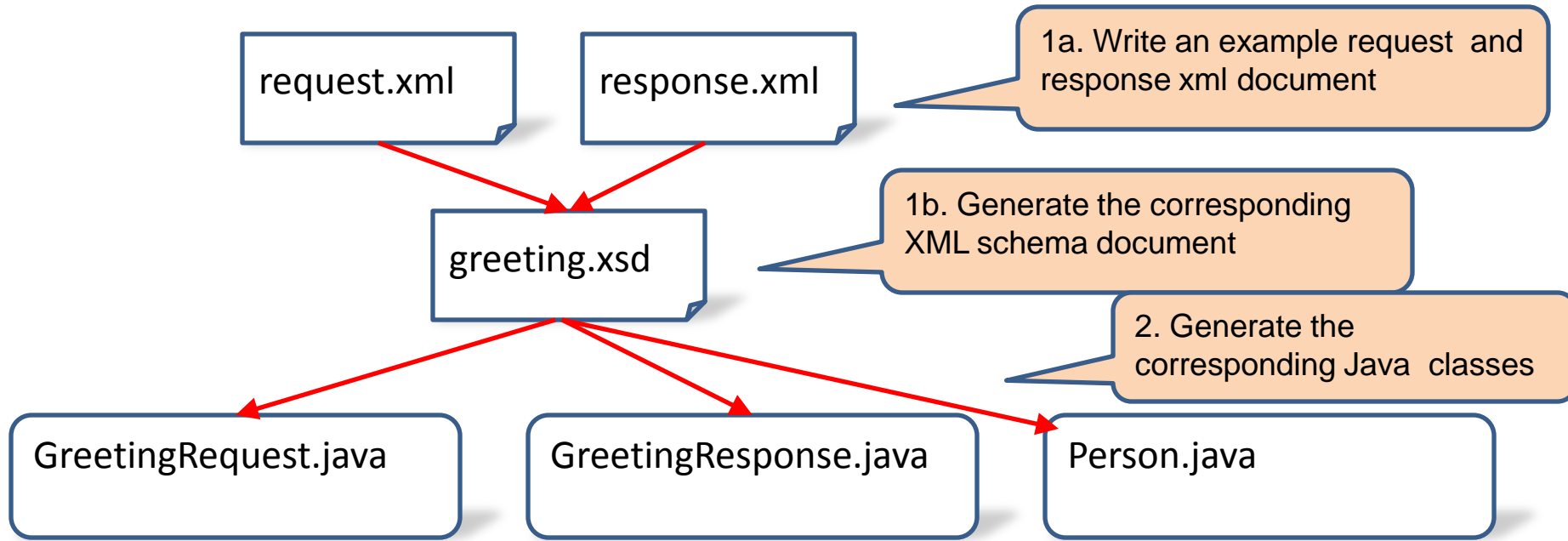


# Spring WS





# Creating a Spring-WS server



```
@Endpoint  
public class GreetingEndpoint {  
    private IGreeting greetingService;
```

```
public class GreetingService  
    implements IGreeting {
```

web.xml

greeting-servlet.xml

3. Write the EndPoint, the Service implementation, web.xml and the Spring configuration file



# Step 1a: Schema Creation

- The easiest way to create an XML schema is to infer it from sample documents.
- We have two sample documents
  1. The XML message that we sent to web service

```
<?xml version="1.0" encoding="UTF-8"?>
<GreetingRequest xmlns="http://springtraining/greeting">
  <Person>
    <FirstName>John</FirstName>
    <LastName>Doe</LastName>
  </Person>
</GreetingRequest>
```

2. The XML message that the web service returns

```
<?xml version="1.0" encoding="UTF-8"?>
<GreetingResponse xmlns="http://springtraining/greeting">
  <Greeting>Hello John Doe!</Greeting>
</GreetingResponse>
```



# Step 1b: Greeting.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  targetNamespace="http://springtraining/greeting"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="GreetingRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Person">
          <xs:complexType>
            <xs:all>
              <xs:element type="xs:string" name="FirstName" />
              <xs:element type="xs:string" name="LastName" />
            </xs:all>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="GreetingResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="Greeting" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Request message schema

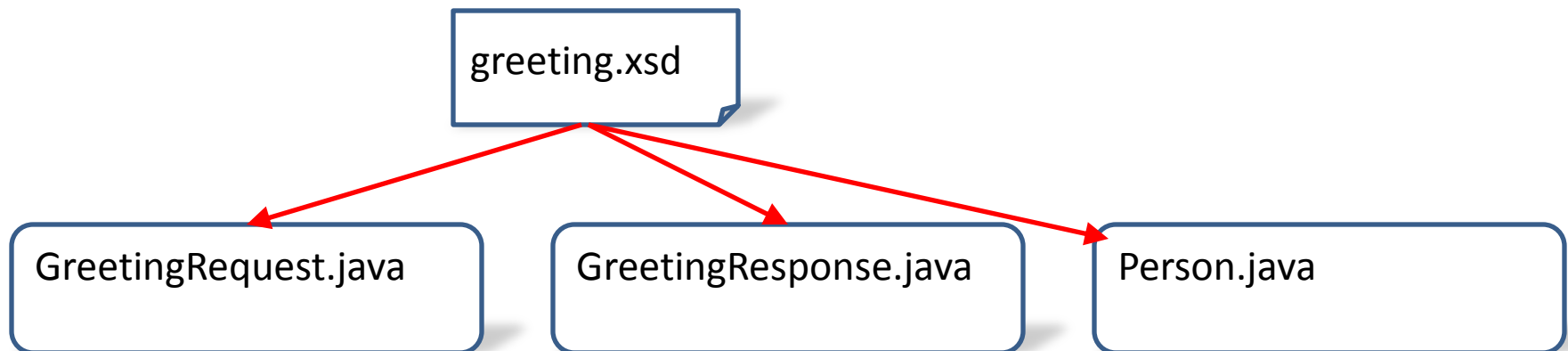
Contained elements may be in any sequence

Response message schema



## Step 2: Generated Code

- We can generate Java classes based on this schema definition.
  - The generated classes will have Java Architecture for XML Binding (JAXB) annotations
  - JAXB is an Object/XML (O/X) mapping framework
  - O/X mapping frameworks automate the conversion of Objects to and from XML





# GreetingRequest

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {"person"})
@XmlRootElement(name = "GreetingRequest")
public class GreetingRequest {

    @XmlElement(name = "Person", required = true)
    protected Person person;

    public Person getPerson() {
        return person;
    }

    public void setPerson(Person value) {
        this.person = value;
    }
}
```

Class JAXB annotations

Attribute JAXB annotations

Single person attribute

greeting.xsd



GreetingRequest.java



# Person

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {})
public class Person {
    @XmlElement(name = "FirstName", required = true)
    protected String firstName;
    @XmlElement(name = "LastName", required = true)
    protected String lastName;

    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String value) {
        this.firstName = value;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String value) {
        this.lastName = value;
    }
}
```

JAXB class annotations

firstName and lastName  
attributes with JAXB  
annotations

Very similar to our  
previous Person class

greeting.xsd



Person.java



# GreetingResponse

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {"greeting"})
@XmlRootElement(name = "GreetingResponse")
public class GreetingResponse {

    @XmlElement(name = "Greeting", required = true)
    protected String greeting;

    public String getGreeting() {
        return greeting;
    }

    public void setGreeting(String value) {
        this.greeting = value;
    }
}
```

GreetingResponse class  
with generated JAXB

Response greeting String

greeting.xsd

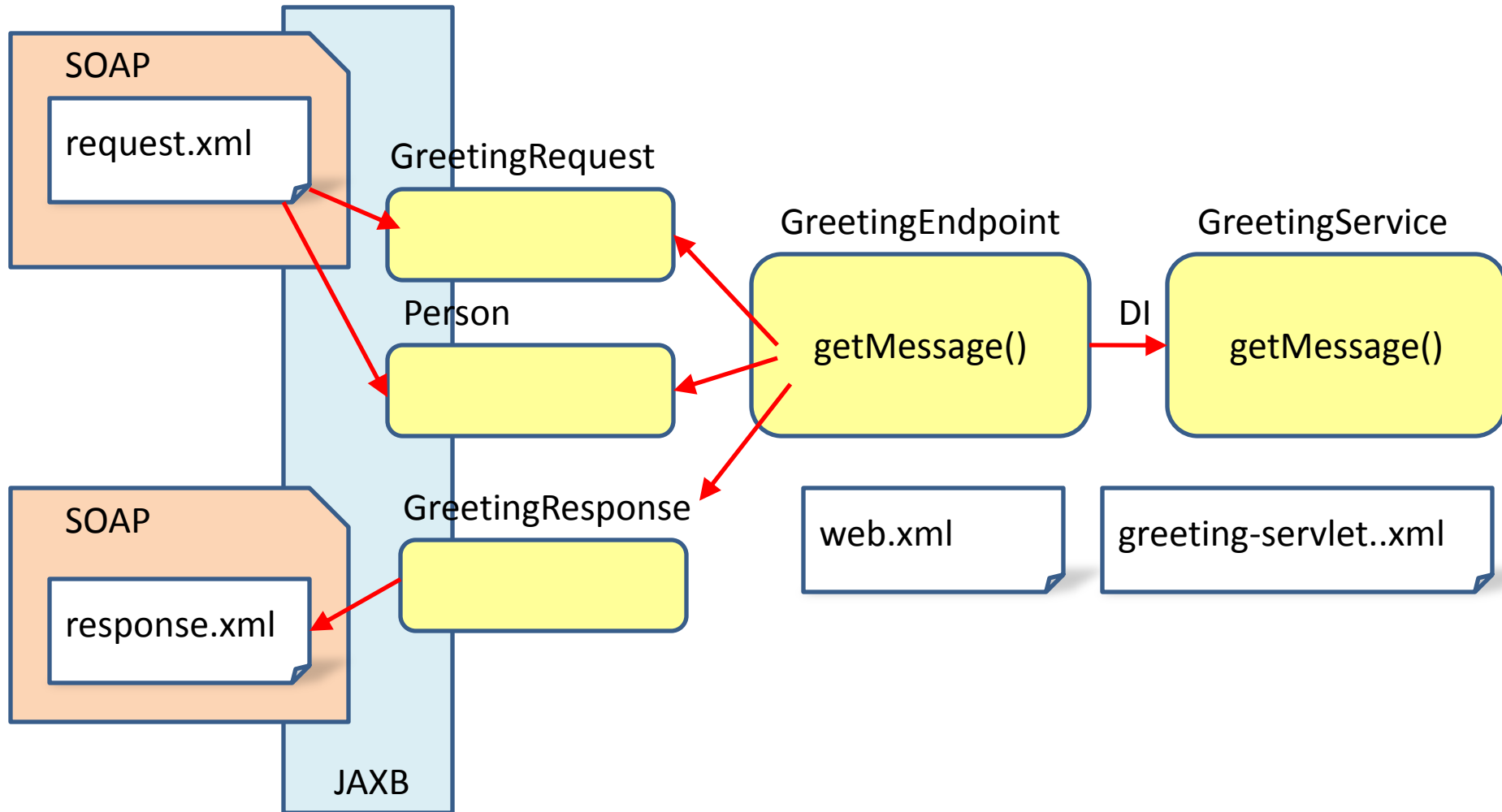


GreetingResponse.java





# Step 3: writing the Webservice implementation





# GreetingEndpoint Implementation

---

```
@Endpoint
public class GreetingEndpoint {

    private IGreeting greetingService;

    public GreetingEndpoint(IGreeting greetingService) {
        this.greetingService = greetingService;
    }

    @PayloadRoot(localPart = "GreetingRequest", namespace =
        "http://springtraining/greeting")
    public GreetingResponse getMessage(GreetingRequest request) {
        GreetingResponse response = new GreetingResponse();
        response.setGreeting(greetingService.getMessage(request.getPerson()));
        return response;
    }
}
```

**@Endpoint**

Normal POJO

DI greetingService

@PayloadRoot, specifies this method as a web service endpoint

Business logic is delegated, method itself just handles request / response



# GreetingService

```
package demo;
```

```
public class GreetingService implements IGreeting {  
    private String greeting;
```

```
    public void setGreeting(String greeting) {  
        this.greeting = greeting;  
    }
```

```
    public String getMessage(Person person) {  
        return greeting + " " + person.getFirstName() + " " + person.getLastName();  
    }  
}
```

Same GreetingService as  
in previous examples

Dependency Injected  
greeting string



# web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">

  <display-name>module13Example-SpringWS</display-name>

  <servlet>
    <servlet-name>greeting</servlet-name>
    <servlet-class>
      org.springframework.ws.transport.http.MessageDispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>greeting</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>

</web-app>
```

Spring Message  
Dispatcher Servlet

Mapped to all incoming requests



# greeting-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

  <bean id="greetingEndpoint" class="demo.GreetingEndpoint">
    <constructor-arg ref="greetingService"/>
  </bean>

  <bean id="greetingService" class="demo.GreetingService" >
    <property name="greeting" value="Hello"/>
  </bean>

  ...
```

greetingEndpoint uses  
DI for greetingService

greetingService bean  
with DI greeting string



# greeting-servlet.xml continued

```
<bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">
  <property name="messageFactory">
    <bean class="com.sun.xml.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_1Impl" />
  </property>
</bean>
```

Indicates annotated endpoints

messageFactory for sending and receiving XML messages

```
<bean
class="org.springframework.ws.server.endpoint.mapping.PayloadRootAnnotationMethodEndpointMapping"
/>
```

```
<bean
class="org.springframework.ws.server.endpoint.adapter.GenericMarshallingMethodEndpointAdapter">
  <constructor-arg ref="marshaller"/>
</bean>
```

Configures O/X Mapping

```
<bean id="marshaller" class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
  <property name="contextPath" value="generated"/>
</bean>
```

```
<bean id="greeting" class="org.springframework.ws.wsd11.DefaultWsd11Definition">
  <property name="schema" value="schema" />
  <property name="portTypeName" value="Greeting" />
  <property name="locationUri" value="/greeting" />
  <property name="targetNamespace" value="http://springtraining/greeting" />
</bean>
```

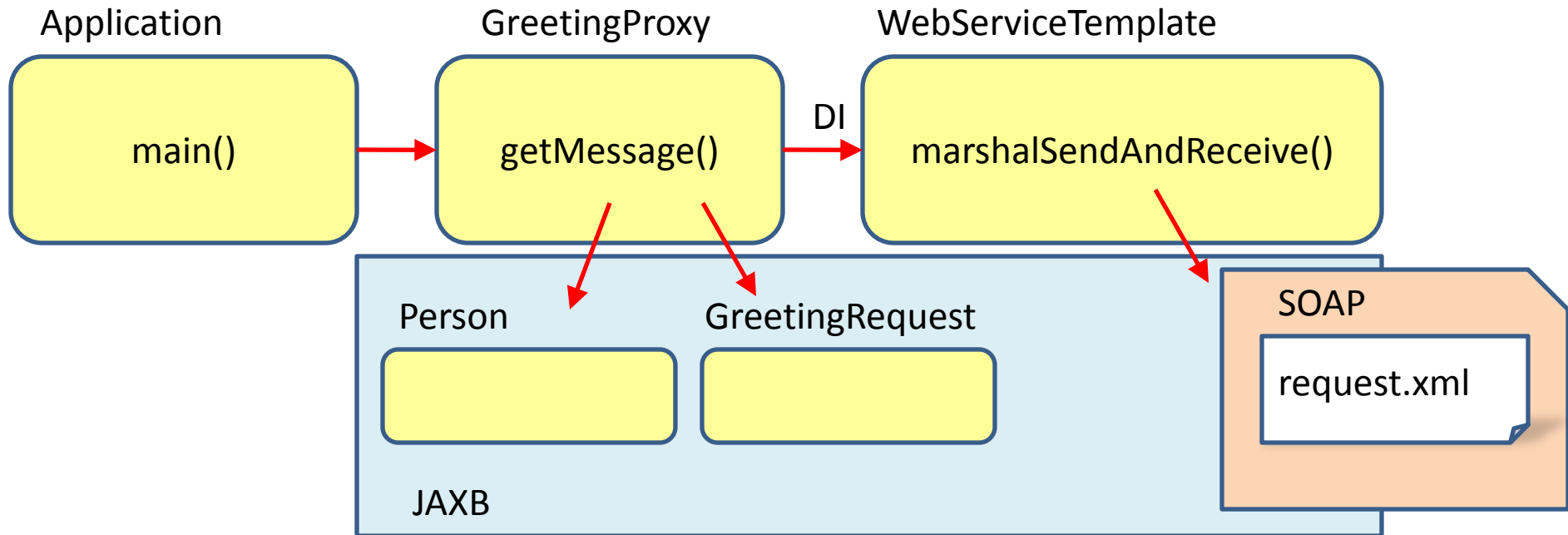
WSDL Generation from schema

```
<bean id="schema" class="org.springframework.xml.xsd.SimpleXsdSchema">
  <property name="xsd" value="/WEB-INF/greeting.xsd" />
</bean>
```

```
</beans>
```

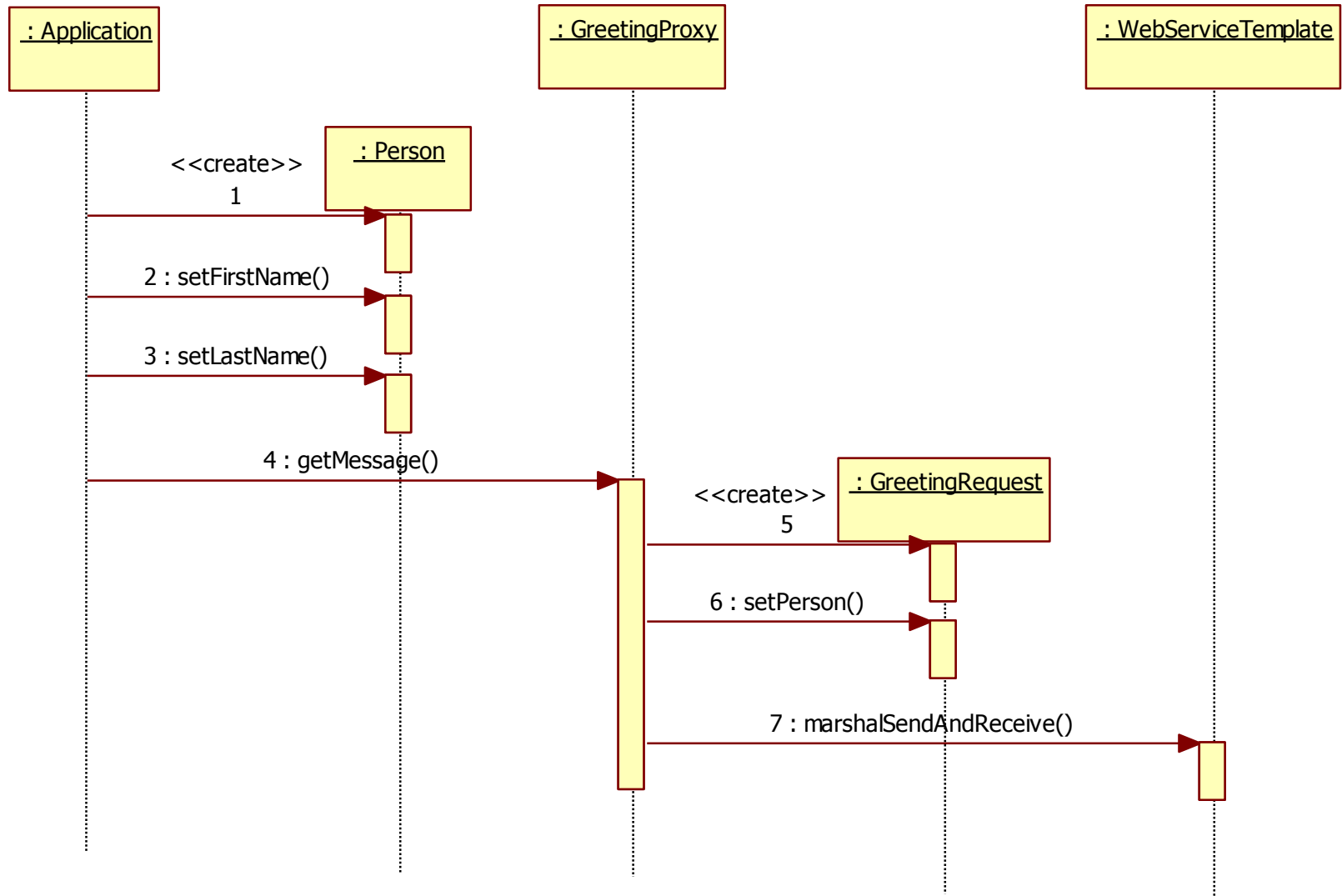


# Spring-WS client





# Spring-WS client





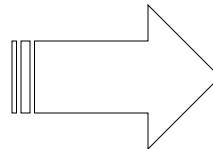


# Spring WS Client

```
public class Application {  
  
    public static void main(String[] args) {  
        ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");  
        IGreeting remoteService = context.getBean("greetingServiceProxy", IGreeting.class);  
  
        Person person = new Person();  
        person.setFirstName("John");  
        person.setLastName("Doe");  
        String result = remoteService.getMessage(person);  
        System.out.println("Receiving result: " + result);  
    }  
}
```

Create Person

Get message



Receiving result: Hello John Doe



# GreetingServiceProxy

```
public class GreetingServiceProxy implements IGreeting {
```

Implements IGreeting

```
    private WebServiceTemplate webServiceTemplate;
```

DI Web Service Template

```
    public void setWebServiceTemplate(WebServiceTemplate webServiceTemplate) {  
        this.webServiceTemplate = webServiceTemplate;  
    }
```

```
    public String getMessage(Person person) {  
        GreetingRequest request = new GreetingRequest();  
        request.setPerson(person);
```

Sends Request, receives  
Response object

```
        GreetingResponse response = (GreetingResponse)  
            webServiceTemplate.marshallSendAndReceive(request);  
        return response.getGreeting();  
    }
```

```
}
```



# springconfig.xml

```
<beans xmlns="...">
```

```
<bean id="greetingServiceProxy" class="demo.GreetingServiceProxy">
  <property name="webServiceTemplate" ref="webServiceTemplate" />
</bean>
```

GreetingProxy bean using  
webServiceTemplate

```
<bean id="webServiceTemplate" class="org.springframework.ws.client.core.WebServiceTemplate">
  <constructor-arg ref="messageFactory" />
  <property name="defaultUri" value="http://localhost:8080/SpringWS" />
  <property name="marshaller" ref="marshaller"/>
  <property name="unmarshaller" ref="marshaller"/>
</bean>
```

webServiceTemplate has  
the URI of our web service

```
<bean id="marshaller" class="org.springframework.xml.jaxb.Jaxb2Marshaller">
  <property name="contextPath" value="generated"/>
</bean>
```

Jaxb2 O/X marshalling

```
<bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory">
  <property name="messageFactory">
    <bean class="com.sun.xml.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_1Impl" />
  </property>
</bean>
```

messageFactory for sending /  
receiving messages

```
</beans>
```



# Endpoint Implementations

---

- We used an O/X framework in our example
- Spring also provides many Endpoint templates that help you parse the XML on your own:
  - AbstractDomPayloadEndpoint
  - AbstractJDomPayloadEndpoint
  - AbstractDom4jPayloadEndpoint
  - AbstractSaxPayloadEndpoint
  - AbstractXomPayloadEndpoint



Spring Web Services:

# **SPRING HTTP INVOKER**



# Spring HTTP Invoker

---

- The Spring HTTP Invoker provides a simple and efficient way to implement Java-to-Java web services
- Objects are simply serialized and sent back and forth over HTTP
- The advantages are that it is faster to implement and faster to transmit
- The disadvantage is that it is only possible if both sides are Java applications



# Java Implementation

```
public interface IGreeting {  
    public String getMessage(Person person);  
}
```

```
public class GreetingService implements IGreeting {  
    private String greeting;  
  
    public void setGreeting(String greeting) {  
        this.greeting = greeting;  
    }  
  
    public String getMessage(Person person) {  
        return greeting + " " + person.getFirstName() + " " + person.getLastName();  
    }  
}
```

IGreeting interface, GreetingService and Person class exactly as we've used them before.

```
public class Person implements Serializable {  
    private static final long serialVersionUID = 1L;  
    private String firstName;  
    private String lastName;  
  
    public Person() {  
    }  
    public Person(String firstName, String lastName) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
    }  
    ...  
}
```



# web.xml

```
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <servlet>
    <servlet-name>greeting</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>greeting</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>

</web-app>
```

Spring Dispatcher Servlet

All URLs Servlet Mapping





# greeting-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
```

Regular Bean with DI greeting

```
<bean id="greetingService" class="httpInvoker.GreetingService">
  <property name="greeting" value="Hello" />
</bean>
```

HttpInvokerServiceExporter

```
<bean name="/GreetingService"
  class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
  <property name="service" ref="greetingService" />
  <property name="serviceInterface" value="httpInvoker.IGreeting" />
</bean>
```

URL mapping

Sets service provider

Service Interface

```
</beans>
```



# HttpInvoker Client

```
public class Application {
```

```
    public static void main(String[] args) {
```

```
        ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");  
        IGreeting remoteService = context.getBean("greetingHttpInvokerProxy", IGreeting.class);
```

```
        Person person = new Person("John", "Doe");
```

```
        String result = remoteService.getMessage(person);
```

```
        System.out.println("Receiving result: " + result);
```

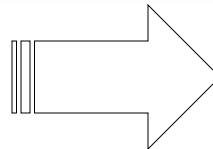
```
    }
```

```
}
```

Get Remote Service

Preform getMessage Call

Print Result



Receiving result: Hello John Doe



# springconfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

  <bean id="greetingHttpInvokerProxy"
    class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean">
    <property name="serviceUrl" value="http://localhost:8080/HttpInvoker/GreetingService" />
    <property name="serviceInterface" value="httpInvoker.IGreeting" />
  </bean>

</beans>
```



# Active Learning

---

- Describe the difference between the POST and the PUT method
- Describe the differences between SOAP and REST



# Summary

---

- There are different ways to implement a web service server and client with Spring
  - Integrate Spring with Axis2 (SOAP)
  - Integrate Spring with CXF (SOAP)
  - Using Spring-WS (SOAP)
  - Using Spring REST (REST)
  - Using the Spring HttpInvoker (Serialized object over HTTP)
- Spring hides most of the webservice implementation details.