Architecture of Enterprise Applications 8
Web Services

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

- Contents
  - WS Overview
  - SOAP WS
  - RESTful WS

- Objectives
  - 能够根据业务需求，识别需要封装为Web服务的业务功能，并能够将其实现为Restful Web服务

Web Services Overview

- Web Services
  - present the opportunity for real interoperability across hardware, operating systems, programming languages, and applications.
- Web
  - Access with web protocols
- Services
  - Independent of the implementation

SOAP

- SOAP is defined by its own XML Schema and relies heavily on the use of XML Namespaces.
  - Here's a SOAP request message that might be sent from a client to a server:

```
<?xml version='1.0' encoding='UTF-8' ?>
<env:Envelope
          xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
    <env:Header />
    <env:Body>
        <reservation xmlns="http://www.titan.com/Reservation">
           <customer>
                <!-- customer info goes here -->
           </customer>
        </reservation>
    </env:Body>
</env:Envelope>
```

- Imagine that you want to develop a web services component that implements the following interface:

```
public interface TravelAgent {
   public String makeReservation(int cruiseID,
         int cabinID, int customerId, double price);
}
```

WSDL

- A WSDL document that describes the `makeReservation( )` method might look like this:

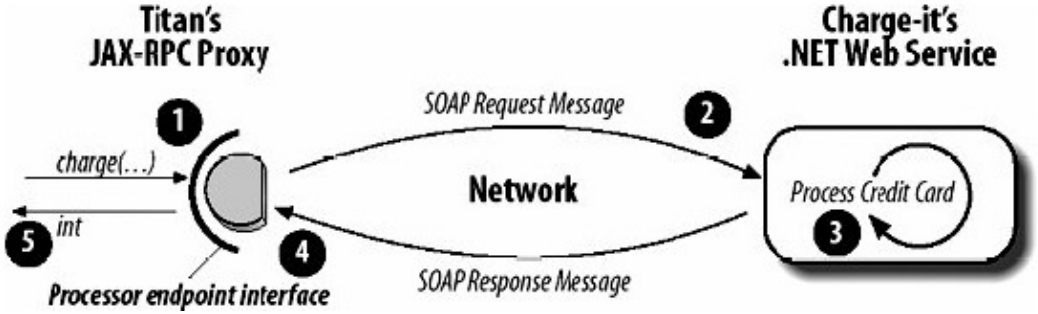
```
<?xml version="1.0"?>
 <definitions name="TravelAgent"
       xmlns="http://schemas.xmlsoap.org/wsdl/"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:titan="http://www.titan.com/TravelAgent"
       targetNamespace="http://www.titan.com/TravelAgent">
 <!-- message elements describe the parameters and return values -->
 <message name="RequestMessage">
         <part name="cruiseId" type="xsd:int" />
         <part name="cabinId" type="xsd:int" />
         <part name="customerId" type="xsd:int" />
         <part name="price" type="xsd:double" />
 </message>
 <message name="ResponseMessage">
         <part name="reservationId" type="xsd:string" />
 </message>
```

WSDL


```
<!-- portType element describes the abstract interface of a web service -->
<portType name="TravelAgent">
   <operation name="makeReservation">
      <input message="titan:RequestMessage"/>
      <output message="titan:ResponseMessage"/>
   </operation>
</portType>
<!--binding element tells us which protocols and encoding styles are used -->
<binding name="TravelAgentBinding" type="titan:TravelAgent">
 <soap:binding style="rpc“ transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="makeReservation">
     <soap:operation soapAction="" />
     <input>
      <soap:body use="literal" namespace="http://www.titan.com/TravelAgent"/>
     </input>
     <output>
      <soap:body use="literal" namespace="http://www.titan.com/TravelAgent"/>
     </output>
 </operation>
</binding>
```

```
  <!-- service element tells us the Internet address of a web service -->
  <service name="TravelAgentService">
    <port name="TravelAgentPort" binding="titan:TravelAgentBinding">
      <soap:address location="http://www.titan.com/webservices/TravelAgent" />
    </port>
  </service>
</definitions>
```

Accessing Web Service



Producing a SOAP Web Service

REliable, INtelligent & Scalable Systems

- pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web-services</artifactId>
  </dependency>
  <!-- tag::springws[] -->
  <dependency>
    <groupId>wsdl4j</groupId>
    <artifactId>wsdl4j</artifactId>
  </dependency>
  <!-- end::springws[] -->
</dependencies>
```

Producing a SOAP Web Service

- pom.xml

```
<profiles>
  <profile>
    <id>java11</id>
    <activation>
      <jdk>[11,)</jdk>
    </activation>
    <dependencies>
      <dependency>
        <groupId>org.glassfish.jaxb</groupId>
        <artifactId>jaxb-runtime</artifactId>
      </dependency>
    </dependencies>
  </profile>
</profiles>
```

Producing a SOAP Web Service

- pom.xml

```
<!-- tag::xsd[] -->
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>jaxb2-maven-plugin</artifactId>
  <version>2.5.0</version>
  <executions>
    <execution>
      <id>xjc</id>
      <goals>
        <goal>xjc</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <sources>
      <source>${project.basedir}/src/main/resources/countries.xsd</source>
    </sources>
  </configuration>
</plugin>
<!-- end::xsd[] -->
```

Producing a SOAP Web Service

- countries.xsd

```
<xs:element name="getCountryRequest">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="getCountryResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="country" type="tns:country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Producing a SOAP Web Service

- countries.xsd

```
<xs:complexType name="country">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="population" type="xs:int"/>
    <xs:element name="capital" type="xs:string"/>
    <xs:element name="currency" type="tns:currency"/>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="currency">
  <xs:restriction base="xs:string">
    <xs:enumeration value="GBP"/>
    <xs:enumeration value="EUR"/>
    <xs:enumeration value="PLN"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Producing a SOAP Web Service

- CountryRepository.java

```
import io.spring.guides.gs_producing_web_service.Country;
import io.spring.guides.gs_producing_web_service.Currency;

@Component
public class CountryRepository {
  private static final Map<String, Country> countries = new HashMap<>();

  @PostConstruct
  public void initData() {
    Country spain = new Country();
    spain.setName("Spain");
    spain.setCapital("Madrid");
    spain.setCurrency(Currency.EUR);
    spain.setPopulation(46704314);
    countries.put(spain.getName(), spain);
```

Producing a SOAP Web Service

- CountryRepository.java

```
    Country poland = new Country();
    poland.setName("Poland");
    poland.setCapital("Warsaw");
    poland.setCurrency(Currency.PLN);
    poland.setPopulation(38186860);
    countries.put(poland.getName(), poland);

    Country uk = new Country();
    uk.setName("United Kingdom");
    uk.setCapital("London");
    uk.setCurrency(Currency.GBP);
    uk.setPopulation(63705000);
    countries.put(uk.getName(), uk);
  }

  public Country findCountry(String name) {
    Assert.notNull(name, "The country's name must not be null");
    return countries.get(name);
  }
}
```

Producing a SOAP Web Service

- CountryEndpoint.java

```
import io.spring.guides.gs_producing_web_service.GetCountryRequest;
import io.spring.guides.gs_producing_web_service.GetCountryResponse;

@Endpoint
public class CountryEndpoint {
  private static final String NAMESPACE_URI = "http://spring.io/guides/gs-producing-web-service";
  private CountryRepository countryRepository;

  @Autowired
  public CountryEndpoint(CountryRepository countryRepository) {
    this.countryRepository = countryRepository;
  }

  @PayloadRoot(namespace = NAMESPACE_URI, localPart = "getCountryRequest")
  @ResponsePayload
  public GetCountryResponse getCountry(@RequestPayload GetCountryRequest request) {
    GetCountryResponse response = new GetCountryResponse();
    response.setCountry(countryRepository.findCountry(request.getName()));

    return response;
  }
}
```

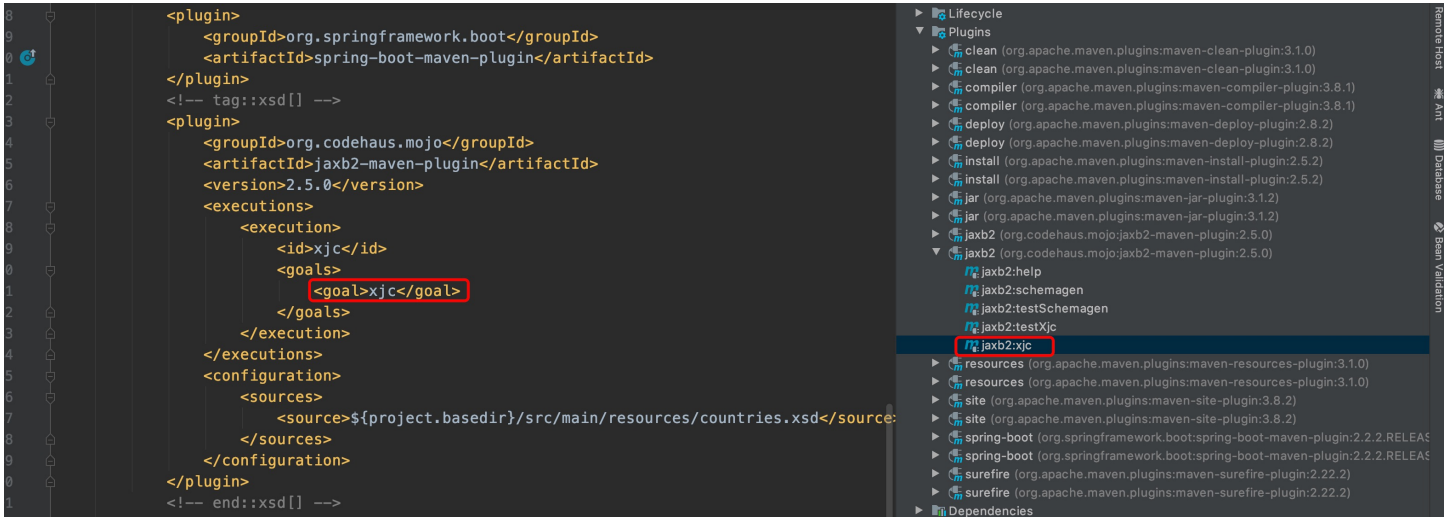
Producing a SOAP Web Service

- WebServiceConfig.java

```
@EnableWs
@Configuration
public class WebServiceConfig extends WsConfigurerAdapter {
  @Bean
  public ServletRegistrationBean messageDispatcherServlet(ApplicationContext applicationContext) {
    MessageDispatcherServlet servlet = new MessageDispatcherServlet();
    servlet.setApplicationContext(applicationContext);
    servlet.setTransformWsdlLocations(true);
    return new ServletRegistrationBean(servlet, "/ws/*");
  }
  @Bean(name = "countries")
  public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema countriesSchema) {
    DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
    wsdl11Definition.setPortTypeName("CountriesPort");
    wsdl11Definition.setLocationUri("/ws");
    wsdl11Definition.setTargetNamespace("http://spring.io/guides/gs-producing-web-service");
    wsdl11Definition.setSchema(countriesSchema);
    return wsdl11Definition;
  }
  @Bean
  public XsdSchema countriesSchema() {
    return new SimpleXsdSchema(new ClassPathResource("countries.xsd"));
  }
}
```

Producing a SOAP Web Service

- Generate Domain Classes Based on an XML Schema



Producing a SOAP Web Service

- request.xml

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:gs="http://spring.io/guides/gs-producing-web-service">
   <soapenv:Header/>
   <soapenv:Body>
      <gs:getCountryRequest>
         <gs:name>Spain</gs:name>
      </gs:getCountryRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Producing a SOAP Web Service

REliable, INtelligent & Scalable Systems

- Test the Web Service with Postman

http://localhost:8080/ws

POST http://localhost:8080/ws

Params Authorization Headers (10) Body Pre-request Script Tests Settings

Headers 9 hidden

KEY	VALUE	DES
Content-Type	text/xml	
Key	Value	Des

Postman

New Import Runner My Workspace Invite Upgrade

Launchpad POST http://localhost:8080/ws No Environment

POST http://localhost:8080/ws Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL XML Beautify

```
1  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2                    xmlns:gs="http://spring.io/guides/gs-producing-web-service">
3     <soapenv:Header/>
4     <soapenv:Body>
5        <gs:getCountryRequest>
6           <gs:name>Spain</gs:name>
7        </gs:getCountryRequest>
8     </soapenv:Body>
9  </soapenv:Envelope>
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 7ms Size: 667 B Save Response

Pretty Raw Preview Visualize XML

```
1  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
2     <SOAP-ENV:Header/>
3     <SOAP-ENV:Body>
4        <ns2:getCountryResponse xmlns:ns2="http://spring.io/guides/gs-producing-web-service">
5           <ns2:country>
6              <ns2:name>Spain</ns2:name>
7              <ns2:population>46704314</ns2:population>
8              <ns2:capital>Madrid</ns2:capital>
9              <ns2:currency>EUR</ns2:currency>
10          </ns2:country>
11       </ns2:getCountryResponse>
12    </SOAP-ENV:Body>
13 </SOAP-ENV:Envelope>
```

Bootcamp Build Browse

Consuming a SOAP web service

- pom.xml

```
<!-- tag::dependency[] -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web-services</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<!-- end::dependency[] -->
```

Consuming a SOAP web service

- pom.xml

```
<!-- tag::profile[] -->
<profiles>
  <profile>
    <id>java11</id>
    <activation>
      <jdk>[11,)</jdk>
    </activation>

    <dependencies>
      <dependency>
        <groupId>org.glassfish.jaxb</groupId>
        <artifactId>jaxb-runtime</artifactId>
      </dependency>
    </dependencies>
  </profile>
</profiles>
<!-- end::profile[] -->
```

Consuming a SOAP web service

- pom.xml

```
<plugin>
  <groupId>org.jvnet.jaxb2.maven2</groupId>
  <artifactId>maven-jaxb2-plugin</artifactId>
  <version>0.14.0</version>
  <executions>
    <execution>
      <goals>
        <goal>generate</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <schemaLanguage>WSDL</schemaLanguage>
    <generatePackage>org.reins.wssample.wsdl</generatePackage>
    <schemas>
      <schema>
        <url>http://localhost:8080/ws/countries.wsdl</url>
      </schema>
    </schemas>
  </configuration>
</plugin>
<!-- end::wsdl[] -->
```

Consuming a SOAP web service

- CountryClient.class

```
public class CountryClient extends WebServiceGatewaySupport {

  private static final Logger log = LoggerFactory.getLogger(CountryClient.class);

  public GetCountryResponse getCountry(String country) {

    GetCountryRequest request = new GetCountryRequest();
    request.setName(country);

    log.info("Requesting location for " + country);

    GetCountryResponse response = (GetCountryResponse) getWebServiceTemplate()
        .marshalSendAndReceive("http://localhost:8080/ws/countries", request,
            new SoapActionCallback(
                "http://spring.io/guides/gs-producing-web-service/GetCountryRequest"));

    return response;
  }
}
```

Consuming a SOAP web service

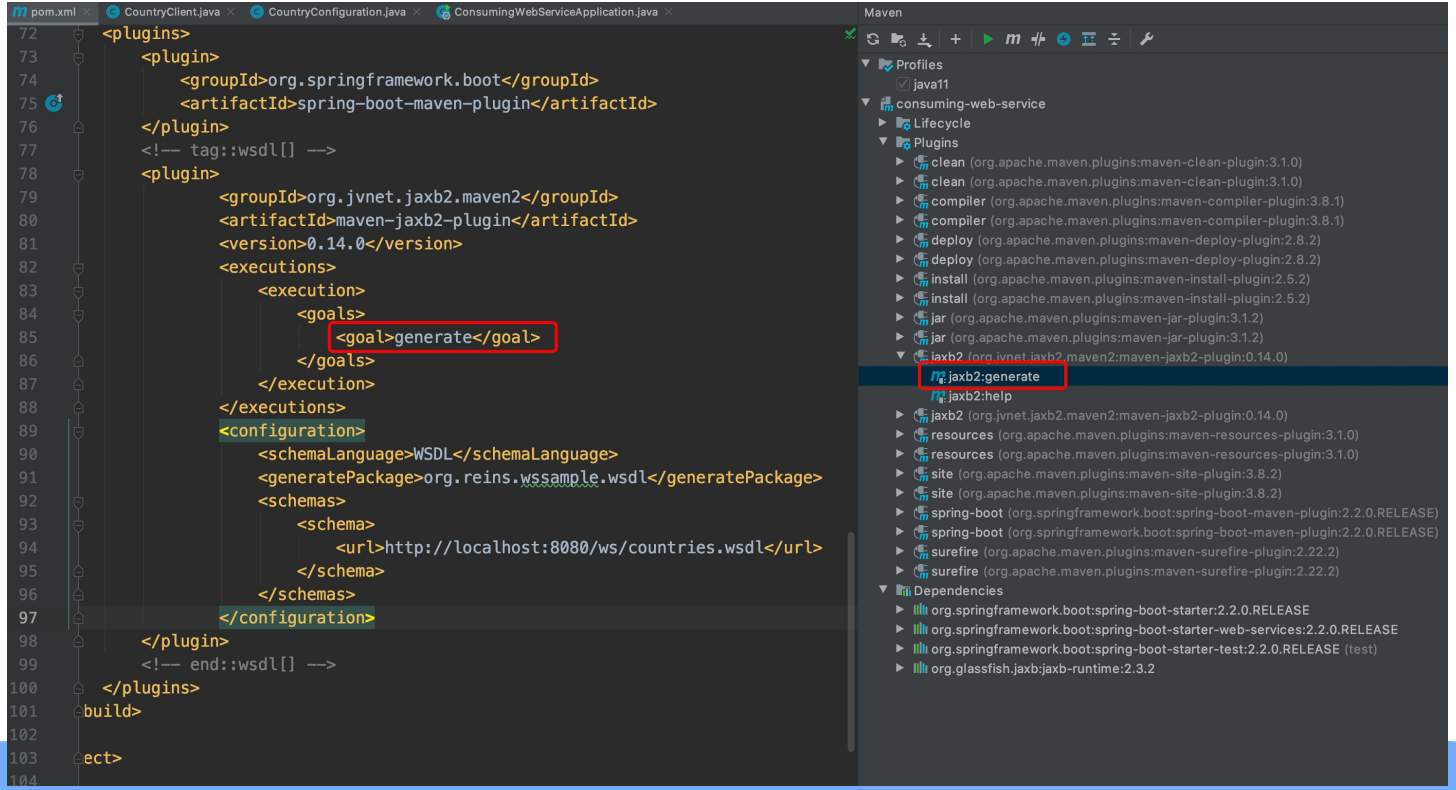
- CountryConfiguration.class

```
@Configuration
public class CountryConfiguration {

  @Bean
  public Jaxb2Marshaller marshaller() {
    Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
    // this package must match the package in the <generatePackage> specified in
    // pom.xml
    marshaller.setContextPath("org.reins.wssample.wsdl");
    return marshaller;
  }

  @Bean
  public CountryClient countryClient(Jaxb2Marshaller marshaller) {
    CountryClient client = new CountryClient();
    client.setDefaultUri("http://localhost:8080/ws");
    client.setMarshaller(marshaller);
    client.setUnmarshaller(marshaller);
    return client;
  }
}
```

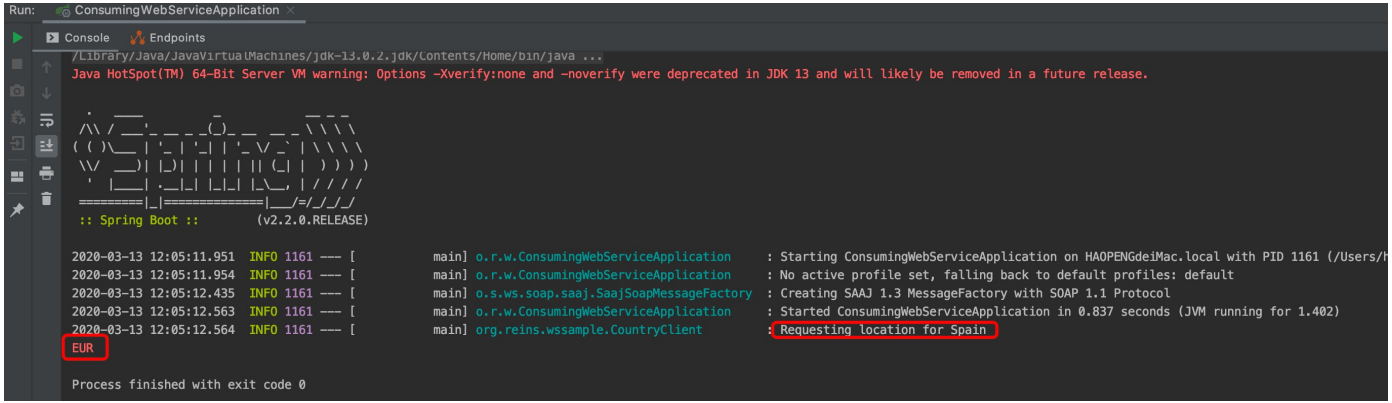
Consuming a SOAP web service

REliable, INtelligent & Scalable Systems

- Generate Domain Objects Based on a WSDL



Consuming a SOAP web service



JAX-WS Web Service

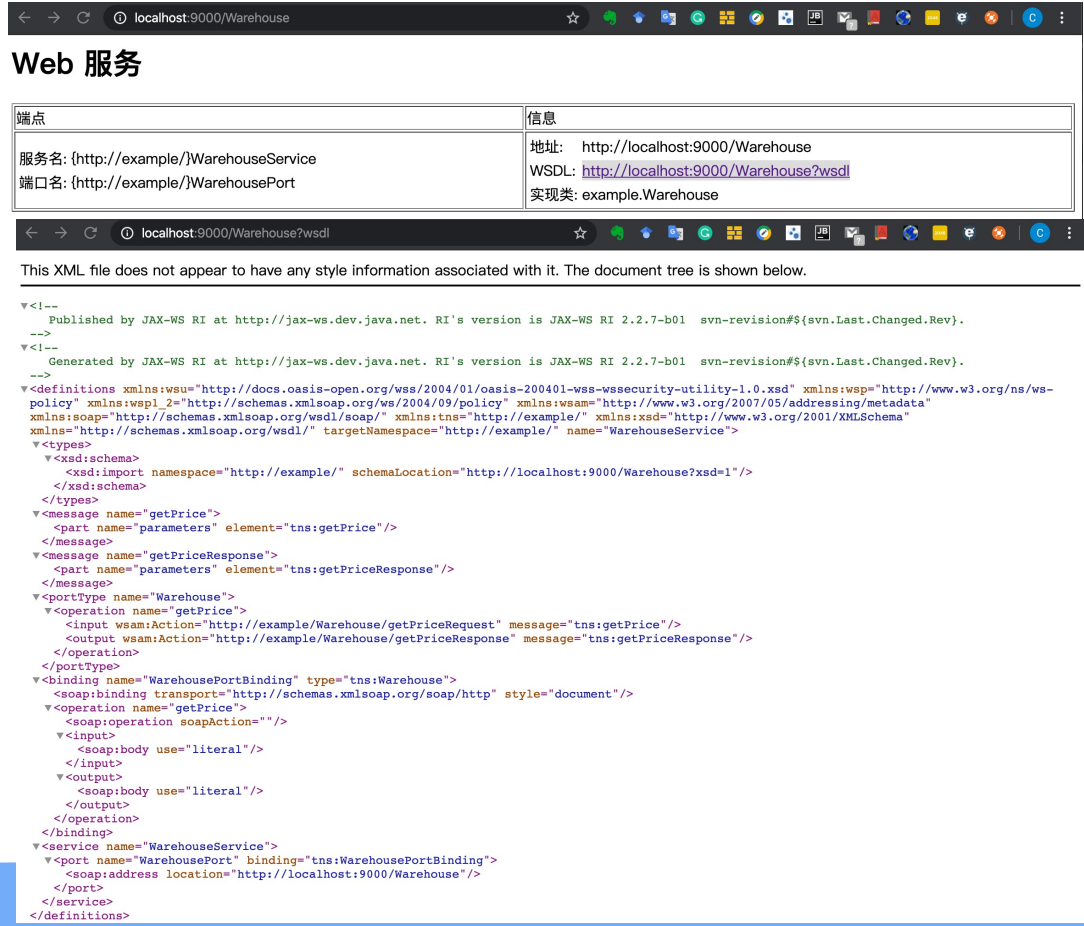
- Warehouse.java

```
@WebService
public class Warehouse {
 public Warehouse() {
  prices = new HashMap<String, Double>();
  prices.put("Blackwell Toaster", 24.95);
  prices.put("ZapXpress Microwave Oven", 49.95);
 }

 public double getPrice(@WebParam(name=”parameters") String description)
 {
  Double price = prices.get(description.trim());
  return price == null ? 0 : price;
 }

 private Map<String, Double> prices;

 public static void main(String[] argv) {
  Object implementor = new Warehouse ();
  String address = "http://localhost:9000/Warehouse";
  Endpoint.publish(address, implementor);
 }
}
```

JAX-WS Web Service

REIn
REliable, INtelligent & Scalable Systems

http://localhost:9000/Warehouse?wsdl



JAX-WS Web Service


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:gs="http://example/">
    <soapenv:Header/>
    <soapenv:Body>
        <gs:getPrice>
            <parameters>Blackwell Toaster</parameters>
        </gs:getPrice>
    </soapenv:Body>
</soapenv:Envelope>
```

JAX-WS Web Service


```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
        <S:Body>
                <ns2:getPriceResponse xmlns:ns2="http://example/">
                        <return>24.95</return>
                </ns2:getPriceResponse>
        </S:Body>
</S:Envelope>
```

JAX-WS Web Service

REliable, INtelligent & Scalable Systems

POST http://localhost:9000/Warehouse Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL XML Beautify

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2                   xmlns:gs="http://example/">
3    <soapenv:Header/>
4    <soapenv:Body>
5       <gs:getPrice>
6          <parameters>Blackwell Toaster</parameters>
7       </gs:getPrice>
8    </soapenv:Body>
9 </soapenv:Envelope>
```

Body Cookies Headers (3) Test Results 200 OK 26 ms 350 B Save Response

Pretty Raw Preview Visualize XML

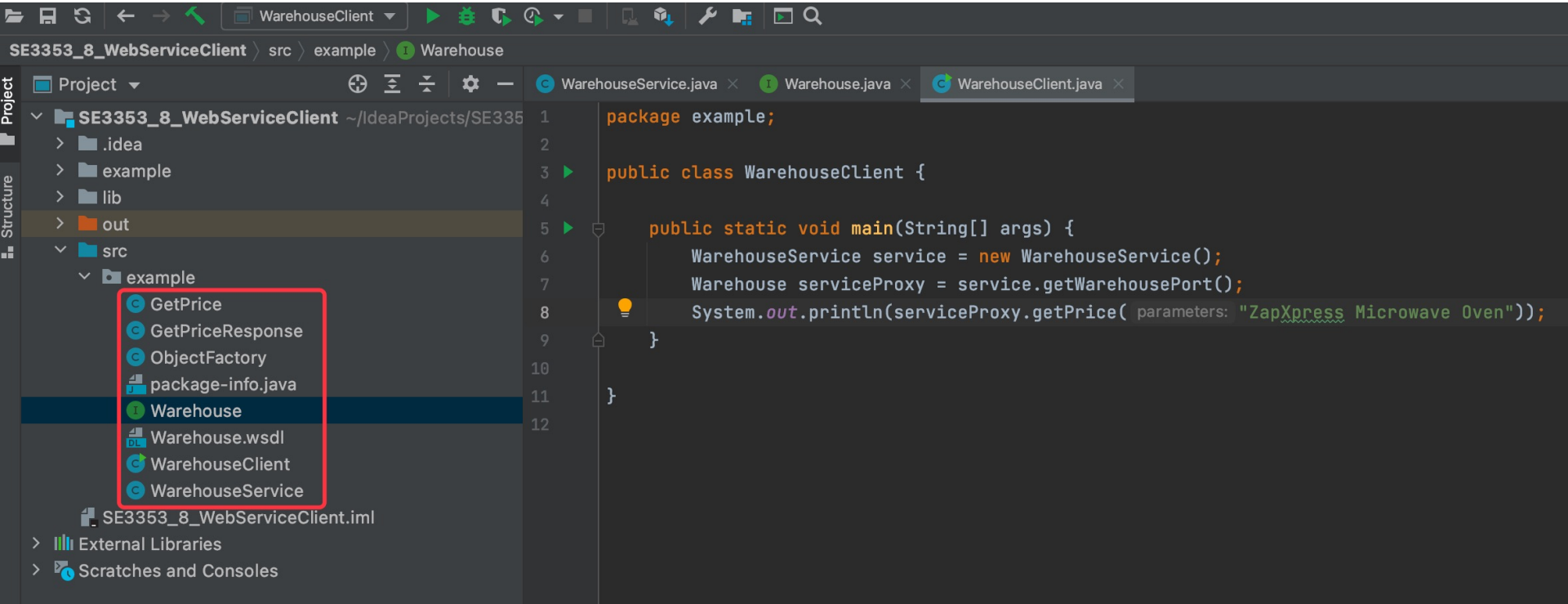
```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
3     <S:Body>
4         <ns2:getPriceResponse xmlns:ns2="http://example/">
5             <return>24.95</return>
6         </ns2:getPriceResponse>
7     </S:Body>
8 </S:Envelope>
```

Consuming JAX-WS in Java

REliable, INtelligent & Scalable Systems

- Generate classes with wsimport

  wsimport -s <generate your source file location> http://localhost:9000/Warehouse?wsdl



Consuming JAX-WS in Java

- WarehouseClient.java

```
package example;

public class WarehouseClient {
   public static void main(String[] args) {
      WarehouseService service = new WarehouseService();
      Warehouse serviceProxy = service.getWarehousePort();
      System.out.println(serviceProxy.getPrice("ZapXpress Microwave Oven"));
   }
}
```

Why

- SOAP-based Web Services
  - Coupling with the message format
  - Coupling with the encoding of WS
  - Parse and assemble SOAP
  - Need a WSDL to describe the details of WS
  - Need a proxy generated from WSDL

- It is a time-cost way to implement Web Service with SOAP
  - We should find a new way to implement WS

Restful Web Service

- REpresentational State Transfer
  - Representational:
    - All data are resources. Representation for client.
    - Each resource can have different representations
    - Each resource has its own unique identity(URI)
  - State:
    - It refers to state of client. Server is stateless.
    - The representation of resource is a state of client.
  - Transfer:
    - Client's representation will be transferred when client access different resources by different URI.
    - It means the states of client are also transferred.
    - That is Representation State Transfer

What

- REST is a kind of architecture, but not a specification
  - REST is a typical Client-Server architecture, but it is stateless server
  - All states are hold in the messages delivered between clients and server
  - Server only process the requirements of data, displaying is completely depended on clients
  - REST is idempotent which means server will return same results for same require. So the results can be cached on either clients or server

What

- In REST, all operations are preformed in unified way
  - Each resource has a unique identity
  - Process resource by representation
    - Client can not directly manipulate resources.
    - Client only can manipulate its representation, and send requires.
    - Server process requires and return response.
  - Any message between clients and server is self-described.
    - The context for processing a message is contained in the message itself.
  - Multimedia interaction system.
    - The content delivered between clients and server can be documents, pictures, audios, and videos
    - It is the base for resource to be rendered as different representations.

How to design REST

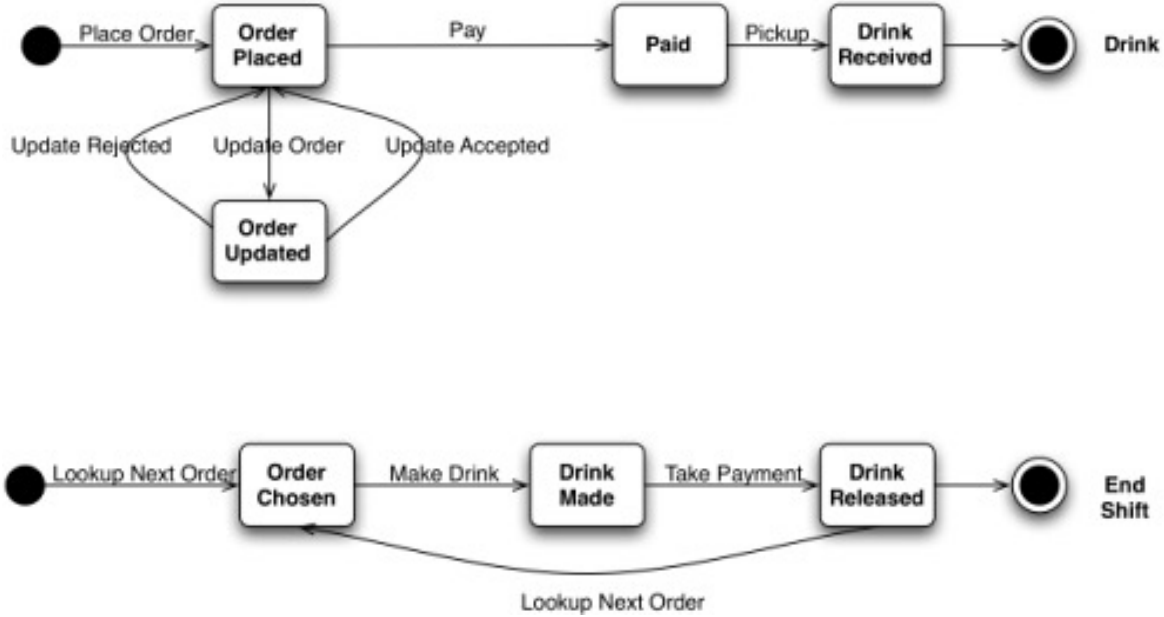
- Design rules
  - Anything on web is abstracted as resource
  - Any resource has a unique resource identifier
  - Access resource by generic connector interface
  - Any manipulation to resource doesn't change resource identifier
  - All operations are stateless

- Resources are not data, but the combination of data and representation
  - Same data with different representation will be abstracted as different resources.

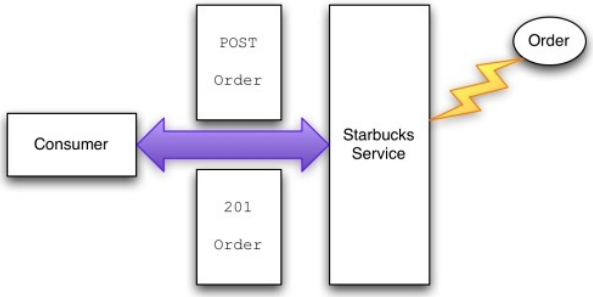
How to design REST

- CRUD
  - Atomic operations: Create, Read, Update, Delete
  - Composite them to build complex manipulate
- HTTP-based
  - GET-read
  - POST-create
  - PUT-update
  - DELETE-delete
- Design by URL
  - We just need to design suitable URLs which directly represent user interface
  - Developers just need to abstract resources according to URLs
  - URL without parameters is more convenient for user
  - Quite different from action-based design method, such as MVC
- Notice: it is very difficult to abstract anything on web as resource
  - Mix MVC and REST

How to Get a Cup of Coffee



How to Get a Cup of Coffee



Request:

```
POST /order HTTP1.1
Host: starbucks.example.org
Content-Type: application/xml
Content-Length: . . .

<order xmlns="http://starbucks.example.org/">
    <drink>latte</drink>
</order>
```

Response:

```
201 Created
Location: http://starbucks.example.org/order/1234
Content-Type: application/xml
Content-Length: . . .

<order xmlns="http://starbucks.example.org/">
    <drink>latte</drink>
    <cost>3.00</cost>
    <next xmlns="http://example.org/state-machine"
        rel="http://starbucks.example.org/payment"
        uri="http://starbucks.example.com/payment/order/1234"
        type="application/xml"/>
</order>
```

How to Get a Cup of Coffee

- Response code
  - 200 OK
  - 201 Created
  - 202 Accepted
  - 303 See Other
  - 400 Bad Request
  - 404 Not Found
  - 409 Conflict
  - 412 Precondition Failed
  - 417 Expectation Failed
  - 500 Internal Server Error

How to Get a Cup of Coffee

- Update order

Request	Response
`OPTIONS /order/1234 HTTP 1.1` `Host: starbucks.example.org`	`200 OK Allow: GET, PUT`

Request	Response

`PUT /order/1234 HTTP 1.1`
`Host: starbucks.example.com`
`Expect: 100-Continue`
	`100 Continue`

How to Get a Cup of Coffee

- Update order

Request:

```
PUT /order/1234 HTTP1.1
Host: starbucks.example.com
Content-Type: application/xml
Content-Length: . . .

<order xmlns="http://starbucks.example.org/">
    <additions>shot</additions>
</order>
```

Response:

```
200 OK
Location: http://starbucks.example.org/order/1234
Content-Type: application/xml
Content-Length: . . .

<order xmlns="http://starbucks.example.org/">
    <drink>latte</drink>
    <additions>shot</additions>
    <cost>4.00</cost>
    <next xmlns="http://example.org/state-machine"
        rel="http://starbucks.example.org/payment"
        uri="http://starbucks.example.com/payment/order/1234"
        type="application/xml"/>
</order>
```

How to Get a Cup of Coffee

- Update order

Request:

```
PUT /order/1234 HTTP1.1
Host: starbucks.example.com
Content-Type: application/xml
Content-Length: . . .

<order xmlns="http://starbucks.example.org/">
    <additions>shot</additions>
</order>
```

Response:

```
409 conflict
Location: http://starbucks.example.org/order/1234
Content-Type: application/xml
Content-Length: . . .

<order xmlns="http://starbucks.example.org/">
    <drink>latte</drink>
    <cost>4.00</cost>
    <next xmlns="http://example.org/state-machine"
        rel="http://starbucks.example.org/payment"
        uri="http://starbucks.example.com/payment/order/1234"
        type="application/xml"/>
</order>
```

How to Get a Cup of Coffee

- Payment

```
<next xmlns="http://example.org/state-machine"
      rel="http://starbucks.example.org/payment"
      uri="http://starbucks.example.com/payment/order/1234"
      type="application/xml"/>
```

Request	Response
`OPTIONS/payment/order/1234 HTTP 1.1` `Host: starbucks.example.com`	`Allow: GET, PUT`

How to Get a Cup of Coffee

- Payment

Request

```
PUT /payment/order/1234 HTTP 1.1
Host: starbucks.example.com
Content-Type: application/xml
Content-Length: ...
Authorization: Digest username="Jane Doe"
realm="starbucks.example.org"
nonce="..."
uri="payment/order/1234"
qop=auth
nc=00000001
cnonce="..."
reponse="..."
opaque="..."
123456789
07/07
John Citizen
4.00
```

Response

```
201 Created
Location:
https://starbucks.example.com/payment/order/1234
Content-Type: application/xml
Content-Length: ...
    123456789
    07/07
    John Citizen
    4.00
```

How to Get a Cup of Coffee

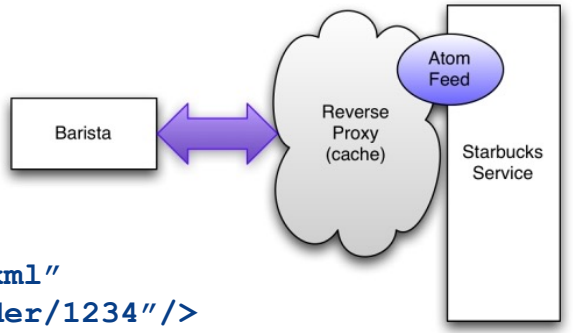
- Get a list of orders

Response:

```
200 OK
Expires:Thu, 12Jun2008 17:20:33 GMT
Content-Type: application/xml
Content-Length: . . .

<?xml version="1.0" ?>
<feed xmlns="http://www.w3.org/2005/Atom">
    <title>Coffees to make</title>
    <link rel="alternate"
        uri="http://starbucks.example.com/orders"/>
    <updated>2014-05-16T08:18:43Z</update>
```

How to Get a Cup of Coffee

- Get a list of orders



Response:

```
<entry>
    <published>2014-05-16T08:18:43Z</title>
    <updated>2014-05-16T08:20:32Z</update>
    <link rel="alternate" type="application.xml"
        uri="http://starbucks.example.com/order/1234"/>
    <id>http://starbucks.example.com/order/1234</id>
    <content type="text+xml">
       <order xmlns="http://starbucks.example.com/">
          <drink>latte</drink>
          <additions>shot</additions>
          <cost>4.00</cost>
       </order>
       <link rel="edit"
          type="applicatioin/atom+xml"
          href="http://starbucks.example.com/order/1234"/>
       . . .
    </content>
</entry>
```

How to Get a Cup of Coffee

- Get a list of orders

Request

```
PUT /order/1234 HTTP 1.1
 Host: starbucks.example.com
 Content-Type: application/atom+xml
 Content-Length: ...

 <entry>
   . . .
  <content type="text+xml">
     <order xmlns="http://starbucks.example.com/">
        <drink>latte</drink>
        <additions>shot</additions>
        <cost>4.00</cost>
        <status>preparing</status>
     </order>
     . . .
  </content>
</entry>
```

How to Get a Cup of Coffee

REliable, INtelligent & Scalable Systems

- Check payment of orders

Request	Response

`GET /payment/order/1234 HTTP 1.1`
`Host: starbucks.example.org`
	`401 Unauthorized`
	`WWW-Authenticate: Digest`
	`realm="starbucks.example.com",`
	`qop="auth",`
	`nonce="ab656...",`
	`opaque="b6a9..."`

Request	Response

`GET /payment/order/1234 HTTP 1.1`
`Host: starbucks.example.org`
`Authorization: Digest`
`username="barista joe"`
`realm="starbucks.example.com"`
`nonce="..."`
`uri="payment/order/1234"`
`qop=auth`
`nc=00000001 c`
`nonce="..."`
`reponse="..."`
`opaque="..."`
	`200 OK`
	`Content-Type: application/xml`
	`Content-Length: ...`
	`123456789`
	`07/07`
	`John Citizen`
	`4.00`

How to Get a Cup of Coffee

- Delete a order

Request	Response
`DELETE /order/1234 HTTP 1.1` `Host: starbucks.example.org`	`200 OK`

Building a RESTful Web Service

- GreetingController.java

```
@RestController
public class GreetingController {

  private static final String template = "Hello, %s!";
  private final AtomicLong counter = new AtomicLong();

  @GetMapping("/greeting")
  public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
    return new Greeting(counter.incrementAndGet(), String.format(template, name));
  }

  @PostMapping("/greeting")
  public Greeting greetingpost(@RequestParam(value = "name", defaultValue = "Spring") String name) {
    return new Greeting(counter.incrementAndGet(), String.format(template, name));
  }
}
```

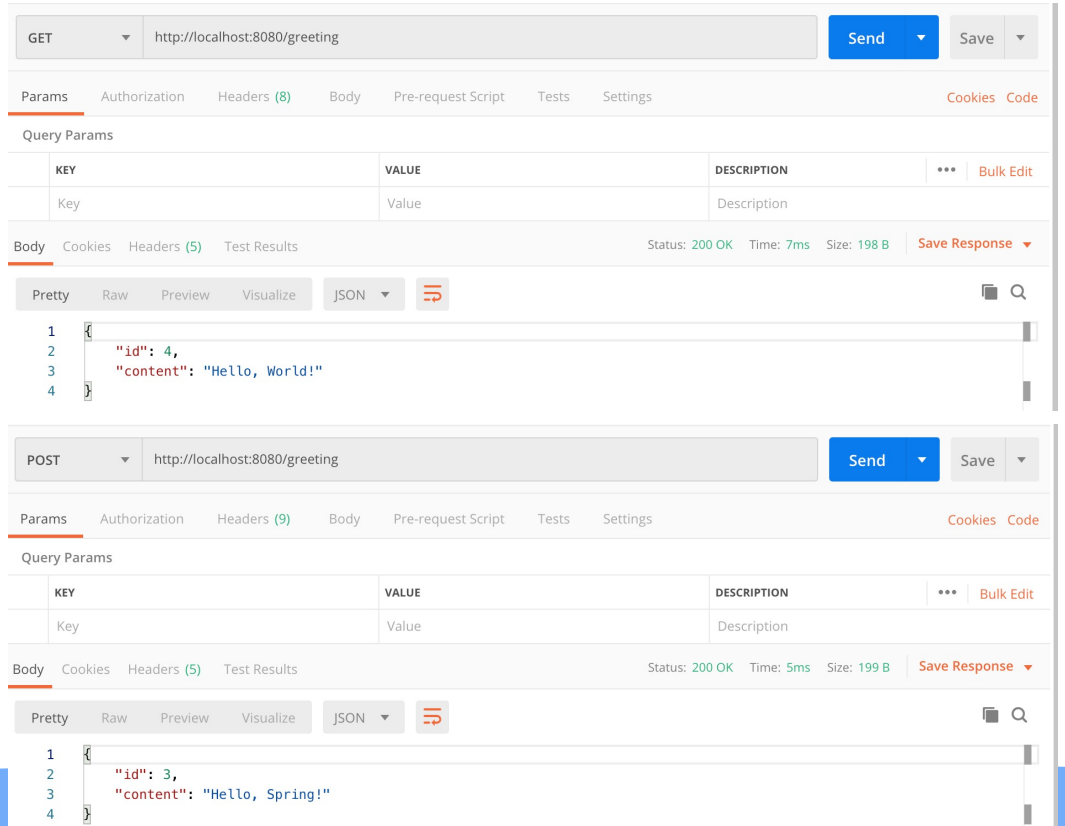
Building a RESTful Web Service

- Greeting.java

```
public class Greeting {

  private final long id;
  private final String content;

  public Greeting(long id, String content) {
    this.id = id;
    this.content = content;
  }

  public long getId() {
    return id;
  }

  public String getContent() {
    return content;
  }
}
```

Consuming a RESTful Web Service

- Test the RESTful Web Service



Consuming a RESTful Web Service

- App.js

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

function App() {
  fetch('http://localhost:8080/greeting')
    .then(response => response.json())
    .then(data => {
      document.getElementById("anh").innerText = JSON.stringify(data)
      console.log('parsed json', data)
    }).catch(function (ex) {
      console.log('parsing failed', ex)
  })
  return (
    <div className="App">
      <h1 id="anh"></h1>
    </div>
  );
}
export default App;
```

{"id":6,"content":"Hello, World!"}

Consuming a RESTful Web Service

- CorsConfig.java

Cross origins

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {

   @Override
   public void addCorsMappings(CorsRegistry registry) {
      registry.addMapping("/**")
            .allowedOrigins("*")
            .allowedMethods("*")
            .allowedHeaders("*")
            .exposedHeaders(HttpHeaders.SET_COOKIE)
            .allowCredentials(true).maxAge(1800);
   }
}
```

Trade-off of WS

- Advantages:
  - Across platforms
    - XML-based, independent of vendors
  - Self-described
    - WSDL: operations, parameters, types and return values
  - Modulization
    - Encapsulate components
  - Across Firewall
    - HTTP
- Disadvantages:
  - Lower productivity
    - Not suitable for stand-alone applications
  - Lower performance
    - Parse and assembly
  - Security
    - Depend on other mechanism, such as HTTP+SSL

When

- When we should use WS:
  - Support communication across firewall
  - Support application integration
  - Support B2B integration
  - Encourage reusing software

- When we should NOT use WS:
  - Stand-alone applications
    - Such as MS Office
  - Homogeneous applications in LAN
    - Such as communication between COM+s or EJBs

The business drivers for a new approach

- While IT executives have been facing the challenge of
  - cutting costs
  - maximizing the utilization of existing technology
- At the same time they have to
  - continuously strive to serve customers better
  - be more competitive
  - be more responsive to the business's strategic priorities.
- There are two underlying themes behind all of these pressures: Heterogeneity and change

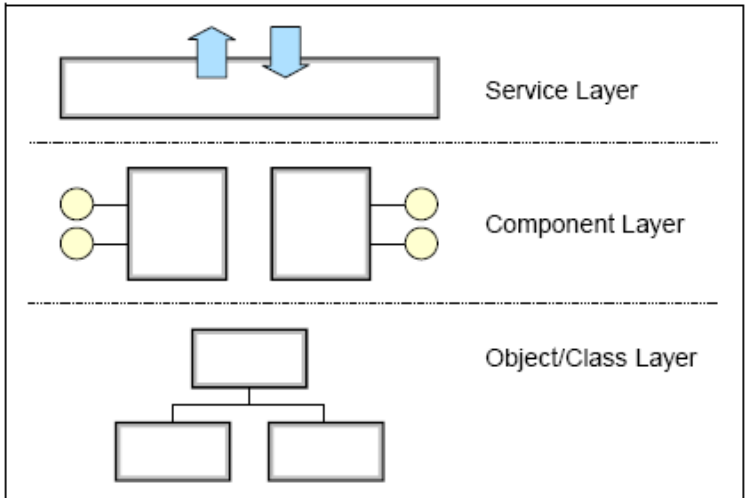
The business drivers for a new approach

- In order to alleviate the problems of heterogeneity, interoperability and ever changing requirements, such an architecture should provide a platform for building application services with the following characteristics:
  - Loosely coupled
  - Location transparent
  - Protocol independent

- Based on such a service-oriented architecture,
  - a service consumer does not even have to care about a particular service it is communicating with because the underlying infrastructure,
  - or service "bus", will make an appropriate choice on behalf of the consumer.

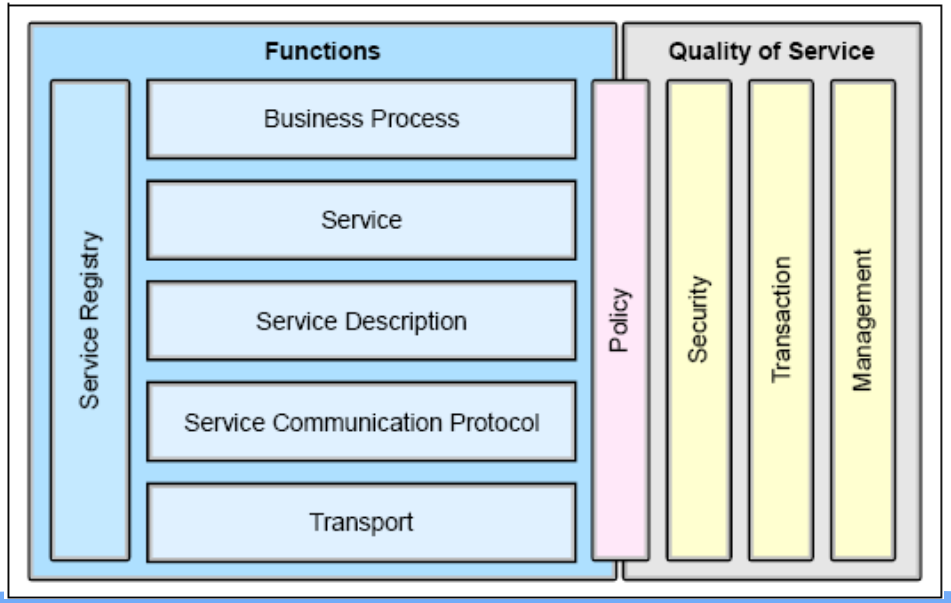
Layered application architectures

- Object-oriented technology and languages are great ways to implement components.
  - While components are the best way to implement services, though one has to understand that a good component-based application does not necessarily make an good service-oriented application.

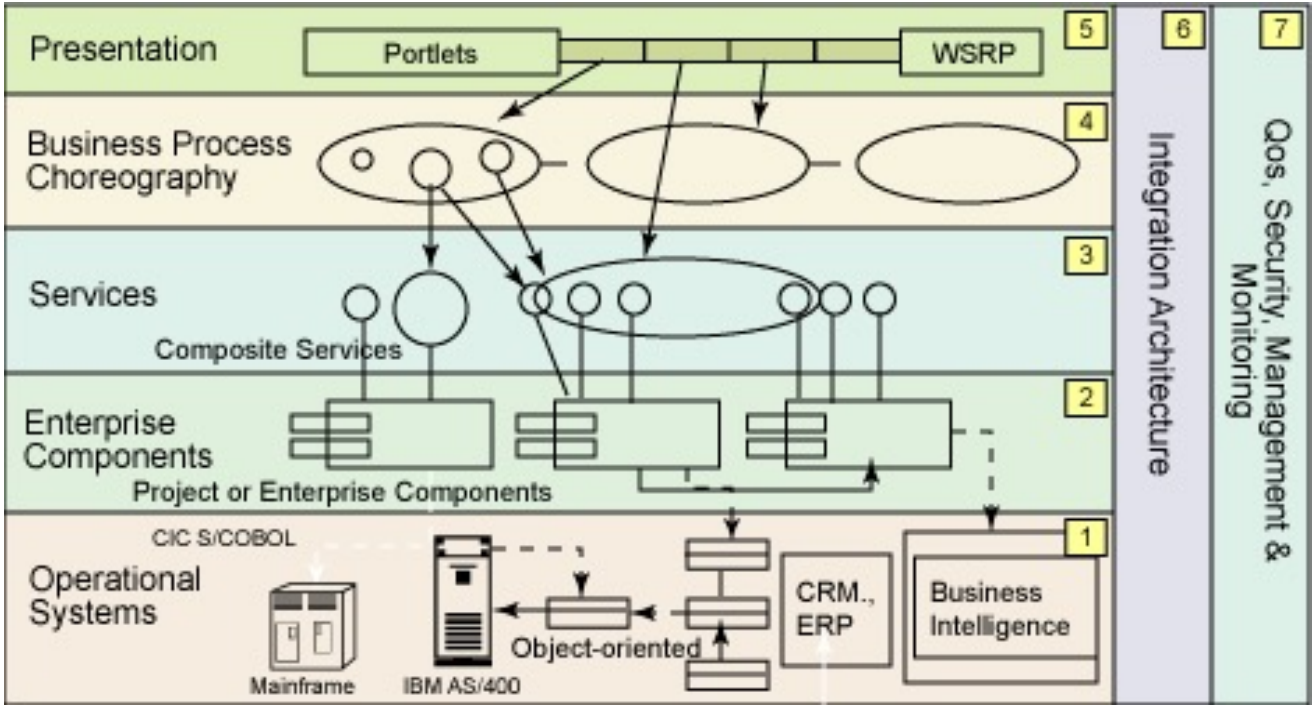


A closer look at SOA

- Service-oriented architecture
  - presents an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services.
  - It is comprised of elements that can be categorized into functional and quality of service.



The SOA Layers



作业四

- 请你在大二开发的E-Book系统的基础上，完成下列任务：
  1. 在你的项目中增加基于Solr或Lucene的针对书籍简介的全文搜索功能，用户可以在搜索界面输入搜索关键词，你可以通过全文搜索引擎找到书籍简介中包含该关键词的书籍列表。为了实现起来方便，你可以自己设计文本文件格式来存储书籍简介信息。例如，你可以将所有书籍的简介信息存储成为JSON对象，包含书的ID和简介文本，每行存储一本书的JSON对象。
  2. 请将上述全文搜索功能开发并部署为Web Service。

  – 请将你编写的相关代码整体压缩后上传，请勿压缩整个工程提交。

- 评分标准：
  1. 能够正确实现上述搜索功能，并且集成到工程中(3分)
  2. 能够正确实现上述Web Service功能，并且集成到工程中(2分)

References

- Producing a SOAP web service
  - https://spring.io/guides/gs/producing-web-service/
- Consuming a SOAP web service
  - https://spring.io/guides/gs/consuming-web-service/
- Building a RESTful Web Service
  - https://spring.io/guides/gs/rest-service/
- 手把手教你如何使用IDEA开发WebService服务器端，顺便填了一些莫名其妙的坑（原创）
  - https://www.jianshu.com/p/de004acd673d
- 用IDEA生成webService客户端时抛出Exception in thread "main" java.lang.AssertionError: org.xml.sax.SAXParseExce
  - https://blog.csdn.net/u013246459/article/details/78458020

Thank You!