

RESTful Java

JBoss User Group
김병곤(fharenheit@gmail.com)

- ❑ JBoss User Group 대표
- ❑ 분산 컴퓨팅 기반 개인화 시스템 구축
 - Process Designer – ETL, Input/Output, Mining Algorithm, 통계...
 - Apache Hadoop/Pig/HBase
 - Apache Cassandra
 - Distributed Cache
 - Log Aggregator
 - CEP(Complex Event Processing)
 - Mining Algorithm – Association Rule, K-Means, CF, ...
- ❑ 다수 책 집필 및 번역
 - JBoss Application Server 5, EJB 2/3
 - O'Reilly RESTful Java 번역중



- ❑ Chapter 1. Introduction
- ❑ Chapter 2. JAX-RS In Action
- ❑ Chapter 3. JAXB와 XML/JSON
- ❑ Chapter 4. Spring Framework Integration
- ❑ Chapter 5. Unit/Integration Test
- ❑ Chapter 6. JBoss RESTEasy

❑ Subversion

- URL : <https://dev.naver.com/svn/edward/trunk>
- Username : anonsvn
- Password : anonsvn

❑ Site : <http://dev.naver.com/projects/edward>

❑ 참고

- http://docs.jboss.org/resteasy/docs/1.2.GA/userguide/html_single

Chapter 1

Introduction

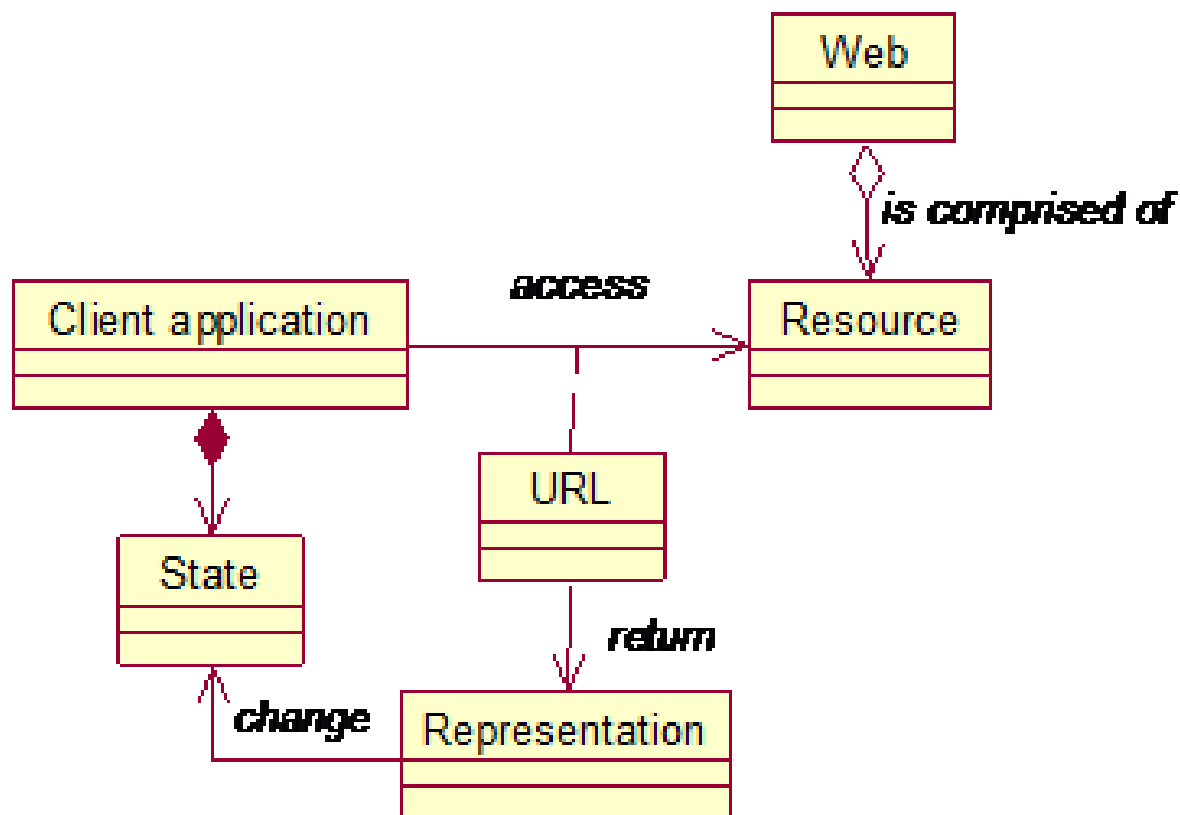
JBoss User Group
김병곤(fharenheit@gmail.com)

- ❑ 이기종 시스템간 통신을 보다 빠르고 편한 방법으로 구현할 수 있을까?
- ❑ Ajax 애플리케이션에 적합한 통신 방법은 없을까?
- ❑ 간단한 기술 및 아키텍처였으면 좋겠다.
- ❑ 모바일(WIPI/Android/iPhone)에 연결할 방법은 없을까?
- ❑ 쉬웠으면 좋겠다.

REST!

- ❑ 2000년 Roy Fielding의 박사학위 논문에서 네트워크 시스템의 구조적 형식을 설명하기 위해 만든 용어
- ❑ REST : REpresentational State Transfer
 - Representation, State
- ❑ REST is non-standard, architecture style
- ❑ RESTful WebService : REST 규칙을 따르는 웹 서비스 구현 방식
- ❑ HTTP의 재발견

REST?

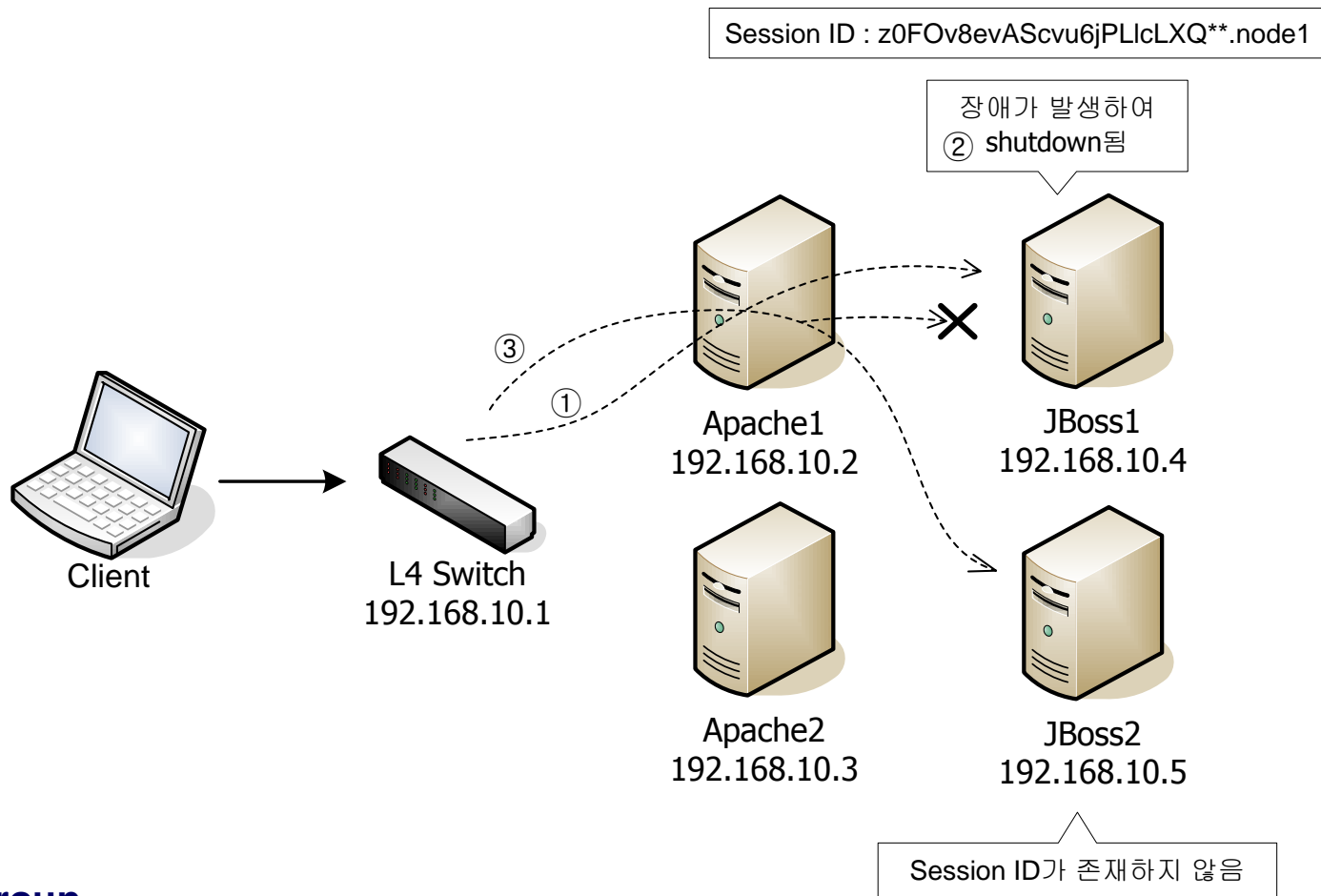


REST Architectural Principles

- ❑ Addressable resources
 - 모든 것은 URI가 있다
- ❑ A uniform, constrained interface
 - 모든 요청은 HTTP Method로
- ❑ Representation-oriented
 - 애플리케이션 마다 다른 포맷 필요
- ❑ Statelessness
 - 확장성 및 클라이언트/서버 간 decoupling
- ❑ Hypermedia As The Engine Of Application State (HATEOAS)

참고: Statelessness

- 상태를 유지하면 확장성이 떨어진다. 그래서 REST에서는 상태를 서버쪽에서 유지하지 않는다.



- ❑ URI 기반으로 리소스에 접근하는 기술
 - 예) Username이 1인 사용자의 정보를 보내줘
 - Request : `http://www.mydomain.com/user/1`
 - Response : XML or JSON or String or ...
- ❑ 프로토콜은 어느 장비에서나 지원하는 HTTP를 사용
- ❑ HTTP 프로토콜의 간단함을 그대로 시스템간 통신시 사용
- ❑ HTTP 프로토콜 그 자체에 집중

WebServices, JAX-RS, REST

- ❑ 자바의 웹 서비스 규격
 - SOAP Based WebServices
 - JAX-RPC(J2EE 1.4), JAX-WS(Java EE 5)
 - REST Based WebServices
 - JAX-RS
- ❑ JAX-RPC, JAX-WS는 모두 SOAP 규격으로써 최근 많이 사용하지 않는 추세
- ❑ JAX-RS는 REST로써 Ajax, Mobile이 증가하면서 향후 많이 사용할 규격
- ❑ 자바 웹 서비스 규격과 함께 배워야 하는 규격 – JAXB
 - Object와 XML 간의 상호 변환(바인딩) 규격
 - JAX-RS에서는 Object와 JSON/XML을 변환할 때 사용

- ❑ JCP Specification
- ❑ Java EE 6를 구성하는 규격
- ❑ Annotation 기반 프레임워크
- ❑ HTTP 요청을 자바 메소드 호출로 매핑

□ HTTP Method

- @javax.ws.rs.GET, @javax.ws.rs.POST
- @javax.ws.rs.PUT, @javax.ws.rs.DELETE
- @javax.ws.rs.HEAD

□ Annotation은 자바 메소드에 추가 가능

```
@Path("/customers")
public class CustomerService {
    @GET
    @Produces("application/xml")
    public String getAllCustomers() {
        ...
    }
}
```

GET http://<IP>:8080/customers

□ @Path annotation으로 처리할 URI를 지정

@Path("/orders")

```
public class OrderResource {
```

```
    @GET
```

```
    public String getAllOrders() {...} → URI : /orders
```

```
    @GET
```

@Path("unpaid")

```
    public String getUnpaidOrders() {...} → URI: /orders/unpaid
```

```
    @GET
```

@Path("{id}")

```
    public String getOrder(@PathParam("id") int id) {...} → URI : /orders/1
```

```
}
```

```
@Path("{id : \\d+}")  
@Path("orders/{date}-{id}")
```

@Produces & @Consumes Annotation

- ❑ 내용의 유형을 식별하는 HTTP 속성
 - Accept 헤더 : 응답으로 받을 수 있는 가능한 미디어 유형
 - Content-Type 헤더 : Req/Res Body의 미디어 유형(예; text/plain)
- ❑ 요청과 응답에 대한 header를 제어하는 annotation
 - MIME Type을 값으로 함 (text/plain, application/json, image/jpeg)
- ❑ @Consumes annotation
 - Content-Type 헤더를 이용하여 요청의 유형을 지정할 때 사용
 - 헤더에 맞지 않는 요청인 경우 406 Not Acceptable 에러 발생
- ❑ @Produces annotation
 - 응답을 보내줄 수 있는 유형을 지정할 때 사용
 - 클라이언트의 accept-header에 따라서 다른 응답을 주게 됨

- ❑ REST는 HTTP 프로토콜 그 자체에 집중하므로 응답 코드도 지극히 HTTP 프로토콜 답다
 - 200~399 : Successful HTTP Response
 - 200 : OK (HTTP Message Body O)
 - 204 : OK but No Content (HTTP Message Body X)
 - 400~599 : Error HTTP Response
 - 404 : Not Found (URI가 없는 경우)
 - 406 : Not Acceptable (accept-header가 맞지 않는 경우)
 - 405 : Method Not Allowed (PUT, GET, DELETE, POST 등이 맞지 않는 경우)
- ❑ 클라이언트에게 적절하게 응답 코드를 돌려주는 것은 매우 중요
 - Ajax 작성시에 응답의 내용에만 집중하는 경향이 있음
 - 응답 코드도 중요함

- REST는 HTTP 응답 코드에도 집중해야 한다!!

```
@Path("/textbook")
public class TextBookService {

    @GET
    @Path("/restfuljava")
    @Produces("text/plain")
    public Response getBook() {
        String book = ...;
        ResponseBuilder builder = Response.ok(book);
        return builder.build();
    }
}
```

응답 코드와 Exception 처리

❑ Exception과 Response Code를 매핑할 방법이 없을까?

- 예) NotFoundException : 404 Not Found

❑ JAX-RS는 WebApplicationException을 제공

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Path("{id}")
    @Produces("application/xml")
    public Customer getCustomer(@PathParam("id") int id) {
        Customer cust = findCustomer(id);
        if (cust == null) {
            throw new WebApplicationException(Response.Status.NOT_FOUND);
        }
        return cust;
    }
}
```

- ❑ URI 기반 파라미터
- ❑ HTTP Body를 직접 로딩
 - InputStream
 - Reader
 - File
 - byte[]
 - String
 - ...

```
@PUT
@Path("/stuff")
public void putStuff(InputStream is) {...}
```

```
@PUT
@Path("/morestuff")
public void putMore(Reader reader) {...}
```

```
@POST
@Path("/morestuff")
public void post(File file) {...}
```

```
@POST
@Consumes("text/plain")
public void post(byte[] bytes) {...}
```

```
@Consumes("text/plain")
public void post(String str) {...}
```

❑ 다양한 유형의 응답 가능

- String
- void
- Integer
- JPEG 등의 이미지
- OutputStream
- XML, JSON
- File
- Response Code

Q & A

Chapter 2

JAX-RS In Action

JBoss User Group
김병곤(fharenheit@gmail.com)

□ HTTP Method에 대해서 기본으로 4개의 annotation 제공

- @javax.ws.rs.GET – 정보의 획득
- @javax.ws.rs.PUT – 정보의 추가
- @javax.ws.rs.POST – 정보의 변경
- @javax.ws.rs.DELETE – 정보의 삭제

□ Body 없이 헤더만 처리하는 경우

- @javax.ws.rs.HEAD

```
@Path("/customers")
public class CustomerService {
    @GET
    @Produces("application/xml")
    public String getAllCustomers() {
    }
}
```


URI Match: @Path annotation

```
@Path("/orders")
public class OrderResource {

    @GET
    public String getAllOrders() { // URI: GET /orders
        ...
    }

    @GET
    @Path("unpaid")
    public String getUnpaidOrders() { // URI: GET /orders/unpaid
        ...
    }
}
```

URI Match: @Path Expressions

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Path("{id}") // URI: GET /customers/1
    public String getCustomer(@PathParam("id") int id) {
        ...
    }
    @GET
    @Path("{firstname}-{lastname}") // URI: GET /customers/edward-kim
    public String getCustomer(@PathParam("firstname") String first,
                              @PathParam("lastname") String last) {
        ...
    }
}
```

URI Match: @Path Expressions

Regular Expression

```
@Path("/customers")
```

```
public class CustomerResource {
```

```
    @GET
```

```
    @Path("{id : \\d+}") // URI: GET /customers/1231
```

```
    public String getCustomer(@PathParam("id") int id) {
```

```
        ...
```

```
    }
```

```
    @GET
```

```
    @Path("{id : .+}/address") // URI: GET /customers/edward/kim/address
```

```
    public String getAddress(@PathParam("id") String id) {
```

```
        ...
```

```
    }
```

```
}
```

URI Match: Encoding

- URI를 구성하는데 허용하는 문자가 존재 하면서 예약어도 있음
 - US-ASCII 알파벳(a-z A-Z) 문자
 - 0~9 까지의 정수 문자
 - 기타 다른 문자 : _-! . ~ ' () *
 - URI에서 예약되어 있는 문자 : , ; : \$ & + = ? / [] @
- 예약되어 있는 특수 문자의 처리할 필요 있음

```
@GET
```

```
@Path("roy&fielding") // URI: GET /customers/roy%26fielding
```

```
public String getOurBestCustomer() {
```


```
    ...
```

```
}
```

URI Match: Matrix Parameter

❑ Matrix Parameter를 가진 URI

- <http://example.cars.com/mercedes/e55;color=black/2006>



Matrix Parameter

```
@Path("/mercedes")
public class MercedesService {
    @GET
    @Path("/e55/{year}")
    @Produces("image/jpeg")
    public Jpeg getE55Picture(@PathParam("year") String year) {
        ...
    }
}
```


- ❑ URI에서 파라미터를 꼭 뽑아야 하는 것은 아님
- ❑ 경우에 따라서 헤더, Query, Form, Cookie 등에서도 뽑아야 함
- ❑ JAX-RS는 파라미터 이외의 정보에 접근하는 방법을 제공

JAX-RS Injection: Basic

- ❑ `@javax.ws.rs.PathParam`
- ❑ `@javax.ws.rs.MatrixParam`
- ❑ `@javax.ws.rs.QueryParam`
- ❑ `@javax.ws.rs.FormParam`
- ❑ `@javax.ws.rs.HeaderParam`
- ❑ `@javax.ws.rs.CookieParam`
- ❑ `@javax.ws.rs.core.Context`

JAX-RS Injection: PathSegment와 Matrix Parameter

```
@Path("/cars/{make}")
public class CarResource {
    @GET
    @Path("/{model}/{year}") // URI: /cars/mercedes/e55;color=black/2006
    @Produces("image/jpeg")
    public Jpeg getPicture(@PathParam("make") String make,
                           @PathParam("model") PathSegment car,
                           @PathParam("year") String year) {
        String carColor = car.getMatrixParameters().getFirst("color");
        ...
    }
}
```



JAX-RS Injection: URI Information

```
public interface UriInfo {
    public String getPath();
    public String getPath(boolean decode);
    public List<PathSegment> getPathSegments();
    public List<PathSegment> getPathSegments(boolean decode);
    public MultivaluedMap<String, String> getPathParameters();
    ...
}

@Path("/cars/{make}")
public class CarResource {
    @GET
    @Path("/{model}/{year}")
    @Produces("image/jpeg")
    public Jpeg getPicture(@Context UriInfo info) {
        String make = info.getPathParameters().getFirst("make");
        PathSegment model = info.getPathSegments().get(1);
        String color = model.getMatrixParameters().getFirst("color");
        ...
    }
}
```

JAX-RS Injection: @QueryParam annotation

❑ GET /customers?**start=0**&**size=10**

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Produces("application/xml")
    public String getCustomers(@QueryParam("start") int start,
                              @QueryParam("size") int size) {
        ...
    }
}
```

JAX-RS Injection: @FormParam annotation

```
<FORM action="http://example.com/customers" method="post">
```

```
<P>
```

```
    First name: <INPUT type="text" name="firstname"><BR>
```

```
    Last name: <INPUT type="text" name="lastname"><BR>
```

```
    <INPUT type="submit" value="Send">
```

```
</P>
```

```
</FORM>
```

```
@Path("/customers")
```

```
public class CustomerResource {
```

```
    @POST
```

```
    public void createCustomer(@FormParam("firstname") String first,  
                               @FormParam("lastname") String last) {
```

```
        ...
```

```
    }
```

```
}
```

JAX-RS Injection: @HeaderParam annotation

```
@Path("/myservice")
public class MyService {
    @GET
    @Produces("text/html")
    public String get(@HeaderParam("Referer") String referer) {
        ...
    }
}
```

JAX-RS Injection: @CookieParam annotation

```
@Path("/myservice")
public class MyService {
    @GET
    @Produces("text/html")
    public String get(@CookieParam("customerId") int custId) {
        ...
    }
}
```



JAX-RS Injection: @DefaultValue annotation

```
@Path("/customers")
public class CustomerResource {

    @GET
    @Produces("application/xml")
    public String getCustomers(@DefaultValue("0") @QueryParam("start") int start,
                              @DefaultValue("10") @QueryParam("size") int size) {
        ...
    }
}
```

JAX-RS Injection: @Encoded annotation

- HTTP 표준에 따라서 URI, Matrix, Query, Form 파라미터는 모두 인코딩 되어야 함
- JAX-RS는 기본으로 파라미터를 디코딩한 후 자바로 형변환
- 디코딩 하지 않고 인코딩한 값을 그대로 받으려면?

```
@GET
```

```
@Produces("application/xml")
```

```
public String get(@Encoded @QueryParam("something") String str) {...}
```

- ❑ REST는 HTTP 프로토콜 그 자체에 집중하므로 응답 코드도 지극히 HTTP 프로토콜 답다
 - 200~399 : Successful HTTP Response
 - 200 : OK (HTTP Message Body O)
 - 204 : OK but No Content (HTTP Message Body X)
 - 400~599 : Error HTTP Response
 - 404 : Not Found (URI가 없는 경우)
 - 406 : Not Acceptable (accept-header가 맞지 않는 경우)
 - 405 : Method Not Allowed (PUT, GET, DELETE, POST 등이 맞지 않는 경우)
- ❑ 클라이언트에게 적절하게 응답 코드를 돌려주는 것은 매우 중요
 - Ajax 작성시에 응답의 내용에만 집중하는 경향이 있음
 - 응답 코드도 중요함

Response Object

```
public abstract class Response {
    ...
    public static ResponseBuilder status(Status status) {...}
    public static ResponseBuilder status(int status) {...}
    public static ResponseBuilder ok() {...} // 200
    public static ResponseBuilder ok(Object entity) {...}
    public static ResponseBuilder ok(Object entity, MediaType type) {...}
    public static ResponseBuilder ok(Object entity, String type) {...}
    public static ResponseBuilder ok(Object entity, Variant var) {...}
    public static ResponseBuilder serverError() {...}
    public static ResponseBuilder created(URI location) {...}
    public static ResponseBuilder noContent() {...}
    public static ResponseBuilder notModified() {...}
    public static ResponseBuilder notModified(EntityTag tag) {...}
    public static ResponseBuilder seeOther(URI location) {...}
    public static ResponseBuilder temporaryRedirect(URI location) {...}
    public static ResponseBuilder notAcceptable(List<Variant> variants) {...}
    public static ResponseBuilder fromResponse(Response response) {...}
    ...
}
```

Response Object

```
@Path("/textbook")
public class TextBookService {
    @GET
    @Path("/restfuljava")
    @Produces("text/plain")
    public Response getBook() {
        String book = ...;
        ResponseBuilder builder = Response.ok(book);
        builder.language("fr").header("Some-Header", "some value");
        return builder.build();
    }
}
```

Exception Handling

- ❑ JAX-RS의 기본 예외: `javax.ws.rs.WebApplicationException`
 - `RuntimeException`을 상속받아서 try catch 필요없음
 - JAX-RS가 이 예외를 적절하게 처리

```
public class WebApplicationException extends RuntimeException {  
    public WebApplicationException() {...}  
    public WebApplicationException(Response response) {...}  
    public WebApplicationException(int status) {...}  
    public WebApplicationException(Response.Status status) {...}  
    public WebApplicationException(Throwable cause) {...}  
    public WebApplicationException(Throwable cause, Response response) {...}  
    ...  
}
```

Exception Handling

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Path("{id}")
    @Produces("application/xml")
    public Customer getCustomer(@PathParam("id") int id) {
        Customer cust = findCustomer(id);
        if (cust == null) {
            throw new WebApplicationException(Response.Status.NOT_FOUND);
        }
        return cust;
    }
}
```

- 내용의 유형을 판단하는 헤더
 - Accept: 응답을 받는쪽이 원하는 미디어 유형
 - Content-Type: 응답을 보내는 쪽의 미디어 유형

GET http://example.com/stuff

Accept: application/xml, application/json

Contents Negotiation

응답 헤더

[view source](#)

```
Date Fri, 09 Jul 2010 17:45:20 GMT
Server Apache
Cache-Control no-cache, no-store, must-revalidate, no-cache, no-store, must-revalidate
Pragma no-cache, no-cache
P3P CP="CAO DSP CURa ADMa TAIA PSAa OUR LAW STP PHY ONL UNI PUR FIN COM NAV INT DEM STA PRE"
Vary Accept-Encoding, User-Agent
Content-Encoding gzip
Connection close
Transfer-Encoding chunked
Content-Type text/html; charset=UTF-8
```

요청 헤더

[view source](#)

```
Host www.naver.com
User-Agent Mozilla/5.0 (Windows; U; Windows NT 6.1; ko; rv:1.9.2.6) Gecko/20100625 Firefox/3.6.6
Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language ko-kr,ko;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding gzip,deflate
Accept-Charset EUC-KR,utf-8;q=0.7,*;q=0.7
Keep-Alive 115
Connection keep-alive
Cookie npic=gn5t3YHm+cL71HGdRUuSoFR6WebQUIhzyKJy0ywf hRt y8hH/srHhat n19MZ7kESzCA==; NB=GMZT1MZZGAYT
; DA_HC=LZ41465580, LB: refreshx=0
Cache-Control max-age=0
```

Contents Negotiation: Preference Ordering

GET http://example.com/stuff

Accept: text/*, text/html;level=1, */*, application/xml

1. text/html
2. application/xml
3. text/*
4. */*

Contents Negotiation: @Consumes @Produces

❑ @Consumes annotation

- 미디어 유형 지정시 사용
- 클라이언트가 보낸 요청을 처리할 수 있음을 명시
- 요청의 Content-Type으로 판단

❑ @Produces annotation

- 미디어 유형 지정시 사용
- 서버가 클라이언트로 보낼때 미디어 유형
- 클라이언트의 요청에 대해서 Accept 헤더로 판단하고
- 응답은 Content-Type에 미디어 유형을 넣는다.

Contents Negotiation


요청의 Accept 헤더로 판단

```
@Path("/customers")
public class CustomerResource {
    @GET
    @Path("{id}")
    @Produces("application/xml")
    public Customer getCustomerXml(@PathParam("id") int id) {...}

    @GET
    @Path("{id}")
    @Produces("text/plain")
    public String getCustomerText(@PathParam("id") int id) {...}

    @GET
    @Path("{id}")
    @Produces("application/json")
    public Customer getCustomerJson(@PathParam("id") int id) {...}
}
```

GET http://example.com/customers/1
Accept: application/json;q=1.0, application/xml;q=0.5



```
<customers>
  <link rel="next" href="http://example.com/customers?start=2&size=2"
        type="application/xml"/>
  <customer id="123">
    <name>Bill Burke</name>
  </customer>
  <customer id="332">
    <name>Roy Fielding</name>
  </customer>
</customers>
```

- ❑ URI 설계가 중요
 - 모든 주문 : `http://<IP>/orders`
 - 1234 주문 : `http://<IP>/orders/1234`
 - 1234 주문 취소 : `http://<IP>/orders/1234/cancel`
- ❑ JAXB를 이용하여 XML/JSON을 모두 처리할 수 있도록 구현
 - XSD 기반 메시지 설계
- ❑ Response에 대한 설계 중요
 - 정상 처리지만 데이터가 없는 경우
 - 404 Not Found?
 - 204 No Content?
- ❑ Domain Model 설계시 일관성 있는 표현이 가능하도록 설계
 - XML/JSON – Domain Object – Table

Q & A

Chapter 3

JAXB와 XML/JSON

JBoss User Group
김병곤(fharenheit@gmail.com)

- JAX-RS를 기반으로 보다 쉽게 애플리케이션을 구현하는 방법
 - ➔ 생산성 확보 방안
- JSON/XML을 고려하지 않고 코딩하는 방법
- 클라이언트에 따라서 JSON/XML을 선택해서 보내주는 방법

자바 개발자는 좀더 편리한 것을 선호한다

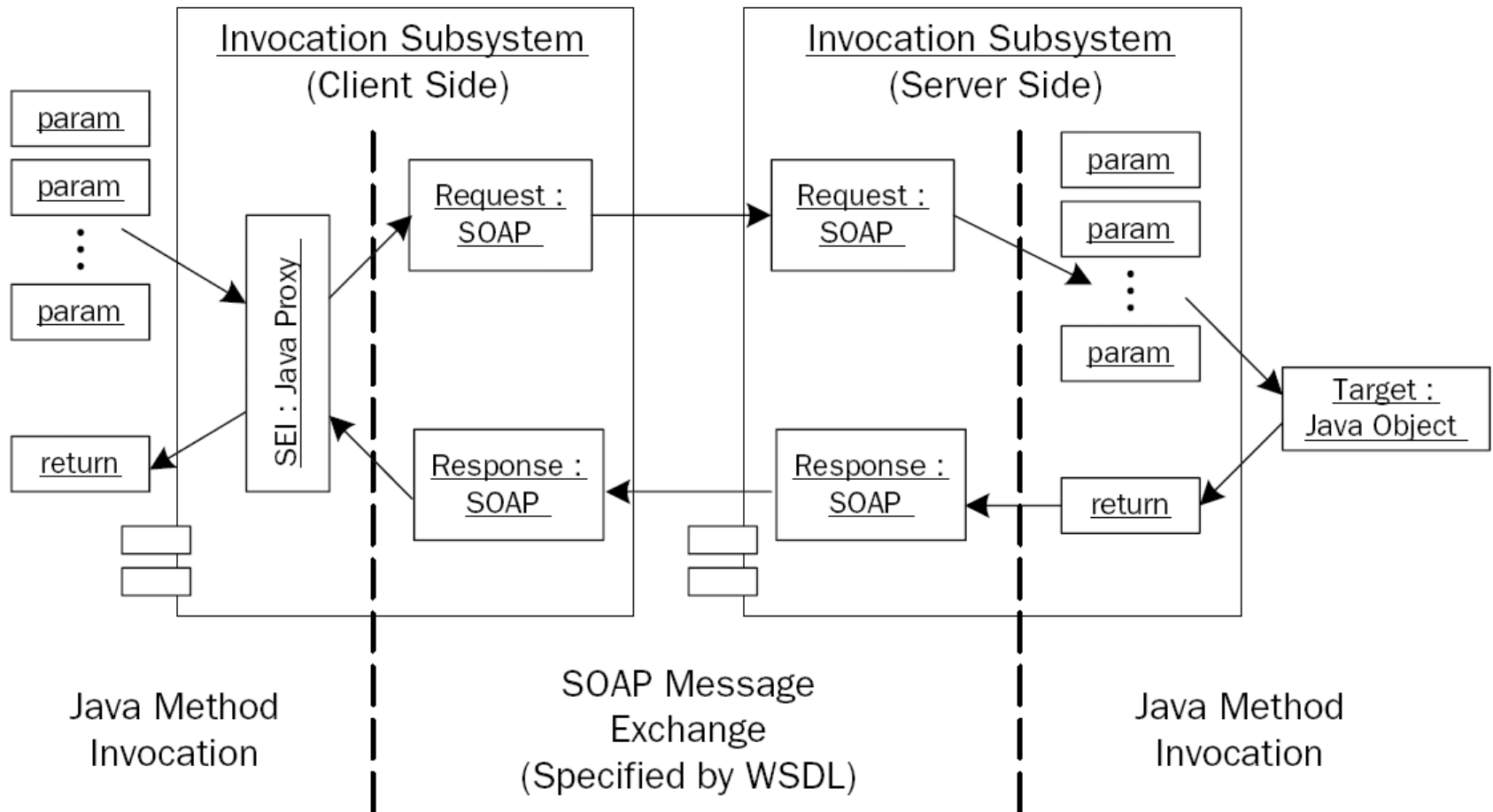
- 시스템로 사용하는 다양한 데이터 타입
 - Ajax는 주로 JSON
 - 이기종 장비는 XML
 - 모바일은 JSON과 XML

- 코드는 최대한 단순하게 작성하고 JSON/XML을 자동으로 처리할 수 있다면?
 - 개발자는 자바 언어를 기반으로한 코드 작성이 쉽다
 - 시스템은 통신시 JSON/XML을 이용한다.

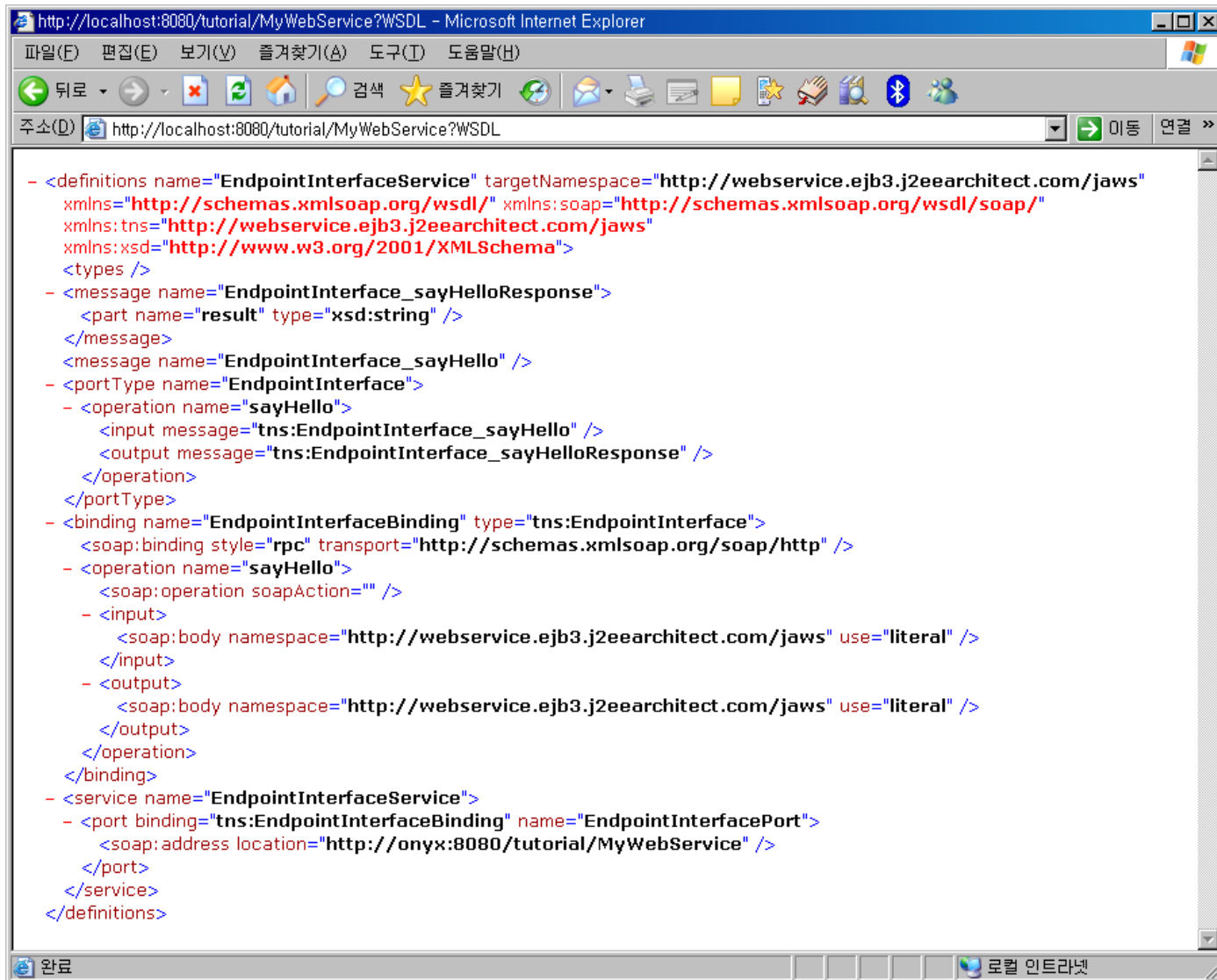
➔ 자바를 사용하면 자동으로 JSON/XML을 JAX-RS가 생성해주면 좋겠다

- ❑ SOAP이라는 XML 기반 메시지로 시스템이 통신
- ❑ 이기종 시스템간 호환성을 보장
- ❑ 인터페이스 명세서는 WSDL 파일
- ❑ WSDL을 로딩하여 상대방 시스템을 호출하는 Stub 코드를 생성
- ❑ 흔히 웹 서비스라고 하면 SOAP 기반 웹 서비스
 - Java 규격 : JAX-RPC, JAX-WS

WebServices Invocation과 SOAP Message



WebService WSDL 파일



```
<?xml version='1.0' encoding='UTF-8'?>
- <definitions name="EndpointInterfaceService" targetNamespace="http://webservice.ejb3.j2eeearchitect.com/jaws"
  xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://webservice.ejb3.j2eeearchitect.com/jaws"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types />
  - <message name="EndpointInterface_sayHelloResponse">
    <part name="result" type="xsd:string" />
  </message>
  <message name="EndpointInterface_sayHello" />
  - <portType name="EndpointInterface">
    - <operation name="sayHello">
      <input message="tns:EndpointInterface_sayHello" />
      <output message="tns:EndpointInterface_sayHelloResponse" />
    </operation>
  </portType>
  - <binding name="EndpointInterfaceBinding" type="tns:EndpointInterface">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    - <operation name="sayHello">
      <soap:operation soapAction="" />
      - <input>
        <soap:body namespace="http://webservice.ejb3.j2eeearchitect.com/jaws" use="literal" />
      </input>
      - <output>
        <soap:body namespace="http://webservice.ejb3.j2eeearchitect.com/jaws" use="literal" />
      </output>
    </operation>
  </binding>
  - <service name="EndpointInterfaceService">
    - <port binding="tns:EndpointInterfaceBinding" name="EndpointInterfacePort">
      <soap:address location="http://onyx:8080/tutorial/MyWebService" />
    </port>
  </service>
</definitions>
```

SOAP Message

□ 웹 서비스의 메시지인 SOAP 메시지는 XML로 구성

```
POST /InStock HTTP/1.1
```

```
Host: www.example.org
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: nnn
```

```
<?xml version="1.0"?>
```

```
<soap:Envelope
```

```
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

```
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
<soap:Body xmlns:m="http://www.example.org/stock">
```

```
<m:GetStockPrice>
```

```
<m:StockName>IBM</m:StockName>
```

```
</m:GetStockPrice>
```

```
</soap:Body>
```

```
</soap:Envelope>
```

public String getStockPrice()

SOAP

HTTP

- ❑ XML과 Java Object를 상호 변환하는 규격
- ❑ 웹 서비스를 구성하는 핵심 규격 중에 하나
 - 메소드의 인자가 객체인 경우 SOAP 메시지 송수신시 상호 변환이 필요하기 때문에
- ❑ XSD를 정의하고 JAXB의 xjc를 통해 Java Object를 자동 생성
- ❑ Marshaller/Unmarshaller를 이용하여 XSD \leftrightarrow Java 상호 변환 (바인딩)

JAXB를 쓰면 좋은 이유

- ❑ XML의 스키마인 XSD를 중심으로 설계가 가능
 - 주고 받는 메시지만 집중

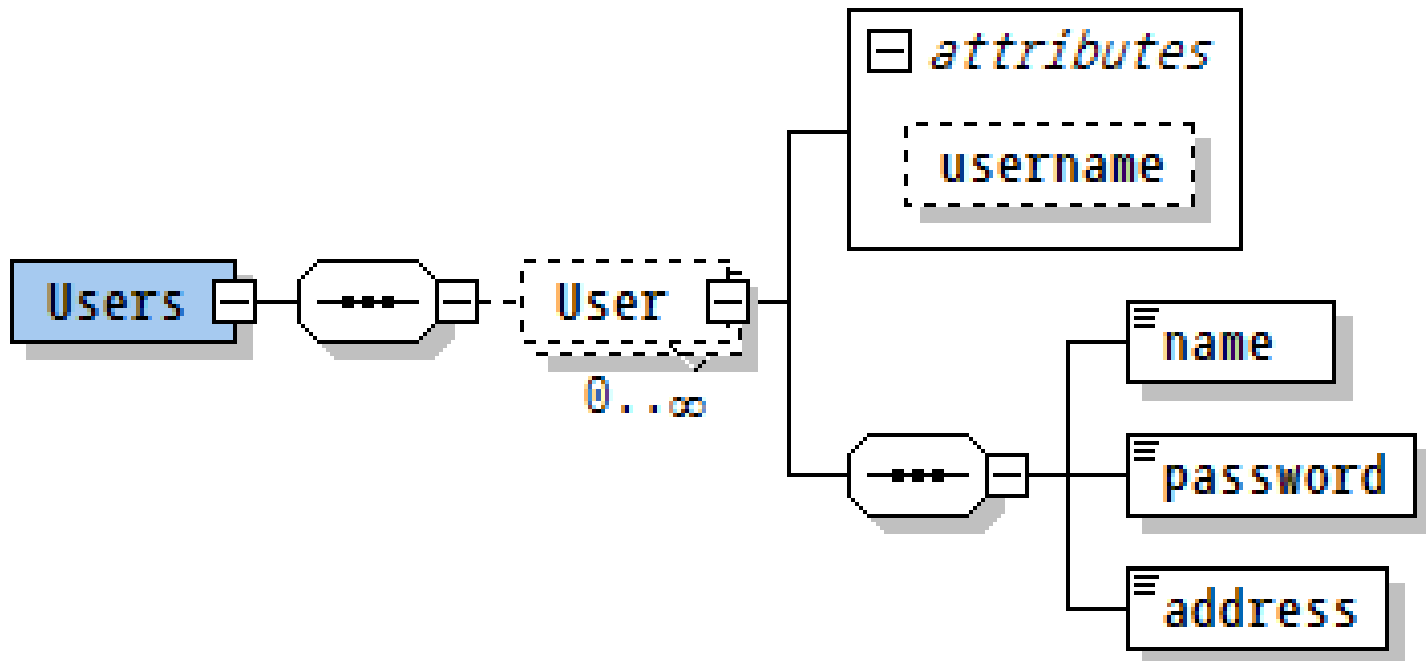
- ❑ JSON/XML을 별도의 코딩 없어 동시에 처리 가능
 - @Produces의 MediaType이 무엇이냐에 따라서 자동 처리
 - XML로 응답 : `MediaType.APPLICATION_XML`
 - JSON으로 응답 : `MediaType.APPLICATION_JSON`

- ❑ 개발 기간 단축

- ❑ 단순한 코드

JAXB Object 생성 절차 (1)

1. Altova XMLSpy/Oxygen XML Editor 등을 이용하여 XSD 정의



JAXB Object 생성 절차 (2)

2. JAXB의 xjc로 XSD의 Java Object를 생성

```
<project name="generate" default="generate" basedir=".">
  <property name="maven.repository" value="${user.home}/.m2/repository"/>
  <taskdef name="xjc" classname="com.sun.tools.xjc.XJCTask">
    <classpath>
      <fileset dir="${maven.repository}/javax/xml/bind/jaxb-api/2.1">
        <include name="**/*.jar"/>
      </fileset>
      ....
      <fileset dir="${maven.repository}/com/sun/xml/bind/jaxb-xjc/2.1.9">
        <include name="**/*.jar"/>
      </fileset>
    </classpath>
  </taskdef>
  <target name="generate">
    <xjc schema="${basedir}/src/main/webapp/movies.xsd" destdir="${basedir}/src/main/java"
      package="com.jbossug.seminar.rest"/>
    <echo>Completed...</echo>
  </target>
</project>
```

JAXB XJC 정의

XJC로 XSD의 Java Object 생성

JAXB Java Object for XSD

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = { "user" })
@XmlRootElement(name = "Users")
public class Users {
    @XmlElement(name = "User")
    protected List<Users.User> user;
    ...

    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "", propOrder = { "name", "password", "address" })
    public static class User {

        @XmlElement(required = true)
        protected String name;
        @XmlElement(required = true)
        protected String password;
        @XmlElement(required = true)
        protected String address;
        @XmlAttribute
        protected String username;
        ...
    }
}
```

JAXB의 각종 annotation



JAX-RS REST Controller

```
@Path("/users")
public class UserResource {

    @GET
    @Produces({MediaType.APPLICATION_XML}) // HTTP Response로 XML을 송신
    public Users getUsers() { // Users가 JAXB Object이므로 JAX-RS는 자동으로 XML로 변환
        Users users = new Users();

        Users.User user = new Users.User();
        user.setUsername("1111");
        user.setAddress("Seoul");
        user.setName("Hong");

        users.getUser().add(user);
        return users;
    }
    ...
}
```

DEMO

Q & A

Chapter 4

Spring Framework Integration

JBoss User Group
김병곤(fharenheit@gmail.com)

- ❑ JAX-RS 규격의 백엔드로 무엇을 쓸것인가?
- ❑ 기존의 백엔드를 그냥 활용할 수 없을까?
- ❑ 어떻게 통합해야 할까?

JAX-RS + Spring Framework

- ❑ JAX-RS가 표준 규격이고 HTTP 처리만 다룸
- ❑ HTTP 요청 이외 부분에서 강력한 지원 기능이 필요
- ❑ Spring Framework과 같은 IoC Container 통합을 고려
 - Spring WebMVC와 어떻게 Integration할 것인가?
 - RESTEasy가 Spring Framework을 잘 지원하는가?
 - Spring WebMVC annotation과 JAX-RS의 annotation 충돌?

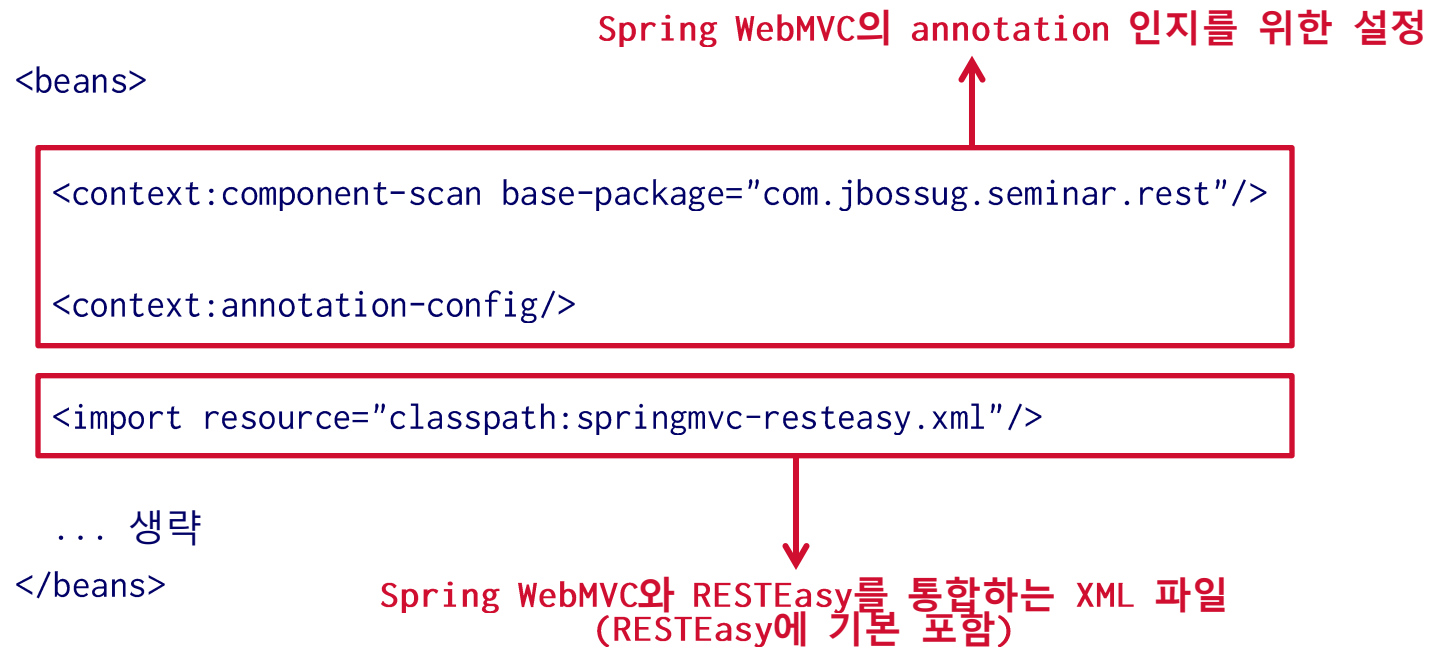
□ Spring Framework Dispatcher Servlet 정의

● 기존 방식 그대로

```
<web-app>
...
<servlet>
  <display-name>Spring Framework Dispatcher Servlet</display-name>
  <servlet-name>main</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>main</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
...
</web-app>
```

Spring Framework Application Context XML

□ WEB-INF/main-servlet.xml 파일



JAX-RS + Spring WebMVC 기반 Controller

- 하나의 컨트롤러에 JAX-RS + Spring annotation이 모두 포함

```
@Controller
@Path("/rest")
public class MovieResource {
    @GET
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Path("movies") // URI is /rest/movies
    public Dataset getMovies() {
        Dataset dataset = new Dataset();
        dataset.setTotal("2");
        Dataset.Row row1 = new Dataset.Row();
        row1.setId("1");
        row1.setTitle("Office Space");
        row1.setDirector("Mike Judge");
        dataset.getRow().add(row1);
        return dataset;
    }
}
```

Q & A

Chapter 5

Unit/Integration Test

JBoss User Group
김병곤(fharenheit@gmail.com)

- HTTP 프로토콜을 다루는 JAX-RS을 어떻게 하면 쉽게 단위 시험을 해볼까?
- GET/POST/DELETE/PUT Method를 어떻게 테스트해볼까?
- 통신을 하지 않고 Mock Object를 이용한 테스트는?

RESTful WebServices의 테스트

- RESTful WebServices는 HTTP Protocol Layer를 통해 호출한다.
 - Web Container가 필요하다
 - 테스트시 Web Container를 구동해야 하고
 - 웹 애플리케이션이 배포되어야 하며
 - 테스트 코드가 실행되어야 한다.
 - Client(예; Android)의 동작 시뮬레이션 테스트에 집중
 - 테스트 코드를 작성하면서 Android 호출 코드가 동시에 작성됨
- HTTP Protocol Layer 없이 Server-Side를 직접 호출하는 방식
 - Web Container가 필요하지 않다
 - JAX-RS Implementation에서 Mock 제공
 - Server-Side 코드를 Protocol Layer를 통하지 않고 테스트 가능
 - Server-Side 코드의 테스트에 집중

Integration Test

Unit Test

Test를 위한 RESTEasy의 지원

- ❑ Client Framework : 직접 Server-Side와 통신하는 방식
 - Apache HttpClient
 - HttpURLConnection
 - RESTEasy ClientRequest/ClientResponse
 - RESTEasy Proxy Framework
 - Client에서 JAX-RS annotation 인식하여 호출

- ❑ Server-Side Mock Framework
 - RESTEasy를 Standalone으로 동작하도록 하는 Embedded Container를 지원
 - Mock HttpRequest/HttpResponse 제공

URLConnection: GET

□ HTTP Body가 존재하지 않는 HTTP GET

```
URL getUrl = new URL("http://localhost:8080/customers/1");
URLConnection connection = (URLConnection) getUrl.openConnection();
connection.setRequestMethod("GET");

BufferedReader reader = new BufferedReader(
    new InputStreamReader(connection.getInputStream()));

String line = reader.readLine();
while (line != null) {
    System.out.println(line);
    line = reader.readLine();
}

connection.disconnect();
```

HTTP GET을 처리하는 Server-Side 코드

□ HTTP Body가 존재하지 않는 HTTP GET

```
@GET
@Path("/{id}")
@Produces("application/xml")
public StreamingOutput getCustomer(@PathParam("id") int id) {
    final Customer customer = customerDB.get(id);
    if (customer == null) { // 고객이 없는 경우 404 NOT FOUND
        throw new WebApplicationException(Response.Status.NOT_FOUND);
    }
    return new StreamingOutput() {
        public void write(OutputStream outputStream)
                                throws IOException, WebApplicationException {
            PrintStream writer = new PrintStream(os);
            writer.println("<customer> ... </customer>");
        }
    };
}
```


URLConnection: POST

□ HTTP Body에 XML을 넣어 전달하는 HTTP POST

```
String newCustomer = "<customer>...</customer>";
```

```
URL postUrl = new URL("http://localhost:8080/customers");
```

```
URLConnection connection = (URLConnection) postUrl.openConnection();
```

```
connection.setDoOutput(true); // XML 내용을 전달하기 위해서 출력 스트림을 사용
```

```
connection.setInstanceFollowRedirects(false); // Redirect 처리 하지 않음
```

```
connection.setRequestMethod("POST");
```

```
connection.setRequestProperty("Content-Type", "application/xml");
```

```
OutputStream os = connection.getOutputStream();
```

```
os.write(newCustomer.getBytes());
```

```
os.flush();
```

```
Assert.assertEquals(URLConnection.HTTP_CREATED, connection.getResponseCode());
```

```
System.out.println("Location: " + connection.getHeaderField("Location"));
```

```
connection.disconnect();
```

HTTP POST을 처리하는 Server-Side 코드

□ HTTP Body가 존재하는 HTTP POST

@POST

```
@Consumes("application/xml")
public Response createCustomer(InputStream is) {
    Customer customer = readCustomer(is); // HTTP Body에서 XML 읽기
    customer.setId(idCounter.incrementAndGet());
    customerDB.put(customer.getId(), customer); // DB INSERT
    System.out.println("Created customer " + customer.getId());

    // 생성한 고객에 접근하는 URI를 리턴
    return Response.created(URI.create("/customers/" + customer.getId())).build();
}
```

HttpURLConnection: PUT

□ HTTP Body에 XML을 넣어 전달하는 HTTP PUT

```
String updateCustomer = "<customer>...</customer>";

URL getUrl = new URL("http://localhost:8080/customers/1");
HttpURLConnection connection = (HttpURLConnection) getUrl.openConnection();
connection.setDoOutput(true);
connection.setRequestMethod("PUT");
connection.setRequestProperty("Content-Type", "application/xml");

OutputStream os = connection.getOutputStream();
os.write(updateCustomer.getBytes());
os.flush();

Assert.assertEquals(HttpURLConnection.HTTP_NO_CONTENT, connection.getResponseCode());
connection.disconnect();
```

HTTP PUT을 처리하는 Server-Side 코드

□ HTTP Body가 존재하는 HTTP PUT

@PUT

@Path("{id}")

@Consumes("application/xml")

```
public void updateCustomer(@PathParam("id") int id, InputStream is) {  
    Customer update = readCustomer(is); // 클라이언트가 송신한 XML  
    Customer current = customerDB.get(id); // Customer Id  
    if (current == null) throw new WebApplicationException(Response.Status.NOT_FOUND);  
  
    current.setFirstName(update.getFirstName());  
    current.setLastName(update.getLastName());  
    current.setStreet(update.getStreet());  
    current.setState(update.getState());  
    current.setZip(update.getZip());  
    current.setCountry(update.getCountry());  
}
```

URLConnection: DELETE

□ HTTP Body가 존재하지 않는 HTTP DELETE

```
URL getUrl = new URL("http://localhost:8080/customers/1");

URLConnection connection = (URLConnection) getUrl.openConnection();
connection.setRequestMethod("DELETE");

Assert.assertEquals(URLConnection.HTTP_NO_CONTENT,
                    connection.getResponseCode());
connection.disconnect();
```

HTTP DELETE을 처리하는 Server-Side 코드

□ HTTP Body가 존재하지 않는 HTTP DELETE

```
@DELETE
@Path("/{id}")
@Produces("application/xml")
public void deleteCustomer(@PathParam("id") int id) {
    final Customer customer = customerDB.get(id);
    if (customer == null) {
        throw new WebApplicationException(Response.Status.NOT_FOUND);
    } else {
        customerDB.remove(id);
    }
}
```

Apache Http Client

```
DefaultHttpClient client = new DefaultHttpClient();

HttpGet get = new HttpGet("http://localhost:8080/customers");
HttpResponse response = client.execute(get);
Assert.assertEquals(200, response.getStatusLine().getStatusCode());

BufferedReader reader = new BufferedReader(
    new InputStreamReader(response.getEntity().getContent()));

String line = reader.readLine();
while (line != null) {
    System.out.println(line);
    line = reader.readLine();
}
```

RESTEasy Client Framework

- ❑ RESTEasy에서 제공하는 ClientRequest/ClientResponse
 - ❑ 요청 및 응답 처리가 매우 단순하여 유용
 - 실제 호출은 다른 구현체로 위임(by ClientExecutor)
 - Apache HttpClient (by ApacheHttpClient4Executor)
 - URLConnection (by URLConnectionClientExecutor)
 - ❑ 요청 송신시 HTTP Method와 매핑하는 메소드 제공
 - ClientRequest.get(), ClientRequest.post() ...
 - ❑ ClientRequest.getTarget() 메소드를 이용하여 HTTP 응답을 적절한 형으로 변환
- ```
String response = clientRequest.getTarget(String.class);
Customer customer = clientRequest.getTarget(Customer.class);
```



# RESTEasy Client Framework

---

```
String url = "http://localhost:8080/customers";

ClientRequest request = new ClientRequest(url);
String output = request.getTarget(String.class);

System.out.println("** XML from " + url);
System.out.println(output);

Customers customers = request.getTarget(Customers.class);
```

# RESTEasy Proxy 기반 테스트: Server-Side

```
@Path("/customers")
public interface CustomerResource { // Proxy 기반 테스트를 위해서 Interface가 필요
 @POST
 @Consumes("application/xml")
 Response createCustomer(InputStream is);
}

public class CustomerResourceImpl implements CustomerResource {
 private Map<Integer, Customer> customerDB = new ConcurrentHashMap<Integer, Customer>();
 private AtomicInteger idCounter = new AtomicInteger();
 public Response createCustomer(InputStream is) {
 Customer customer = readCustomer(is);
 int id = idCounter.incrementAndGet();
 customer.setId(id);
 System.out.println("Created customer's id is " + id);
 customerDB.put(customer.getId(), customer);
 System.out.println("Created customer " + customer.getId());
 return Response.created(URI.create("/customers/" + customer.getId())).build();
 }
}
```

# RESTEasy Proxy 기반 테스트: Client Proxy

```
String newCustomer = "<customer>"
+ "<first-name>Bill</first-name>"
+ "<last-name>Burke</last-name>"
+ "<street>256 Clarendon Street</street>"
+ "<city>Boston</city>"
+ "<state>MA</state>"
+ "<zip>02115</zip>"
+ "<country>USA</country>"
+ "</customer>";
```

CustomerResource 인터페이스의 Proxy 생성

```
CustomerResource customerService =
 ProxyFactory.create(CustomerResource.class, "http://localhost:8080");
```

```
ByteArrayInputStream is = new ByteArrayInputStream(newCustomer.getBytes("UTF-8"));
```

```
Response response = customerService.createCustomer(is); // 직접 호출
```

```
Assert.assertEquals(response.getStatus(), 201);
```

```
MultivaluedMap<String, Object> map = response.getMetadata();
```

```
Assert.assertEquals(map.get("Location").get(0), "http://localhost:8080/customers/0");
```

# Apache Maven & Jetty Integration

---

- ❑ Client가 Server-Side 로직을 HTTP 프로토콜 레이어를 통해 호출하는 경우 프로토콜 레이어가 필요하게 됨
- ❑ Apache Maven Project의 경우 Jetty와 Integration시 Integration Test가 가능
  1. Jetty Start
  2. Integration Test
  3. Jetty Stop
- ❑ Maven의 Integration Test Phase에 Jetty를 구동하고 실제 배포 과정을 진행한 후 테스트하는 방식

# Maven Lifecycle Phase 매핑

- ❑ Client가 Server-Side를 HTTP 프로토콜 레이어를 통과하는 경우 Web Container가 필요함
- ❑ Jetty or Tomcat을 Maven의 Integration Test Phase에 통합하여 테스트 전에 배포 작업을 진행

| Maven Lifecycle Phase | Action                                                          |
|-----------------------|-----------------------------------------------------------------|
| Test Phase            | Test 하지 않음                                                      |
| Pre Integration Test  | Jetty Start                                                     |
| Integration Test      | Maven의 단위 테스트 플러그인인 Surefire 플러그인 동작<br>→ 테스트 소스코드를 실행하여 테스트 진행 |
| Post Integration Test | Jetty Stop                                                      |

# Surefire Plugin Configuration

```
<project>
 <build>
 <plugins>
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-surefire-plugin</artifactId> → Maven의 단위 테스트 플러그인 = Surefire
 <configuration>
 <skip>true</skip> → Maven Test Phase때 플러그인을 동작하지 않음
 </configuration>
 <executions>
 <execution>
 <id>surefire-it</id>
 <phase>integration-test</phase> → Maven Integration Test Phase때
 <goals>
 <goal>test</goal> → Surefire의 test goal을 실행
 </goals>
 <configuration>
 <skip>false</skip> → Maven의 Integration Phase때 플러그인을 동작하여 테스트 진행
 </configuration>
 </execution>
 </executions>
 </plugin>
 </plugins>
 </build>
</project>
```

# Jetty Configuration (1)

```
<project>
 <build>
 <plugins>
 <plugin>
 <groupId>org.mortbay.jetty</groupId>
 <artifactId>maven-jetty-plugin</artifactId>
 <version>6.1.15</version>
 <configuration>
 <contextPath>/</contextPath> → 웹 애플리케이션의 Context Path
 <scanIntervalSeconds>2</scanIntervalSeconds>
 <stopKey>foo</stopKey>
 <stopPort>9999</stopPort>
 <connectors>
 <connector implementation="org.mortbay.jetty.nio.SelectChannelConnector">
 <port>8080</port> → Jetty 구동 포트
 <maxIdleTime>60000</maxIdleTime>
 </connector>
 </connectors>
 </configuration>
 ... 생략
 </plugin>
 </plugins>
 </build>
</project>
```

# RESTEasy Server-Side Mock

```
// Embedded Container 생성
Dispatcher dispatcher = MockDispatcherFactory.createDispatcher();
POJOResourceFactory noDefaults = new POJOResourceFactory(CustomerResource.class);
dispatcher.getRegistry().addResourceFactory(noDefaults);
```

```
// Request XML
String newCustomer = "<customer>...</customer>";
```

```
// Mock Request & Repsonse 생성
MockHttpRequest request = MockHttpRequest.post("/customers");
ByteArrayInputStream is = new ByteArrayInputStream(newCustomer.getBytes("UTF-8"));
request.setInputStream(is);
request.contentType("application/xml");
MockHttpResponse response = new MockHttpResponse();
```

```
dispatcher.invoke(request, response);
```

```
Assert.assertEquals(201, response.getStatus()); // 정상적으로 Customer가 생성되면 201 코드
```



# Q & A

# Chapter 6

## JBoss RESTEasy

JBoss User Group  
김병곤(fharenheit@gmail.com)

# JAX-RS Implementation

---

- ❑ Jersey(<https://jersey.dev.java.net>)
  - Reference Implementation(RI)
  
- ❑ Apache CXF
  - OpenSource WebServices Framework
  - 최근 JAX-RS 지원 추가
  
- ❑ JBoss RESTEasy
  - Asynchronous HTTP(Server-Side Pushing) → COMET
  - Embedded Container와 JUnit을 이용한 단위 테스트 지원
  - GZIP Compression, Server-Side Caching, Browser Cache

- ❑ JAX-RS Implementation
- ❑ AppServer/Tomcat Portability
- ❑ Embedded Container for JUnit Testing
- ❑ Client Browser Cache
- ❑ Server In-Memory Cache
- ❑ Rich Provider Set : XML JSON YAML FastInfoset Multipart ...
- ❑ JAXB Marshalling Into XML JSON Jackson FastInfoset Atom ...
- ❑ GZIP Compression
- ❑ Asynchronous HTTP(Comet)
- ❑ Asynchronous Job Service
- ❑ Rich Interceptor Model
- ❑ EJB Seam Guice Spring

- ❑ MessageBodyReader/Writer Interceptors
- ❑ PreProcessInterceptor
- ❑ PostProcessInterceptors
- ❑ ClientExecutionInterceptors
- ❑ ...

```
<feed xmlns="http://www.w3.org/2005/Atom">
 <title>Example Feed</title>
 <subtitle>A subtitle.</subtitle>
 <link href="http://example.org/feed/" rel="self" />
 <link href="http://example.org/" />
 <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
 <updated>2003-12-13T18:30:02Z</updated>
 <author>
 <name>John Doe</name>
 <email>johndoe@example.com</email>
 </author>
 <entry>
 <title>Atom-Powered Robots Run Amok</title>
 <link href="http://example.org/2003/12/13/atom03" />
 <link rel="alternate" type="text/html" href="http://example.org/2003/12/13/atom03.html"/>
 <link rel="edit" href="http://example.org/2003/12/13/atom03/edit"/>
 <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
 <updated>2003-12-13T18:30:02Z</updated>
 <summary>Some text.</summary>
 </entry>
</feed>
```



```
@Path("atom")
public class MyAtomService {
 @GET
 @Path("feed")
 @Produces("application/atom+xml")
 public Feed getFeed() throws URISyntaxException {
 Feed feed = new Feed();
 feed.setId(new URI("http://example.com/42"));
 feed.setTitle("My Feed");
 feed.setUpdated(new Date());
 Link link = new Link();
 link.setHref(new URI("http://localhost"));
 link.setRel("edit");
 feed.getLinks().add(link);
 feed.getAuthors().add(new Person("Bill Burke"));
 ...
 return feed;
 }
}
```



- ❑ Request/Response를 GZIP으로 압축
  - Request : Accept-Encoding → gzip deflate
  - Response : Content-Encoding → gzip
- ❑ 압축을 지원하도록 헤더에 포함시키면 알아서 Compress/Uncompress

```
@Path("/")
public class MyService {

 @GET
 @Produces("application/xml")
 @GZIP
 public String getData() {...}
}
```



# Asynchronous HTTP (Comet)

```
@Path("/")
public class SimpleResource {

 @GET
 @Path("basic")
 @Produces("text/plain")
 public void getBasic(final @Suspend(10000) AsynchronousResponse response) throws Exception {
 Thread t = new Thread() {
 @Override
 public void run() {
 try {
 Response jaxrs = Response.ok("basic").type(MediaType.TEXT_PLAIN).build();
 response.setResponse(jaxrs);
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
 };
 t.start();
 }
}
```

# Tomcat Conf for Asynchronous HTTP (Comet)

## □ Tomcat의 server.xml 파일

```
<Connector port="8080" address="${jboss.bind.address}"
 emptySessionPath="true" protocol="org.apache.coyote.http11.Http11NioProtocol"
 enableLookups="false" redirectPort="6443"
 acceptorThreadCount="2" pollerThreadCount="10"
>
```

## □ 웹 애플리케이션의 WAR#WEB-INF/web.xml 파일

```
<servlet>
 <servlet-name>Resteasy</servlet-name>
 <servlet-class>
 org.jboss.resteasy.plugins.server.servlet.Tomcat6CometDispatcherServlet
 </servlet-class>
</servlet>
```

# Q & A