

Les services Web Java EE

François Robert

francois.robert@shipstone.org



1

Le plan !

- Java EE & REST, définition
- JAXB
- SOAP
- REST
- Services REST vs SOAP ?
- Introduction WOA

3

De quoi parle t'on ?

- Java EE
- Service Web
- SOAP, REST & RESTFul
- Web Oriented Architecture

2

Les outils du cours

- VM Linux (xubuntu)
- Java 8
- IDEs
 - IntelliJ
 - Netbeans
- Maven
- Serveur d'application
 - Glassfish
 - Wildfly
 - Jetty (container de servlet)
- Git
- shell



maven

4



Initialisation



- Lancer la VM
 - login : siad - password : javaeesiad
- Ouvrez une console et rendez vous dans le repertoire de dev
 - Créez un repertoire workspaces et placez vous dedans
 - Astuce :mkdir workspaces && cd \$_
- Il s'agit maintenant de cloner le repository de source (hébergé par github)
 - Exécutez git clone https://github.com/ptitbob/siad_m2_ws.git
 - Placez vous dans le repertoire siad_m2_ws
 - Executez mvn clean compile (il devrait télécharger le web)

Pour faire les mise à jour pour chaque cours (si la VM n'a pas été réinitialisée) :

- git pull origin
- mvn clean compile

5

Java EE 7

Les principaux composants

CDI 1.1	BeanValidation 1.1	Interceptor 1.2	Concurrency 1.0	JSP JSTL	EL 3.0	JSF 2.2	JAX-RS 2.0
				Servlet 3.1	Websocket 1.0	JSON-P 1.0	
				JTA 1.2	EJB 3.2	JMS 2.0	Batch 1.0
				JPA 2.1	JCA 1.7		JavaMail 1.5
Java EE 7 - JSR 342							

7

Références

	Beginning Java EE 7 Antonio Goncalves aPress 2013
	RESTful Java with Jax-RS 2.0 Bill Burke O'Reilly 2013
	RESTful Web APIs Leonard Richardson, Mike Amundsen, Sam Ruby O'Reilly 2013
	RESTful Java with JAXRS 2.0 Bill Burke

8

Java EE 7

Les principaux composants

Vu en MI

CDI 1.1	BeanValidation 1.1	Interceptor 1.2	Concurrency 1.0	JSP JSTL	EL 3.0	JSF 2.2	JAX-RS 2.0
				Servlet 3.1	Websocket 1.0	JSON-P 1.0	
				JTA 1.2	EJB 3.2	JMS 2.0	Batch 1.0
				JPA 2.1	JCA 1.7		JavaMail 1.5
Java EE 7 - JSR 342							

Java EE 7

Les principaux composants

Les web Services



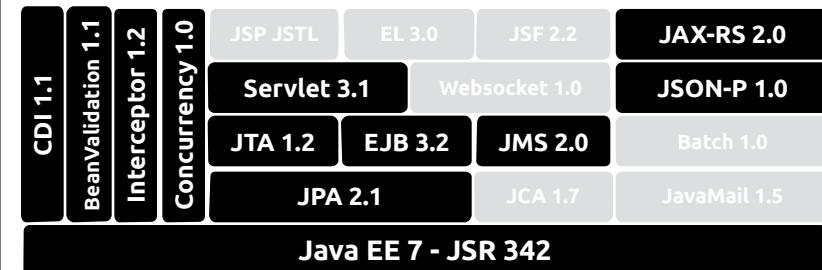
9

Java EE 7

Les principaux composants

Les briques d'une application d'entreprise basé sur les WS

C'est évidemment une architecture presque minimale



10

Définitions

- Echange de **services** de façon **faiblement couplée** entre un **fournisseur** et un **demandeur**
- **Service** : fonction métier
- **Fournisseur** : celui qui offre le service
- **Demandeur** : celui qui consomme le service
- **Faiblement couplée** : Fournisseur et Demandeur ignorent les détails de l'implémentation du partenaire

Type de service Web

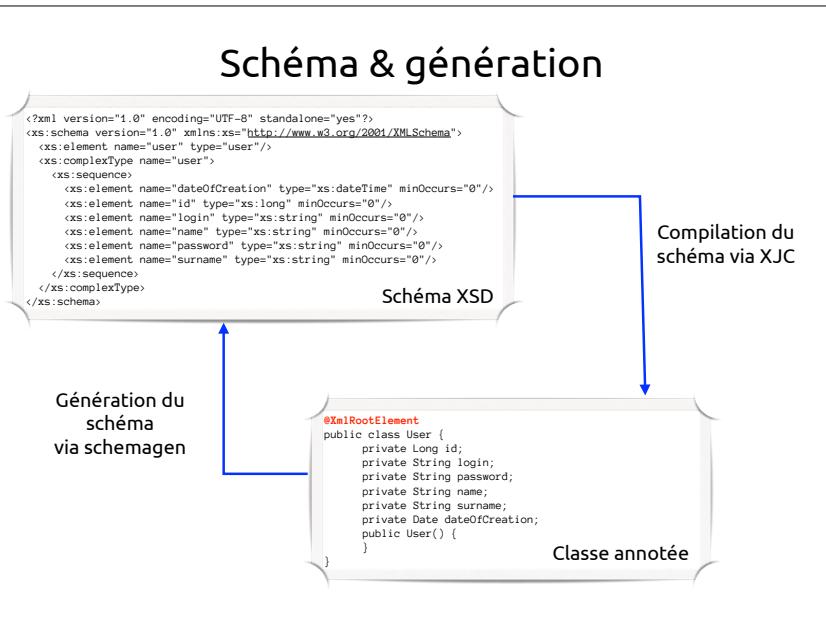
- WS-* : Web Services - * : repose sur les standards SOAP et WSDL
- REST : Representational State Transfer : repose sur les standards HTTP et URI

11

12

XML & JAXB

13

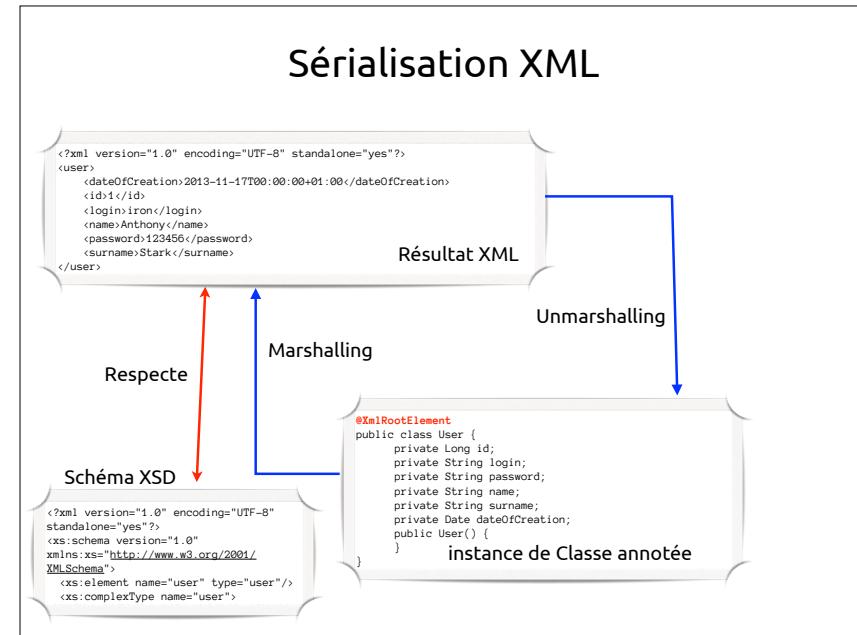


15

JAXB

- **Java Architecture for XML Binding (Java SE)**
- Présent dans JavaEE (7) et JavaSE (1.8)
- JSR 222 - révision 2 - version 2.2.3
- Permet transformation XML <-> Java
- Marshalling : transformer un objet en XML
- Unmarshalling : transformer du XML en objet
- Gestion du XSD
- xjc et schemagen pour la génération

14



16

Annotation (1)

```
@XmlRootElement(name = "book", namespace = "http://univ-blois.fr/ws/book")
@XmlAccessorType(XmlAccessType.FIELD)
@Entity
public class Book {

    @XmlAttribute(name = "id", required = true)
    @Id
    private Long id;

    private String title;

    private String isbn;

    @Enumerated(EnumType.STRING)
    private Type type;

    public Book() {
    }
...
}
```

17

@XmlRootElement

```
@Retention(RUNTIME)
@Target({TYPE})
public @interface XmlRootElement {

    String namespace() default "##default";

    String name() default "##default";
}
```

18

@ XmlAccessorType

```
@Inherited @Retention(RUNTIME) @Target({PACKAGE, TYPE})
public @interface XmlAccessorType {

    XmlAccessType value() default XmlAccessType.PUBLIC_MEMBER;
}

public enum XmlAccessType {
    PROPERTY,
    FIELD,
    PUBLIC_MEMBER,
    NONE
}
```



19

@XmlAttribute

```
@Retention(RUNTIME) @Target({FIELD, METHOD})
public @interface XmlAttribute {

    String name() default "##default";

    boolean required() default false;

    String namespace() default "##default" ;
}
```

20

@ Enumerated

```
@Target({METHOD, FIELD})  
@Retention(RUNTIME)  
public @interface Enumerated {  
  
    EnumType value() default ORDINAL;  
}  
  
public enum EnumType {  
    ORDINAL,  
  
    STRING  
}
```

21

Annotation (1)

```
@XmlRootElement(name = "book", namespace = "http://univ-blois.fr/ws/book")  
@XmlAccessorType(XmlAccessType.FIELD)  
@Entity  
public class Book {  
  
    @XmlAttribute(name = "id", required = true)  
    @Id  
    private Long id;  
    @XmlElement(name = "title")  
    private String title;  
    @XmlElement(name = "isbn")  
    private String isbn;  
  
    @Enumerated(EnumType.STRING)  
    private Type type;  
  
    public Book() {  
    }  
...  
}
```

22

Génération XML

```
public class BookGeneration {  
  
    public static void main(String... arg) {  
        Book book = new Book(1L, "Pawn of Prophecy", "XXXX", Type.ROMAN);  
        try {  
            JAXBContext jaxbContext = JAXBContext.newInstance(Book.class);  
            Marshaller marshaller = jaxbContext.createMarshaller();  
            marshaller.setProperty("jaxb.encoding", "UTF-8");  
            marshaller.setProperty("jaxb.formatted.output", true);  
            marshaller.marshal(book, new File("book.xml"));  
        } catch (JAXBException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

23

Résultat

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<ns2:book xmlns:ns2="http://univ-blois.fr/ws/book" id="1">  
    <title>Pawn of Prophecy</title>  
    <isbn>XXXX</isbn>  
    <type>ROMAN</type>  
</ns2:book>
```

24

Génération des schéma

- Compilation via Maven : mvn clean compile
- Utilisation de schemagen
- schemagen -cp target/classes/ fr.univ.blois.siad.m2.ws.jaxb.Book

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xss: schema version="1.0" xmlns:xss="http://www.w3.org/2001/XMLSchema">

<xss:complexType name="book">
  <xss:sequence>
    <xss:element name="title" type="xs:string" minOccurs="0"/>
    <xss:element name="isbn" type="xs:string" minOccurs="0"/>
    <xss:element name="type" type="type" minOccurs="0"/>
  </xss:sequence>
  <xss:attribute name="id" type="xs:long" use="required"/>
</xss:complexType>

<xss:simpleType name="type">
  <xss:restriction base="xs:string">
    <xss:enumeration value="COMICS"/>
    <xss:enumeration value="SCIFI"/>
    <xss:enumeration value="ROMAN"/>
    <xss:enumeration value="POLAR"/>
    <xss:enumeration value="INFORMATIQUE"/>
    <xss:enumeration value="EDUCATION"/>
  </xss:restriction>
</xss:simpleType>
</xss: schema>
```

25

Génération des classes

Utilisation de l'utilitaire xjc
xjc schema1.xsd

- 
- Génération de 3 classes
 - Book.java
 - Type.java
 - Une factory

26

Génération via annotation

```
@XmlRootElement
@XmlAccessorType(XmlAccessType.ALPHABETICAL)
@XmlAccessorType(XmlAccessType.FIELD)
public class User {
  @XmlAttribute
  private Long id;

  private String login;
  @XmlTransient
  private String password;
  @XmlElement(name = "firstname")
  private String name;

  private String surname;
  @XmlTransient
  private Date dateOfCreation;

  public User() {
  }
}
```

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<user id="1">
  <login>iron</login>
  <firstname>Anthony</firstname>
  <surname>Stark</surname>
</user>
```

27

JAXB les annotations

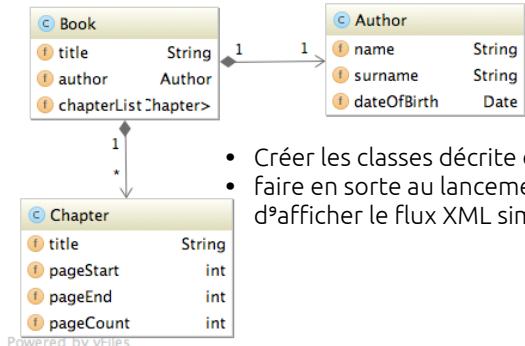
Annotation	Description
@XmlAccessorType	FIELD, NONE, PROPERTY, PUBLIC_MEMBER
@XmlRootElement	mappe une classe comme racine d'un XML
@XmlAttribute	mappe comme un attribut XML
@XmlElement	mappe comme un élément XML
@XmlTransient	empêche le mapping XML
@XmlList	Mappe une propriété comme une liste XML
@XmlType	Mappe une classe comme type complexe XML

@XmlElement, @Enum, @EnumValue, @XmlID, @XmlIDREF, @XmlMimeType,
@XmlNs, @XmlSchema, @XmlValue, **@XmlAccessorType**, **@XmlAccessorTypeOrder**

28

Hands on

Avec le projet JaxbTest



- Créer les classes décrite dans le diagramme joint
- faire en sorte au lancement de la classe Main d'afficher le flux XML simple (basé sur les champs).

29

Hands on

Avec le projet JaxbTest

Dans un second temps

- Utilisant les même sources
- Faire en sorte que le flux XML de Book
• ai comme attribut le titre
- la liste des chapitre ai comme balise `<chapters>`
- Author ai comme attribut le nom et le prénom et que les champs soit trié par ordre alpha.
- Chapter
• ai comme attribut le titre, et ne présente pas le numéro de page de fin
- que les champs ne soit pas triés
- Puis générer à l'aide schemagen le schema xsd

30

Web Services SOAP

31

Spécifications

Spécifications	Rôle
JAX-WS 2.0	Spécifications
JAXB 2.2.3	Binding XML (utilisé pour le WSDL)
WS-BP 1.1	Interopérabilité .NET
WS-Metadata	Metadata approche pour définition des WS
JAX-RPC 1.1	Compatibilité avec J2EE 1.4 WS

32

Technologie et protocoles

- UDDI
- WSDL
- SOAP
- Protocole de transport
- XML

33

UDDI

- Annuaire de web-services
- fournit les informations sur les web- services sous forme de document WSDL

34

WSDL

- Web Service Description Language
- définit le type de message, le port, le protocole, les opérations, l'adresse...
- Fichier XML

35

SOAP

- Simple Object Access Protocol
- Protocole standard de communication des web services
- Fichier XML

36

Protocole de transport

- Moyen d'échange des messages
 - HTTP souvent
 - HTTPS,TCP/IP, SMTP, FTP, JMS...

37

XML

- Permet l'indépendance et l'interopérabilité des web services
- dans la définition (WSDL) et l'échange (SOAP)
- xsd permet la validation des messages

38

Différentes implémentations ?

- Choix entre 2 type d'implémentations
 - Classe standard Java (POJO)
 - EJB Stateless

39

Différentes implémentations ?

Fonctionnalité	Java WS	EJB
POJO	OUI	OUI
DI de ressources, PU, ...	OUI	OUI
Lifecycle methods	OUI	OUI
Transaction déclarative	NON	OUI
Sécurité déclarative	NON	OUI
Annotation Processing dans un APT externe	OUI	non pour la plupart des containers EJB
Fonctionne dans un container web (Tomcat)	OUI	NON

40

Contraintes d'implémentation

- Annotation @WebService ou équivalent XML dans le descripteur de déploiement
- classe publique, non final et non abstract
- un constructeur public par défaut
- pas de définition de finalize()
- objet stateless

41

Exemples...

```
@Stateless  
@LocalBean  
@WebService  
public class UserServices {  
  
    ...  
  
    public String getUserName(long userId) {  
        User user = entityManager.find(User.class, userId);  
        return user.getName();  
    }  
  
    ...  
}
```

43

Approche de développement

- Bottom up : de l'implémentation au WSDL
- Top Down : du WSDL à l'implémentation (contract first)
- Meet in the middle : Rattache le WSDL à l'implémentation

42

@WebService

- S'utilise sur l'interface ou le bean (le container génère alors l'interface)
- Toutes les méthodes publiques sont exposées

44

@WebService

```
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = {ElementType.TYPE})
public @interface WebService {
    public String name() default "";
    public String targetNamespace() default "";
    public String serviceName() default "";
    public String portName() default "";
    public String wsdlLocation() default "";
    public String endpointInterface() default "";
}
```

45

@WebService

- name : nom du WS
- targetNamespace : précise le namespace du WSDL
- serviceName : quand on annote le bean
- wsdlLocation : précise l'emplacement du WSDL (pas de génération)
- endpointInterface : précise le nom du Service Endpoint Interface
- portname : nom du port

46

Configuration

- plusieurs styles :
 - DOCUMENT
 - RPC
- style de message :
 - LITERAL
 - ENCODED
- Style de paramètres
 - BARE
 - WRAPPED

47

@SOAPBinding

```
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = {ElementType.TYPE, ElementType.METHOD})
public @interface SOAPBinding {
    public enum ParameterStyle {
        BARE, WRAPPED;
    }
    public enum Style {
        DOCUMENT, RPC;
    }
    public enum Use {
        LITERAL, ENCODED;
    }
    public Style style() default DOCUMENT;
    public Use use() default LITERAL;
    public ParameterStyle parameterStyle() default WRAPPED;
}
```

48

@WebMethod

- Permet de choisir les méthodes exposées.

```
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = {ElementType.METHOD})
public @interface WebMethod {

    public String operationName() default "";
    public String action() default "";
    public boolean exclude() default false;
}
```

49

@WebMethod

- operationName : operation name du WSDL
- action : soap:operation du WSDL
- exclude : pour empêcher l'exposition

50

@WebParam

```
@Retention(value = RetentionPolicy.RUNTIME)
@Target(value = {ElementType.PARAMETER})
public @interface WebParam {
    public enum Mode {
        IN, OUT, INOUT;
    }
    public String name() default "";
    public String partName() default "";
    public String targetNamespace() default "";
    public Mode mode() default IN;
    public boolean header() default false;
}
```

51

@WebParam

- name : name du paramètre dans le message du WSDL
- targetNamespace : configuration du WSDL
- mode : mode d'échange du paramètre
- header : true si paramètre dans le header et pas le message body
- partName : name element du WSDL

52

@OneWay

- se positionne sur une méthode
- quand il n'y a pas de valeur de retour

@HandlerChain

- Equivalent des Intercepteurs des EJB
- WebServiceContext.getMessageContext() équivalent de InvocationContext.getContextData()

53

WebServiceContext

- Injection par @Resource
- méthodes :
 - getMessageContext
 - getUserPrincipal
 - isUserInRole
 - getEndPointReference

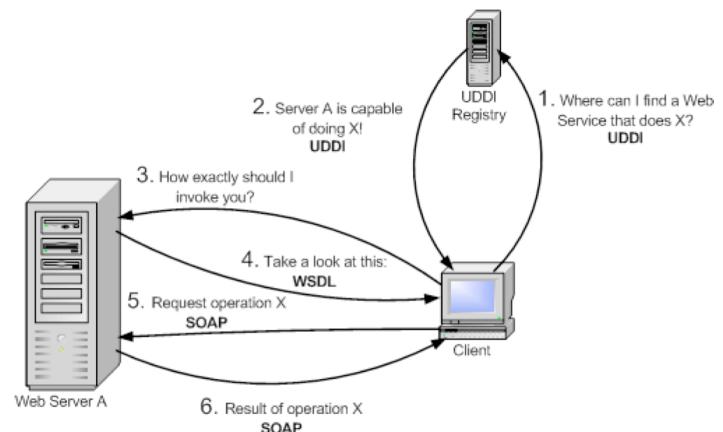
55

Cycle de vie et injection

- Comme pour les EJB, entité : @PostConstruct & @PreDestroy
- Injection de dépendance possible (EntityManager par exemple)

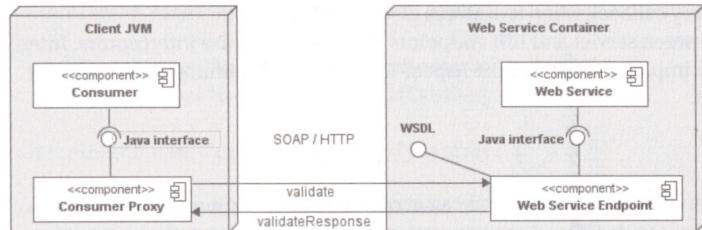
54

Appeler un WebService



56

Appeler un WebService



Beginning Java EE 6 Platform with GlassFish 3, Antonio Goncalves, Apress, p. 418

57

Référence par programmation

- HelloWSService service = new HelloWSService();
- HelloWS hello = service.getPort(HelloWS.class);
- String result = hello.sayHello(@WebServiceB);

59

Appeler un WebService

- Utilisation de
 - wsgen : génération du WSDL
 - wsimport : génération du SEI et des classes à partir du WSDL
- Récupération du SEI par programmation ou par injection, puis du proxy et invocation

58

Référence par annotation

```
@WebServiceRef  
HelloWSService service ;  
.....  
HelloWS hello = service.getPort(HelloWS.class);  
String result = hello.sayHello(@WebServiceB);
```

60

@WebServiceRef

```
@Target({TYPE})
public @interface WebServiceRef {
    String name() default >?;
    String mappedName() default >?;
    Class type() default java.lang.Object.class;
    Class value() default java.lang.Object.class;
    String wsdlLocation() default >?;
};
```

61

@WebServiceRef

- name : nom JNDI lié à java:comp/env/<name>
- mappedName : nom JNDI vendeur spécifique
- type : type de la resource
- value : classe du service (extends javax.xml.ws.Service)
- wsdlLocation : URL du WSDL

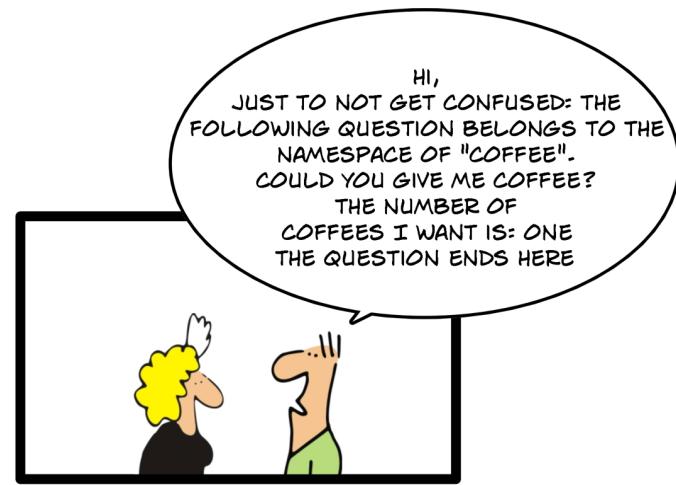
62

WS-*

- Quelques spécifications complémentaires (pas toutes standardisées) :
 - WS-Security
 - WS-Reliability
 - WS-Transaction
 - WS-Addressing

63

Webservice en résumé...



64

Hands on

Avec le projet Activity

- Proposer de Service pour
 - UserServices
 - ActivityServices (Attention au stackOverflow)
- Nommer les méthodes
- Tester les méthodes
- Générer le Wsdl et le xsd de validation



65

WebService REST



66

REST ?

- REST : Acronyme de REpresentational State Transfert
- Le retour du Web dans le Web Service
- Principes définis dans la thèse de Roy FIEDLING en 2000
- Style d'architecture inspiré de l'architecture du web
- Repose sur HTTP (HyperText Transfert Protocol) et URIs (Uniform Resource Identifiers)
- Standardisé dans Java EE 6 : JSR 311 : JAX-RS

67

REST ?

- REST est
 - Un style d'architecture
 - Une approche pour construire une application
- REST n'est pas
 - un format
 - un protocol
 - un standard

68

REST ?

- Les services web REST sont développés pour mettre en place des architecture orienté ressources
- Les applications respectant les architectures orientées ressources sont appelé RESTful.

69

Ressource ?

- Tout ce que le client peut vouloir
- En général concepts concrets
- Pour une boutique de livres :
 - la liste des livres d'une catégorie
 - le résumé d'un livre
 - la bio d'un auteur

70

Représentation

- Format de présentation de la ressource (texte, JSON, XML, PDF...), la ressource reste sur le serveur
- La ressource lié à un bon de commande est représenté par un document XML
- La création de la commande est l'association de la méthode POST et du document XML
- Distinction par URI ou par ⁸Content Negotiation⁹
- L'état est maintenu par la représentation de la ressource
- De ce fait, c'est le client qui est responsable de l'état.

71

WebService RESTful

- Les services REST sont sans états (Stateless)
- Toutes les requêtes envoyées au serveurs doivent contenir les informations propre à leur traitement
- Minimisation des ressources système (pas de session ni de d'état)
- Les services REST fournissent une interface basé sur les verbe HTTP
 - GET, PUT, DELETE & POST
- Les chemin REST sont basé sur les URI (identification unique des ressources)
- ex : <http://www.flickr.com/explore/2013/10/08>

72

URI

- Généralement combinaison d'URL (Uniform Resource Locator) et URN (Uniform Resource Name)
- le plus descriptif possible
- Identifie de manière unique la ressource

`http://shopbook/books/fantasy/belgariad/1`

Ressource de type collection
Identifiant primaire de la ressource

- une ressource peut avoir plusieurs URI

`http://shopbook/books/fantasy/belgariad/1`
`http://shopbook/books/fantasy/belgariad/Pawn_of_Prophesy`
`http://shopbook/books/fantasy/belgariad` Tous les livres de la Belgariade
`http://shopbook/books/fantasy` Tous les livres d'heroic fantasy

73

REST & Méthodes

- Une ressource peut subir 4 opérations de base, connues sous le nom de CRUD
 - Create
 - Read / Retrieve
 - Update
 - Delete
- REST s'appuie sur les verbes HTTP :
 - Create via POST
 - Read via GET
 - Update via PUT
 - Delete via DELETE
- on n'utilise plus rarement les autres HEAD, TRACE, OPTIONS, CONNECT

74

GET :: Lecture



- Lecture simple
- doit être implémentée de façon *'safe'*
- doit être *'idempotent'*

- **Safe** : se dit d'une méthode qui ne change pas l'état de la ressource
- **Idempotent** : se dit d'une méthode que l'on peut appeler plusieurs fois sans changer le résultat

75

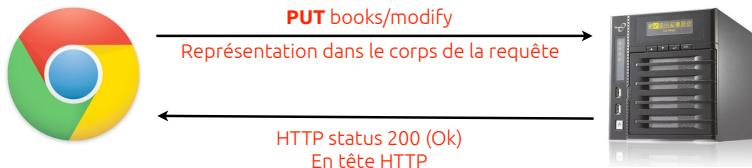
POST :: Création



- pour une représentation (texte, XML...), POST crée un nouvelle ressource comme subalterne d'une ressource principale identifiée par l'URI demandée
- Pas *'safe'* (modifie l'état)
- Pas *idempotent* (Post multiples entraînent des doublons)

76

PUT :: Modification



- Met à jour l'état de la ressource correspond à l'URI
- Crée la ressource si elle n'existe pas
- Pas ^{safe}
- Idempotent

77

Delete :: Suppression



- Supprime la ressource
- Pas ^{safe}
- Idempotent

78

Et les autres ?

- HEAD : comme GET sans le ^{body} de la réponse (test la validité de l'URI ou la taille de l'objet)
- TRACE : renvoie la requête reçue (echo)
- OPTIONS : demande les informations sur les options de communications
- CONNECT : utilisé pour demander le changement d'un proxy en tunnel HTTP

79

Négociation de contenu

- dans les headers de la requête HTTP :
 - Accept
 - Accept-Charset
 - Accept-Encoding
 - Accept-Language
 - User-Agent

80

Types de contenu

Content Types

- dans les champs Content-Type et Accept
- 5 catégories : text, image, audio, video, application
- quelques sous-types :
 - text/html, text/plain (type par défaut)
 - image/gif, image/jpeg, image/png
 - text/xml, application/xml
 - application/json (JavaScript Object Notation)

81

Les codes de retour

Quelques codes

- 200 : Ok
- 201 : Crée
- 301 : déplacé de manière permanente
- 400 : syntaxe erronée
- 401 : non autorisé
- 404 : non trouvé
- 418 : je suis une théière
- 500 : erreur serveur
- etc...

http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

83

Les codes de retour

Ou les codes de statut

- 1xx : Informational : requête reçue, traitement en cours
- 2xx : Success : requête reçue, comprise et traitée
 - 200 : Ok
 - 201 : Crée
- 3xx : Redirection : de nouvelles étapes à faire pour compléter la requête
- 4xx : Client Error : mauvaise syntaxe, la requête ne peut être complétée
 - 404 Not found
- 5xx : Server Error : Le serveur a échoué sur une requête apparemment correcte

http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

82

JEE : Définition service REST

Old flavored JEE 6

Un fichier descriptif : WEB-INF / web.xml Implémentation de référence JSR 311 JAX-RS

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
```

```
    <display-name>toolbox</display-name>
    <servlet>
        <servlet-name>calc</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>fr.univ.blois.jee.rest</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

Définition de la servlet

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

Package de base exposant les services

Mapping de la servlet

</web-app> Composant de l'url ⁸root⁸ des services REST : blois.univ.fr/rest/....

84

JEE : Définition service REST

Mais c^etait avant...

85

JEE : Définition service REST

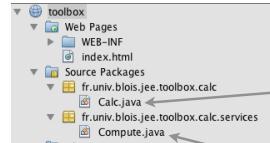
JEE 7

Un fichier descriptif : WEB-INF / **web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
```

```
    <display-name>Univ-blois-M2</display-name>
```

```
</web-app>
```



Classe décrivant l^eapplication

Classe présentant un service REST

86

WS-REST

Description de l^eapplication

```
import fr.univ.blois.jee.toolbox.calc.services.Compute;
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("calc")
public class Calc extends Application {
    @Override
    public Set<Class<?>> getClasses() {
        return new HashSet<Class<?>>(Arrays.asList(Compute.class));
    }
}
```

- Classe étendant la classe Application (implémentation JAX-RS)
- annotation @ApplicationPath avec comme paramètre le root
 - Le root peut être vide (⁸⁸)
- Surcharge de la méthode getClasses pour renvoyer les classes implémentant les services REST

87

WS-REST

Presentation d^eun service

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;

@Path("do")
public class Compute {

    @GET
    @Path("hello")
    public String sayHello() {
        return "hello";
    }
}
```

- programmation Classe annotée avec @javax.ws.rs.Path
- Possibilité d^eavoir des capacités d^eEJB par ajout de @Stateless
- Classe publique, ni finale, ni abstract
- Classe racine de la ressource (annotée @Path) doit avoir un constructeur par défaut

88

@Path

```
@Path("/books")
public class BookRest {

    @GET
    @Path("{themeId}")
    public List<Book> getListBookForTheme(@PathParam("themeId") String themeId) {
        return null;
    }

    @GET
    @Path("{themeId}/{collectionId}")
    public List<Book> getListBookForCollection(
        @PathParam("themeId") String themeId,
        @PathParam("collectionId") String collectionId
    ) {
        return null;
    }
}
```

- si annotation sur la classe et la méthode, l'URI est la concaténation des deux
- <http://blois/shopbook/books/fantasy> appelle getListBookForTheme
- <http://blois/shopbook/books/fantasy/belgariad> appelle getListBookForCollection

89

Extraction des paramètres

- Possibilité de récupérer des paramètres
- @PathParam
- @QueryParam
- @MatrixParam
- @CookieParam
- @HeaderParam
- @FormParam

90

@QueryParam

```
@Path("/books")
public class BookRest {
    @GET
    public Book getBook(@QueryParam("bookId") String bookId) {
        return findingBook;
    }
}
```

Extraction depuis <http://blois/shopbook/books?bookId=123>

91

- @CookieParam et @HeaderParam : utilisation mais on recherche les paramètres dans le Cookie ou le Header
- @FormParam : extraction d'un formulaire mais support pas obligatoire
- @DefaultValue : pour une valeur par défaut
public Book getBook(@DefaultValue("1234")
@QueryParam("bookid") int bookId){

92

@Produces @Consumes

```
@Path("/books")
public class BookRest {

    @GET
    @Path("/{themeId}")
    @Produces("application/xml")
    public List<Book> getListBookForThemeAsXML(@PathParam("themeId") String themeId) {
        return null;
    }

    @GET
    @Path("/{themeId}")
    @Produces("application/json")
    public List<Book> getListBookForThemeAsJSON(@PathParam("themeId") String themeId) {
        return null;
    }
}
```

- Sur la classe ou surcharge sur la méthode
- Permet de définir le Type Mime accepté ou renvoyé par la méthode

93

Choix des méthode

- Négociation de type par les entêtes HTTP
- Accept : text/plain
 - Méthode `getAsTextPlain`
- Accept : text/plain; q=0.8, text/html
 - Méthode `getAsHtml`

94

Hand on

Projet toolbox

- Dans le package ...toolbox.calc.service
 - Créer une classe de calculatrice (idée d'action)
 - Exposant en GET
 - addition, soustraction (2 méthodes) utilisant les paramètres de requête
 - L'ajouter à l'application toolbox
 - Tester via un CURL ou directement via une page web (Netbean peut vous aider)



Exception

En cas de problème, on lève une WebApplicationException qui est converti par JAX-RS fonction du paramètre de constructeur.

```
new WebApplicationException(Response.Status.NOT_FOUND)
```

95

96

Hand on

Projet toolbox (2)

- A partir de vos sources
 - Implémenter les opérations
 - Multiplication
 - Division (faites attention...)



97

Réponse complexe

- Un service peut renvoyer des réponses complexes
 - Un bean issue d'un traitement représenté en
 - XML
 - JSON

98

JSON / XML

```
{"book": {  
    "id": "1234",  
    "authors": [  
        {  
            "firstName": "Pierre",  
            "lastName": "Dupont"  
        },  
        {  
            "firstName": "Paul",  
            "lastName": "Durand"  
        }  
    ]  
}}
```

```
<book>  
    <id>1234</id>  
    <authors>  
        <author>  
            <firstName>Pierre</firstName>  
            <lastName>Dupont</lastName>  
        </author>  
        <author>  
            <firstName>Paul</firstName>  
            <lastName>Durand</lastName>  
        </author>  
    </authors>  
</book>
```

99

EntityProvider

- conversion de la représentation en Java et vice-versa (JAX-B par exemple)
- deux types : MessageBodyReader ou MessageBodyWriter
- Les deux sont annotés @Provider
- On peut restreindre les types consommés par @Consumes et @Produces

100

Provider par défaut

Type	Description
byte[]	All media type (/*)
java.lang.String	All media type (/*)
java.io.InputStream	All media type (/*)
java.io.Reader	All media type (/*)
java.io.File	All media type (/*)
javax.activation.DataSource	All media type (/*)
javax.xml.transform.Source	XML type
javax.xml.bind.JAXBElement	JAXB class, XML Media Types
MultivaluedMap<String, String>	Form content
javax.ws.rs.core.StreamingOutput	all media type (/*), MessageBodyWriter only

101

MessageBodyWriter

- Permet de gérer la génération de format spécifique
- Annotation @Provider
- Utilisation de l'annotation @Produces pour fixer le type produit.
- Implémente MessageBodyWriter<Type>
- Type == Classe à traduire
- Contrat
 - isWriteable
 - getSize
 - writeTo

102

MessageBodyWriter

```
@Provider  
@Produces("application/gson")  
public class UserGsonProvider implements MessageBodyWriter<User> {  
  
    @Override  
    public boolean isWriteable(Class<?> type, Type genericType, Annotation[] annotations,  
    MediaType mediaType) {  
        return true;  
    }  
  
    @Override  
    public long getSize(User t, Class<?> type, Type genericType, Annotation[] annotations,  
    MediaType mediaType) {  
        return 1;  
    }  
  
    @Override  
    public void writeTo(User t, Class<?> type, Type genericType, Annotation[] annotations,  
    MediaType mediaType, MultivaluedMap<String, Object> httpHeaders, OutputStream entityStream)  
    throws IOException, WebApplicationException {  
        try(OutputStreamWriter outputStreamWriter = new OutputStreamWriter(entityStream)) {  
            new Gson().toJson(t, type, outputStreamWriter);  
        }  
    }  
}
```

103

MessageBodyWriter Utilisation

```
@GET  
@Path("user")  
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON, "application/gson"})  
public User getOneUser() {  
    return new User("name", "surname");  
}
```

Fournit par Glassfish

104

MessageBodyReader

- Permet de gérer la lecture de format spécifique
- Annotation `@Provider`
- Utilisation de l'annotation `@Consumes` pour fixer le type produit.
- Implémente `MessageBodyReader<Type>`
- Type == Classe à traduire
 - Contrat
 - `isReadable`
 - `readFrom`

105

MessageBodyReader

```
@Provider  
@Consumes("application/csv")  
public class UserCsvProvider implements MessageBodyReader<User> {  
  
    @Override  
    public boolean isReadable(Class<?> type, Type genericType, Annotation[] annotations,  
    MediaType mediaType) {  
        return type == User.class;  
    }  
  
    @Override  
    public User readFrom(Class<User> type, Type genericType, Annotation[] annotations,  
    MediaType mediaType, MultivaluedMap<String, String> httpHeaders, InputStream entityStream)  
    throws IOException, WebApplicationException {  
        String content = CharStreams.toString(new InputStreamReader(entityStream,  
       Charsets.UTF_8));  
        Doing some stuff !!!!  
        return user;  
    }  
}
```

106

MessageBodyReader Utilisation

```
@POST  
@Path("show")  
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON, "application/gson"})  
@Consumes("application/csv")  
public User getUserFromCommaSeparatedValue(User user) {  
    return user;  
}
```

Fournit par Glassfish

107

Revenons sur les exceptions

- `WebApplicationException` c'est bien...
- Mais pas tant que cela !!!
- Une gestion fine est préférable

108

⁸provider⁸ et exception

- Il est possible de créer des provider pour les exceptions
- Permet de gérer le traitement d'un type d'exception de manière unique
- Gestion de l'héritage des exceptions

109

⁸provider⁸ et exception

- Classe annotée avec @Provider
- implémente ExceptionMapper<Type>
- Type == classe de l'exception
- Contrat
- toResponse pour renvoyer la réponse correspondant à l'exception.

110

⁸provider⁸ et exception

```
@Provider
public class DivideByZeroExceptionMapper implements ExceptionMapper<DivideByZeroException>{

    @Override
    public Response toResponse(DivideByZeroException exception) {
        return Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity("Diviser par
zéro, c'est mal !!!").build();
    }
}
```

111

⁸provider⁸ et exception Ce que cela change ?

```
@GET
@Path("divide")
@Produces(MediaType.TEXT_PLAIN)
public String divide(@QueryParam("valA") String valueA, @QueryParam("valB") String valueB) throws
WebApplicationException {
    try {
        System.out.println("valA = " + valueA);
        System.out.println("valB = " + valueB);
        return String.valueOf(Integer.valueOf(valueA) / Integer.valueOf(valueB));
    } catch (NumberFormatException e) {
        return "N/A";
    } catch (ArithmaticException arithmeticException) {
        throw new WebApplicationException(Response.Status.BAD_REQUEST);
    }
}
```

```
@GET
@Path("divide")
@Produces(MediaType.TEXT_PLAIN)
public String divide(@QueryParam("valA") String valueA, @QueryParam("valB") String valueB) throws
WebApplicationException, NumberFormatException, DivideByZeroException {
    if (Integer.valueOf(valueB) == 0) {
        throw new DivideByZeroException();
    }
    return String.valueOf(Integer.valueOf(valueA) / Integer.valueOf(valueB));
}
```

112



113