<*XML num="5"*/>
Application Web en Java

- Emilien Bondu (Airbus Defence and Space)
- emilien.bondu@cassidian.com

Sommaire

- Cours 1 : Introduction à XML et validation (XSD)
- Cours 2 : Validation (DTD, Relax NG, Schematron) et transformation (XPath, XSL; XSL-FO)
- Cours 3 : Recherche XML (XQuery), Base de données XML, Liens XML, Manipulation XML en Java: Dom, Sax
- Cours 4 : Manipulation XML en Java : StAX, Data-binding, JavaEE : tomcat
- Cours 5 : Manipulation XML en JavaEE: servlet, JSP, ExpressionLanguage, TagLib
- Cours 6 : TagLib (suite), Spring, JSF, AJAX

Rappels

- StAX
  - est une API de lecture de flux XML
  - est une API d'écriture de flux XML
  - permet d'itérer sur les événements d'un flux XML
  - est compatible avec XPath

Rappels

- Le Data Binding :
  - est un mécanisme de mapping XML-Objet
  - est un mécanisme de mapping XML-Relationnel
  - permet de créer une représentation mémoire sous la forme d'objets métiers
  - est utilisé lors de la programmation de services Web

Rappels

- JEE & JSE:
  - JEE est une extension de JSE
  - la couche « business tier» traite les requêtes web
  - JEE permet de développer des clients lourds
  - JEE permet de développer des EJB

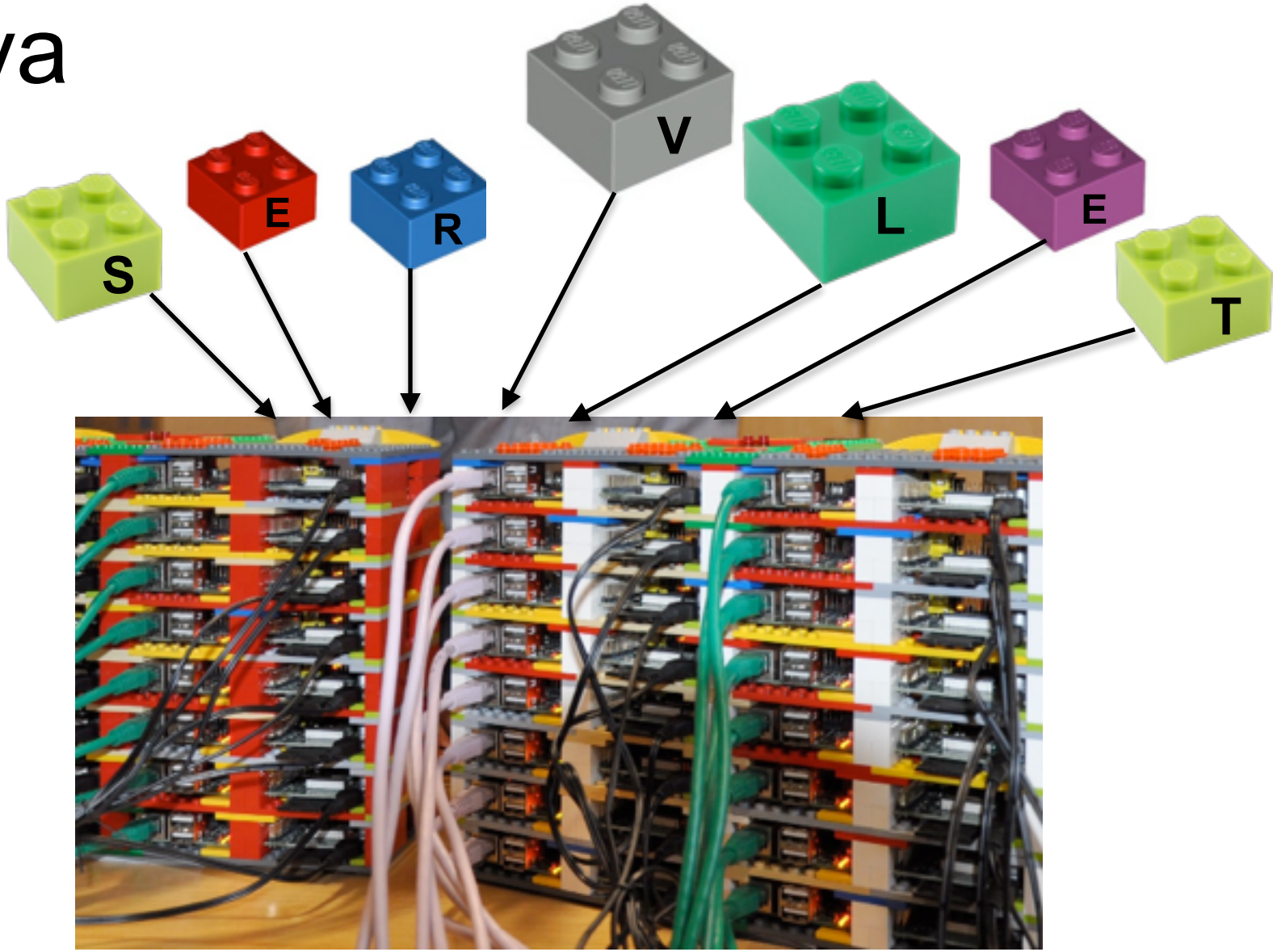
Rappels

- Applications web:
  - une application web est basée sur une architecture n-tiers
  - les JSP sont des composants web
  - un composant web est déployable sous forme d'une archive WAR
  - les données de l'application sont centralisées au sein du serveur JEE

Rappels

- Tomcat :
  - est certifié JEE
  - héberge un ensemble d'applications web
  - fourni un contexte unique par application
  - permet l'inspection des applications pour faciliter le développement

Java



Java Servlet

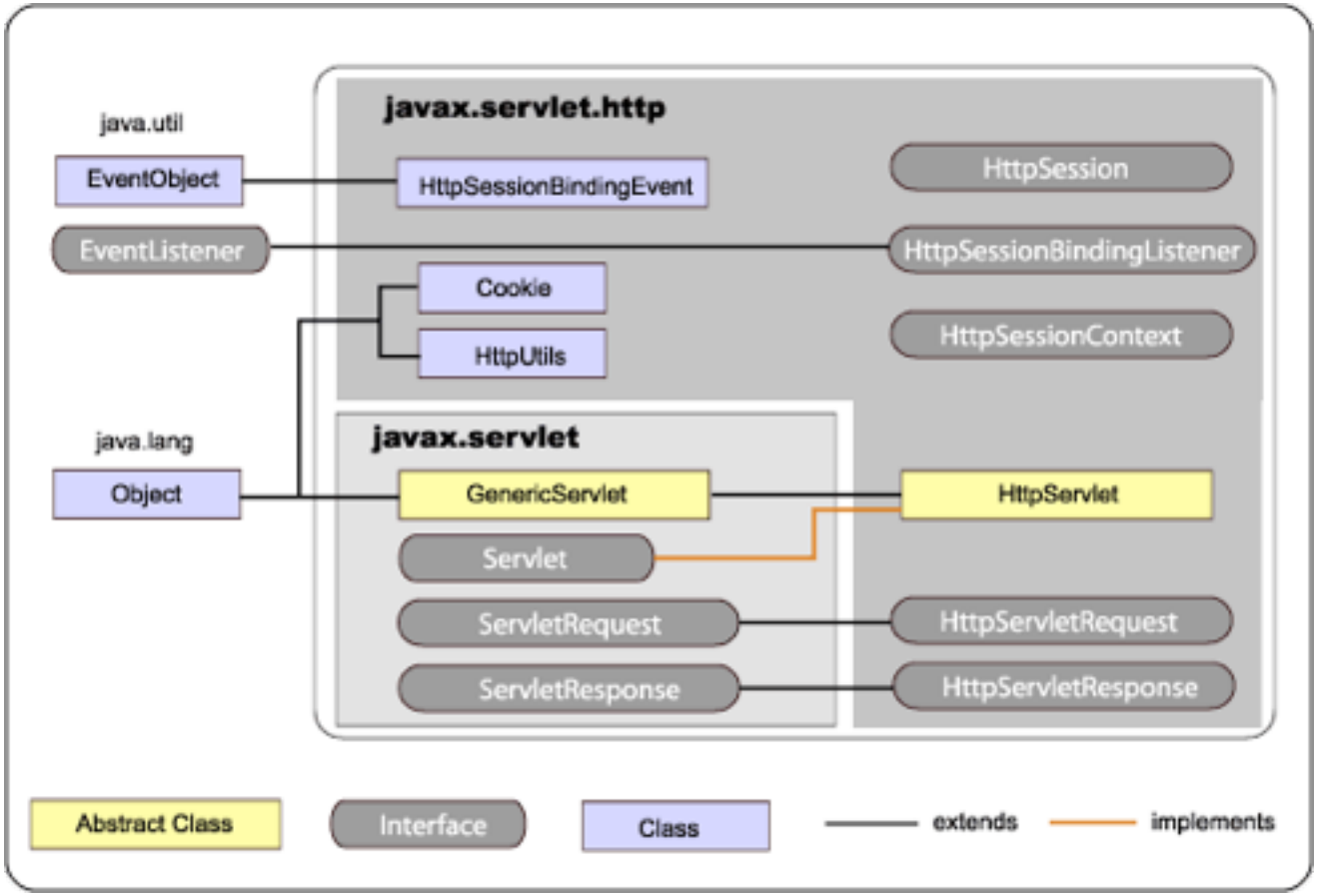
Rappels

- GenericServlet
  - classe qui étend les possibilités d'un serveur
  - fonctionne par un mécanisme requête/réponse
  - fourni une implémentation conforme au cycle de vie des Servlets

- HTTPServlet
  - traitement des requêtes HTTP (post, get, put, …)

Java HttpServlet

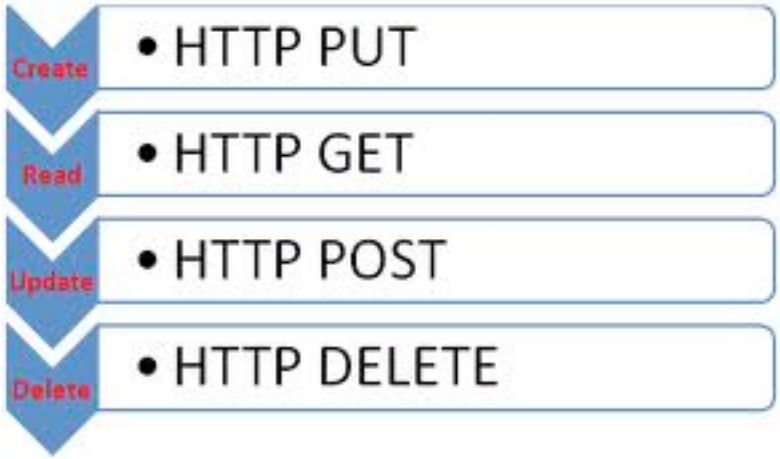
API - rappel



Java HttpServlet

API – rappel de quelques méthodes…

- `init()`
- `destroy()`
- `doPut()`
- `doGet()`
- `doPost()`
- `doDelete()`



Java HttpServlet

API – rappel de quelques méthodes…

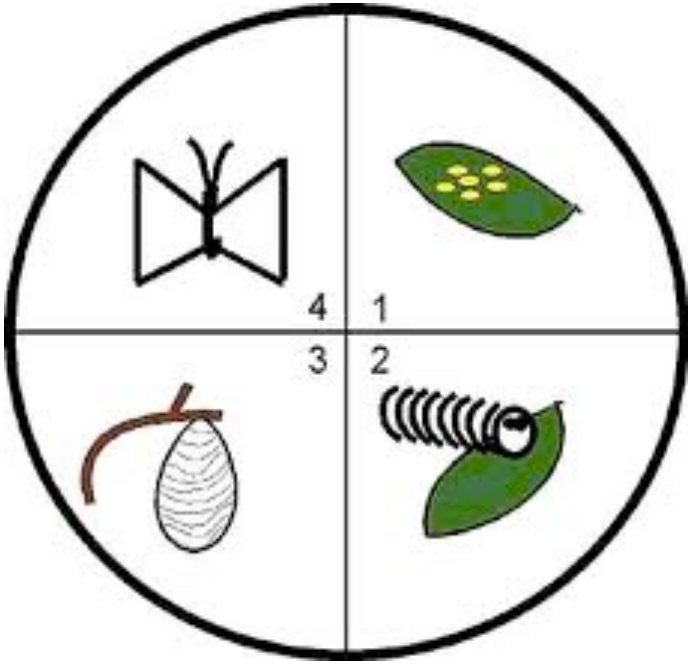
- `getRequestDispatcher()`
  - expédier la requête à un autre composant web
- `include()`
  - inclure la réponse d'un autre composant web à la réponse
  - l'autre composant ne peut modifier le header/statu de la réponse
- `forward()`
  - laisser l'autre composant écrire la réponse
  - l'appelant peut effectuer un prétraitement avant la transmission
- `getWriter()`
  - écrire directement dans la réponse

Java HttpServlet

API – rappel de quelques méthodes…

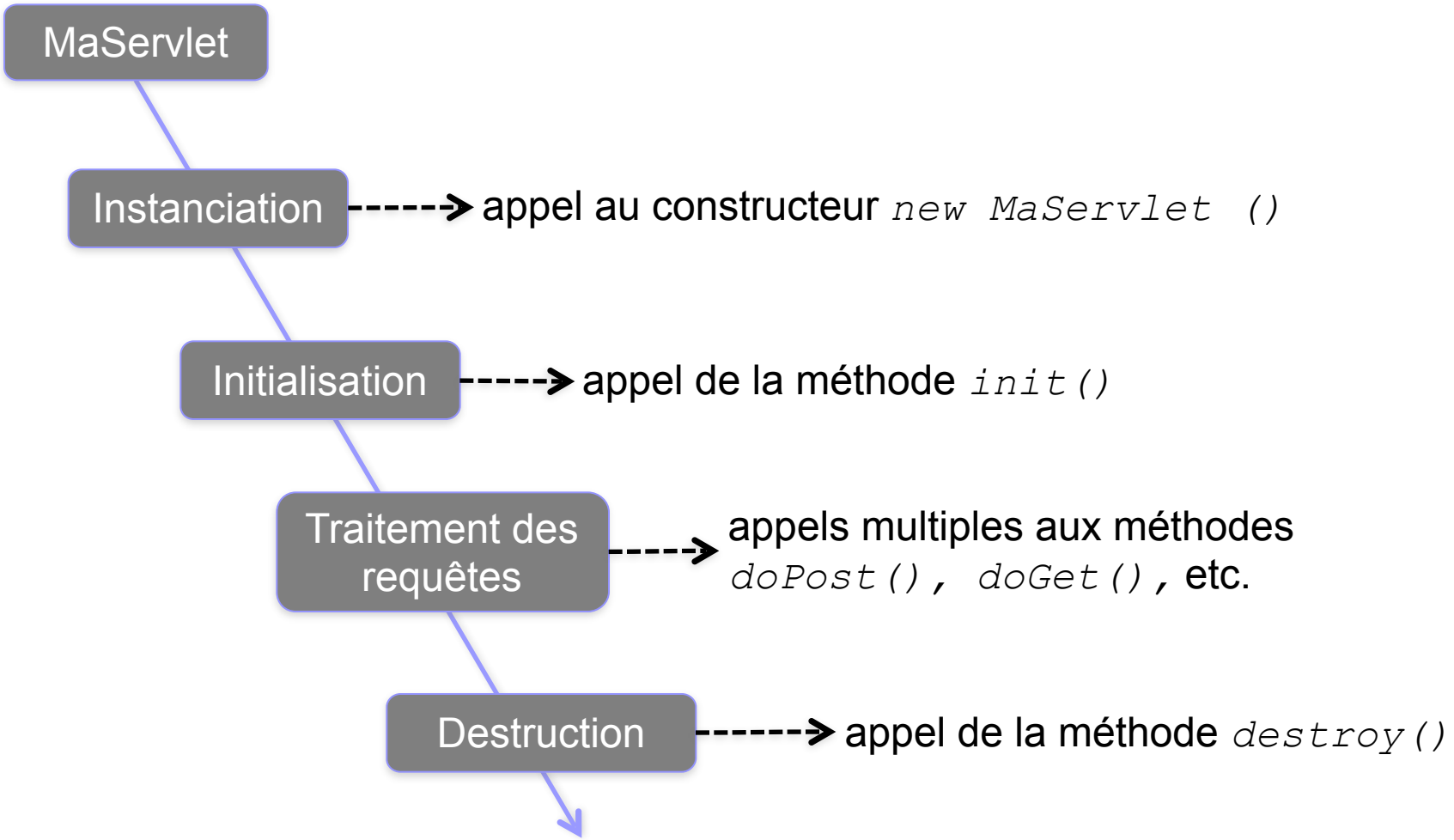
- `ServletConfig()`
  - à utiliser lors de l'initialisation
- `getServletContext()`
  - méthodes pour communiquer avec le container
  - log, attributs du contexte, ressources, dispatcher, type mime, version, etc.
- `getInitParameter(name)`
- `getSession()`
- `setContentType()`
- `sendRedirect()`
  - `redirection vers une URL.`

Java Servlet Lifecycle



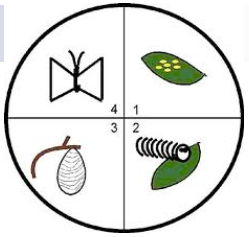
Java Servlet

cycle de vie

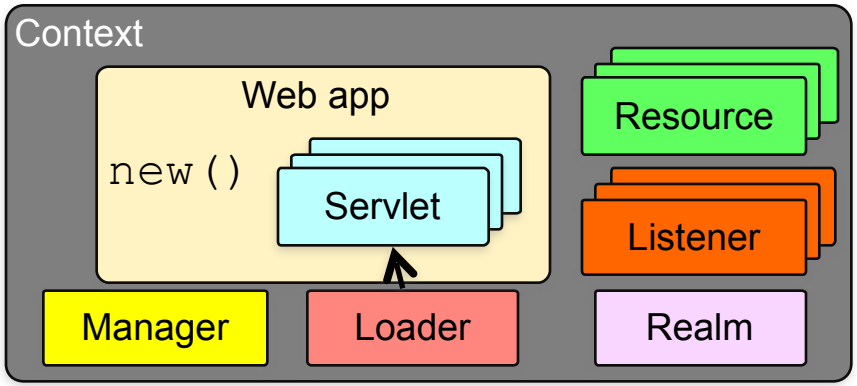


Application & Servlet

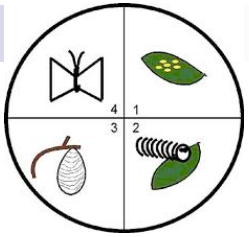
instanciation



- Au déploiement d'une application, Tomcat :
  - créer le contexte (session manger, class loader, ressources, etc)
  - le loader du contexte instancie les servlets & beans de l'application



Application & Servlet



context – context.xml

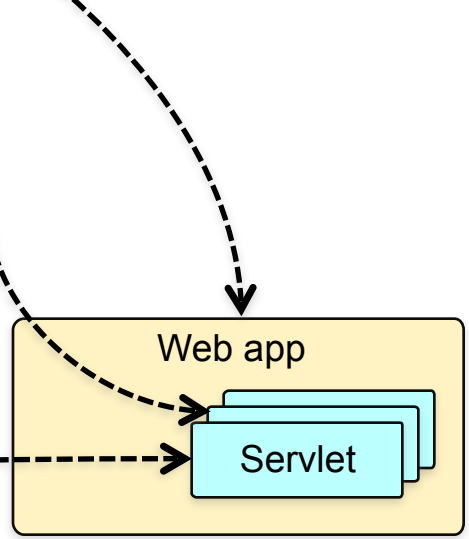
```
<?xml version='1.0' encoding='utf-8'?>

<!-- The contents of this file will be loaded for each web application -->
<Context>

    <!-- Default set of monitored resources -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!--
    <Manager pathname="" />
    -->

    <!-- Uncomment this to enable Comet connection tacking (provides events
         on session expiration as well as webapp lifecycle) -->
    <!--
    <Valve className="org.apache.catalina.valves.CometConnectionManagerValve" />
    -->

       <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
               maxActive="100" maxIdle="30" maxWait="10000"
               username="javauser" password="javadude" driverClassName="com.mysql.jdbc.Driver"
               url="jdbc:mysql://localhost:3306/javatest"/>
</Context>
```

Context

Application & Servlet

déploiement – web.xml

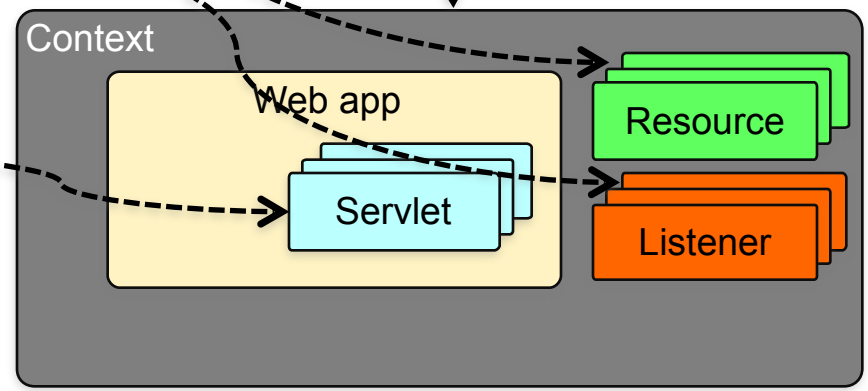
```xml
<?xml version="1.0" encoding="UTF-8"?>

<web-app id="WebApp_ID" version="2.3"
         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_3.xsd">

         <!-- nom de l'application -->
         <display-name>Biography App</display-name>

         <!-- Déclaration de la servlet de recherche -->
         <servlet>
                  <description>Biography searcher servlet</description>
                  <display-name>Biography - search</display-name>
                  <servlet-name>BiographySearcher</servlet-name>

                  <!-- Déclaration de la classe servlet à implémenter -->
                  <servlet-class>univ.rouen.fr.BiographySearcher</servlet-class>

                  <!-- paramètre pour cette servlet -->
                  <init-param>
                           <description>Nombre de recherches autorisées par session</description>
                           <param-name>max-search-by-user</param-name>
                           <param-value>20</param-value>
                  </init-param>
         </servlet>

         <!-- Déclaration de la servlet de recherche -->
         <servlet>
                  <description>Bio viewer servlet</description>
                  <display-name>Biography - view</display-name>
                  <servlet-name>BiographyViewer</servlet-name>

                  <!-- Déclaration de la classe servlet à implémenter -->
                  <servlet-class>univ.rouen.fr.BiographyViewer</servlet-class>
         </servlet>

         <!-- Mapping URL / servlet -->
         <servlet-mapping>
                  <servlet-name>BiographySearcher</servlet-name>
                  <url-pattern>/search</url-pattern>
         </servlet-mapping>
```



Application & Servlet

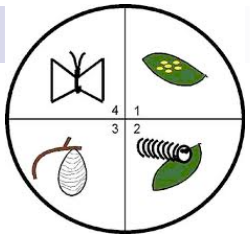
déploiement – web.xml

```
<web-app id="WebApp_ID" version="2.3"
         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_3.xsd">

         <!-- nom de l'application -->
         <display-name>Biography App</display-name>

         <!-- paramètre global à l'application -->
         <context-param>
                  <description>Nombre max de requêtes en parallèle à la base</description>
                  <param-name>max-database-connections</param-name>
                  <param-value>100</param-value>
         </context-param>

         <!-- resources -->
         <resource-ref>
             <description>DB Connection</description>
             <res-ref-name>jdbc/TestDB</res-ref-name>
             <res-type>javax.sql.DataSource</res-type>
             <res-auth>Container</res-auth>
         </resource-ref>

         <!-- listener -->
         <listener>
                  <description>Ecouteur de session</description>
                  <listener-class>univ.rouen.fr.MySessionListener</listener-class>
         </listener>

         <!-- Déclaration de la servlet de recherche -->
         <servlet>
                  <description>Biography searcher servlet</description>
                  <display-name>Biography - search</display-name>
                  <servlet-name>BiographySearcher</servlet-name>

                  <!-- Déclaration de la classe servlet à implémenter -->
                  <servlet-class>univ.rouen.fr.BiographySearcher</servlet-class>

                  <!-- paramètre pour cette servlet -->
                  <init-param>
                           <description>Nombre de recherches authorisées par session</description>
                           <param-name>max-search-by-user</param-name>
                           <param-value>20</param-value>
                  </init-param>
         </servlet>

         ...
```

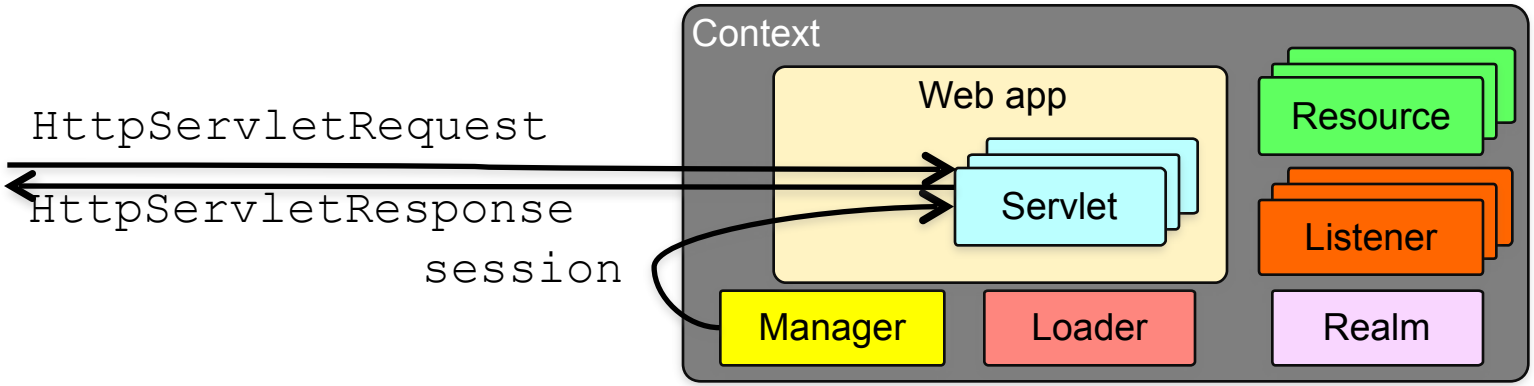


Application & Servlet

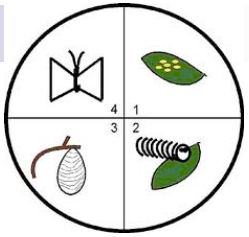
initialisation & traitement des requêtes



- Au premier appel d'une servlet (http), Tomcat :
  - initialise la servlet appelée (possibilité de configurer lors du déploiement)
- Puis…
  - initialise la `HttpSession`, `HttpServletRequest`, `HttpServletResponse`
  - appel la méthode `doPost()`, `doGet()`, etc. en fonction du type de requête

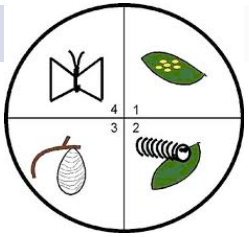


Application & Servlet




```
public class BiographySearcher extends HttpServlet {

        private static int MAX_REQ;
        private static int NB_REQ;
        private static final String URI = "xmldb:exist://localhost:8080/exist/xmlrpc/db";
        private static final String driver = "org.exist.xmldb.DatabaseImpl";
        private static final String collectionName = "CVTheque";
        private static final String dbadmin_user="admin";

        public void init(ServletConfig config) throws ServletException {
                MAX_REQ = Integer.valueOf(config.getInitParameter("max-search-by-user"));
                // Obtain our environment naming context
                Context initCtx;
                try {
                        initCtx = new InitialContext();
                        Context envCtx = (Context) initCtx.lookup("java:comp/env");

                        // Look up our mySQL data source
                        DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");

                        // initialize exist XML database driver
                        Class<Database> cl = (Class<Database>) Class.forName(driver);
                        Database database = (Database) cl.newInstance();
                        database.setProperty("create-database", "true");
                        DatabaseManager.registerDatabase(database);
                } catch (NamingException e) {
                        throw new ServletException("Unable to get JNDI resource");
                } catch (ClassNotFoundException e) {
                        throw new ServletException("DB driver not found");
                } catch (XMLDBException e) {
                        throw new ServletException("DB error");
                } catch (InstantiationException e) {
                        throw new ServletException("DB error");
                } catch (IllegalAccessException e) {
                        throw new ServletException("DB error");
                }
                super.init(config);
        }
…
}
```

Application & Servlet



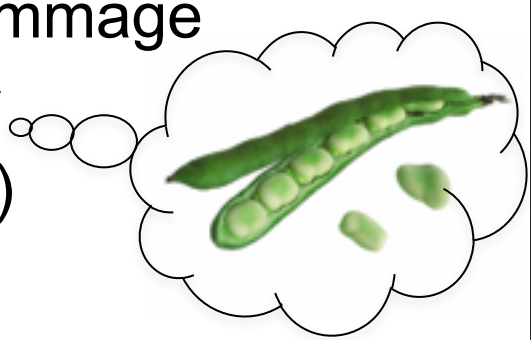

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
{
	MAX_REQ++;
	int nbReq = (Integer) req.getSession().getAttribute("nbReq");
	nbReq++;

	req.getSession().setAttribute("nbReq", nbReq);

	if (nbReq>MAX_REQ) {
		resp.getOutputStream().print("to many requests");
	} else {
		// search using the DB
	}
}
```

Java Beans

JavaBeans

généralités

- JavaBeans
  - recommandations et conventions de nommage
  - encapsule les attributs d'un objet métier
  - sérialisable (sauvegarde et restauration)
  - constructeur sans paramètre
  - getters and setters sur les attributs
  - gestion de listeners (changement des attributs)
  - info / description (BeanInfo, BeanDescriptor)



JavaBeans

pour les GIL…

- JavaBeans
  - recommandations et conventions de nommage
  - encapsule les attributs d'un objet métier
  - sérialisable (sauvegarde et restauration)
  - constructeur sans paramètre
  - getters and setters sur les attributs
  - gestion de listeners (changement des attributs)
  - info / description (BeanInfo, BeanDescriptor)

Utilisé en Swing / AWT / JSF

JavaBeans

exemple

```
public class BioBean {
	// properties
	List<String> faitsNotables;
	String nom;
	String prenom;

	// constructor
	public BioBean() throws FileNotFoundException {
		faitsNotables = new LinkedList<String>();
	}

	/**
	 * @return the faitsNotables
	 */
	public List<String> getFaitsNotables() {
		return faitsNotables;
	}

	/**
	 * @param faitsNotables the faitsNotables to set
	 */
	public void setFaitsNotables(List<String> faitsNotables) {
		this.faitsNotables = faitsNotables;
	}

}
```

JavaBeans

```java
public class BioBean {
	// properties
	List<String> faitsNotables;
	String nom;
	String prenom;
	Image image;


	// constructor
	public BioBean() throws FileNotFoundException {
		faitsNotables = new LinkedList<String>();
	}

	/**
	 * @return the faitsNotables
	 */
	public List<String> getFaitsNotables() {
		return faitsNotables;
	}

	/**
	 * @param faitsNotables the faitsNotables to set
	 */
	public void setFaitsNotables(List<String> faitsNotables) {
		this.faitsNotables = faitsNotables;
	}

	/**
	 * @return the image
	 */
	public Image getImage() {
		return image;
	}

	/**
	 * @param image the image to set
	 */
	public void setImage(Image image) {
		this.image = image;
	}
}
```

exemple 2

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS

SUMMARY: NESTED | FIELD | CONSTR | METHOD

java.awt

Class Image

java.lang.Object
  └ java.awt.Image

Direct Known Subclasses:

BufferedImage, VolatileImage

public abstract class **Image**
extends Object

The abstract class `Image` is the superclass of all classes that represent graphica

Since:

JDK1.0

Field Summary	
static int	**SCALE_AREA_AVERAGING** Use the Area Averaging image scaling algorithm.
static int	**SCALE_DEFAULT** Use the default image-scaling algorithm.

Entreprise Java Beans

aperçu & généralités

- Entreprise JavaBeans (EJB)
  - correspond aux objets métiers
  - permet de créer des composants JEE distribués
  - composants autosuffisants
  - fonctionne dans un contexte transactionnel
  - possède un cycle de vie géré par le conteneur d'EJB (création, activation, passivation, etc…)
  - utilise les annotations Java

Quelques similarités entre JavaBeans et EJB

Entreprise Java Beans

types d'EJB

- Session EJB (SessionBean)
  - propose des services aux clients
  - conserve leur état entre appels : stateful
  - ne conserve PAS leur état entre appels : stateless
- Entité EJB (EntityBean)
  - objet qui a pour vocation de gérer sa persistance
  - utilisation de la Java Persistance API
  - persistance effectuée par le conteneur
  - mapping vers une base de données relationnelle

Entreprise Java Beans

types d'EJB

- Message EJB (Message Bean)
  - traite les messages d'une pile gérée par le serveur
  - le serveur instancie et transmet le message au bean

EJBeans

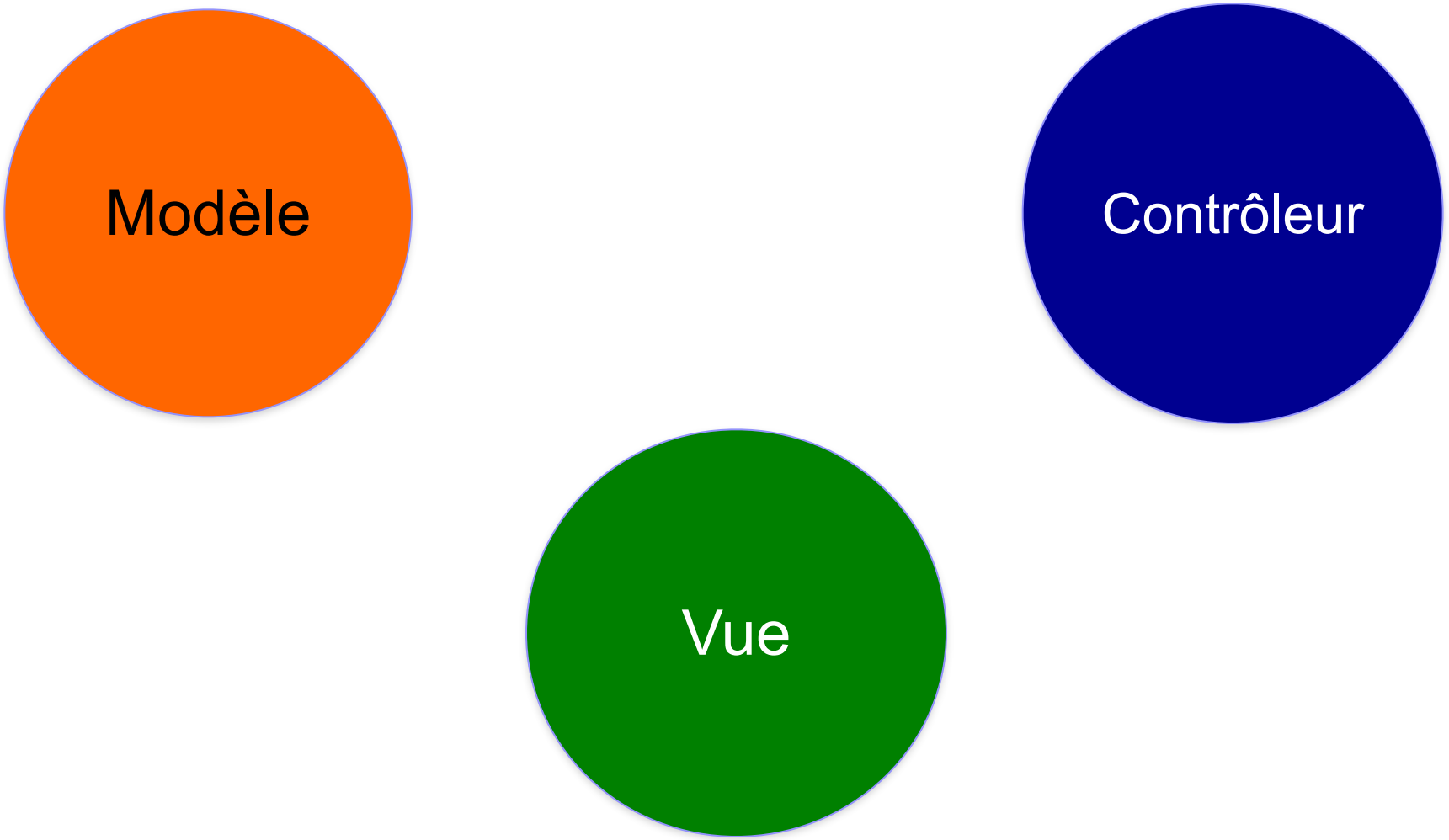
exemple d'un EntityBean

```
package biography;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="WEB_BOOKSTORE_BOOKS")

public class Book implements Serializable {

    private String bookId;
    private String title;

    public Book() { }

    public Book(String bookId, String title, ...) {
        this.bookId = bookId;
        this.title = title;
        ...
    }

    @Id
    public String getBookId() {
        return this.bookId;
    }

    public String getTitle() {
        return this.title;
    }
    ...

    public void setBookId(String id) {
        this.bookId=id;
    }

    public void setTitle(String title) {
        this.title=title;
    }
}
```

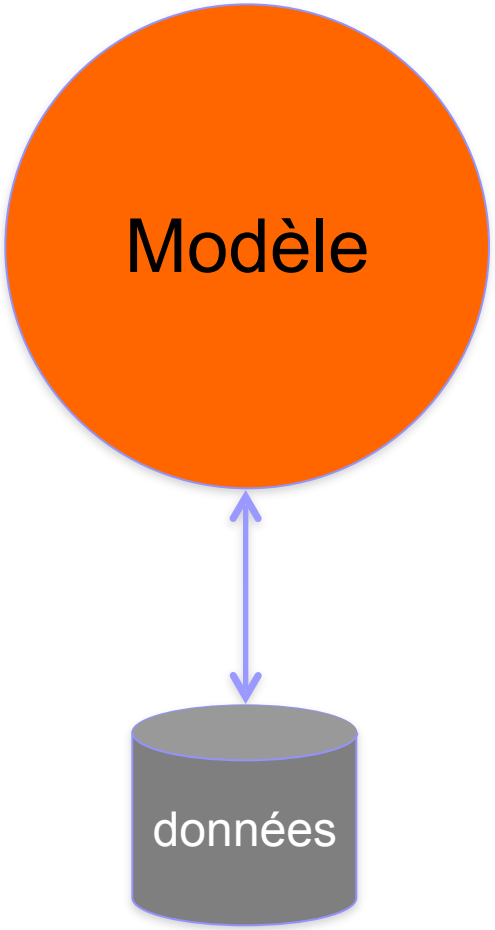
Pas cette année…

De plus, Tomcat n'est pas un conteneur d'EJB…

Application web et MVC



Application web et MVC



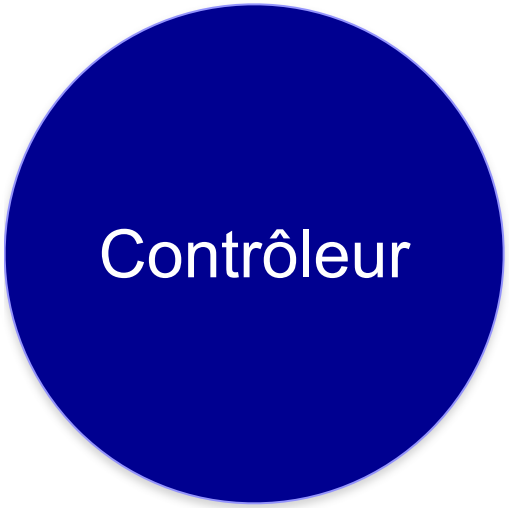
- Correspond au « Business Tier »
- Objets qui permettent d'interagir avec les données
- Opérations liées au métier
  - Ajouter un cv dans la base
  - Modifier un cv
  - Rechercher des cv
  - etc.
- Implémenté dans : EJB, services web & autres classes java

Contrôleur

Vue

Application web et MVC

- Gère les interactions entre l'utilisateur et le modèle
- Assure la cohérence du modèle
- Appelle les opérations du modèle liées au métier

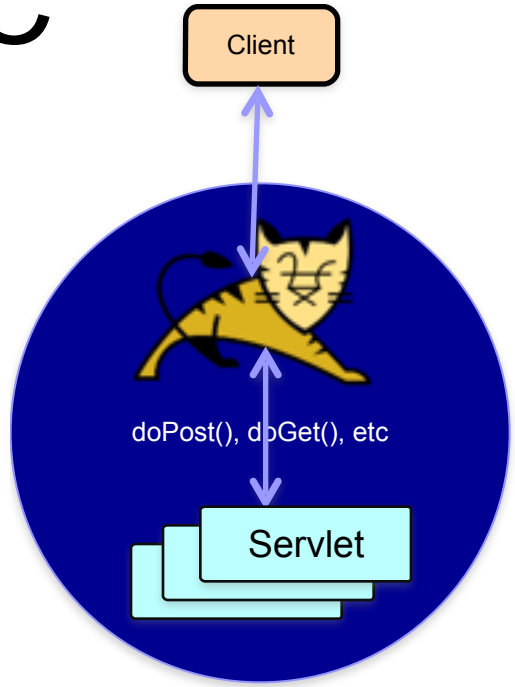


Modèle

Vue

Application web et MVC

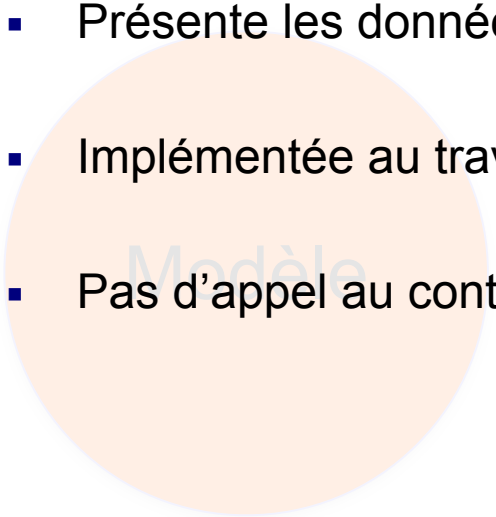
- Dans Tomcat, implémenté par Catalina
  - Produit et contrôle les « Objets requêtes » entrantes et sortante (entêtes, sessions, cookies, etc.)
  - Contrôle et filtre les requêtes (robinets & filtres)
  - Adresse les requêtes aux web apps & web components (appel les méthodes doPost(), doGet(), etc. des servlets)



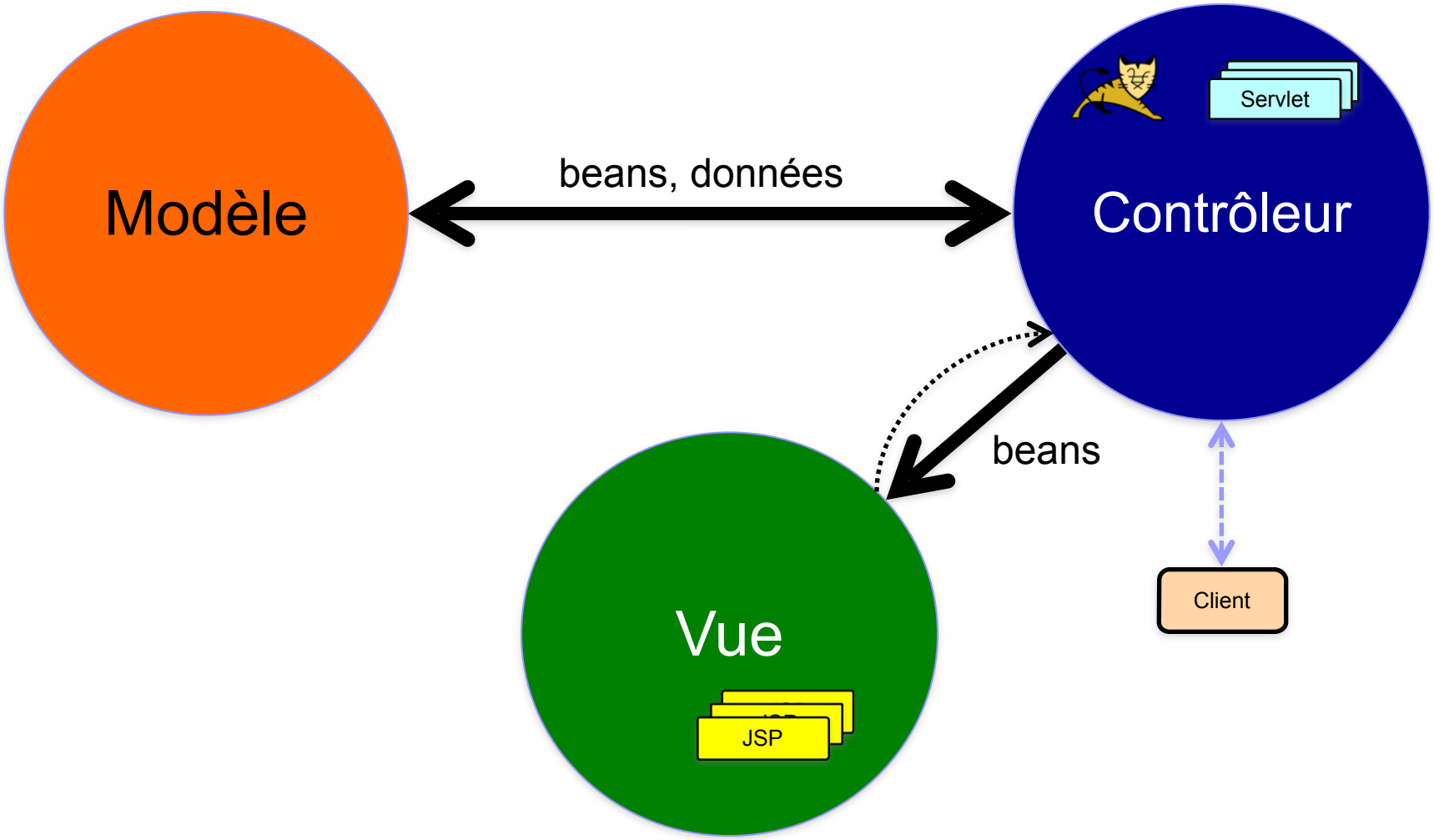
- Dans les servlets
  - Dans les méthodes doPost(), doGet(), etc.
  - Contrôle les paramètres des requêtes et appelle les méthodes du modèle

Application web et MVC

- Présente les données issues du modèle
- Implémentée au travers des pages JSP
- Pas d'appel au contrôleur (pas de listener)



Application web et MVC



Servlet & MVC

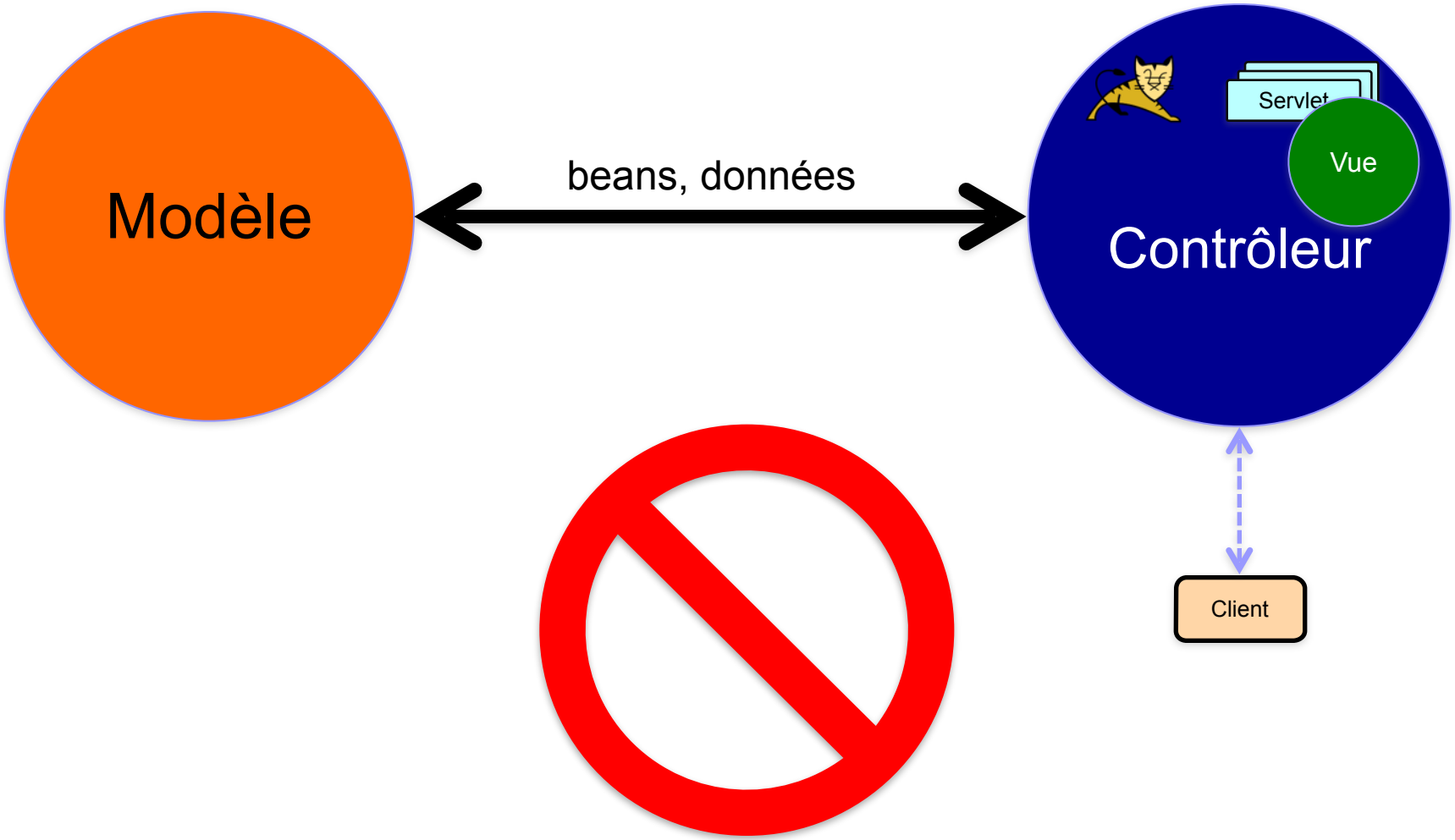
En MVC

Lecture de formulaire

```
1  import javax.servlet.*;
2  import javax.servlet.http.*;
3  import java.util.*;
4  public class UserInfo extends HttpServlet {
5    public void doPost(HttpServletRequest request,
6                       HttpServletResponse response)
7      throws ServletException, IOException {
8      response.setContentType("text/html");
9      PrintWriter out = response.getWriter();
10     out.println("<HTML>\n<BODY>\n" +
11       "<H1>Recapitulatif des informations</H1>\n"
12     "<UL>\n" +
13     "  <LI>Nom: "
14         + request.getParameter("Nom") + "\n" +
15         "  <LI>Prenom: "
16         + request.getParameter("Prenom") + "\n" +
17         "  <LI>Age: "
18         + request.getParameter("Age") + "\n" +
19         "</UL>\n" +
20     "</BODY></HTML>");
21   }
22 }
```

Servlet & MVC

Utilisation du writer pour produire la vue



Utilisation du writer

pour produire la vue…

- Problématiques liées au développement
  - pas d'utilisation du pattern MVC
  - modification complexe de la vue
  - nécessité de recompiler la classe Servlet
  - lisibilité du code HTML
  - lisibilité du code Java
  - pas de contrôle des balises HTML dans l'IDE
- Problématiques liées aux performances
  - gestion du cache
  - pré-compilation et optimisation des JSP

Servlet & MVC

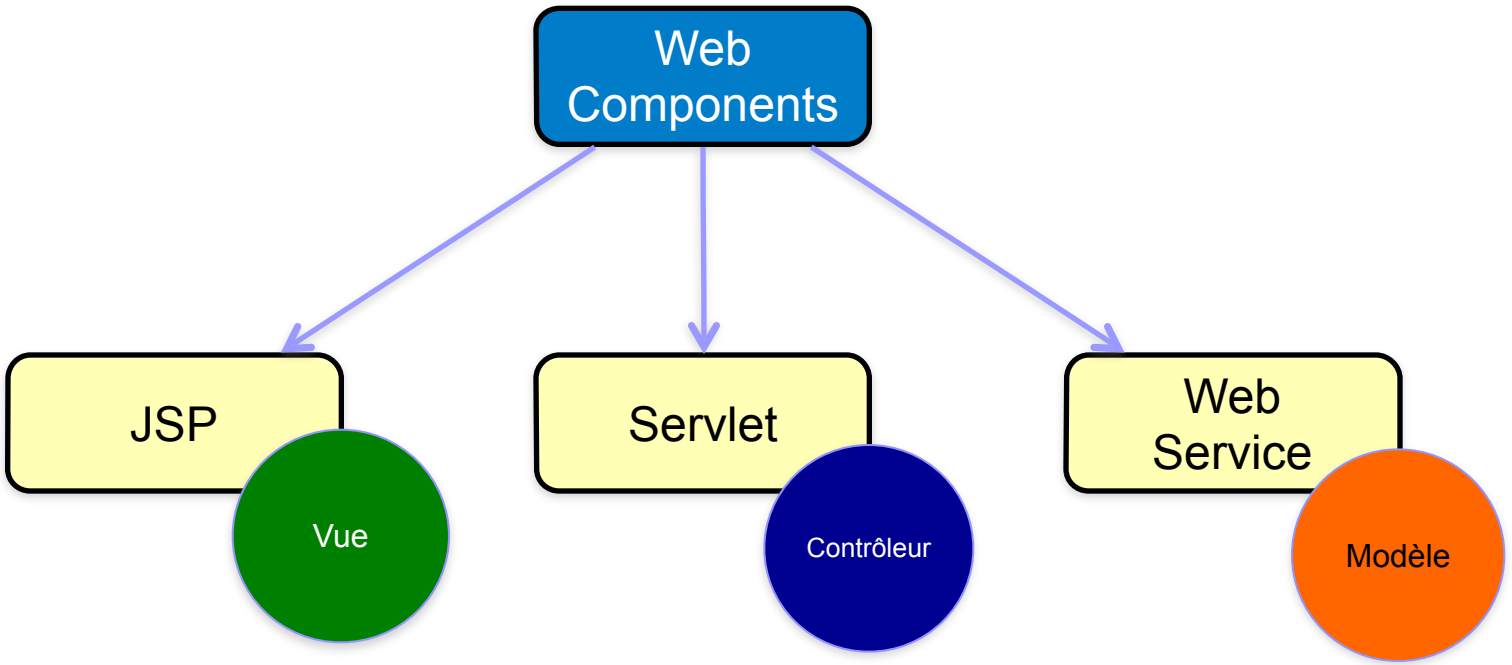
Cas d'utilisation du writer

En MVC sauf…

Cas d'utilisation	Utilisation du writer
Renvoyer une page HTML/XHTML	✗
Renvoyer un arbre XML	✔
Renvoyer un tableau JSON	✔
Renvoyer des données binaires (image / vidéo / fichier)	✔
Renvoyer du texte brut	✔

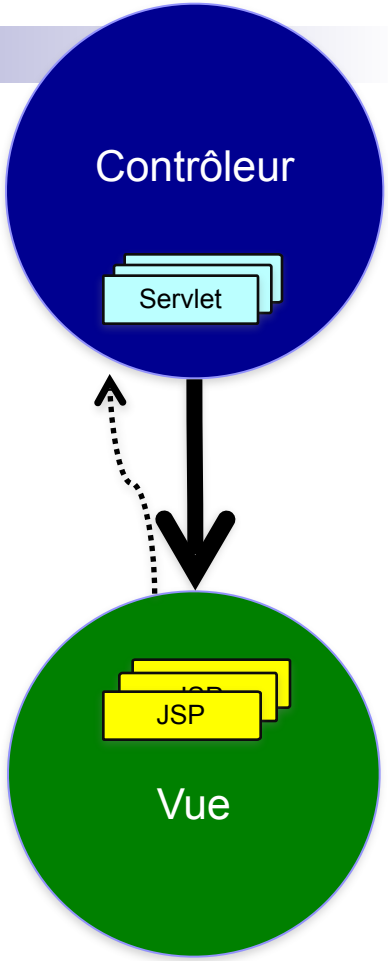
✔ Toujours privilégier l'utilisation des JSP pour produire les vues HTML

Usage acceptable …



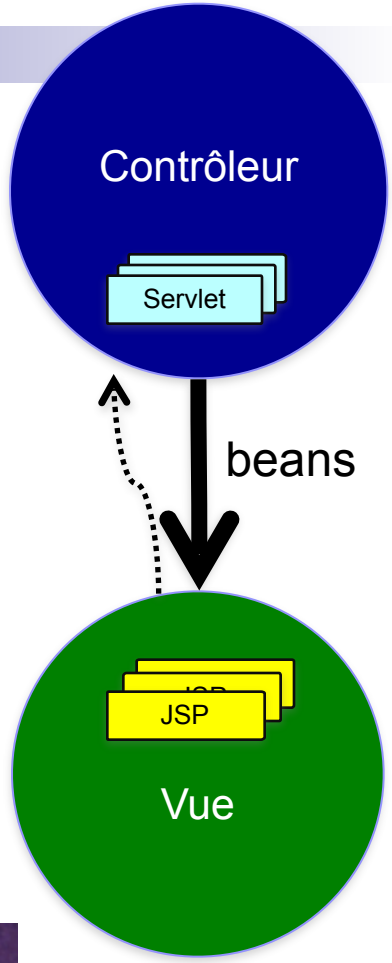
How to MVC ?

- Utilisation d'un **dispatcher**
  - récupération du dispatcher
  - sélection de la destination
  - envoie de la requête à destination
  - incorporation de la réponse (include)
  - délégation à la destination (forward)



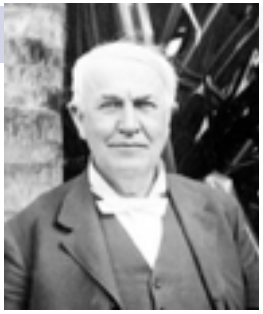
How to MVC ?

- Utilisation des **beans**
  - récupération des beans auprès du modèle
  - ajout des beans dans la requête
  - envoie de la requête à destination (dispatching)
  - utilisation des beans dans la page



How to dispatch ?

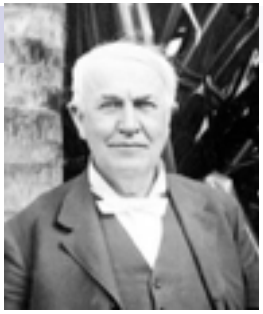
to a JSP…



```
/**
 * @see javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
 */
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
	try {
		// création des beans
		EtatCivilBean etatCivil = new EtatCivilBean("Edison", "Thomas", DateFormat.getDateInstance().parse("1831-10-18"));
		BioBean bio = new BioBean();
		bio.getFaitsNotables().add("Pionnier de l'éléctricté");
		bio.getFaitsNotables().add("Nombre record de brevets (1093)");
		bio.getFaitsNotables().add("Fondateur de General Electric");
		bio.getFaitsNotables().add("Inventeur (auto-proclamé) du téléphone et du cinéma");

		// injection des beans
		req.setAttribute("etatCivilBean", etatCivil);
		req.setAttribute("bioBean", bio);

		// récupération du dispatcher
		RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/jsp/biographie.jsp");

		// envoi à la jsp
		dispatcher.include(req, resp);

	} catch (Exception e) {

	}
}
```

How to dispatch ?

to another servlet…



```
/**
 * @see javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
 */
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
	try {
		// création des beans
		EtatCivilBean etatCivil = new EtatCivilBean("Edison", "Thomas", DateFormat.getDateInstance().parse("1831-10-18"));
		BioBean bio = new BioBean();
		bio.getFaitsNotables().add("Pionnier de l'éléctricté");
		bio.getFaitsNotables().add("Nombre record de brevets (1093)");
		bio.getFaitsNotables().add("Fondateur de General Electric");
		bio.getFaitsNotables().add("Inventeur (auto-proclamé) du téléphone et du cinéma");

		// injection des beans
		req.setAttribute("etatCivilBean", etatCivil);
		req.setAttribute("bioBean", bio);

		// récupération du dispatcher
		RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("AnotherServlet");

		// envoi à la jsp
		dispatcher.forward(req, resp);

	} catch (Exception e) {

	}
}
```

How to MVC ?

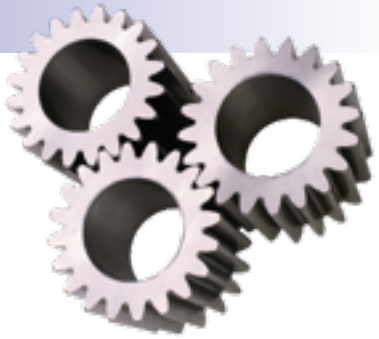
using a model…

```
/** (non-Javadoc)
* @see javax.servlet.http.HttpServlet#doPost(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
 */
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
	// récupération des paramètres
	String vipID = req.getParameter("vip_ID");

	// utilisation du modèle pour récupérer les beans
	BiographyModel model = new BiographyModel();

	EtatCivilBean etatCivil = model.getEtatCivil(vipID);
	BioBean bio = model.getBio(vipID);


	// injection des beans
	req.setAttribute("etatCivilBean", etatCivil);
	req.setAttribute("bioBean", bio);

	// récupération du dispatcher
	RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/jsp/biographie.jsp");

	// envoi à la jsp
	dispatcher.include(req, resp);
}
```

Java Server Pages

Java Server Page

rappels



- Objectif & fonctionnement
  - traite les requêtes transmises par le conteneur
  - construire dynamiquement du code HTML/XML en réponse
  - compilée en pseudo-code Java (servlet) par le compilateur JSP
  - syntaxe JSP basée sur des balises XML

Java Server Page

rappels

- Syntaxe et structure (4 parties)
  - Données statiques
  - Directives
  - Scripts
  - Actions

Java Server Page

rappels

- Données statiques
  - contenu HTML ou XML

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>


<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>CVTheque - Upload</title>
</head>
<body>

        Ajouter un cv

        <form action="/CVTheque/upload" method="post" enctype="multipart/form-data">
                <input type="file" name="cvxml">
                <input type="submit">
        </form>

        <a href="/CVTheque/search"><fmt:message key="searchPage"/></a>
</body>
</html>
```

Java Server Page

rappels

- Directives
  - déclarées entre les balises <%@ … %>
  - directives pour la compilation
    - include : inclusion de fichier
    - page : import de package, type mime, encodage, page d'erreur, threadSafe
    - taglib : préfix et URI de la librairie

Java Server Page

rappels

- Directives
  - déclarées entre les balises <%@ … %>
  - directives pour la compilation

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
```

directive page

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
```

directive taglib

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>CVTheque - Upload</title>
</head>
<body>

        Ajouter un cv

        <form action="/CVTheque/upload" method="post" enctype="multipart/form-data">
                <input type="file" name="cvxml">
                <input type="submit">
        </form>
```

Java Server Page

rappels

- Scripts
  - ajouter du code Java dans la page
  - éléments de la partie script :
    - déclaration : variables ou méthodes Java (balise <%! …%>)
    - scriplet : déclaration du code java (balise <% …%>)
    - expression : évalue et ajoute le résultat d'une expression dans la réponse (balise <%= …%>)
    - commentaire (balise <%-- … --%>)

Java Server Page

rappels

- Scripts
  - Objets Java déclarés implicitement dans la page utilisables dans les scripts
    - request : requête transmise par conteneur
    - response : réponse renvoyée au conteneur
    - session : session utilisateur
    - out : flot de sortie de la réponse
    - méthode log()
    - pageContext : contexte et attributs de la page

Java Server Page

rappels

- Scripts

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

<%! int monEntier = 15; %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<!-- contenu de la page -->
<html>
	<head>
		<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
		<title>Page de test</title>
	</head>

	<body>

	<%
	for (int i=0; i< 10; i++ ) {
		monEntier++;
	}
	%>

	<h1>Mon entier vaut :
		<%=monEntier %>
	</h1>

</body>
</html>
```

déclaration

commentaire

scriplet

expression

Java Server Page

rappels

- Actions
  - action à effectuer
    - include <jsp:include> : inclure la contenu d'une page (balise)
    - forward <jsp:forward> : déléguer à une page (balise)
    - param <jsp:param> : ajouter les paramètres à la requête lors du forward ou include
    - useBean <jsp:useBean> : créer ou utilise un Java Bean dans la page
    - setProperty <jsp:setProperty> : fixe la propriété d'un bean
    - getProperty <jsp:getProperty> : récupère la propriété d'un bean
  - actions étendues proposées par des librairies (taglib)

${data }

Expression Language (EL)

généralités

- Définition
  - Langage simplifiant la manipulation des données dans les pages JSP
  - Manipulation des types Java primaires
  - Manipulation d'objets implicites (liés à la page)
  - Manipulation des Java Beans
  - Utilisé conjointement avec les taglib

Expression Language (EL)

généralités

- Objectifs
  - Se passer (complètement) de l'utilisation des scriptlets
  - Améliorer la lisibilité de la page JSP
  - Comprendre la couche de présentation sans connaissances du particulière du Java

Expression Language (EL)

généralités

- Syntaxe

```
${ expression }
```

- Expression

```
${terme opérateur terme opérateur terme…}
```

- Terme
  - Type primaire

▪utilise le wrapper de la classe correspondante

▪`Long`, `Double`, `String`, `Boolean`

Expression Language (EL)

généralités

- Terme (suite)
  - Objet implicite
    - objets accessibles dans la page
    - `pageContext`
    - attributs scopés: `pageScope`, `requestScope`, `sessionScope`, `applicationScope`
    - paramètre HTTP : `param`, `paramValues` (tableau)
    - paramètre initialisation : `initParam`
    - header : `header`, `headerValues` (tableau)
    - cookies : `cookie`

Expression Language (EL)

- Terme (suite)
  - Fonction EL
    - fonctions accessibles dans la page
    - fonctions déclarées sous la forme de Taglib

```
${prefix:nomFonction([paramètres])}
```

Expression Language (EL)

généralités

- Opérateurs
  - Arithmétique
    - applicable aux valeurs numériques
    - `+` , `-` , `*`, `/` (`div`) , `%` (`mod`)
  - Relationnel
    - applicables aux objets proposant les méthodes equals() & compareTo()
    - `==` (`eq`), `!=` (`ne`), `<` (`lt`), `>` (`gt`), `<=` (`le`), `>=` (`ge`)
  - Logiques
    - applicable aux booléens
    - `&&` (`and`) , `||` (`or`) , `!` (`not`)

Expression Language (EL)

- Opérateurs (suite)
  - Autres
    - applicable aux chaines, map, liste, tableau et null
    - `empty`

Expression Language (EL)

généralités

- Accès aux propriétés d'un objet
  - Map, tableau, List, bean
  - utilise les getters (réflexion)
  - opérateur point

```
${object.property}
```

  - opérateur crochet

```
${object['property']}
```

Expression Language (EL)

exemple

- Exemples

```
${5 > 6 }
${19 mod 6 }
${empty cvList }
${sessionScope.userName == 'Edison'}
${sessionScope['userName'] != 'Bob'}
${initParam.maxConnexions }
${cv[etatCivil.nom] }
${fn:escapeXml(cv[etatCivil][nom]) }
```

Expression Language (EL)

généralités

- Recherche d'attribut
  - recherche d'un attribut dans les différents scopes, par son nom

```
${user}
```

- Ordre et priorité
  - attribut de même nom dans différents scopes ?
  - ordre : page, request, session, application
  - utiliser le scope ! `${sessionScope[user]}`

Expression Language (EL)

exemple

```
<!-- contenu de la page -->
<html>
      <head>
            <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
            <title>Page de test</title>
      </head>

      <body>

      <h1>Mon nom d'utilisateur est ${sessionScope['userName']}</h1>

      19 modulo 6 : ${19 mod 6 }

</body>
</html>
```

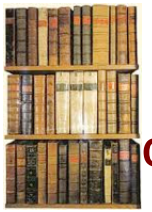
JSP Tag Library

<% taglib

%>

<% taglib

JSP Tag Library



%>

généralités

- Définition
  - Librairie (logicielle) utilisable au moyen de tag
  - Balises basées sur XML
  - Collection d'actions utilisable dans une JSP
  - Étend les actions JSP de base : les actions jsp font partie de la taglib JSP
  - Chaque balise correspond à une action
  - Chaque action correspond à un traitement
  - La librairie (i.e. les traitements) est codée en Java

<% taglib

JSP Tag Library

%>

exemple

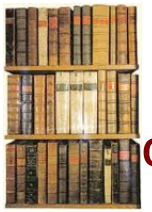
```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

<%! int monEntier = 15; %>

<html>
    <head>
        <%-- Ceci est le header de la page HTML produite --%>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>test</title>
    </head>
    <body>
        <%
        for ( int i = 0; i < 10; i++) {
            monEntier++;
        }
        %>

        <h1>Mon entier vaut :
            <%= monEntier %>
        </h1>
    </body>
</html>
```

Codé dans la TagLib !

<maTagLib:monTraitement param="10"/>

remplacé par

JSP Tag Library

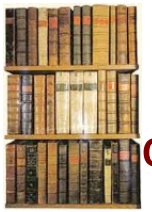


généralités

- Objectifs
  - Limiter l'utilisation des scriptlets
  - Se passer des scriptlets (combiné à EL)
  - Masquer le code Java associé aux actions
  - Améliorer la lisibilité de la page JSP
  - Comprendre la couche de présentation sans connaissances du particulière du Java

<% taglib

JSP Tag Library



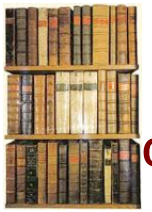
%>

définitions

- Composition : 3 éléments
  - **descripteur** (Tag Librairie Descriptor) : fichier *.tld qui fait le **mapping** entre les tags et les méthodes de la classe Java
  - classes Java **implémentant** les tags
  - classe (optionnelle) apportant les informations supplémentaire de la TagLib (**TagExtraInfo**)

<% taglib

JSP Tag Library



%>

exemple de taglib

- Java Standard Tag Library (JSTL)
  - spécification Sun pour une librairie standard
  - implémentation libre (adaptée au serveur JEE)
  - core (base) : http://java.sun.com/jsp/jstl/core
  - Format : http://java.sun.com/jsp/jstl/fmt
  - XML : http://java.sun.com/jsp/jstl/xml
  - SQL : http://java.sun.com/jsp/jstl/sql
  - Fonctions : http://java.sun.com/jsp/jstl/functions

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Principales actions
  - actions des base
    - afficher une expression
    - définir / supprimer une variable ou une propriété (scopée)
  - action conditionnelles
    - if / choose / when / otherwise
  - itérations
    - boucles ForEach & forTokens
  - URLs
    - création d'URLs complexes, redirection & imports

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Afficher une expression
  - **<c:out />**
    - value (objet) : expression à afficher
    - default (objet) :expression par défaut (si valeur nulle)
    - échappement XML

```
<c:out value="${cv.title}" escapeXml="false"/>
```

Équivalent à *${cv.title}*

```
<c:out value="${cv.title}" default="titre" escapeXml="true" />
```

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Définir & supprimer une variable / propriété
- Changer la valeur d'une propriété d'un bean

<c:set /> <c:remove /> :

  - value : expression à évaluer
  - var : nom de la variable
  - scope : scope où est définie la variable (page, request, session, application)
  - target : l'objet à modifier
  - property : propriété de l'objet à modifier

<% taglib

%>

Taglib core <c:/>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

- Exemple

```
<c:set value="${cv.title}" var="cvTitre" scope="page" />

<c:set target="${session['user']}" property="name">
Edison
</c:set>
```

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Interception des erreurs
  - **<c:catch />** : interception des erreurs

```
<c:catch>
    <c:set value="${cv.title}" var="cvTitre" scope="page" />
</c:catch>
```

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Traitement conditionnel
  - **<c:if />** :
    - test (boolean) : expression à évaluer
    - var (string) : nom de la variable qui contiendra le résultat du test
    - scope (string) : scope où est définie la variable (page, request, session, application)

```
<c:if test="${empty cv.etatCivil.age > 18}" >
   <c:out value="${cv.title}"/>
</c:if>
```

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Traitement conditionnel exclusif (switch/if/then/else)
  - **<c:choose /> <c:when /> <c:otherwise />**
    - test (boolean) : expression à évaluer (dans <c:when />)

```
<c:choose>
    <c:when test="${empty cv.etatCivil.age > 18}" >
            <p><c:out value="${cv.title}"/></p>
        </c:when>
        <c:otherwise>
            <p>Trop jeune !</p>
        </c:otherwise>
</c:choose>
```

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Boucle itérative (for / while)
  - **<c:forEach />**
    - items (object) : collection à itérer
    - var (string) : nom de la variable qui contiendra l'élément courant de l'itération
    - varStatus (string) : nom de la variable qui contiendra le statut de l'itération
    - begin (int) : indexe de départ de l'itération
    - end (int) : indexe de fin de l'itération
    - step (int) : nombre d'itération dans la collection

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Boucle itérative (for / while)
  - Objet varStatus (type **LoopTagStatus**) dans la boucle
    - begin (int) : valeur de l'attribut begin
    - end (int) : valeur de l'attribut end
    - step (int) : valeur de l'attribut step
    - count (int) : le nombre de tour d'itérations effectuées
    - current (object) : item courant de la collection itérée
    - index (int) : position de l'élément courant dans la collection
    - first (boolean) : première itération
    - last (boolean) : dernière itération

<% taglib

%>

Taglib core <c:/>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

- Exemple

```
<c:forEach var="cv" items="${session['listCV']}" />
   <p>${cv.titre}</p>
</c:forEach>

<c:forEach var="cv" items="${session['listCV']}" begin="0"  end="4"/>
   <p>${cv.titre}</p>
</c:forEach>

<c:forEach var="mapEntry" items="${session['mapCV']}"/>
   <p>clé : ${mapEntry.key}</p>
   <p>valeur : ${mapEntry.value}</p>
</c:forEach>
```

<% taglib

%>

Taglib core <c:/>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

- Exemple (suite)

```
<c:forEach var="cv" varStatus="status" items="${session['listCV']}"/>
   <p>CV - ${status.count} : ${cv.titre}</p>
</c:forEach>
```

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Boucle itérative sur des tokens
  - **<c:forTokens />**
    - items (**string**) : chaine à découper
    - delims (string) : caractères servant de délimiteur
    - var (string) : nom de la variable qui contiendra l'élément courant de l'itération
    - varStatus (string) : nom de la variable qui contiendra le statut de l'itération
    - begin (int) : indexe de départ de l'itération
    - end (int) : indexe de fin de l'itération
    - step (int) : nombre d'itération dans la collection

Taglib core <c:/>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

- Exemple

```
<c:forTokens var="token" items="cv1;cv2;cv3;cv4" delims=";" />
   <p>${token}</p>
</c:forEach>
```

<% taglib

%>

Taglib core <c:/>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

- Création d'URLs (complexes)
  - **<c:url />**
    - value (string) : URL absolue ou relative au contexte
    - context (string) : URL relative à un autre contexte (autre application locale)
    - var (string) : nom de la variable qui contiendra l'URL
    - scope(string) : scope dans lequel sera stockée la variable
  - **<c:param/>**
    - name (string) : nom du paramètre
    - value (string) : valeur du paramètre

Taglib core <c:/>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

- Exemple

```
<c:url value="view.jsp"  var="complexURL" >
   <c:forEach var="cvMapEntry" items="${session['mapCV']}"/>
   <c:param name="cvMapEntry.key" value="cvMapEntry.value"/>
   </c:forEach>
</c:url>

<a href="${complexURL}">Lien</a>
```