

Architectures des SI - Les architectures 3-tiers

Partie III : les composants WEB

Informatique, Statistique et IA 4^{ème} année

Olivier Caron¹

`http://ocaron.polytech-lille.net`

¹École d'ingénieurs Polytech Lille
Université de Lille

30 janvier 2026



Luc Fayard

Cookie : Anciennement petit gâteau sucré, qu'on acceptait avec plaisir.
Aujourd'hui : petit fichier informatique drôlement salé, qu'il faut refuser avec véhémence.

Quelques références

- API Jakarta JEE :
<https://jakarta.ee/specifications/platform/9.1/apidocs/>

Quelques références

- API Jakarta JEE :
<https://jakarta.ee/specifications/platform/9.1/apidocs/>
- Un résumé des tags JSTL :
<http://homepage.divms.uiowa.edu/~slonnegr/wpj/jqr.pdf>

Quelques références

- API Jakarta JEE :
<https://jakarta.ee/specifications/platform/9.1/apidocs/>
- Un résumé des tags JSTL :
<http://homepage.divms.uiowa.edu/~slonnegr/wpj/jqr.pdf>
- Patron MVC et JEE :
<http://orm.bdpedia.fr/mvc.html>

Les architectures 3-tiers : le premier niveau

- Le premier niveau : IHM (Interface Homme-Machine)

Les architectures 3-tiers : le premier niveau

- Le premier niveau : IHM (Interface Homme-Machine)
 - ▶ Uniquement l'aspect visuel :

Les architectures 3-tiers : le premier niveau

- Le premier niveau : IHM (Interface Homme-Machine)
 - ▶ Uniquement l'aspect visuel :
 - ★ Affichage des données

Les architectures 3-tiers : le premier niveau

- Le premier niveau : IHM (Interface Homme-Machine)
 - ▶ Uniquement l'aspect visuel :
 - ★ Affichage des données
 - ★ Transfert d'informations (formulaire)

Les architectures 3-tiers : le premier niveau

- Le premier niveau : IHM (Interface Homme-Machine)
 - ▶ Uniquement l'aspect visuel :
 - ★ Affichage des données
 - ★ Transfert d'informations (formulaire)
 - ▶ Pas de code métier !

Les architectures 3-tiers : le premier niveau

- Le premier niveau : IHM (Interface Homme-Machine)
 - ▶ Uniquement l'aspect visuel :
 - ★ Affichage des données
 - ★ Transfert d'informations (formulaire)
 - ▶ Pas de code métier !
 - ▶ Plusieurs interfaces possibles pour une même application (Wap, PC, PDA,...)

Les architectures 3-tiers : le premier niveau

- Le premier niveau : IHM (Interface Homme-Machine)
 - ▶ Uniquement l'aspect visuel :
 - ★ Affichage des données
 - ★ Transfert d'informations (formulaires)
 - ▶ Pas de code métier !
 - ▶ Plusieurs interfaces possibles pour une même application (Wap, PC, PDA, ...)
 - ▶ Un protocole privilégié : le WEB (http)

Les architectures 3-tiers : le premier niveau

- Le premier niveau : IHM (Interface Homme-Machine)
 - ▶ Uniquement l'aspect visuel :
 - ★ Affichage des données
 - ★ Transfert d'informations (formulaires)
 - ▶ Pas de code métier !
 - ▶ Plusieurs interfaces possibles pour une même application (Wap, PC, PDA, ...)
 - ▶ Un protocole privilégié : le WEB (http)
 - ★ déploiement automatique des applications !

Les architectures 3-tiers : le premier niveau

- Le premier niveau : IHM (Interface Homme-Machine)
 - ▶ Uniquement l'aspect visuel :
 - ★ Affichage des données
 - ★ Transfert d'informations (formulaires)
 - ▶ Pas de code métier !
 - ▶ Plusieurs interfaces possibles pour une même application (Wap, PC, PDA, ...)
 - ▶ Un protocole privilégié : le WEB (http)
 - ★ déploiement automatique des applications !
 - ▶ Les qualités attendues : ré-utilisabilité, évolutivité

Le langage HTML

- CERN de Genève , les Normes (<http://www.w3c.org>)
- Langage à base de "tags" (balises) :

```
<nomCommande attribut1="valeur1" ... attributN="valeurN">  
...  
</nomCommande>  
ou  
<nomCommande ... />
```

- être conforme à la norme : <http://validator.w3.org/>

Les pages webs dynamiques : les applets Java (1/3)

Définition

Une applet Java est un programme **compilé**, téléchargé sur le web, et interprété au sein d'un navigateur.

```
import java.applet.Applet;  
import java.awt.Graphics;  
  
public class HelloWorld extends Applet {  
    public void paint(Graphics g) {  
        g.drawString("Hello world!", 50, 25);  
    }  
}
```

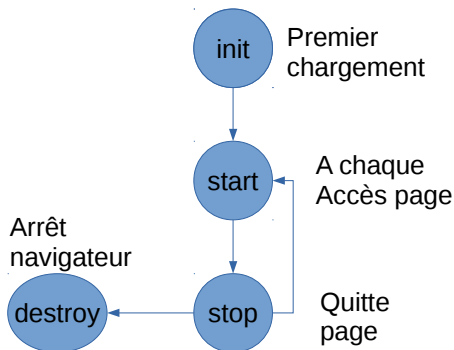
Pages webs dynamiques : les applets Java (2/3)

- Clause ARCHIVE et clause PARAM

```
<!DOCTYPE HTML>
<html>
  <head>
    <title> A Simple Program </title>
    <meta charset="UTF-8" / >
  </head>
  <body>
    Here is the output of my program:
    <applet code="HelloWorld.class"
            width="150" height="25">
    </applet>
  </body>
</html>
```

Pages webs dynamiques : les applets Java (3/3)

- Contraintes de sécurité : accès disque, accès réseau
- Sécurité de l'applet (ex : mot de passe dans le code)
- Le navigateur manipule également l'applet (`init`, `start`, `stop` et `destroy`)



Pages webs dynamiques : les programmes CGI (1/3)

- Les programmes C.G.I. : Common Gateway Interface

Pages webs dynamiques : les programmes CGI (1/3)

- Les programmes C.G.I. : Common Gateway Interface
- Exécution sur le serveur web

Pages webs dynamiques : les programmes CGI (1/3)

- Les programmes C.G.I. : Common Gateway Interface
- Exécution sur le serveur web
- Tout langage qui :

Pages webs dynamiques : les programmes CGI (1/3)

- Les programmes C.G.I. : Common Gateway Interface
- Exécution sur le serveur web
- Tout langage qui :
 - ▶ peut lire des variables d'environnement

Pages webs dynamiques : les programmes CGI (1/3)

- Les programmes C.G.I. : Common Gateway Interface
- Exécution sur le serveur web
- Tout langage qui :
 - ▶ peut lire des variables d'environnement
 - ▶ peut écrire sur la sortie standard

Pages webs dynamiques : les programmes CGI (1/3)

- Les programmes C.G.I. : Common Gateway Interface
- Exécution sur le serveur web
- Tout langage qui :
 - ▶ peut lire des variables d'environnement
 - ▶ peut écrire sur la sortie standard
- Pas de notion de session

Pages webs dynamiques : les programmes CGI (2/3)

- Un exemple :

```
<form action="http://sweethome/cgi-bin/prog.cgi" method="post">  
  Votre nom : <input type="text" name="nom" /><br />  
  Votre adresse Email : <input type="text" name="email" />  
  <br /> <input type="submit" value="valider" />  
  
</form>
```

Pages webs dynamiques : les programmes CGI (2/3)

- Un exemple :

```
<form action="http://sweethome/cgi-bin/prog.cgi" method="post">  
  Votre nom : <input type="text" name="nom" /><br />  
  Votre adresse Email : <input type="text" name="email" />  
  <br /> <input type="submit" value="valider" />  
  
</form>
```

- Envoi des données par get : données concaténées à l'URL, limitation taille des données, données visibles, utilisation possible sans formulaire

Pages webs dynamiques : les programmes CGI (2/3)

- Un exemple :

```
<form action="http://sweethome/cgi-bin/prog.cgi" method="post">  
  Votre nom : <input type="text" name="nom" /><br />  
  Votre adresse Email : <input type="text" name="email" />  
  <br /> <input type="submit" value="valider" />  
  
</form>
```

- Envoi des données par get : données concaténées à l'URL, limitation taille des données, données visibles, utilisation possible sans formulaire
- Envoi des données par post : données envoyées à part, chiffrement possible (option encrypt), pas de limitation de taille des données.

Pages webs dynamiques : les programmes CGI (3/3)

- Les actions du serveur web :

Pages webs dynamiques : les programmes CGI (3/3)

- Les actions du serveur web :
 - ▶ Stocke les données du formulaire dans des variables d'environnement (noms des variables standardisés)

Pages webs dynamiques : les programmes CGI (3/3)

- Les actions du serveur web :
 - ▶ Stocke les données du formulaire dans des variables d'environnement (noms des variables standardisés)
 - ▶ Lance le programme CGI correspondant

Pages webs dynamiques : les programmes CGI (3/3)

- Les actions du serveur web :

- ▶ Stocke les données du formulaire dans des variables d'environnement (noms des variables standardisés)
- ▶ Lance le programme CGI correspondant
- ▶ La sortie standard du programme est redirigée vers le navigateur client

Pages webs dynamiques : les programmes CGI (3/3)

- Les actions du serveur web :
 - ▶ Stocke les données du formulaire dans des variables d'environnement (noms des variables standardisés)
 - ▶ Lance le programme CGI correspondant
 - ▶ La sortie standard du programme est redirigée vers le navigateur client
- Programmes C, shell, Perl, ... sont compatibles

Pages webs dynamiques : Attention aux données fournies

- Exemple de fraude, programme CGI :

Pages webs dynamiques : Attention aux données fournies

- Exemple de fraude, programme CGI :
 - ▶ code CGI (script shell) : `mail $adresse < doc.txt`

Pages webs dynamiques : Attention aux données fournies

- Exemple de fraude, programme CGI :
 - ▶ code CGI (script shell) : `mail $adresse < doc.txt`
 - ▶ saisie champ email : `xx@bidon.fr ; mail badBoy@badLand < /etc/passwd;`

Pages webs dynamiques : Attention aux données fournies

- Exemple de fraude, programme CGI :
 - ▶ code CGI (script shell) : `mail $adresse < doc.txt`
 - ▶ saisie champ email : `xx@bidon.fr ; mail badBoy@badLand < /etc/passwd;`
- Exemple de fraude (vérification login/password), injection SQL :

Pages webs dynamiques : Attention aux données fournies

- Exemple de fraude, programme CGI :
 - ▶ code CGI (script shell) : `mail $adresse < doc.txt`
 - ▶ saisie champ email : `xx@bidon.fr ; mail badBoy@badLand < /etc/passwd;`
- Exemple de fraude (vérification login/password), injection SQL :
 - ▶ code SQL (script PHP) : `select 1 from user where login='$login' and password='$password'`

Pages webs dynamiques : Attention aux données fournies

- Exemple de fraude, programme CGI :
 - ▶ code CGI (script shell) : `mail $adresse < doc.txt`
 - ▶ saisie champ email : `xx@bidon.fr ; mail badBoy@badLand < /etc/passwd;`
- Exemple de fraude (vérification login/password), injection SQL :
 - ▶ code SQL (script PHP) : `select 1 from user where login='$login' and password='$password'`
 - ▶ saisie champ login : `' or '1'='1' --`

Pages webs dynamiques : Attention aux données fournies

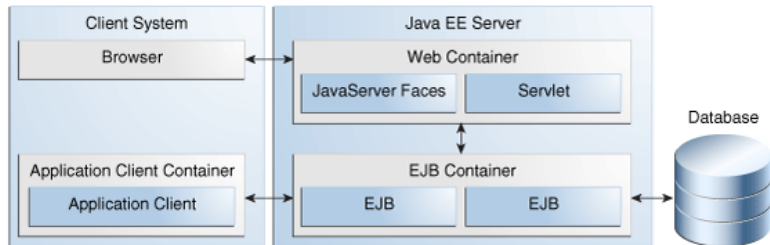
- Exemple de fraude, programme CGI :
 - ▶ code CGI (script shell) : `mail $adresse < doc.txt`
 - ▶ saisie champ email : `xx@bidon.fr ; mail badBoy@badLand < /etc/passwd;`
- Exemple de fraude (vérification login/password), injection SQL :
 - ▶ code SQL (script PHP) : `select 1 from user where login='$login' and password='$password'`
 - ▶ saisie champ login : `' or '1'='1' --`

En résumé

Toujours vérifier les données envoyées à la fois coté client (html et javascript) mais aussi coté serveur (PHP, EJB, ...)

La vision Java des architectures logicielles de type 3-tiers

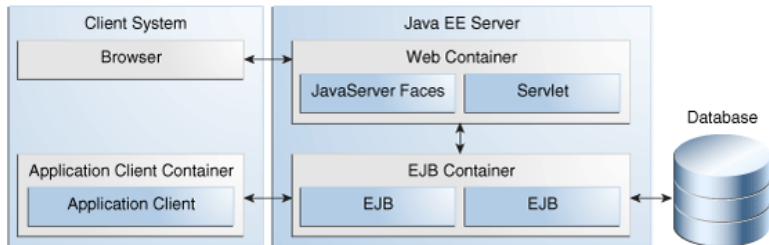
- Un conteneur web au sein du serveur JEE :



(source image : Oracle Javasoft)

La vision Java des architectures logicielles de type 3-tiers

- Un conteneur web au sein du serveur JEE :



(source image : Oracle Javasoft)

- Accepte des modules web (.war)

Cycle de vie des composants webs JEE

- 1 Développement du composant (html, css, images, java,...)

Cycle de vie des composants webs JEE

- ➊ Développement du composant (html, css, images, java,...)
- ➋ Archivage du composant dans un fichier normalisé (.war)

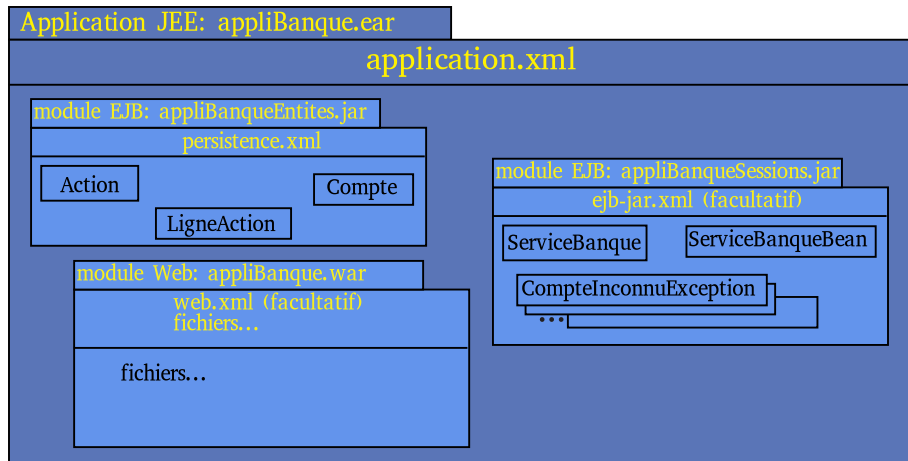
Cycle de vie des composants webs JEE

- ➊ Développement du composant (html, css, images, java,...)
- ➋ Archivage du composant dans un fichier normalisé (.war)
- ➌ Archivage dans une application JEE (.ear) avec informations de déploiement

Cycle de vie des composants webs JEE

- ➊ Développement du composant (html, css, images, java,...)
- ➋ Archivage du composant dans un fichier normalisé (.war)
- ➌ Archivage dans une application JEE (.ear) avec informations de déploiement
- ➍ Déploiement de l'application vers un serveur d'applications compatible JEE.

Exemple : application Banque avec module web



Code complet : <https://gitlab.univ-lille.fr/olivier.caron/demoCoursal.git>

Descripteur application.xml

Contenu archive :

```
appliBanque/  
META-INF/  
  application.xml  
appliBanqueSessions.jar  
appliBanqueEntites.jar  
appliBanque.war
```

Fichier application.xml :

```
<?xml version="1.0" encoding="UTF-8"?>  
<application xmlns="https://jakarta.ee/xml/ns/jakartaee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee/application_10.xsd"  
  version="10">  
  <display-name>Appli Banque</display-name>  
  <module><ejb>appliBanqueSessions.jar</ejb></module>  
  <module><ejb>appliBanqueEntites.jar</ejb></module>  
  <module>  
    <web>  
      <web-uri>appliBanque.war</web-uri>  
      <context-root>appBanque</context-root>  
    </web>  
  </module>  
</application>
```

Pages webs accessible à partir de :

<http://serveurJEE:8080/appBanque/...>

- Objectif : exploiter Java pour faire des applications webs

JEE et modules web

- Objectif : exploiter Java pour faire des applications webs
- Utiliser le ou les services (EJB sessions) pour obtenir et/ou modifier les données.

JEE et modules web

- Objectif : exploiter Java pour faire des applications webs
- Utiliser le ou les services (EJB sessions) pour obtenir et/ou modifier les données.
- Bénéficier du conteneur web où s'exécutent les programmes (transparent pour le développeur Java de composants webs) :

- Objectif : exploiter Java pour faire des applications webs
- Utiliser le ou les services (EJB sessions) pour obtenir et/ou modifier les données.
- Bénéficier du conteneur web où s'exécutent les programmes (transparent pour le développeur Java de composants webs) :
 - ▶ Le conteneur web intègre un contrôleur global chargé de recevoir les requêtes http et transmettre aux composants Java (routage web).

- Objectif : exploiter Java pour faire des applications webs
- Utiliser le ou les services (EJB sessions) pour obtenir et/ou modifier les données.
- Bénéficier du conteneur web où s'exécutent les programmes (transparent pour le développeur Java de composants webs) :
 - ▶ Le conteneur web intègre un contrôleur global chargé de recevoir les requêtes http et transmettre aux composants Java (routage web).
 - ▶ Le conteneur se charge de propager le résultat au client web

- Objectif : exploiter Java pour faire des applications webs
- Utiliser le ou les services (EJB sessions) pour obtenir et/ou modifier les données.
- Bénéficier du conteneur web où s'exécutent les programmes (transparent pour le développeur Java de composants webs) :
 - ▶ Le conteneur web intègre un contrôleur global chargé de recevoir les requêtes http et transmettre aux composants Java (routage web).
 - ▶ Le conteneur se charge de propager le résultat au client web
- Plusieurs technologies disponibles : servlet, jsp, jstl, ...

- Objectif : exploiter Java pour faire des applications webs
- Utiliser le ou les services (EJB sessions) pour obtenir et/ou modifier les données.
- Bénéficier du conteneur web où s'exécutent les programmes (transparent pour le développeur Java de composants webs) :
 - ▶ Le conteneur web intègre un contrôleur global chargé de recevoir les requêtes http et transmettre aux composants Java (routage web).
 - ▶ Le conteneur se charge de propager le résultat au client web
- Plusieurs technologies disponibles : servlet, jsp, jstl, ...
- Importance de la structuration du logiciel pour la qualité du logiciel résultant (lisibilité, efficacité, évolutivité, ...)

Une première technologie : les servlets (page 1/3)

Attention, ce code n'est pas la bonne solution technique ! ⚠

```
package web.controleurs;  
  
import java.io.IOException;  
import java.io.PrintWriter;  
  
import jakarta.ejb.EJB;  
import jakarta.servlet.ServletException;  
import jakarta.servlet.annotation.WebServlet;  
import jakarta.servlet.http.HttpServlet;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
  
import ejb.entites.Compte;  
import ejb.sessions.CompteInconnuException;  
import ejb.sessions.ServiceBanqueLocal;
```

Une première technologie : les servlets (page 2/3)

Ici, cette partie du code est correcte

```
@WebServlet(value={"badDisplayCompte"})
public class BadServlet extends HttpServlet {
    @EJB ServiceBanqueLocal service ;

    public BadServlet() {}

    public void doGet(HttpServletRequest req,
                      HttpServletResponse res)
        throws ServletException, IOException {
        int numeroCompte=
            Integer.parseInt(req.getParameter("numeroCompte")) ;
        Compte c=null ;
        try {
            c=service.getCompte(numeroCompte) ;
        } catch (CompteInconnuException e) { } // à éviter!!
    }
}
```

Une première technologie : les servlets (page 3/3)

Cette partie du code est un désastre! ⚠

```
res.setContentType("text/html");
PrintWriter out = res.getWriter();
out.println("<html><head><title>demo servlet"+
            "</title></head>");
out.println("<body>");
if (c==null)
    out.println("<h2>Compte "+numeroCompte+" inconnu</h2>")
    ;
else {
    out.println("<h2>Compte "+numeroCompte+"</h2>") ;
    out.println("<ul><li>Titulaire:<b>"+c.getTitulaire()+
                "</b></li>");
    out.println("<li>Solde:<b>"+c.getSolde()+
                "</b></li></ul>");
}
out.println("</body></html>");
}
```

Bilan : utilisation du servlet seul (1/2)

Requête http obtenue (c'est un exemple) :

<http://serveurJEE:8080/appBanque/badDisplayCompte?numeroCompte=2>

👍 Suite à la requête http, le contrôleur global du serveur d'applications :

Bilan : utilisation du servlet seul (1/2)

Requête http obtenue (c'est un exemple) :

<http://serveurJEE:8080/appBanque/badDisplayCompte?numeroCompte=2>

- 👍 Suite à la requête http, le contrôleur global du serveur d'applications :
- Sélectionne le servlet (grâce à l'annotation Java `@WebServlet` ligne 16),

Bilan : utilisation du servlet seul (1/2)

Requête http obtenue (c'est un exemple) :

<http://serveurJEE:8080/appBanque/badDisplayCompte?numeroCompte=2>

- 👍 Suite à la requête http, le contrôleur global du serveur d'applications :
- Sélectionne le servlet (grâce à l'annotation Java `@WebServlet` ligne 16),
 - Procède à l'instanciation,

Bilan : utilisation du servlet seul (1/2)

Requête http obtenue (c'est un exemple) :

<http://serveurJEE:8080/appBanque/badDisplayCompte?numeroCompte=2>

👍 Suite à la requête http, le contrôleur global du serveur d'applications :

- Sélectionne le servlet (grâce à l'annotation Java `@WebServlet` ligne 16),
- Procède à l'instanciation,
- Invoque la méthode `doGet` ou `doPost` en fournissant les paramètres :
 - `HttpServletRequest req` : descriptif de la requête (paramètres, informations sur l'origine de la requête : numéro IP, navigateur,...)
 - `HttpServletResponse res` : les données à envoyer au client

Bilan : utilisation du servlet seul (2/2)



le contrôleur global du serveur d'applications assure aussi :

- l'accès optimisé au(x) session(s) (annotation @EJB ligne 18)

La ligne de code suivante :

```
@EJB ServiceBanqueLocal service ;
```

permet l'exécution du code suivant assuré par le contrôleur global du serveur :

```
InitialContext ic = new InitialContext();  
Object obj = ic.lookup("java:app/appliBanqueSessions/  
    ServiceBanqueBean!ejb.sessions.ServiceBanqueLocal");  
service= (ServiceBanqueLocal) obj ;
```



Les inconvénients :

- Du code peu lisible (du code HTML dans du code Java)
- Couplage fort entre services et vues (réutilisation)

Une seconde technologie : les JSP (page 1/3)

Le code décrit ici est à proscrire aussi! ⚠

```
<%@ page language="html" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>demo JSP</title>
</head>

<%@ import="jakarta.ejb.*" %>
<%@ import="javax.naming.*" %>
<%@ import="ejb.sessions.ServiceBanqueLocal" %>
<%@ import="ejb.entites.Compte" %>
<%@ import="ejb.sessions.CompteInconnuException" %>
```

Une seconde technologie : les JSP (page 2/3)

```
<%!  
Compte c ;  
int numCompte ;  
%>  
<body>  
<%  
    numCompte=Integer.parseInt(request.getParameter("numeroCompte")) ;  
    Compte c=null ;  
    InitialContext ic = new InitialContext();  
    Object obj = ic.lookup("java:app/appliBanqueSessions/  
        ServiceBanqueBean!ejb.sessions.ServiceBanqueLocal");  
    ServiceBanqueLocal service= (ServiceBanqueLocal) obj ;
```

Une seconde technologie : les JSP (page 3/3)

```
try {  
    c=service .getCompte(numeroCompte) ;  
%>  
    <h2>Compte <%= numeroCompte%></h2>  
    <ul><li>Titulaire:<b><%= c.getTitulaire() %></b></li>  
        <li>Solde:<b><%= c.getSolde() %></b></li>  
    </ul>  
    } catch (CompteInconnuException e) { %>  
    <h2>Compte <%= numeroCompte %> inconnu</h2> ;  
    } // fin du try catch %>  
</body>  
</html>
```

Programmation JSP

- Notion de scriptlet (du code Java dans du code HTML)

Programmation JSP

- Notion de scriptlet (du code Java dans du code HTML)
- Les différentes sections de code JSP :

Programmation JSP

- Notion de scriptlet (du code Java dans du code HTML)
- Les différentes sections de code JSP :
 - ▶ `<%@ page import="..." %>` : importation de packages Java

Programmation JSP

- Notion de scriptlet (du code Java dans du code HTML)
- Les différentes sections de code JSP :
 - ▶ `<%@ page import="..." %>` : importation de packages Java
 - ▶ `<%! .... %>` : déclaration de variables et fonctions Java

Programmation JSP

- Notion de scriptlet (du code Java dans du code HTML)
- Les différentes sections de code JSP :
 - ▶ `<%@ page import="..." %>` : importation de packages Java
 - ▶ `<%! .... %>` : déclaration de variables et fonctions Java
 - ▶ `<% .... %>` : sections de code Java

Programmation JSP

- Notion de scriptlet (du code Java dans du code HTML)
- Les différentes sections de code JSP :
 - ▶ `<%@ page import="..." %>` : importation de packages Java
 - ▶ `<%! .... %>` : déclaration de variables et fonctions Java
 - ▶ `<% .... %>` : sections de code Java
 - ▶ `<%= expression %>` : affichage d'expressions Java

Programmation JSP

- Notion de scriptlet (du code Java dans du code HTML)
- Les différentes sections de code JSP :
 - ▶ `<%@ page import="..." %>` : importation de packages Java
 - ▶ `<%! .... %>` : déclaration de variables et fonctions Java
 - ▶ `<% .... %>` : sections de code Java
 - ▶ `<%= expression %>` : affichage d'expressions Java
- Le programmeur JSP dispose de variables prédéfinies (request, session,out,...)

Programmation JSP

- Notion de scriptlet (du code Java dans du code HTML)
- Les différentes sections de code JSP :
 - ▶ `<%@ page import="..." %>` : importation de packages Java
 - ▶ `<%! .... %>` : déclaration de variables et fonctions Java
 - ▶ `<% .... %>` : sections de code Java
 - ▶ `<%= expression %>` : affichage d'expressions Java
- Le programmeur JSP dispose de variables prédéfinies (request, session, out, ...)
- Lors de la **première** demande d'exécution d'un JSP, transformation du JSP en servlet puis compilation
 - ➔ **Servlet et JSP ne font donc qu'un !**

Bilan : utilisation du JSP seul

Requête http obtenue (c'est un exemple) :

<http://serveurJEE:8080/appBanque/badDisplayCompte2.jsp?numeroCompte=2>

 Points positifs

Bilan : utilisation du JSP seul

Requête http obtenue (c'est un exemple) :

<http://serveurJEE:8080/appBanque/badDisplayCompte2.jsp?numeroCompte=2>



Points positifs

- On améliore (un peu) la lisibilité du code (notamment du point de vue html)

Bilan : utilisation du JSP seul

Requête http obtenue (c'est un exemple) :

<http://serveurJEE:8080/appBanque/badDisplayCompte2.jsp?numeroCompte=2>



Points positifs

- On améliore (un peu) la lisibilité du code (notamment du point de vue html)



Points négatifs :

Bilan : utilisation du JSP seul

Requête http obtenue (c'est un exemple) :

<http://serveurJEE:8080/appBanque/badDisplayCompte2.jsp?numeroCompte=2>



Points positifs

- On améliore (un peu) la lisibilité du code (notamment du point de vue html)



Points négatifs :

- Couplage fort entre services et vues : on ne peut pas réutiliser la vue pour d'autres sessions.

Bilan : utilisation du JSP seul

Requête http obtenue (c'est un exemple) :

<http://serveurJEE:8080/appBanque/badDisplayCompte2.jsp?numeroCompte=2>



Points positifs

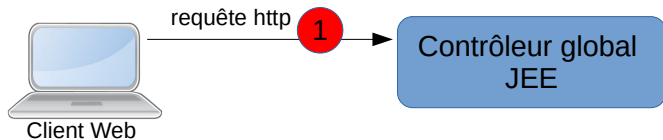
- On améliore (un peu) la lisibilité du code (notamment du point de vue html)



Points négatifs :

- Couplage fort entre services et vues : on ne peut pas réutiliser la vue pour d'autres sessions.
- Code reste difficile à lire : Java et HTML

La bonne architecture : le patron MVC (Modèle-Vue-Contrôleur)

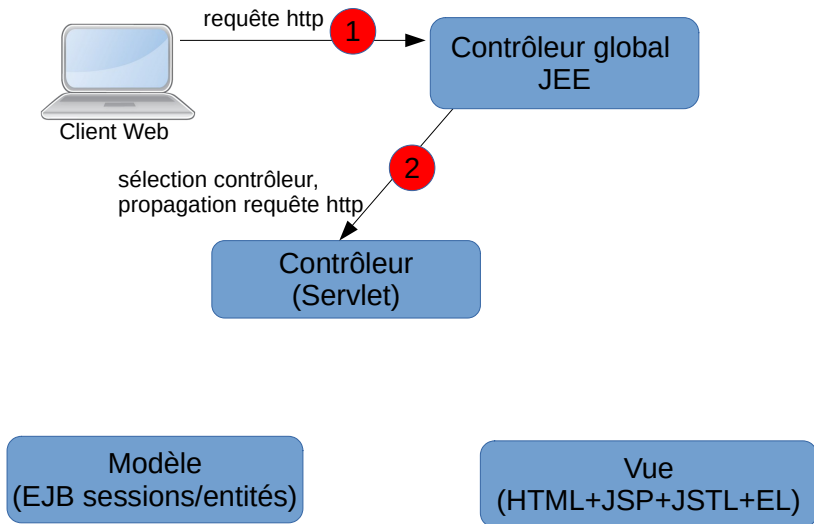


Contrôleur
(Servlet)

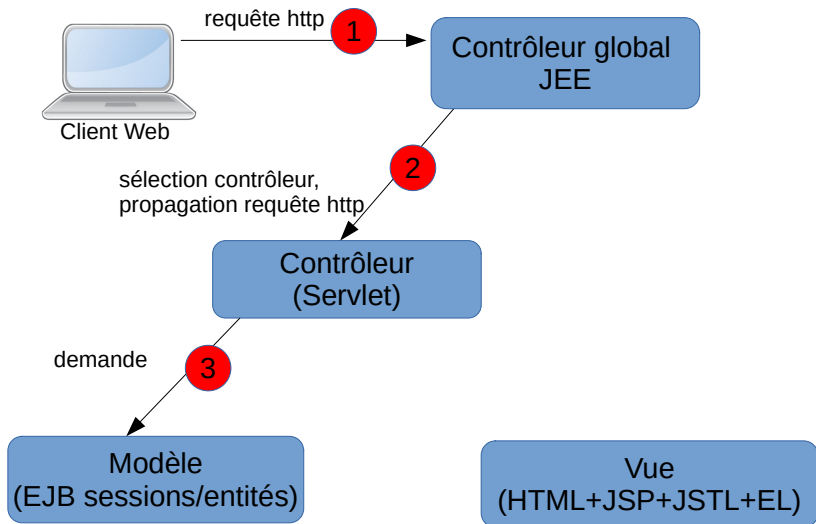
Modèle
(EJB sessions/entités)

Vue
(HTML+JSP+JSTL+EL)

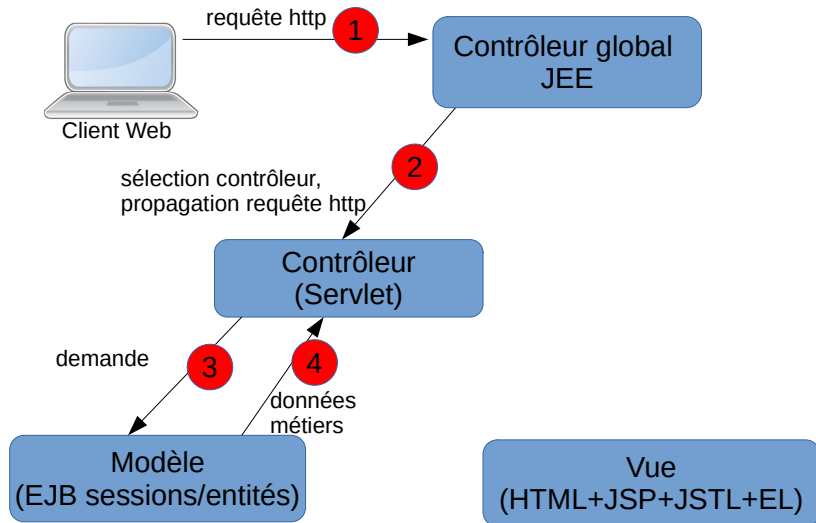
La bonne architecture : le patron MVC (Modèle-Vue-Contrôleur)



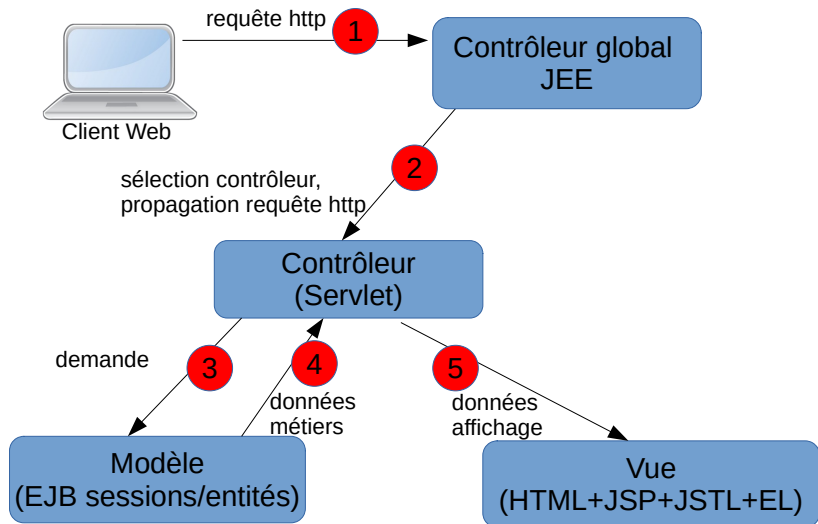
La bonne architecture : le patron MVC (Modèle-Vue-Contrôleur)



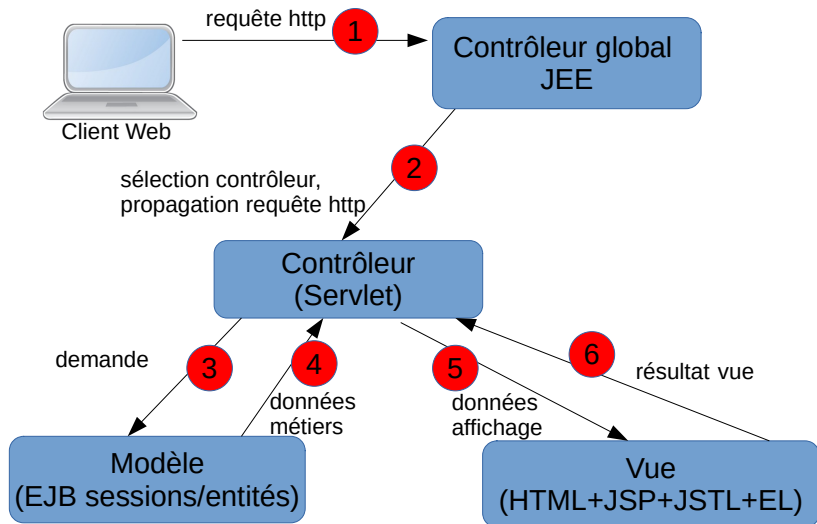
La bonne architecture : le patron MVC (Modèle-Vue-Contrôleur)



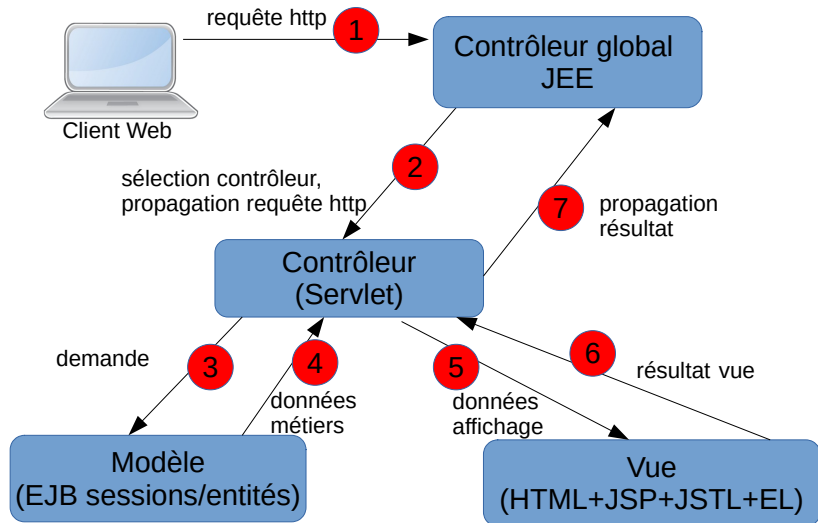
La bonne architecture : le patron MVC (Modèle-Vue-Contrôleur)



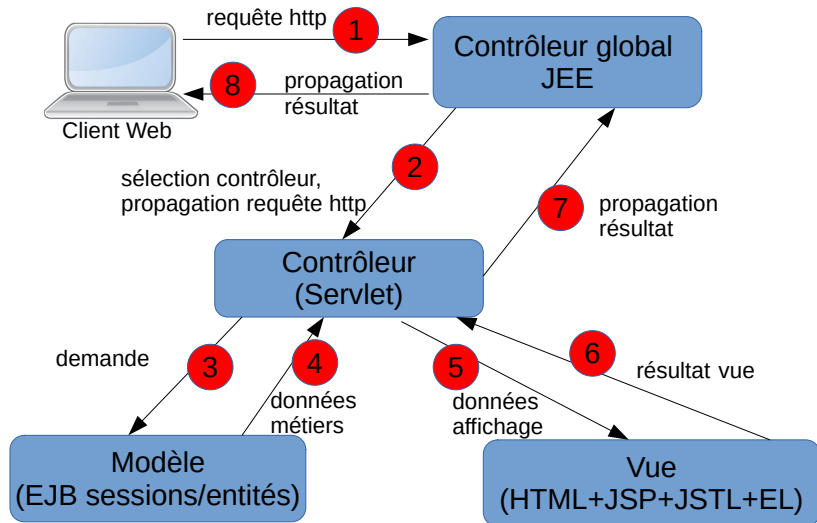
La bonne architecture : le patron MVC (Modèle-Vue-Contrôleur)



La bonne architecture : le patron MVC (Modèle-Vue-Contrôleur)



La bonne architecture : le patron MVC (Modèle-Vue-Contrôleur)



Application banque : exemple d'une partie web MVC

- 2 requêtes http possibles :

Application banque : exemple d'une partie web MVC

- 2 requêtes http possibles :
 - ▶ URL : `http://.../appBanque/rechCompte` : formulaire HTML de recherche d'un compte, la vue associée sera le fichier `rechCompte.html`

Application banque : exemple d'une partie web MVC

- 2 requêtes http possibles :
 - ▶ URL : `http://.../appBanque/rechCompte` : formulaire HTML de recherche d'un compte, la vue associée sera le fichier `rechCompte.html`
 - ▶ URL : `http://.../appBanque/afficherCompte` : permet d'afficher le descriptif d'un compte, la vue associée sera le fichier `afficherCompte.jsp`

Application banque : exemple d'une partie web MVC

- 2 requêtes http possibles :
 - ▶ URL : `http://.../appBanque/rechCompte` : formulaire HTML de recherche d'un compte, la vue associée sera le fichier `rechCompte.html`
 - ▶ URL : `http://.../appBanque/afficherCompte` : permet d'afficher le descriptif d'un compte, la vue associée sera le fichier `afficherCompte.jsp`
- Le servlet qui joue le rôle de contrôleur sera le fichier `Controleur.java`

Que fait le contrôleur de l'architecture MVC ?

- Le contrôleur est invoqué pour différentes actions référencés par des adresses web.

Que fait le contrôleur de l'architecture MVC ?

- Le contrôleur est invoqué pour différentes actions référencés par des adresses web.
- L'annotation `@WebServlet` énumère les dernières parties de l'adresse web.

`http://serveurJEE:8080/appliBanque/rechCompte`

première partie liée au serveur, 2nde partie liée au context-root

Que fait le contrôleur de l'architecture MVC ?

- Le contrôleur est invoqué pour différentes actions référencés par des adresses web.
- L'annotation `@WebServlet` énumère les dernières parties de l'adresse web.
`http://serveurJEE:8080/appliBanque/rechCompte`
première partie liée au serveur, 2nde partie liée au context-root
- Pour chaque adresse, le contrôleur :

Que fait le contrôleur de l'architecture MVC ?

- Le contrôleur est invoqué pour différentes actions référencés par des adresses web.
- L'annotation `@WebServlet` énumère les dernières parties de l'adresse web.
`http://serveurJEE:8080/appliBanque/rechCompte`
première partie liée au serveur, 2nde partie liée au context-root
- Pour chaque adresse, le contrôleur :
 - ❶ récupère les données (exemple : données d'un formulaire)

Que fait le contrôleur de l'architecture MVC ?

- Le contrôleur est invoqué pour différentes actions référencés par des adresses web.
- L'annotation `@WebServlet` énumère les dernières parties de l'adresse web.

`http://serveurJEE:8080/appliBanque/rechCompte`
première partie liée au serveur, 2nde partie liée au context-root

- Pour chaque adresse, le contrôleur :
 - 1 récupère les données (exemple : données d'un formulaire)
 - 2 invoque un service pour effectuer un traitement

Que fait le contrôleur de l'architecture MVC ?

- Le contrôleur est invoqué pour différentes actions référencés par des adresses web.
- L'annotation `@WebServlet` énumère les dernières parties de l'adresse web.

`http://serveurJEE:8080/appliBanque/rechCompte`
première partie liée au serveur, 2nde partie liée au context-root

- Pour chaque adresse, le contrôleur :
 - ➊ récupère les données (exemple : données d'un formulaire)
 - ➋ invoque un service pour effectuer un traitement
 - ➌ prépare les données de présentation pour la vue

Que fait le contrôleur de l'architecture MVC ?

- Le contrôleur est invoqué pour différentes actions référencés par des adresses web.
- L'annotation `@WebServlet` énumère les dernières parties de l'adresse web.

`http://serveurJEE:8080/appliBanque/rechCompte`
première partie liée au serveur, 2nde partie liée au context-root

- Pour chaque adresse, le contrôleur :
 - ➊ récupère les données (exemple : données d'un formulaire)
 - ➋ invoque un service pour effectuer un traitement
 - ➌ prépare les données de présentation pour la vue
 - ➍ sélectionne et invoque la vue associée

Que fait le contrôleur de l'architecture MVC ?

- Le contrôleur est invoqué pour différentes actions référencés par des adresses web.
- L'annotation `@WebServlet` énumère les dernières parties de l'adresse web.

`http://serveurJEE:8080/appliBanque/rechCompte`
première partie liée au serveur, 2nde partie liée au context-root

- Pour chaque adresse, le contrôleur :
 - ① récupère les données (exemple : données d'un formulaire)
 - ② invoque un service pour effectuer un traitement
 - ③ prépare les données de présentation pour la vue
 - ④ sélectionne et invoque la vue associée
- Selon les cas, les points 1 à 3 sont inutiles.

Le contrôleur MVC de l'application banque (page 1/3)

Enfin !, le code qu'il faut retenir

```
package web.controleurs;

import java.io.IOException;
import ejb.sessions.* ;
import ejb.entites.* ;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.*;

@WebServlet(value={"rechCompte","afficherCompte"})
public class Controleur extends HttpServlet {
    private static final long serialVersionUID = 1L;
    @jakarta.ejb.EJB private ServiceBanqueLocal service ;

    public Controleur() {}
}
```

Le contrôleur MVC de l'application banque (page 2/3)

```
protected void doGet(HttpServletRequest request ,
    HttpServletResponse response)
throws ServletException , IOException {
    String url = request.getRequestURL().toString();
    String maVue = "/rechCompte.html"; // vue par défaut
    if (url.endsWith("/rechCompte")) {
        // sélection de la vue :
        maVue = "/rechCompte.html"; // rien d'autre à faire
    } else if (url.endsWith("/afficherCompte")) {
        // sélection de la vue :
        maVue = "/afficherCompte.jsp";
        // récupération des données :
        String str=request.getParameter("numeroCompte");
        int numeroCompte=Integer.parseInt(str) ;
```

Le contrôleur MVC de l'application banque (page 3/3)

```
Compte compte=null ;
try { // traitement métier :
    compte= service.getCompte(numeroCompte);
    java.util.Collection<LigneAction> l_la = service.
        getActionsAchetees(numeroCompte);
    // préparation des données :
    request.setAttribute("listeLA", l_la);
} catch (CompteInconnuException e) { }
// préparation des données :
request.setAttribute("compte", compte) ;
}
RequestDispatcher dispatcher = getServletContext().
    getRequestDispatcher(maVue);
dispatcher.forward(request, response);
}
```

Résumé de l'activité du contrôleur

	Actions gérées (ligne 11)	
	rechCompte	afficherCompte
1. Récupération des données	—	obtention numeroCompte lignes 28-29
2. service métier	—	appel à getCompte lignes 30-36
3. préparation des données	—	le compte ("compte") et les lignes d'actions ("listeLA") lignes 35 et 38
4. appel à la vue associée	rechCompte.html lignes 23 ;40-41	afficherCompte.jsp lignes 26 ;40-41

Vue n°1 : rechCompte

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Application Banque</title> <meta charset="UTF-8" />
</head>
<body>
  <header><h1>Application banque</h1></header>
  <h2>Rechercher un compte</h2>
  <form action="afficherCompte">
    Numéro du compte : <input type="text" name="numeroCompte"
                           required placeholder="numéro du
                           compte..." />
    <input type="submit" value="Rechercher" >
  </form>
</body>
</html>
```

Vue n°2 : afficherCompte (1/2)

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
%>
<html>
  <head>
    <title>Application banque</title><meta charset="UTF-8" />
  </head>
  <body>
    <header><h1>Application banque</h1></header>
    <h2>Information compte:</h2>
    <c:if test = "${not empty requestScope.compte}">
      <h3>Compte no: ${requestScope.compte.numero}</h3>
      Titulaire : <b>${requestScope.compte.titulaire}</b><br />
      Solde <b>${requestScope.compte.solde}</b>
    </c:if>
  </body>
</html>
```

Vue n°2 : afficherCompte (2/2)

```
<ul>
  <c:forEach items="{requestScope.listeLA}" var="la">
    <li>${la.nombre} action(s) ${la.action.nom}
      au taux de ${la.action.taux}
    </li>
  </c:forEach>
</ul>
</c:if>
<c:if test = "{empty requestScope.compte}">
  <h3>Compte Inconnu</h3>
</c:if>
</body>
</html>
```

Bilan architecture MVC

 Points positifs

Bilan architecture MVC

Points positifs

- Facilité d'ajout de nouvelles fonctionnalités, soit par ajout d'un "test if" dans le contrôleur, soit par ajout d'autre(s) contrôleurs.

Bilan architecture MVC

Points positifs

- Facilité d'ajout de nouvelles fonctionnalités, soit par ajout d'un "test if" dans le contrôleur, soit par ajout d'autre(s) contrôleurs.
- Code des vues plus facile à lire : uniquement des tags.

Bilan architecture MVC

Points positifs

- Facilité d'ajout de nouvelles fonctionnalités, soit par ajout d'un "test if" dans le contrôleur, soit par ajout d'autre(s) contrôleurs.
- Code des vues plus facile à lire : uniquement des tags.
- Indépendance entre vues et appel aux sessions : les vues sont plus facilement réutilisables pour d'autres applications

Bilan architecture MVC

Points positifs

- Facilité d'ajout de nouvelles fonctionnalités, soit par ajout d'un "test if" dans le contrôleur, soit par ajout d'autre(s) contrôleurs.
- Code des vues plus facile à lire : uniquement des tags.
- Indépendance entre vues et appel aux sessions : les vues sont plus facilement réutilisables pour d'autres applications

Point négatif :

Bilan architecture MVC

Points positifs

- Facilité d'ajout de nouvelles fonctionnalités, soit par ajout d'un "test if" dans le contrôleur, soit par ajout d'autre(s) contrôleurs.
- Code des vues plus facile à lire : uniquement des tags.
- Indépendance entre vues et appel aux sessions : les vues sont plus facilement réutilisables pour d'autres applications

Point négatif :

- Beaucoup de fonctionnalités impliquent beaucoup de if au niveau du contrôleur

Solution 1 : un contrôleur par vue ou groupe de vues

Solution 2 : des frameworks compatibles MVC viennent simplifier l'écriture des contrôleurs (hors programme IS^{ia}4).

Explications de la vue n°2 : afficherCompte

- Technologies employées : JSP, EL et JSTL

Explications de la vue n°2 : afficherCompte

- Technologies employées : JSP, EL et JSTL
- EL et JSTL introduisent des expressions combinées avec des balises qui viennent se substituer au code Java des JSP ➔ **lisibilité**

Explications de la vue n°2 : afficherCompte

- Technologies employées : JSP, EL et JSTL
- EL et JSTL introduisent des expressions combinées avec des balises qui viennent se substituer au code Java des JSP ➔ **lisibilité**
- Plus aucune référence au(x) composant(s) sessions utilisés ➔ **ré-utilisabilité**

Technologie EL (Expression Language)

- Au sein de JSP, les expressions EL permettent :

Technologie EL (Expression Language)

- Au sein de JSP, les expressions EL permettent :
 - ▶ de manipuler des objets et attributs d'objets dans une page JSP

Technologie EL (Expression Language)

- Au sein de JSP, les expressions EL permettent :
 - ▶ de manipuler des objets et attributs d'objets dans une page JSP
 - ▶ d'établir des tests simples

Technologie EL (Expression Language)

- Au sein de JSP, les expressions EL permettent :
 - ▶ de manipuler des objets et attributs d'objets dans une page JSP
 - ▶ d'établir des tests simples
- Syntaxe : `${expression}`
: ce qui se trouve à l'intérieur des accolades est interprété.

Technologie EL : opérateurs

- Opérateurs arithmétiques : +, - , *, / et %
- Opérateurs logiques : && , || , !
- Opérateurs relationnels : == ou eq, != ou ne, < ou lt, > ou gt, <= ou le, >= ou ge
- Opérateur empty : test si null ou vide : \$empty var
remarque : objet null de type String est considéré comme vide
- Opérateur de condition ternaire test ? partie_si_vraie :
partie_si_faux

Exemples

```
${ age < 18 } <!-- affiche true ou false -->
${ empty prenom ? "prénom inconnu" : prenom }
dites ${ 32 + 1 } <!-- affiche 33 -->
```

Technologie EL : variables (1/2)

- Permet d'accéder à des objets et/ou à des propriétés Java Bean :
pers1.nom est équivalent au code Java pers1.getNom()
- Pas de NullPointerException ! (n'affiche rien)
- Les données fournies par le contrôleur sont accessibles via la variable requestScope
Ex : request.setAttribute("v1", ...) ➔ requestScope.v1
- Gère les collections (opérateur get(i) ou [i])

Exemples

```
${compte.solde} <!-- affiche la propriété solde de l'objet compte  
-->  
${listePersonne.get(3)} <!-- affiche le 4ième élément de la liste--  
>  
${listePersonne[0]} <!-- affiche le 1er élément -->
```

Technologie EL : variables (2/2)

- Quelques variables prédéfinies :

Technologie EL : variables (2/2)

- Quelques variables prédéfinies :

`param` paramètres sous forme de String

Exemple : `${param["numeroCompte"]}`

Technologie EL : variables (2/2)

- Quelques variables prédéfinies :

`param` paramètres sous forme de String

Exemple : `${param["numeroCompte"]}`

`requestScope` collection des variables de la requête

Technologie EL : variables (2/2)

- Quelques variables prédéfinies :

`param` paramètres sous forme de String

Exemple : `${param["numeroCompte"]}`

`requestScope` collection des variables de la requête

`sessionScope` collection des variables de la session

Technologie EL : variables (2/2)

- Quelques variables prédéfinies :

`param` paramètres sous forme de String

Exemple : `${param["numeroCompte"]}`

`requestScope` collection des variables de la requête

`sessionScope` collection des variables de la session

`cookie` collection de tous les cookies

Technologie EL : variables (2/2)

- Quelques variables prédéfinies :

`param` paramètres sous forme de String

Exemple : `${param["numeroCompte"]}`

`requestScope` collection des variables de la requête

`sessionScope` collection des variables de la session

`cookie` collection de tous les cookies

...

Technologie JSTL

Definition

JSP Standard Tag Library définit des balises qui se substituent au code Java JSP.

Utilisation des balises JSTL de base

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
    %>
<!DOCTYPE html>
<html>
    ...
```

JSTL core : affichage des données

- Soit par `${expression}` soit par balise `c:out` :

balise <code>c:out</code>		
Attribut	Signification	Par défaut
<code>value</code>	la valeur à afficher	
<code>default</code>	une valeur par défaut si l'expression évaluée dans <code>value</code> est vide	
<code>escapeXml</code>	désactive le code HTML	true

Exemples (ici, `var` contient "`<b>essai</b>`")

<code><c:out value="test" /></code>	affiche "test"
<code><c:out value="\${var}" escapeXml="false"/></code>	affiche "essai" en gras
<code><c:out value="\${var}" /></code> ou <code>\${var}</code>	affiche " <code>essai</code> "
<code><c:out value="\${v2}" default="rien" /></code>	affiche <code>v2</code> , ou "rien" si <code>v2</code> vide ou null

JSTL : portée des variables

- Portée d'une donnée : endroit(s) où la variable est accessible.
- On distingue les 4 portées (scope) suivantes :
 - page** la variable n'est accessible que dans la page courante
 - request** la variable est disponible pour les pages qui traitent de la même requête http.
 - session** la variable est disponible durant toute la session avec un client web.
 - application** la variable est accessible sur toutes les pages de l'application.

Exemples (dans un contrôleur (Servlet))

```
request.setAttribute("v1","test") ; // variable de portée  
    request  
request.getSession().setAttribute("v2","test 2") ; //  
    variable de portée session  
Object caddy=request.getSession().getAttribute("caddy25") ;
```

JSTL core : définition des variables

balise <code>c:set</code>		
Attribut	Signification	Par défaut
<code>var</code>	le nom de la variable	
<code>value</code>	le contenu de la variable	
<code>scope</code>	portée de la variable	<code>page</code>

- Ordre de recherche : `page` puis `request`, puis `session` puis `application`
- Objets implicite : `<portée>Scope` pour délimiter la recherche dans un scope donné

Exemples

```
<c:set var="v1" value="3" /> variable v1 de scope page
<c:set var "v2" value="{3*v1}" scope="session" /> variable session
v2
${v2} // on recherche une variable v2 dans page, puis request puis
session
${requestScope.v3} // on recherche une variable v3 dans espace
request
```

JSTL core : définition/modification des propriétés Java Bean

balise c:set

Attribut	Signification	Par défaut
target	l'objet concerné	
property	le nom de la propriété à modifier	
value	la nouvelle valeur	

Exemples

```
<c:set target="${compte}" property="solde" value="2000.0" />
<c:set target="${compte}" property="titulaire" value="${null}" />
```

JSTL core : suppression de variables

balise <code>c:remove</code>		
Attribut	Signification	Par défaut
<code>var</code>	nom de la variable	
<code>scope</code>	portée de la variable	page

Exemples

```
<c:remove var="v1" />  
<c:remove var="v2" scope="session" />
```

JSTL core : structure de contrôle if

balise <code>c:if</code>		
Attribut	Signification	Par défaut
test	expression du test	
var	optionnel : nom de la variable booléenne contenant le résultat du test	
scope	optionnel : portée de la variable	page

- le `else` n'existe pas !

Exemples

```
<c:if test="${empty compte}" var="pasDeCompte">
  compte inconnu
</c:if>
<c:if test="${age < 18}">mineur</c:if>
```

JSTL core : structure de contrôle "switch like"

balise `c:choose`

sous-balise `c:when`

Attribut	Signification	Par défaut
----------	---------------	------------

test	test du switch	
------	----------------	--

sous-balise `c:otherwise`

- Les conditions sont exclusives

Exemples

```
<c:choose>
  <c:when test="${systeme=='windows'}">Windows</c:when>
  <c:when test="${systeme=='macos'}">Mac OS X</c:when>
  <c:otherwise>Linux</c:otherwise>
</c:choose>
```

JSTL core : structure de contrôle "for à la C"

balise c:forEach		
Attribut	Signification	Par défaut
begin	valeur indice départ	0
end	valeur indice fin	
var	nom de variable contenant l'indice	
step	pas d'incrément	

- équivalent C de for (var=begin; var<end; var+=step)

Exemples

Les nombres pairs entre 1 et 50 :

```
<c:forEach begin="2" end="50" step="2" var="i">
  ${i}
</c:forEach>
```

JSTL core : structure de contrôle "foreach"

balise c:forEach

Attribut	Signification	Par défaut
items	la collection à itérer	
var	nom de variable contenant l'objet courant	
varStatus	nom de variable contenant des informations sur l'exécution de l'itération (index, count, first, last)	

Exemples

```
<c:forEach items="${comptes}" var="compte" varStatus="status">
  <c:if test="${status.first}"><ul></c:if>
  <li>${compte.titulaire} : ${compte.solde}</li>
  <c:if test="${status.last}"></ul></c:if>
</c:forEach>
```

JSTL core : gestion des URL

balise c:url		
Attribut	Signification	Par défaut
value	l'adresse URL	
sous-balise c:param		
name	nom du paramètre	
value	valeur du paramètre	

- Intérêt 1 : encodage de la chaîne
- Intérêt 2 : ajout du contexte à l'URL (`context-root`)

Exemples

```
<a href="<c:url value="rechCompte" />">rechercher un compte</a>
<a href="<c:url value="/afficherCompte">
  <c:param name="numeroCompte" value="1" />
</c:url>">afficher compte 1 </a>
```

JSTL core : importation de fragment de page

balise <code>c:import</code>		
Attribut	Signification	Par défaut
<code>url</code>	référence de la page à importer	

- Intérêt : ré-utilisation de partie de pages.
- Utilisation possible de sous-balise `c:param`

Exemples

```
<c:import url="header.html" />
```

JSTL core : redirection de page

balise c:redirect

Attribut	Signification	Par défaut
url	l'adresse de redirection	

- Utilisation possible de sous-balise c:param

Exemples

```
<c:if test="${empty login}">  
  <c:redirect url="Connexion" />  
</c:if>
```

JSTL, c'est aussi

- Des bibliothèques supplémentaires :

JSTL, c'est aussi

- Des bibliothèques supplémentaires :
 xml manipulation de données xml

JSTL, c'est aussi

- Des bibliothèques supplémentaires :
 - `xml` manipulation de données xml
 - `sql` interrogation de bases de données (pas très MVC...)

JSTL, c'est aussi

- Des bibliothèques supplémentaires :

`xml` manipulation de données xml

`sql` interrogation de bases de données (pas très MVC...)

`fn` traitement de chaînes de caractères

JSTL, c'est aussi

- Des bibliothèques supplémentaires :

- `xml` manipulation de données xml

- `sql` interrogation de bases de données (pas très MVC...)

- `fn` traitement de chaînes de caractères

- `fmt` formatage des données (ex : dates), internationalisation

JSTL, c'est aussi

- Des bibliothèques supplémentaires :

- `xml` manipulation de données xml

- `sql` interrogation de bases de données (pas très MVC...)

- `fn` traitement de chaînes de caractères

- `fmt` formatage des données (ex : dates), internationalisation

Memento JSTL

<http://homepage.divms.uiowa.edu/~slonnegr/wpj/jqr.pdf>

Exemple : Structure d'un composant web

```
1 web/  
2 WEB-INF/  
3   web.xml  
4   lib/  
5     xxx.jar ...  
6   classes/  
7     web/  
8       controleurs/  
9         BadServlet.  
10          class  
11          Controleur.  
12           class  
13 badDisplayCompte2.jsp  
14 rechCompte.html  
15 afficherCompte.jsp  
16 ...
```

```
jar cvf appliBanque.war web/*
```

- Ligne 3 : fichier descriptif
WEB-INF/web.xml facultatif
- Lignes 4-5 : les éventuelles
bibliothèques
- Lignes 6-10 : les contrôleurs
- Lignes 11-* : les fichiers web
(html, jsp, css, ...)

Fonctionnement serveur JEE (1/3)

- Phase de déploiement (deploy) :

Fonctionnement serveur JEE (1/3)

- Phase de déploiement (deploy) :
 - ① Réception fichier application (.ear)

Fonctionnement serveur JEE (1/3)

- Phase de déploiement (deploy) :
 - ➊ Réception fichier application (.ear)
 - ➋ Désarchivage, analyse fichier XML META-INF/application.xml

Fonctionnement serveur JEE (1/3)

- Phase de déploiement (deploy) :
 - ➊ Réception fichier application (.ear)
 - ➋ Désarchivage, analyse fichier XML META-INF/application.xml
 - ➌ Pour chaque module, identification du type de module (ejb, web)

Fonctionnement serveur JEE (1/3)

- Phase de déploiement (deploy) :
 - ➊ Réception fichier application (.ear)
 - ➋ Désarchivage, analyse fichier XML META-INF/application.xml
 - ➌ Pour chaque module, identification du type de module (ejb, web)
 - ➍ Pour chaque module de type web :

Fonctionnement serveur JEE (1/3)

- Phase de déploiement (deploy) :
 - ➊ Réception fichier application (.ear)
 - ➋ Désarchivage, analyse fichier XML META-INF/application.xml
 - ➌ Pour chaque module, identification du type de module (ejb, web)
 - ➍ Pour chaque module de type web :
 - ➊ Création du répertoire défini dans context-root
Pour notre exemple : context-root="appBanque"

Fonctionnement serveur JEE (1/3)

- Phase de déploiement (deploy) :
 - ➊ Réception fichier application (.ear)
 - ➋ Désarchivage, analyse fichier XML META-INF/application.xml
 - ➌ Pour chaque module, identification du type de module (ejb, web)
 - ➍ Pour chaque module de type web :
 - ➊ Création du répertoire défini dans context-root
Pour notre exemple : context-root="appBanque"
 - ➋ Désarchivage du module dans cet espace

Fonctionnement serveur JEE (1/3)

- Phase de déploiement (deploy) :
 - ➊ Réception fichier application (.ear)
 - ➋ Désarchivage, analyse fichier XML META-INF/application.xml
 - ➌ Pour chaque module, identification du type de module (ejb, web)
 - ➍ Pour chaque module de type web :
 - ➊ Création du répertoire défini dans context-root
Pour notre exemple : context-root="appBanque"
 - ➋ Désarchivage du module dans cet espace
 - ➌ Analyse fichier WEB-INF/web.xml **ou** introspection des contrôleurs dans WEB-INF/classes/ (lecture des annotations @WebServlet)

Fonctionnement serveur JEE (1/3)

- Phase de déploiement (deploy) :
 - ① Réception fichier application (.ear)
 - ② Désarchivage, analyse fichier XML META-INF/application.xml
 - ③ Pour chaque module, identification du type de module (ejb, web)
 - ④ Pour chaque module de type web :
 - ① Création du répertoire défini dans context-root
Pour notre exemple : context-root="appBanque"
 - ② Désarchivage du module dans cet espace
 - ③ Analyse fichier WEB-INF/web.xml **ou** introspection des contrôleurs dans WEB-INF/classes/ (lecture des annotations @WebServlet)
 - ④ Enrichissement d'une table d'associations : nom symbolique - localisation programme

Fonctionnement serveur JEE (2/3)

- Phase d'exécution page web (run) :

Fonctionnement serveur JEE (2/3)

- Phase d'exécution page web (run) :
 - ① GET `http://nomServeur:8080/cheminContextRoot/Action`

Fonctionnement serveur JEE (2/3)

- Phase d'exécution page web (run) :
 - 1 GET `http://nomServeur:8080/cheminContextRoot/Action`
 - 2 Recherche du contrôleur associé au nom symbolique

Fonctionnement serveur JEE (2/3)

- Phase d'exécution page web (run) :
 - ① GET `http://nomServeur:8080/cheminContextRoot/Action`
 - ② Recherche du contrôleur associé au nom symbolique
 - ③ 1ère demande : compilation de la vue associée (jsp) :

Fonctionnement serveur JEE (2/3)

- Phase d'exécution page web (run) :
 - ① GET `http://nomServeur:8080/cheminContextRoot/Action`
 - ② Recherche du contrôleur associé au nom symbolique
 - ③ 1ère demande : compilation de la vue associée (jsp) :
 - ★ Pas d'erreurs : préservation du fichier `.class` et exécution

Fonctionnement serveur JEE (2/3)

- Phase d'exécution page web (run) :
 - ① GET `http://nomServeur:8080/cheminContextRoot/Action`
 - ② Recherche du contrôleur associé au nom symbolique
 - ③ 1ère demande : compilation de la vue associée (jsp) :
 - ★ Pas d'erreurs : préservation du fichier `.class` et exécution
 - ★ Erreur : Affichage des erreurs de compilation via web

Fonctionnement serveur JEE (2/3)

- Phase d'exécution page web (run) :
 - ① GET `http://nomServeur:8080/cheminContextRoot/Action`
 - ② Recherche du contrôleur associé au nom symbolique
 - ③ 1ère demande : compilation de la vue associée (jsp) :
 - ★ Pas d'erreurs : préservation du fichier `.class` et exécution
 - ★ Erreur : Affichage des erreurs de compilation via web
 - ④ ième demande : exécution du fichier `.class`

Fonctionnement serveur JEE (3/3)

- Phase de retrait (undeploy) :

Fonctionnement serveur JEE (3/3)

- Phase de retrait (undeploy) :
 - 1 Analyse fichier XML META-INF/application.xml

Fonctionnement serveur JEE (3/3)

- Phase de retrait (undeploy) :
 - ➊ Analyse fichier XML META-INF/application.xml
 - ➋ Pour chaque module (ejb ou web), suppression des espaces créés et des programmes compilés

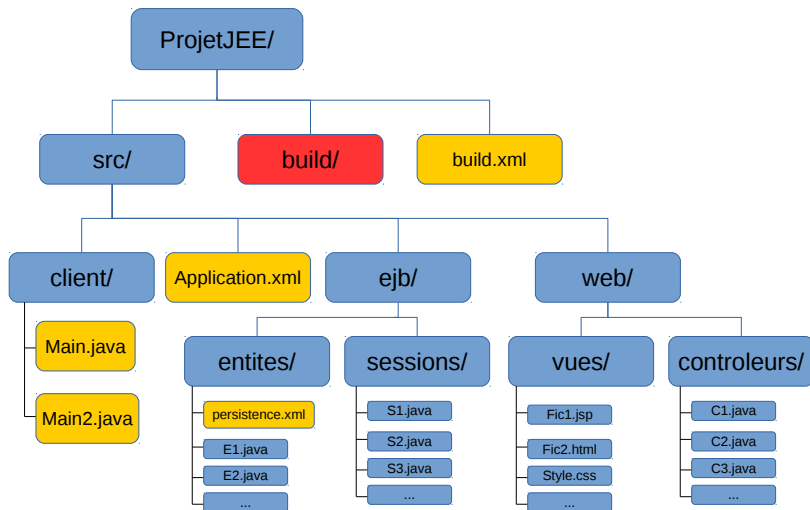
Fonctionnement serveur JEE (3/3)

- Phase de retrait (undeploy) :
 - ➊ Analyse fichier XML META-INF/application.xml
 - ➋ Pour chaque module (ejb ou web), suppression des espaces créés et des programmes compilés
 - ➌ Pour chaque module de type web :

Fonctionnement serveur JEE (3/3)

- Phase de retrait (undeploy) :
 - ➊ Analyse fichier XML META-INF/application.xml
 - ➋ Pour chaque module (ejb ou web), suppression des espaces créés et des programmes compilés
 - ➌ Pour chaque module de type web :
 - ➊ Retrait des liens de correspondance de la table d'associations (nom symbolique - localisation programme)

Le framework Polytech au complet



Web :clients légers versus clients riches

- Constats clients légers webs :

Web : clients légers versus clients riches

- Constats clients légers webs :
 - ▶ + déploiement quasi instantané : un navigateur web suffit

Web : clients légers versus clients riches

- Constats clients légers webs :
 - ▶ + déploiement quasi instantané : un navigateur web suffit
 - ▶ – une interface graphique limitée

Web : clients légers versus clients riches

- Constats clients légers webs :
 - ▶ + déploiement quasi instantané : un navigateur web suffit
 - ▶ – une interface graphique limitée
 - ▶ – programmation web fastidieuse (succession de pages HTML)
peut-on parler de programmation ?

Web : clients légers versus clients riches

- Constats clients légers webs :
 - ▶ + déploiement quasi instantané : un navigateur web suffit
 - ▶ – une interface graphique limitée
 - ▶ – programmation web fastidieuse (succession de pages HTML)
peut-on parler de programmation ?
- Vers des clients webs riches :

Web :clients légers versus clients riches

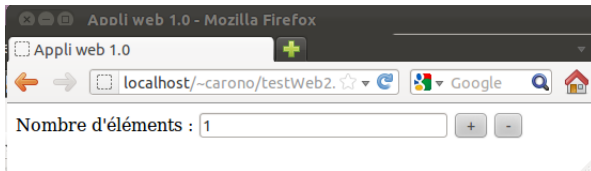
- Constats clients légers webs :
 - ▶ + déploiement quasi instantané : un navigateur web suffit
 - ▶ – une interface graphique limitée
 - ▶ – programmation web fastidieuse (succession de pages HTML)
peut-on parler de programmation ?
- Vers des clients webs riches :
 - ▶ – nécessite un runtime intégré au navigateur (exemple plugin flash)

Web : clients légers versus clients riches

- Constats clients légers webs :
 - ▶ + déploiement quasi instantané : un navigateur web suffit
 - ▶ – une interface graphique limitée
 - ▶ – programmation web fastidieuse (succession de pages HTML)
peut-on parler de programmation ?
- Vers des clients webs riches :
 - ▶ – nécessite un runtime intégré au navigateur (exemple plugin flash)
 - ▶ + objectifs : meilleure ergonomie, retour à la "vraie" programmation

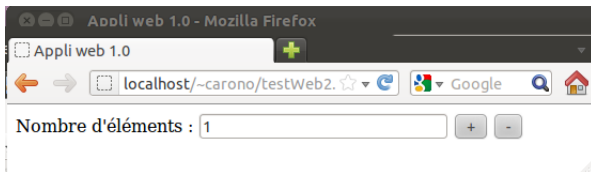
Qu'est ce qui ne va pas dans Web 1.0 ?

- Une comparaison Web 1.0 - 2.0 par l'exemple :



Qu'est ce qui ne va pas dans Web 1.0 ?

- Une comparaison Web 1.0 - 2.0 par l'exemple :



- Objectif : modifier la valeur du champ de texte par l'appui sur des boutons '+' et '-'
Exemple classique de quantité d'un produit dans les applications de commerce électronique.

Implémentation Web 1.0

- Technologies HTML et PHP, fichier web1.php

```
<html>
<head> <title>Appli web 1.0</title> </head>
<body>
  <?php
    extract($_POST) ;
    if (isset($_POST['nbElement'])) {
      if ($action=="+") $nbElement=$nbElement+1 ;
      else $nbElement=$nbElement-1 ;
    } else $nbElement=1 ;
  ?>
  <form action="web1.php" method="post">
    Nombre d'elements :
    <input type="text" name="nbElement"
      value="<? echo $nbElement ; ?>" />
    <input type="submit" name="action" value="+" />
    <input type="submit" name="action" value="-" />
  </form> </body> </html>
```

Implémentation Web 2.0

- Technologies HTML et Javascript, fichier web2.html

```
<html>
<head> <title>Appli web 2.0</title>
<script type="text/javascript">
    function modifierNbElement(action) {
        textfield=document.getElementById('tf_nbEl') ;
        valeur=parseInt(textfield.getAttribute('value')) ;
        if (action=='+')
            textfield.setAttribute('value',valeur+1) ;
        else
            textfield.setAttribute('value',valeur-1) ;
    }
</script> </head> <body>
    Nombre d'elements : <input id="tf_nbEl" type="text"
                           name="nbElement" value="1" />
    <button onclick="modifierNbElement(' + ');">+</button>
    <button onclick="modifierNbElement(' - ');">-</button>
</body> </html>
```

Comparaison coût réseau des deux approches

- Approche Web 1.0 :

Comparaison coût réseau des deux approches

- Approche Web 1.0 :
 - ▶ Premier chargement : appel puis transfert web1.php (interface graphique + données)

Comparaison coût réseau des deux approches

- Approche Web 1.0 :

- ▶ Premier chargement : appel puis transfert web1.php (interface graphique + données)
- ▶ à chaque modification : appel puis transfert web1.php (interface graphique + données)

Comparaison coût réseau des deux approches

- Approche Web 1.0 :
 - ▶ Premier chargement : appel puis transfert web1.php (interface graphique + données)
 - ▶ à chaque modification : appel puis transfert web1.php (interface graphique + données)
- Approche Web 2.0 :

Comparaison coût réseau des deux approches

- Approche Web 1.0 :
 - ▶ Premier chargement : appel puis transfert web1.php (interface graphique + données)
 - ▶ à chaque modification : appel puis transfert web1.php (interface graphique + données)
- Approche Web 2.0 :
 - ▶ Premier chargement : appel puis transfert web2.html (interface graphique + données)

Comparaison coût réseau des deux approches

- Approche Web 1.0 :
 - ▶ Premier chargement : appel puis transfert web1.php (interface graphique + données)
 - ▶ à chaque modification : appel puis transfert web1.php (interface graphique + données)
- Approche Web 2.0 :
 - ▶ Premier chargement : appel puis transfert web2.html (interface graphique + données)
 - ▶ à chaque modification : coût nul.

Qu'est ce qui ne va pas dans Web 1.0 ?

- En résumé :

Qu'est ce qui ne va pas dans Web 1.0 ?

- En résumé :
 - ▶ L'interface graphique est figée :
changer une partie de l'interface nécessite de recharger toute la page !

Qu'est ce qui ne va pas dans Web 1.0 ?

- En résumé :

- ▶ L'interface graphique est figée :
changer une partie de l'interface nécessite de recharger toute la page !
- ▶ Problème de latence réseau (insupportable pour l'utilisateur) dues à :

Qu'est ce qui ne va pas dans Web 1.0 ?

- En résumé :

- ▶ L'interface graphique est figée :
changer une partie de l'interface nécessite de recharger toute la page !
- ▶ Problème de latence réseau (insupportable pour l'utilisateur) dues à :
 - ★ mode synchrone d'envoi des données d'un formulaire

Qu'est ce qui ne va pas dans Web 1.0 ?

- En résumé :

- ▶ L'interface graphique est figée :
changer une partie de l'interface nécessite de recharger toute la page !
- ▶ Problème de latence réseau (insupportable pour l'utilisateur) dues à :
 - ★ mode synchrone d'envoi des données d'un formulaire
 - ★ les données récupérées concernent l'interface ET les données (toute la nouvelle page HTML), c'est lourd et ça prend du temps à se télécharger et s'afficher

La technologie AJAX

- La technologie du moment, "Quoi, tu ne connais pas Ajax?"

La technologie AJAX

- La technologie du moment, "Quoi, tu ne connais pas Ajax?"
- Symbole du web 2.0 (c'est du marketing)

La technologie AJAX

- La technologie du moment, "Quoi, tu ne connais pas Ajax?"
- Symbole du web 2.0 (c'est du marketing)
- Une bonne référence : "AJAX en pratique" (Ajax in Action)

La technologie AJAX

- La technologie du moment, "Quoi, tu ne connais pas Ajax?"
- Symbole du web 2.0 (c'est du marketing)
- Une bonne référence : "AJAX en pratique" (Ajax in Action)
- Rien de révolutionnaire mais la synergie de technologies existantes :

La technologie AJAX

- La technologie du moment, "Quoi, tu ne connais pas Ajax?"
- Symbole du web 2.0 (c'est du marketing)
- Une bonne référence : "AJAX en pratique" (Ajax in Action)
- Rien de révolutionnaire mais la synergie de technologies existantes :
- Acronyme pour "Asynchronous JavaScript + XML"

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript
 - ▶ Langage de script. . .

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript
 - ▶ Langage de script. . .
 - ▶ Orienté objet

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript
 - ▶ Langage de script. . .
 - ▶ Orienté objet
 - ▶ Code embarqué dans les pages HTML

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript

- ▶ Langage de script. . .
- ▶ Orienté objet
- ▶ Code embarqué dans les pages HTML
- ▶ Fortes connexions avec le cycle de vie du navigateur

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript
 - ▶ Langage de script. . .
 - ▶ Orienté objet
 - ▶ Code embarqué dans les pages HTML
 - ▶ Fortes connexions avec le cycle de vie du navigateur
- Technologie 2 : la technologie CSS

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript
 - ▶ Langage de script. . .
 - ▶ Orienté objet
 - ▶ Code embarqué dans les pages HTML
 - ▶ Fortes connexions avec le cycle de vie du navigateur
- Technologie 2 : la technologie CSS
 - ▶ Acronyme pour "Cascading Style Sheet"

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript
 - ▶ Langage de script. . .
 - ▶ Orienté objet
 - ▶ Code embarqué dans les pages HTML
 - ▶ Fortes connexions avec le cycle de vie du navigateur
- Technologie 2 : la technologie CSS
 - ▶ Acronyme pour "Cascading Style Sheet"
 - ▶ Découpage propre charte graphique / les données à afficher

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript
 - ▶ Langage de script. . .
 - ▶ Orienté objet
 - ▶ Code embarqué dans les pages HTML
 - ▶ Fortes connexions avec le cycle de vie du navigateur
- Technologie 2 : la technologie CSS
 - ▶ Acronyme pour "Cascading Style Sheet"
 - ▶ Découpage propre charte graphique / les données à afficher
 - ▶ Styles visuels **réutilisables**

Les 4 technologies associées à AJAX (1/3)

- Technologie 1 : le langage JavaScript
 - ▶ Langage de script. . .
 - ▶ Orienté objet
 - ▶ Code embarqué dans les pages HTML
 - ▶ Fortes connexions avec le cycle de vie du navigateur
- Technologie 2 : la technologie CSS
 - ▶ Acronyme pour "Cascading Style Sheet"
 - ▶ Découpage propre charte graphique / les données à afficher
 - ▶ Styles visuels **réutilisables**
 - ▶ Qualités évolutives (un seul fichier à évoluer)

CSS en un seul slide !

- `h1 { color : red } →` les balises `<h1>` sont en rouge
- `div h1 { color : red ; } →` les balises `<h1>` contenues dans une balise `<div>` sont en rouge
- `.attention { border: solid blue 1px; background-color: cyan} →` classe de style applicable à toute balise html (exemple `<div class="attention">`
- `#id121 { color:blue } →` style applicable à l'**unique** élément `<XXX id="id121"> ...`
- Appliquer une feuille de style :
`<link rel="stylesheet" href="./css/style.css" />`
- Démo : <http://www.csszengarden.com/tr/francais/>
- Des frameworks CSS : charte, **placement** des composants
ex : framework Twitter Bootstrap <http://getbootstrap.com>

Les 4 technologies associées à AJAX (2/3)

- Technologie 3 : la technologie XML

Les 4 technologies associées à AJAX (2/3)

- Technologie 3 : la technologie XML
 - ▶ Les pages HTML sont en fait conformes à XML

Les 4 technologies associées à AJAX (2/3)

- Technologie 3 : la technologie XML
 - ▶ Les pages HTML sont en fait conformes à XML
 - ▶ On peut parcourir le contenu de cette page grâce à l'API DOM avec JavaScript

Les 4 technologies associées à AJAX (2/3)

- Technologie 3 : la technologie XML

- ▶ Les pages HTML sont en fait conformes à XML
- ▶ On peut parcourir le contenu de cette page grâce à l'API DOM avec JavaScript
- ▶ On peut modifier la structure de la page (et pas la totalité!)!!

Petite illustration du DOM avec JavaScript/JQuery

- La fonction Javascript/JQuery `$(se1)` sélectionne le ou les objets HTML selon le sélecteur CSS `se1` donné en paramètre (lignes 6,8)

```
1 <html>
2 <head>
3   <script type="text/javascript" src="./js/jquery.min.js"></script>
4   <script type="text/javascript">
5     function ajout() {
6       nb=$("#zone p").length+3 ; // analyse du DOM
7       if (nb==6) return ;
8       $("#zone").append("<p>IS et IS 2A "+nb+"</p>") ;// modification DOM
9     }
10  </script>
11 </head>
12 <body>
13   <h1 id="titre">bonjour les IS <button onclick="ajout();">...</button>
14   </h1> <div id="zone"></div>
15 </body>
16 </html>
```

Les 4 technologies associées à AJAX (3/3)

- Technologie 4 : l'asynchronisme

Les 4 technologies associées à AJAX (3/3)

- Technologie 4 : l'asynchronisme
 - ▶ Invocation d'un traitement sans bloquer l'utilisateur

Les 4 technologies associées à AJAX (3/3)

- Technologie 4 : l'asynchronisme
 - ▶ Invocation d'un traitement sans bloquer l'utilisateur
 - ▶ Diminue notablement les problèmes de latence réseau

Les 4 technologies associées à AJAX (3/3)

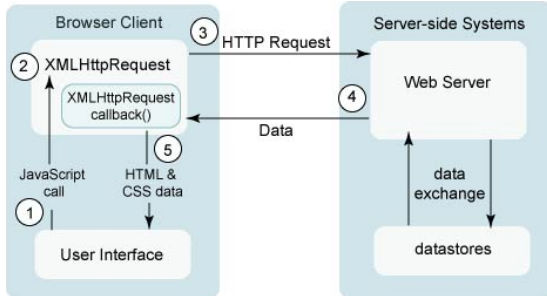
- Technologie 4 : l'asynchronisme

- ▶ Invocation d'un traitement sans bloquer l'utilisateur
- ▶ Diminue notablement les problèmes de latence réseau
- ▶ Charge uniquement des données (pas l'interface) de manière asynchrone

Les 4 technologies associées à AJAX (3/3)

● Technologie 4 : l'asynchronisme

- ▶ Invocation d'un traitement sans bloquer l'utilisateur
- ▶ Diminue notablement les problèmes de latence réseau
- ▶ Charge uniquement des données (pas l'interface) de manière asynchrone
- ▶ "The" méthode : XMLHttpRequest et ses multiples extensions



source : <https://marcautran.developpez.com>

Architecture Logicielle Web 2.0 (1/2)

- Structuration de l'application en niveaux :

Architecture Logicielle Web 2.0 (1/2)

- Structuration de l'application en niveaux :
 - ▶ Niveau données (exemple SGBD)

Architecture Logicielle Web 2.0 (1/2)

- Structuration de l'application en niveaux :
 - ▶ Niveau données (exemple SGBD)
 - ▶ Niveau code métier (exemple PHP)

Architecture Logicielle Web 2.0 (1/2)

- Structuration de l'application en niveaux :
 - ▶ Niveau données (exemple SGBD)
 - ▶ Niveau code métier (exemple PHP)
 - ▶ Niveau Interface Homme Machine (exemples : HTML et PHP, HTML et javascript)

Architecture Logicielle Web 2.0 (2/2)

- Interactions entre les niveaux :

Architecture Logicielle Web 2.0 (2/2)

- Interactions entre les niveaux :
 - ▶ Données - code métier : API bases de données (ex : PDO)

Architecture Logicielle Web 2.0 (2/2)

- Interactions entre les niveaux :
 - ▶ Données - code métier : API bases de données (ex : PDO)
 - ▶ Code métier - IHM : échanges de données (ex : XML ou JSON)

Échange de données

- Données XML :

Échange de données

- Données XML :
 - ▶ Utilisé pour services webs

Échange de données

- Données XML :
 - ▶ Utilisé pour services webs
 - ▶ Inconvénient : verbeux, lourd à décoder

Échange de données

- Données XML :
 - ▶ Utilisé pour services webs
 - ▶ Inconvénient : verbeux, lourd à décoder
- Données JSON (JavaScript Object Notation) :

Échange de données

- Données XML :
 - ▶ Utilisé pour services webs
 - ▶ Inconvénient : verbeux, lourd à décoder
- Données JSON (JavaScript Object Notation) :
 - ▶ Format de données textuel, générique

Échange de données

- Données XML :
 - ▶ Utilisé pour services webs
 - ▶ Inconvénient : verbeux, lourd à décoder
- Données JSON (JavaScript Object Notation) :
 - ▶ Format de données textuel, générique
 - ▶ Avantages : simplicité, décodage rapide, typage des données

Structure de documents JSON

- Un document JSON ne comprend que deux éléments structurels :

Structure de documents JSON

- Un document JSON ne comprend que deux éléments structurels :
 - ▶ des ensembles de paires nom / valeur

Structure de documents JSON

- Un document JSON ne comprend que deux éléments structurels :
 - ▶ des ensembles de paires nom / valeur
 - ▶ des listes ordonnées de valeurs

Structure de documents JSON

- Un document JSON ne comprend que deux éléments structurels :
 - ▶ des ensembles de paires nom / valeur
 - ▶ des listes ordonnées de valeurs
- Ces mêmes éléments représentent 3 types de données :

Structure de documents JSON

- Un document JSON ne comprend que deux éléments structurels :
 - ▶ des ensembles de paires nom / valeur
 - ▶ des listes ordonnées de valeurs
- Ces mêmes éléments représentent 3 types de données :
 - ▶ des objets ('{' ... '}')

Structure de documents JSON

- Un document JSON ne comprend que deux éléments structurels :
 - ▶ des ensembles de paires nom / valeur
 - ▶ des listes ordonnées de valeurs
- Ces mêmes éléments représentent 3 types de données :
 - ▶ des objets ('{' ... '}')
 - ▶ des tableaux ('[' ... ']')

Structure de documents JSON

- Un document JSON ne comprend que deux éléments structurels :
 - ▶ des ensembles de paires nom / valeur
 - ▶ des listes ordonnées de valeurs
- Ces mêmes éléments représentent 3 types de données :
 - ▶ des objets ('{' ... '}')
 - ▶ des tableaux ('[' ... ']')
 - ▶ des valeurs génériques de type tableau, objet, booléen (true, false) , nombre, chaîne ou null.

Un exemple JSON

```
{  
  "name": "Frank",  
  "age": 24,  
  "engaged": true,  
  "favorite_tv_shows": [  
    "Lost", "Dirty Jobs",  
    "Deadliest Catch", "Man vs Wild"  
  ]  
}
```

- JSON sous-ensemble de la notation objet de JavaScript

```
var objetJSON = {  
    "name": "Franck", "age":24, "engaged" : true ,  
    "favorite_tv_shows" : [  
        "Lost", "Dirty Jobs", "Deadliest Catch", "Man vs Wild"  
    ]  
};  
alert(objetJSON.name) // "Franck"  
alert(objetJSON.favorite_tv_shows[1]) ; // "Dirty Jobs"  
var chaineJSON = JSON.stringify(objetJSON) ; alert(chaineJSON) ;  
var secondObjetJSON = eval( '(' + chaineJSON + ')' ) ; // String to Json
```

- Utilisation des tableaux associatifs PHP
- Deux fonctions : `json_encode($tab)` et `json_decode($chaine)`

```
$tab=array (  
    "name" => "Franck", "age" => 24, "engaged" => true ,  
    "favorite_tv_shows" => array ("Lost", "Dirty Jobs" ,  
        "Deadliest catch", "Man vs Wild")  
);  
$chaineJSON=json_encode($tab);
```

PHP, bases de données et JSON

```
<?php
    header( 'Access-Control-Allow-Origin: *' );
    header( "Content-type: application/json" );

    ... // connexion à une base
    $resultatRequete=pg_query( $connect , $requeteSQL ) ;
    $data=pg_fetch_all( $resultatRequete );
    $chaineJSON=json_encode( $data );
    echo $chaineJSON ;

?>
```

Java et JSON

- Java et JSON : <http://www.json.org/java/>

Java et JSON

- Java et JSON : <http://www.json.org/java/>
 - ▶ Des fonctions "Json to String" et "String to Json"

Java et JSON

- Java et JSON : <http://www.json.org/java/>
 - ▶ Des fonctions "Json to String" et "String to Json"
 - ▶ Parseur JSON

Java et JSON

- Java et JSON : <http://www.json.org/java/>
 - ▶ Des fonctions "Json to String" et "String to Json"
 - ▶ Parseur JSON
- D'autres API orientées "Json to Java Objets" : Jackson, Google Json (Gson),...

Java et JSON : codage JSON

```
import org.json.* ;  
...  
  
JSONObject jsonObj = new JSONObject();  
jsonObj.put("name", "Franck");  
jsonObj.put("age", 24);  jsonObj.put("engaged", true) ;  
JSONArray jsArray = new JSONArray() ;  
jsArray.put("Lost") ; jsArray.put("DirtyJobs") ;  
jsArray.put("Deadliest catch") ; jsArray.put("Man vs Wild") ;  
jsonObj.put("favorite_tv_shows", jsArray) ;  
System.out.println("json"+jsonObj.toString()) ;
```

Java et JSON : décodage JSON

```
JSONObject jsObj = new JSONObject(chaineJSON) ;  
String name = jsObj.getString("name") ;  
int age = jsObj.getInt("age") ;  
boolean engaged = jsObj.getBoolean("engaged") ;  
String favoriteTvShows[] = new String[4] ;  
JSONArray jsA= jsObj.getJSONArray("favorite_tv_shows") ;  
for (int index=0; index < jsA.length();index++)  
    favoriteTvShows[index]= (String) jsA.get(index) ;
```

Un exemple complet AJAX - le service Web (JSON/PHP)

partie : I

```
<?php
/**  Fichier : serviceWebPromo.php
 * service Web au format JSON écrit en PHP qui retourne le
 * responsable de la promo
 * l'acronyme "promo" est passé en paramètre de type get
 */
header('Access-Control-Allow-Origin: *');
header("Content-type: application/json");
if (! isset($_GET["promo"]))
    $result=array("code" => -1, "message" => "erreur paramètre
    incorrect") ;
else {
    $server="localhost"; $base="promos"; $user="demo"; $password="
    postgres";
    $base=pg_connect("host=$server dbname=$base user=$user password=
    $password") ;
    if (! $base) // échec connexion
        $result=array("code" => -2, "message" => "erreur connexion BD :
        ".pg_last_error()) ;
    else {
```

Un exemple complet AJAX - le service Web (JSON/PHP)

partie : II

```
$requeteSQL="select responsable from promo where acronyme='".
    $_GET["promo"]."'";
$resultat=pg_query($base,$requeteSQL) ;
if (! $resultat) // échec requête
    $result=array("code" => -3, "message" => "Erreur SQL : ".
        pg_last_error()) ;
elseif (pg_num_rows($resultat)!=1) // aucune ligne trouvée
    $result=array("code"=> -4, "message"=> $_GET["promo"]."
        inconnue") ;
else
    $result=array(
        "code" => 0, "message" => pg_fetch_assoc($resultat)["
            responsable"]
    ) ;
}
}
echo json_encode($result) ; // résultat du service :
?>
```

Un exemple complet AJAX - la vue partie : I

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Démo AJAX</title>
  <script type="text/javascript" src="./js/jquery-3.2.1.min.js" >
  </script>
</head>
<body >
  <h1> Les responsables pédagogiques IS 2A</h1>
  <p>
    <input type="radio" name="promo" onclick="affResp ( ' is2a3 ' );" />
      IS 2A 3
    <input type="radio" name="promo" onclick="affResp ( ' is2a4 ' );" />
      IS 2A 4
    <input type="radio" name="promo" onclick="affResp ( ' is2a5 ' );" />
      IS 2A 5
  </p>

  <h1> Les responsables pédagogiques IS</h1>
  <p>
```

Un exemple complet AJAX - la vue partie : II

```
<input type="radio" name="promo" onclick="affResp('is3');" />IS  
3  
<input type="radio" name="promo" onclick="affResp('is4');" />IS  
4  
<input type="radio" name="promo" onclick="affResp('is5');" />IS  
5
```

</p>

<p>Responsable : </p>

```
<script type="text/javascript">  
function affResp(promo) {  
    $("#resp").html("patientez ...") ;  
    $.ajax ( { // fonction asynchrone JQuery  
        url: './serviceWebPromo.php',  
        type: 'GET',  
        data: "promo="+promo ,  
        success: function(result){  
            if (result.code ==0) $("#resp").html(result.message) ;  
            else $("#resp").html("Echec: "+result.message) ;  
        }  
    } ) ;  
}
```

Un exemple complet AJAX - la vue partie : III

```
</script>  
</body>  
</html>
```

L'industrie Web 2.0

- Essor très rapide, multitude de frameworks

L'industrie Web 2.0

- Essor très rapide, multitude de frameworks
- Des frameworks ajax : rico, prototype (copier/coller, drag and drop, ...)

L'industrie Web 2.0

- Essor très rapide, multitude de frameworks
- Des frameworks ajax : rico, prototype (copier/coller, drag and drop, ...)
- Des applications vedettes : Google Calendar, Google maps, Google Office, Google mail, ...

L'industrie Web 2.0

- Essor très rapide, multitude de frameworks
- Des frameworks ajax : rico, prototype (copier/coller, drag and drop, ...)
- Des applications vedettes : Google Calendar, Google maps, Google Office, Google mail, ...
- Une contrainte : JavaScript, langage qui ne suscite pas l'unanimité.

L'industrie Web 2.0

- Essor très rapide, multitude de frameworks
- Des frameworks ajax : rico, prototype (copier/coller, drag and drop, ...)
- Des applications vedettes : Google Calendar, Google maps, Google Office, Google mail, ...
- Une contrainte : JavaScript, langage qui ne suscite pas l'unanimité.
 - ▶ Les "pro-javascript" se tournent vers le "full-stack Javascript"
Exemple : AngularJS (UI) +Node.js (Backend)

L'industrie Web 2.0

- Eessor très rapide, multitude de frameworks
- Des frameworks ajax : rico, prototype (copier/coller, drag and drop, ...)
- Des applications vedettes : Google Calendar, Google maps, Google Office, Google mail, ...
- Une contrainte : JavaScript, langage qui ne suscite pas l'unanimité.
 - ▶ Les "pro-javascript" se tournent vers le "full-stack Javascript"
Exemple : AngularJS (UI) + Node.js (Backend)
 - ▶ Les "Anti-javascript"
Exemple 1 : JEE : JSTL+EL(UI)+ Servlet+EJB
Exemple 2 : Google Web Toolkit (inclus un compilateur Java vers Javascript)

L'industrie Web 2.0

- Eessor très rapide, multitude de frameworks
- Des frameworks ajax : rico, prototype (copier/coller, drag and drop, ...)
- Des applications vedettes : Google Calendar, Google maps, Google Office, Google mail, ...
- Une contrainte : JavaScript, langage qui ne suscite pas l'unanimité.
 - ▶ Les "pro-javascript" se tournent vers le "full-stack Javascript"
Exemple : AngularJS (UI) +Node.js (Backend)
 - ▶ Les "Anti-javascript"
Exemple 1 : JEE : JSTL+EL(UI)+ Servlet+EJB
Exemple 2 : Google Web Toolkit (inclus un compilateur Java vers Javascript)
 - ▶ Des solutions hybrides apparaissent : AngularJS+JEE,
d'où l'importance d'une architecture logicielle modulaire.

Conclusion

- La technologie Web :

Conclusion

- La technologie Web :
 - ▶ Incontournable !

Conclusion

- La technologie Web :
 - ▶ Incontournable !
 - ▶ Technologie JEE (JSP, JSTL, Servlets) très implantée gros systèmes, des frameworks JEE compatible MVC (JSF, Spring, ...)

Conclusion

- La technologie Web :
 - ▶ Incontournable !
 - ▶ Technologie JEE (JSP, JSTL, Servlets) très implantée gros systèmes, des frameworks JEE compatible MVC (JSF, Spring, ...)
 - ▶ Veille technologique vitale (période de profonds changements)

Le mot de la fin

© Edsger Dijkstra

Si "debugger", c'est supprimer des bugs, alors programmer ne peut être que les ajouter.