Technologies serveur

Benjamin Canou - Christian Queinnec

Cours 4 du 10/12/2012

Architecture des Applications Réticulaires

Tâches du serveur

Génération de documents
Accès aux données, concurrence, sécurité
Conversations et sessions

Choix technologiques

Les choix d'implantations dépendent de plusieurs critères

- Niveau de fiabilité requis
- Temps/budget de développement
- Temps/budget de maintenance
- Forme et taille de l'équipe de développement
- Nombre d'utilisateurs et fréquentation de l'application

Pas de solution miracle et universelle

Génération de documents

Dans n'importe quel langage :

- On peut produire un document (X)HTML
- On peut même forger des réponses HTTP complètes

En fait, dans n'importe quel langage disposant de `print`.

Beaucoup de solutions

- De plus ou moins haut niveau
- Plus ou moins sûres
- Plus ou moins facile d'accès et souples
- Un thème important, pourtant souvent mis de côté

Accès aux données

Vu au cours 1 : dialogue entre la base de données et le langage

- Sépararation des rôles / tâches
- Traitement des très grands volumes de données
- Traitement des accès concurrents très fréquents

Un domaine en mutation

Concurrence

Une problématique majeure, historiquement gérée en reposant :

- Sur un serveur Web éprouvé
- La base de données

Le serveur Apache (par exemple) :

- S'occupe d'ordonner les connexions
- Gère les entrées / sorties
- Est configurable (ex pour la concurrence : fork, prefork,threads)

Les solutions à la mode (REST, Node.JS,etc.) demandent plus au programmeur

La base de données :

- Est utilisée pour partager des informations entre toutes les requêtes
- Assure la cohérence par le modèle transactionnel

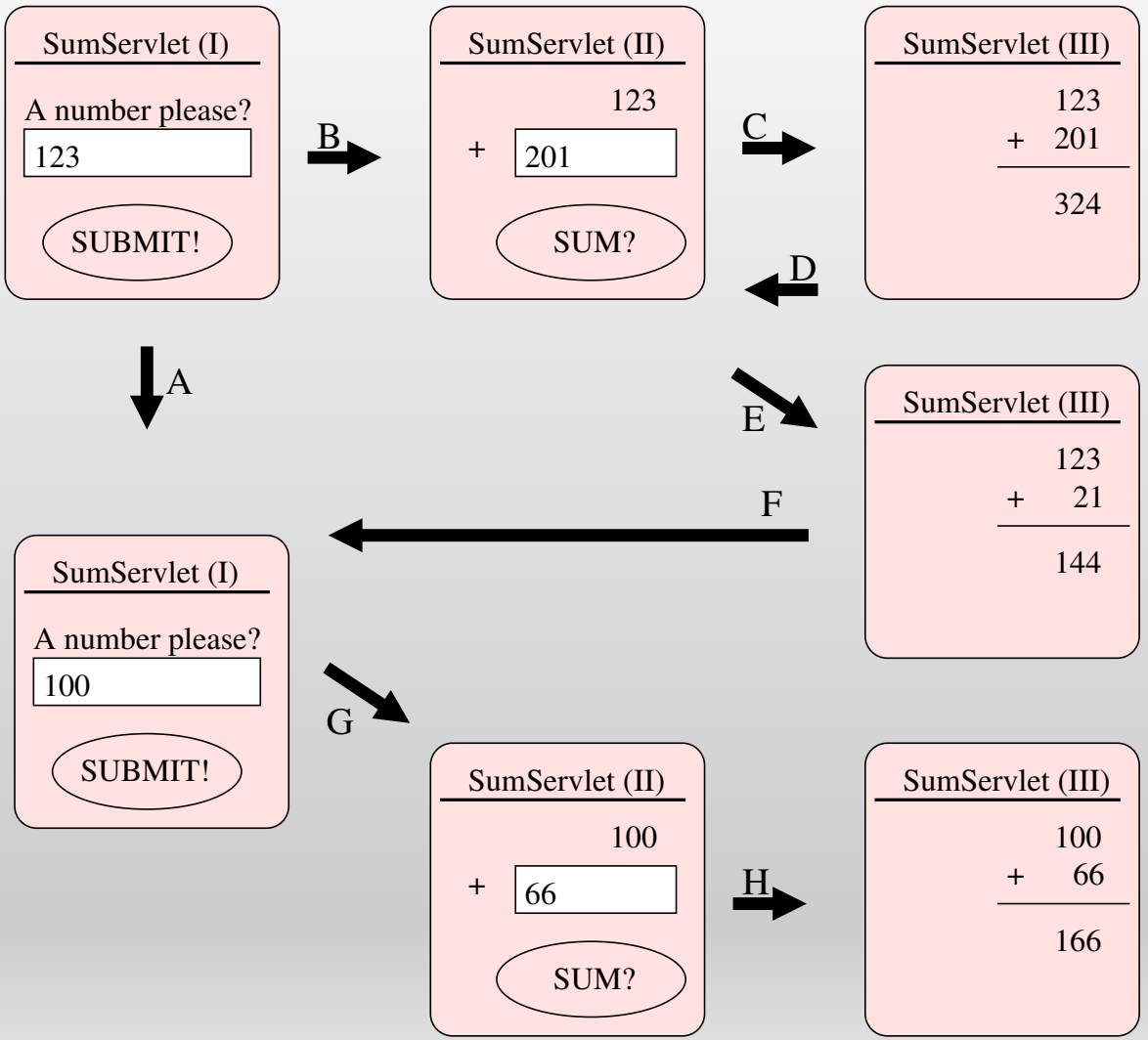
Mais les bases à la mode n'assurent plus ni l'un ni l'autre

Pour les services,on a vu :

- REST : requêtes idépendantes
- SOAP : séquences simples de requêtes

Dans une application pour humains, on a :

- De vraies conversations (formulaires multiples, etc.)
- Le retour en arrière et le clonage du navigateur

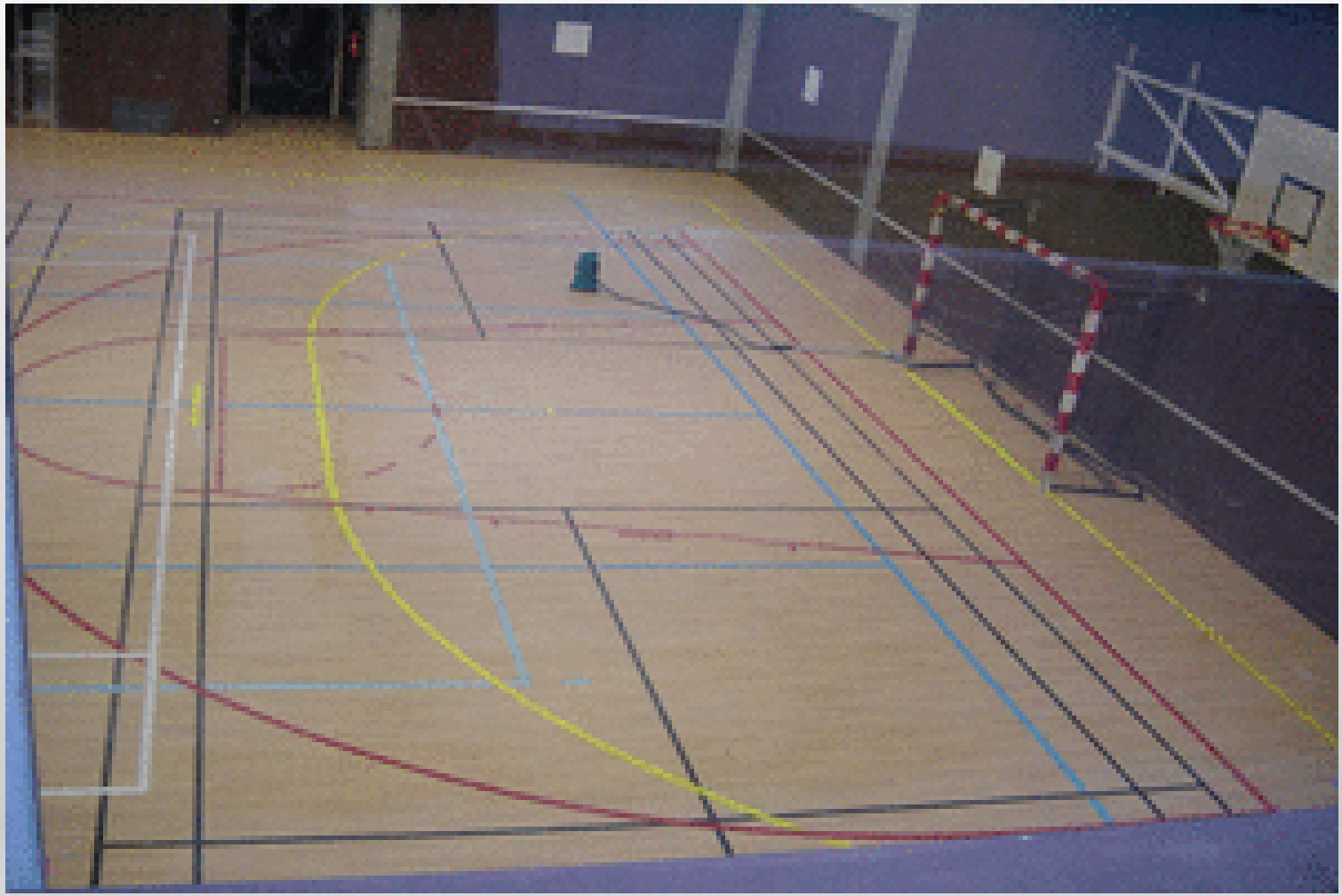

SumServlet (I)
A number please?
123
SUBMIT!
B
SumServlet (II)
123
+ 201
SUM?
C
SumServlet (III)
123
+ 201
324
D
A
E
SumServlet (III)
123
+ 21
144
F
SumServlet (I)
A number please?
100
SUBMIT!
G
SumServlet (II)
100
+ 66
SUM?
H
SumServlet (III)
100
+ 66
166

Fiabilité, rapidité de développement

L'approche langage

Architecture des Applications Réticulaires

Modèle historique : métissage



Un programme en plusieurs langages mêlés.

En réalité : un langage principal + inclusions d'un ou plusieurs langages

- C et cpp (pour les macros)
- notation *backquote* en Lisp/Scheme
- programmation littéraire (TEX)
- Java et javadoc, commentaires structurés, annotations
- *X* et SQL (par exemple, jsql = Java + SQL)

Constat : pour une page basique, le HTML statique est le plus important
Idée : Inclure des parties de langage généraliste dans HTML
On utilise un Compilateur traduisant le langage HTML+X en X.

- Une JSP est compilée en une servlet (une classe Java)
- Shervlet = sh+HTML

```
1 : <html><head><title> shervlet </title></head><body>
2 : Il est <?sh date ?> en ce moment<br>
3 : dit <?sh uname —a ?></body></html>
4 :
5 : <html><head><title> shervlet </title></head><body>
6 : <?sh if [ "$REMOTE_HOST" = '1.2.3.4' ] ; then ?>
7 : Salut vieille branche, <?sh ;else ?>
8 : Bonjour, <?sh fi ?> </body></html>
```

- Compilation : `sed -e 's/(^|?>)/echo "/g' -e 's/($|<?)/"/g'`

Première idée : HTML bien caché derrière des composants

- Utilisé par la plupart des bibliothèques de composants propriétaires
- Astuce de typage : on empaquette chaque composant dans un DIV
- Permet des implantations Web de toolkits existants (ex. Gtk + HTML5)

Meilleure idée : Représentation intermédiaire du HTML

- On construit un arbre avec des primitives

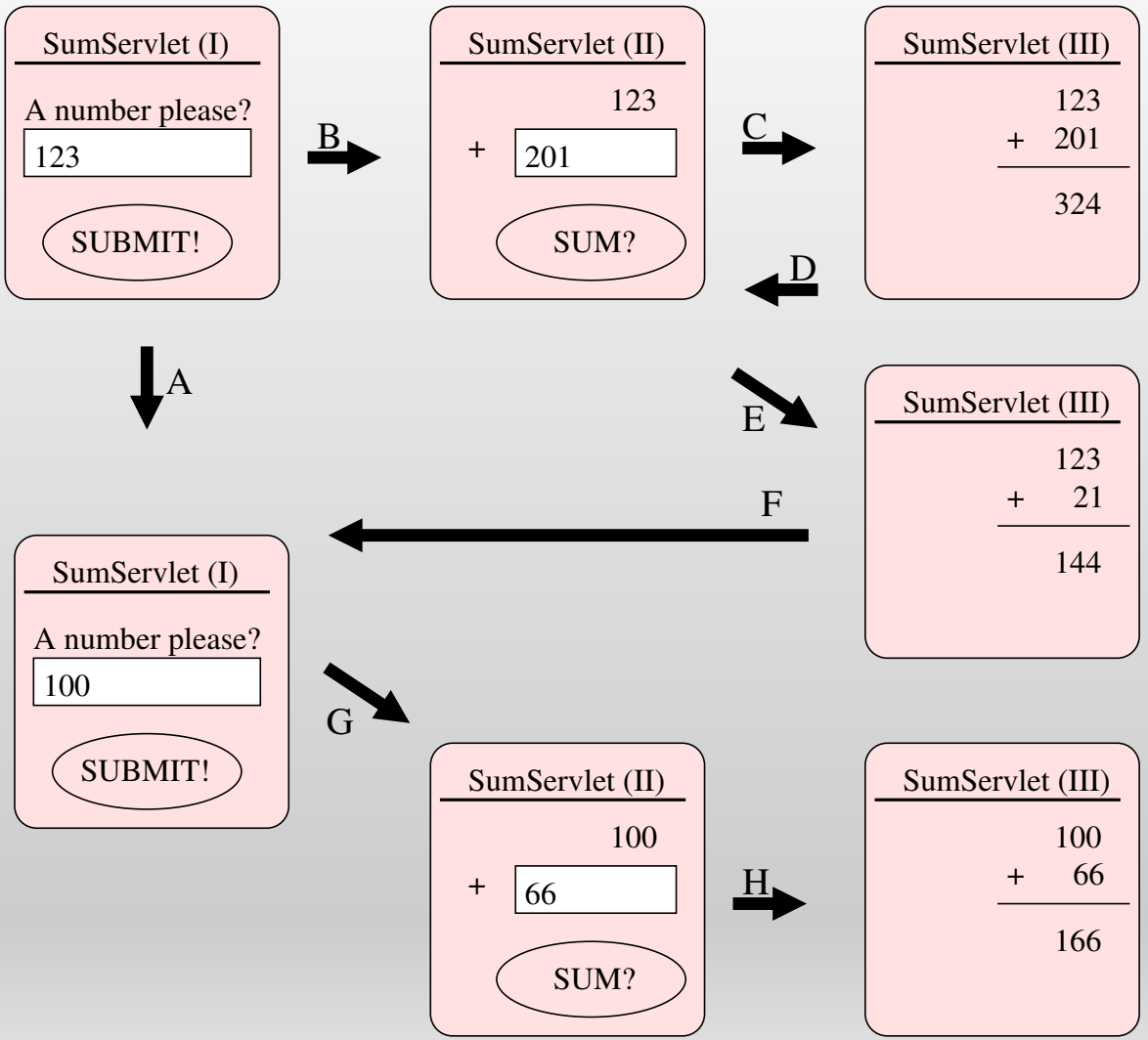
```
1 : (<BODY>
2 :    (<H1> "Bonjour")
3 :    (<P> "Comment allez-vous ? ")
4 :    (<BR>)
5 :    ; ...
```

- Pour Hop, achetez vos places pour le cours de Manuel Serrano en vague 3
- Typage statique de la validité : CDuce, Ocsigen
- Couche basse propre $\rightarrow$ composants plus fiables

La création de document est intégrée au langage, pas mélangée

On introduit plusieurs notions :

- Session : stockage d'informations plus ou moins volatiles
  - Espace sur le serveur + cookie / champs cachés / URL
  - En JSP : requête, session, application, à vie
  - En Ocsigen : plusieurs niveaux de session
  - Le programmeur doit gérer l'empreinte mémoire
- Continuations : stockage de points d'exécution rappelables
  - Sauvegarde de l'état courant pour reprise future possible
  - Nécessite une prise en charge par le langage
  - Implantations serveur : Seaside
  - Notion de service : HOP, Ocsigen, un service = une continuation
  - Links : la continuation est dans l'URL
- Historique : mémorisation des actions et états successifs
  Ex : programmation fonctionnelle réactive (FRP) :
  - Temps vu comme une ligne ponctuée par les évènements
  - Retour en arrière possible
  - Contraintes d'expressivité sur le langage / système de types
  - En fait, utilise des continuations


SumServlet (I)
A number please?
123
SUBMIT!
B
SumServlet (II)
123
+ 201
SUM?
C
SumServlet (III)
123
+ 201
324
D
A
E
SumServlet (III)
123
+ 21
144
F
SumServlet (I)
A number please?
100
SUBMIT!
G
SumServlet (II)
100
+ 66
SUM?
H
SumServlet (III)
100
+ 66
166

Gestion des données persistantes dans le langage

Intégration base - langage :

- Vu au premier cours ; ORM
- LinQ : langages de requêtes SQL intégré à C#
- Macaque : langage de requêtes déclaratives intégré à OCaml
- En OPA : seules données persistantes = DB

On peut même typer les accès :

- HaskellDB :
  - On donne une description déclarative de la base
  - Les requêtes sont vérifiées statiquement
- Plus fort : PGOCaml :
  - On vérifie au déploiement la compatibilité des requêtes avec la base
  - Soit une erreur arrive tout de suite, soit le programme ne peut échouer

Typage des requêtes $\Rightarrow$ pas d'injection par construction

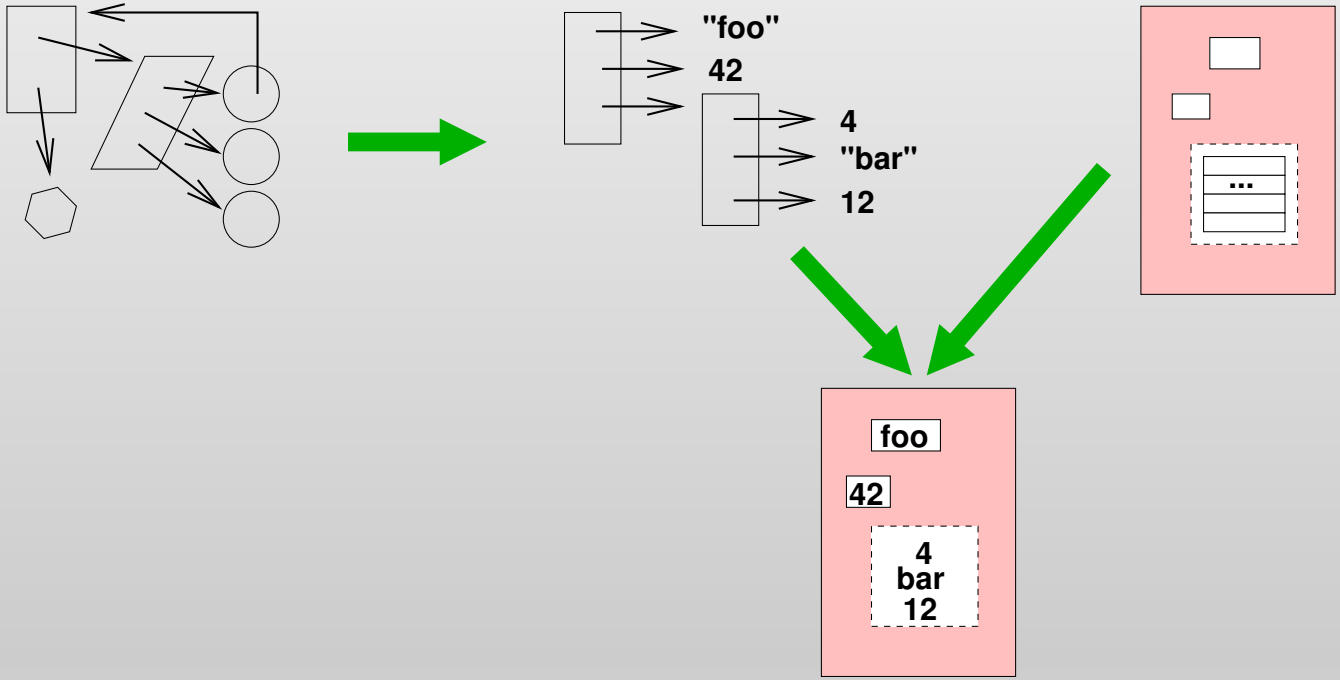
Séparation des tâches

L'approche templates
Tissage, Modèle-Vue-Contrôleur

Architecture des Applications Réticulaires

Tissage (templates)

Linéarisation des objets métiers en des collections arborescentes de données élémentaires puis tissage à un patron (par macro-génération (texte, DOM, *taglib*) ou incrustation) (sur serveur ou client)



Perl -- HTML::Template

Utiliser plutot TemplateToolkit!
Substitution textuelle (HTML, XML, Java, JSON, etc.)

```
1 : my $template = HTML::Template->new(filename => 'test.tmpl');
2 : $template->param('Title' => "L'Aar");
3 : my @names = [ {Nom => 'Lin'}, {Nom => 'Clet'} ];
4 : $template->param('LesNoms' => \@names);
```

```
1 : <html><head><title><TMPL_VAR name='Title' ESCAPE="HTML"></title>
2 : <body> Les voici: <ul>
3 :   <TMPL_LOOP name='LesNoms'>
4 :     <li><TMPL_IF name='Nom'>
5 :           <a href='<TMPL_VAR␣name='Nom'␣ESCAPE="URL">'>?</a>
6 :         </TMPL_IF></li>
7 :   </TMPL_LOOP></ul>
8 : </body></html>
```

L'équivalent en Java: Velocity, WebMacro.

Perl -- Template

```
1 : my $config = {
2 :   INCLUDE_PATH => '/search/path',  # or list ref
3 :   EVAL_PERL    => 1,              # evaluate Perl code blocks
4 : };
5 : my $template = Template->new($config);
6 : my $vars = {
7 :   var1  => $value,
8 :   var2  => \%hash,
9 :   var3  => \@list,
10 :  var4  => \&code,
11 :  var5  => $object,
12 : };
13 : $template->process('myfile.html', $vars)
14 :   || die $template->error();
```

Perl -- Template exemple

```
1 : [% IF hash.key %] Bonjour [% ELSE %] Salut [% END %]
2 :   [%  GET variable %]
3 : [% FOREACH v = list %] et [% v %], [% END %]
```

Autres mots clés: SET, INSERT, INCLUDE, MACRO, FILTER, PERL, etc.

PHP -- SPIP

langage de patrons permettant l'extraction de données venant de base.

```
 1 : <B_personne>
 2 : Voici la liste des #TOTAL_BOUCLE des ...:
 3 : <ul>
 4 :  <BOUCLE_personne(edb:SpipHDR){par
 5 :      LABORATORY_NICKNAME}{par LASTNAME}>
 6 :   <li>[#LABORATORY_NICKNAME]<strong>
 7 :     <a href='mailto:[(#EMAIL|replace{@,(a)})]'>
 8 :      <span style='text-transform: lowercase;'>
 9 :       #FIRSTNAME</span>
10 :      [(#LASTNAME|majuscules)]</a></strong>
11 :   </li>
12 :  </BOUCLE_personne>
13 : </ul>
14 : </B_personne>
15 : Il n'y a personne!
16 : <//B_personne>
```

Java -- Tomcat

- Apache Tomcat est un conteneur de servlets (comme Jetty)
- Il implante une série de standards visant à rendre le déploiement aisé.
- Il peut fonctionner seul comme un serveur httpd ou fonctionner derrière Apache.

Une application Web (une WebApp) est définie par un fichier .war (un .jar avec un répertoire WEB—INF/web.xml) à installer dans le répertoire webapps de Tomcat.
Accès à l'URL utilisée getContextPath, getServletPath et getPathInfo tel que

```
1 : requestURI = ContextPath + ServletPath + PathInfo
```

```
1 : <?xml version="1.0" encoding="ISO-8859-1"?>
2 : <!DOCTYPE web-app
3 :    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
4 :    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
5 : <web-app>
6 :  <display-name>ICFP2000 talk</display-name>
7 :  <description>The icfp2000 talk as a war file.</description>
8 :
9 :  <!-- These context parameters will appear in
10 :       every ServletConfig. -->
11 :  <context-param>
12 :   <param-name>webmaster</param-name>
13 :   <param-value>Christian.Queinnec@lip6.fr</param-value>
14 :   <description>
15 :    The Email address of the administrator to whom questions
16 :    and comments about this application should be addressed.
17 :   </description>
18 :  </context-param>
```

```
1 :  <servlet>
2 :   <servlet-name>FilterServlet</servlet-name>
3 :   <servlet-class>fr.lip6.qnc.videoc2000.FilterServlet</servlet-class>
4 :   <!-- This servlet serves pages from the CDRom, manages an Internet
5 :        cache and a zone for updated files, filters the pages. -->
6 :   <init-param>
7 :    <param-name>configuration.builder</param-name>
8 :    <param-value>
9 :     fr.lip6.qnc.videoc2000.Videoc2000ConfigurationBuilder
10 :   </param-value>
11 :  </init-param>
12 : </servlet>
```

```
1 :   <servlet-mapping>
2 :     <servlet-name>FilterServlet</servlet-name>
3 :     <url-pattern>/document/*</url-pattern>
4 :   </servlet-mapping>
5 :   <servlet-mapping>
6 :     <servlet-name>LamlPage</servlet-name>
7 :     <url-pattern>*.laml</url-pattern>
8 :   </servlet-mapping>
9 :   <servlet-mapping>
10 :    <servlet-name>XMLPage</servlet-name>
11 :    <url-pattern>*.xml</url-pattern>
12 :  </servlet-mapping>
```

```
1 :     <session-config>
2 :       <session-timeout>120</session-timeout><!-- 2 hours -->
3 :     </session-config>
4 :
5 :     <mime-mapping>
6 :       <extension>pdf</extension>
7 :       <mime-type>application/pdf</mime-type>
8 :     </mime-mapping>
9 :     <mime-mapping>
10 :      <extension>README</extension>
11 :      <mime-type>text/plain</mime-type>
12 :    </mime-mapping>
13 :    <mime-mapping>
14 :      <extension>js</extension>
15 :      <mime-type>application/x-javascript</mime-type>
16 :    </mime-mapping>
17 :
18 :    <welcome-file-list>
19 :      <welcome-file>index.html</welcome-file>
20 :    </welcome-file-list>
21 :
22 : </web-app>
```

Une page = une classe; greffe des composants UI dans (simili-)DOM

```
1 : <!-- Hello.html -->
2 : <html xmlns:wicket="http://wicket.apache.org">
3 : Hello <span wicket:id="name">you</span>!
4 : </html>
```

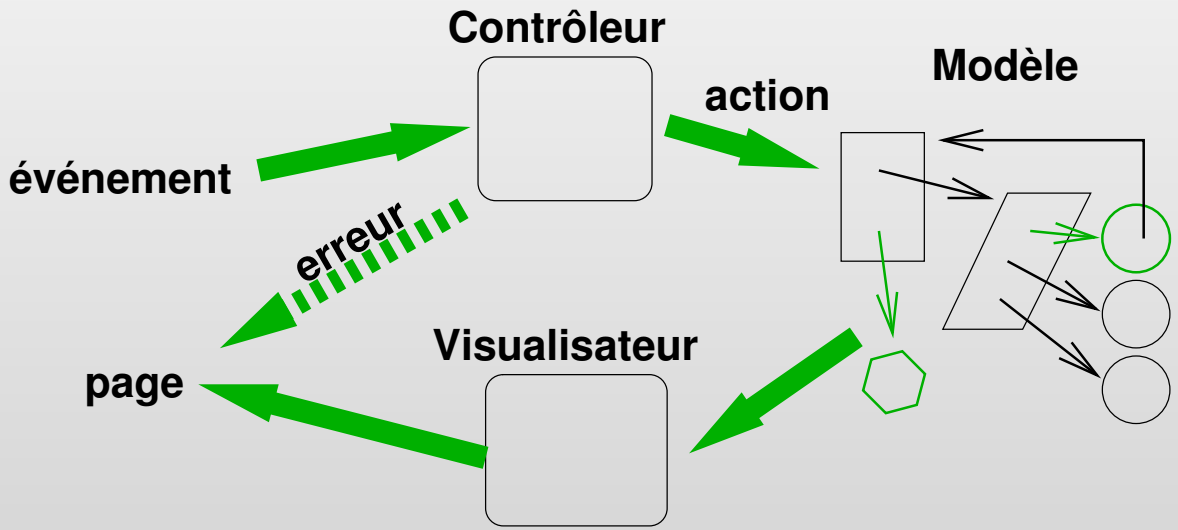
```
1 : // Hello.java
2 : public class Hello extends WebPage {
3 :     public Hello() {
4 :       Label s = new Label("name", "AAR");
5 :       add(s);
6 :     }
7 : }
```

Cascade de transformations d'XML (avec XSLT):

```
1 : <map:sitemap ...>
2 :  <map:pipelines>
3 :   <map:pipeline><map:match pattern="myFirstPipeline">
4 :    <map:generate src="myXmlFile.xml" type="file"/>
5 :       <map:serialize type="xml"/>
6 :    </map:match><map:match pattern="mySecondPipeline">
7 :       <map:generate src="myXmlFile.xml" type="file"/>
8 :       <map:transform src="myXsltFile.xslt" type="xslt"/>
9 :       <map:serialize type="html"/>
10 :   </map:match><map:match pattern="myThirdPipeline">
11 :      <map:generate src="myXmlFile.xml" type="file"/>
12 :      <map:transform src="myXml2PdfFile.xslt" type="xslt"/>
13 :      <map:serialize type="fo2pdf"/>
14 :   </map:match></map:pipeline></map:pipelines></map:sitemap>
```

Model-View-Controler

Depuis Smalltalk 80, la technique du MVC. Les visualisateurs sont souvent des moteurs incrémentiels. Le modèle peut être testé indépendamment des E/S.



Dans le monde du Web, MVC un peu particulier :

- M : fortement couplé à la BD (automatiquement)
- V : composants prédéfinis
- C : la où est toute l'intelligence
- Exemples : RoR, django, Catalyst

- un fichier de configuration faces—config.xml
- des pages .jsp utilisant les bibliothèques d'étiquettes (*taglib*) propres à JSF. La jsp sert de vue et de générateur d'actions pour le contrôleur.
- Une servlet unique FacesServlet
- un état préservé dans FacesContext

```
1 : <h:outputLabel value="Group" for="group" accesskey="g" />
2 : <h:selectOneMenu id="group" validatorMessage="required"
3 :     value="#{contactController.selectedGroupId}">
4 :     <f:selectItems value="#{contactController.groups}" />
5 :     <f:validateLongRange minimum="1" />
6 : </h:selectOneMenu>
```

Java -- JSF cycle de vie

Le cycle de vie:

1. *Restore View:* JSF reconstruit l'arborescence des composants de la page à partir de l'état courant (automatique par JSF)
2. *Apply Request values:* stocke les valeurs présentes dans la requête dans les composants
3. *Process Validations:* valide ces données à l'aide des règles définies dans les composants
4. *Update model values:* met à jour du modèle et engendre les événements signalant les modifications
5. *Invoke Application:* traite des événements et détermine la page suivante
6. *Render Response:* engendre le contenu de la réponse (automatique par JSF)

Java -- JSF navigation

```
1 : <navigation-rule>
2 :     <from-view-id>/previous.jsp</from-view-id>
3 :     <navigation-case>
4 :         <from-outcome>uneAction</from-outcome>
5 :         <to-view-id>/next.jsp</to-view-id>
6 :     </navigation-case>
7 :     <navigation-case>
8 :         <from-outcome>uneAutreAction</from-outcome>
9 :         <to-view-id>/other.jsp</to-view-id>
10 :    </navigation-case>
11 :    <!-- autres navigation-case... -->
12 : </navigation-rule>
```

Java -- JSF

- structure de formulaire assez statique
  - formulaires avec sous-ensembles au choix
  - formulaires avec sous-ensembles dépendant
- nécessaire parallélisme entre page affichée et état sauvegardé
- bibliothèques additionnelles de composants (Tomahawk, Tobago)

Passage à l'échelle

Architecture nuageuse
Élasticité

Architecture des Applications Réticulaires

Gestion de la mémoire par client

Pendant une requête :

- Limiter la taille des documents
- Envoi par morceaux (chunks)
    - Force le modèle d'exécution
    - Difficile avec une représentation intermédiaire
    - Impossible d'annoncer une erreur tardive en HTTP

Entre les requêtes :

- Limiter la taille des données de session
- Dans l'idéal, pas d'information sur les clients : stateless

Cela signifie :

- Fortes implications / limitations sur l'architecture de l'application
- Implications sur les langages / modèles
- Maintenant : on peut déléguer aux clients

Grand nombre de clients :

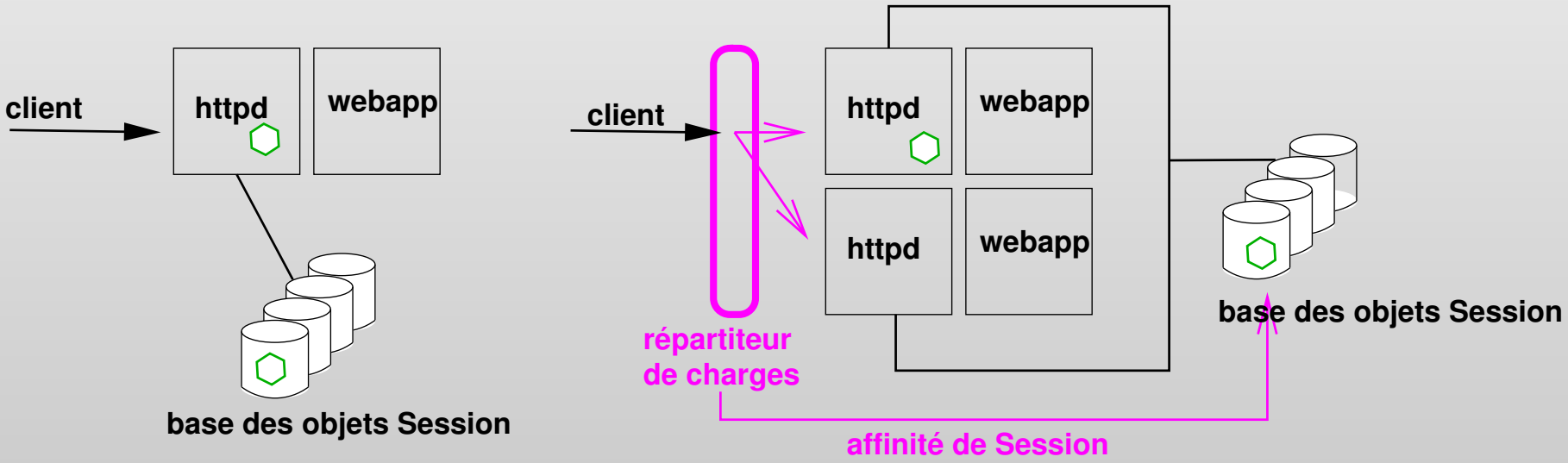
- Plusieurs serveurs, plusieurs BD
- Répartisseur de charge (load balancer) (ex. Wikipédia)
- Utilisation du DNS comme répartiteur (ex. google)

Nombre variable de clients :

- Architecture nuageuse (ex. amazon EC2)
- Trop de clients : louer des machines !
- Pas assez de clients : libérer des machines !

Il faut parfois redistribuer les clients :

- REST : Code HTTP Moved + IP nouveau serveur
- Serveur stateful : pas si simple...



Les Bases de Données Relationnelles (type SQL) savent gérer :

- De très gros volumes de données
- Réparties sur un grand nombre de machines /disques
- Avec une très bonne cohérence grâce au modèle transactionnel

Les bases de données pour réseaux sociaux modernes doivent gérer :

- De trèe très très gros volumes de données
- Réparties sur un très très grand nombre de machines /disques
- En faisant au mieux pour la cohérence...

Les besoins sont spécifiques :

- On n'implanterait pas (par exemple) une gestion bancaire sur ce type de BD
- On ne tient pas à la cohérence parfaite entre les serveurs
- On peut avoir des délais
- Mais on ne veut tout de même pas se tromper de compte / personne

Mouvement NoSQL :

- Émergence de plusieurs nouvelles bases de données, sans standard
- Avec un langage de requêtes ou non
- Des primitives plus simples que l'algèbre relationnel
- Plus ou moins flexibles, plus ou moins efficaces
- Parfois même avec un peu de transactionnel si nécessaire

Quelques noms :

- Clef / Valeur : Cassandra, MemcacheDB, Dynamo, BigTable
- Stockage de documents / records : MongoDB, HBase
- Échanges P2P : BitTorrent

Modèle de base : la Table de Hachage Distribuée (DHT)

- Division horizontale (sharding) : l'ensemble des documents est séparé (pas les documents)
- On hache la clef pour trouver le serveur
- Tolérance aux pannes : réplication
- Variation des serveurs : routage des clefs

Conclusion

De nombreux points de choix
Pas de solution universelle