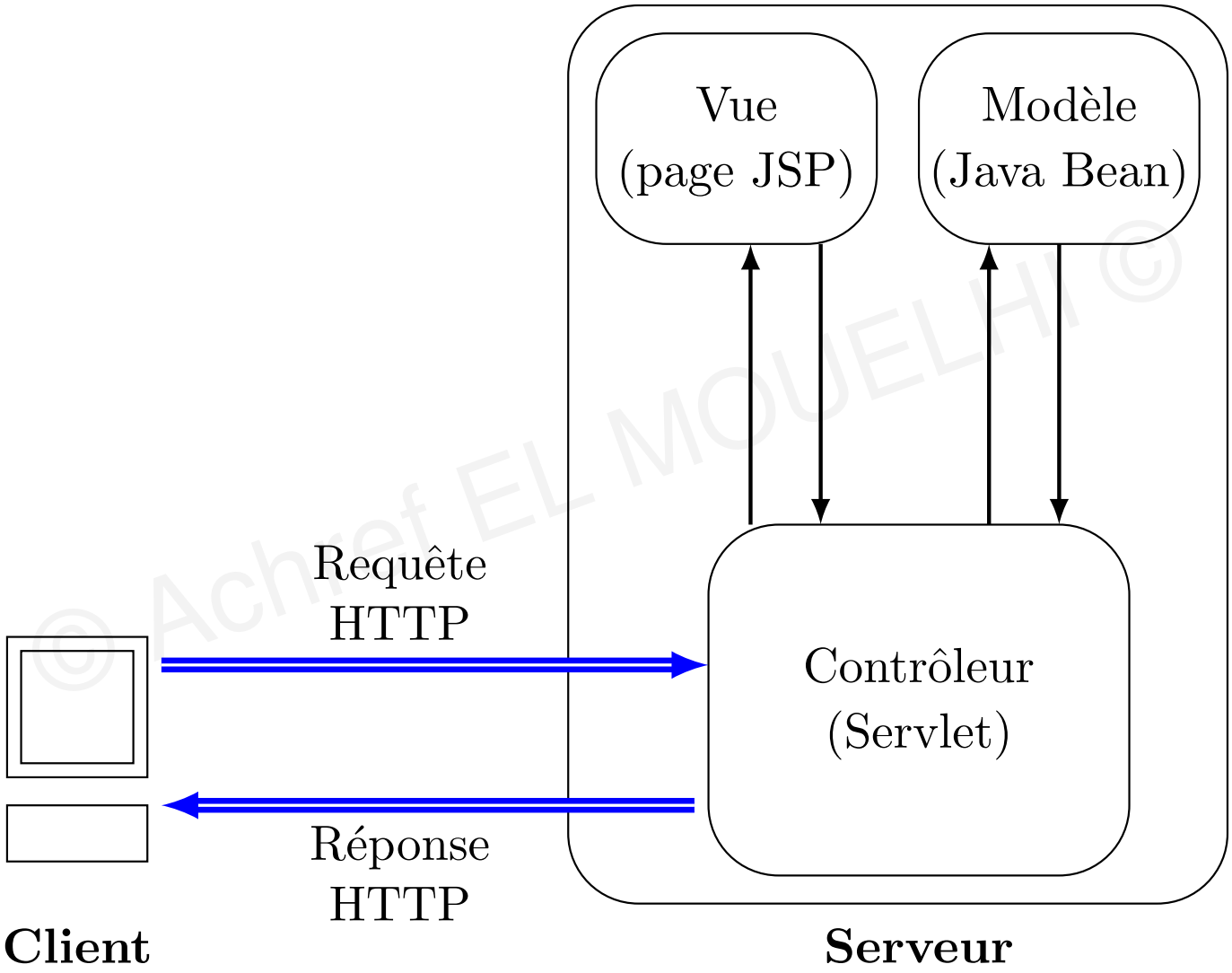
Plan

JEE

JSP

- **Java Server Pages**
- Une technologie de la plateforme **JEE** permettant de créer dynamiquement des pages **HTML** (d'extension `.jsp`)
- Une page **JSP** sera transformée par le compilateur en Servlet
- Les **JSP** sont extensibles : on peut créer nos propres balises **JSP** (avec **JSTL**)

JEE



JEE

Déroulement

- Faire un clic droit sur `WEB-INF` de notre projet
- Aller dans `New` et choisir `JSP File`
- Remplir le champ `File name:` par vue.jsp (par exemple)
- Valider

JEE

Notre vue générée

```
<%@ page language="java" contentType="text/html; charset=
   UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional
   //EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
       charset=UTF-8">
    <title>Insert title here</title>
  </head>
  <body>

  </body>
</html>
```

JEE

Préparons notre Hello World

```
<%@ page language="java" 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>Projet JEE</title>
  </head>
  <body>
    Hello World (depuis une JSP)
  </body>
</html>
```

JEE

Comment l'appeler ?

- C'est toujours le contrôleur (Servlet) qui communique avec les vues

JEE

Pour construire correctement une page HTML

```
  protected void doGet(HttpServletRequest request, HttpServletResponse
     response) throws ServletException, IOException{
    this.getServletContext().getRequestDispatcher("/WEB-INF/vue.jsp").
       forward(request, response);
}
```

JEE

Pour construire correctement une page HTML

```
  protected void doGet(HttpServletRequest request, HttpServletResponse
     response) throws ServletException, IOException{
    this.getServletContext().getRequestDispatcher("/WEB-INF/vue.jsp").
       forward(request, response);
}
```

Explication

- `this.getServletContext()` : permet de communiquer avec d'autres composants (via le conteneur de Servlet).
- `getRequestDispatcher("/WEB-INF/vue.jsp")` : permet d'indiquer l'emplacement de la vue et de la récupérer.
- `forward(request, response)` : pour envoyer la requête et la réponse (on les utilisera plus tard).

JEE

Balises **JSP**

- sont définies par <% ... %>
- Entre ces deux balises, on peut utiliser les bases algorithmiques du langage **Java** :
  - des structures conditionnelles
  - des structures itératives
  - ...
- Les balises **JSP** peuvent être utilisées plusieurs fois dans une page **JSP**.

JEE

Balises spéciales

- `<%-- ... --%>` : pour ajouter un commentaire
- `<%! String var; %>` : pour déclarer une variable directement dans la classe de la servlet.
- `<%= var %>` : pour afficher le contenu de la variable var $\equiv$ `<% out.println(var); %>`

JEE

Attention

Il est déconseillé de mélanger du code **HTML** avec du code **Java**.

JEE

Directives

- Instructions dans des balises **JSP** spéciales
- Structure :

```
<%@ directive {attribut="valeur"} %>
```

JEE

Directives

- Instructions dans des balises **JSP** spéciales
- Structure :
  `<%@ directive {attribut="valeur"} %>`

Rôle

- définir des données relatives à la page (directive page)
- inclure une autre page JSP (directive include)
- inclure des bibliothèques de balise (directive taglib)

Utiliser la directive `page` pour définir des données relatives à la page (code auto-généré à la création d'une JSP)

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

Utiliser la directive `page` pour définir des données relatives à la page (code auto-généré à la création d'une JSP)

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

Utiliser l'attribut `import` pour importer une classe à utiliser dans la page

```
<%@ page import="java.util.Date"  %>
```

Utiliser la directive `page` pour définir des données relatives à la page (code auto-généré à la création d'une JSP)

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

Utiliser l'attribut `import` pour importer une classe à utiliser dans la page

```
<%@ page import="java.util.Date"  %>
```

Autres attributs

- `extends`
- `import`
- `session = "true | false"`
- `isELIgnored = "true | false"`
- ...

Inclure le contenu d'une autre page JSP

```
<%@ include file="maPage.jsp" %>
```

ou

```
<jsp:directive.include file="maPage.jsp" />
```

Inclure le contenu d'une autre page JSP

```
<%@ include file="maPage.jsp" %>
```

ou

```
<jsp:directive.include file="maPage.jsp" />
```

Différence entre les deux solutions

- Avec la première solution, le fichier sera chargé au moment de la compilation (donc le contenu de maPage sera recompilé avec le code de la page appelante)
- Avec la deuxième au moment de l'exécution

Inclure le contenu d'une autre page JSP

```
<%@ include file="maPage.jsp" %>
```

ou

```
<jsp:directive.include file="maPage.jsp" />
```

Différence entre les deux solutions

- Avec la première solution, le fichier sera chargé au moment de la compilation (donc le contenu de maPage sera recompilé avec le code de la page appelante)
- Avec la deuxième au moment de l'exécution

Utilisation

Pour inclure (menu, entête...) qui sont généralement définis dans un fichier spécifique et qui sera inclus dans les autres fichiers de l'application (pour éviter le copier/coller et favoriser la réutilisation).

JEE

Inclure des bibliothèques de balises (à voir dans un prochain chapitre)

```
<%@ taglib uri="maLib" prefix="tag" %>
```

Comme dans les Servlets

```
request.getParameter("nomParameter");
```

JEE

Exemple

```
<%@ page language="java" 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>Projet JEE</title>
  </head>
  <body>
    Hello World (depuis une JSP)
    <%
      String nom = request.getParameter("nom");
      String prenom = request.getParameter("prenom");
      out.println("<br/>Hello " + nom + " " + prenom);
    %>
  </body>
</html>
```

JEE

Transmission de données entre Servlet/JSP

- Et si la Servlet veut transmettre des données (variables, objets...) à la vue ?
- On peut utiliser `request.setAttribute()` pour transmettre et `request.getAttribute()` pour récupérer
  - `request.setAttribute("nomAttribut", "valeur")`
  - `request.getAttribute("nomAttribut")` : récupère l'objet ayant le nom `nomAttribut` qui doit correspondre au nom utilisé lors de l'envoi

Envoi de données par la Servlet

```
protected void doGet(HttpServletRequest request,
   HttpServletResponse response) throws
   ServletException, IOException {

     String ville = "Marseille";
     request.setAttribute("maVille",ville);
     // l'envoi de request se fait après cette
        instruction
     this.getServletContext().getRequestDispatcher("/
        WEB-INF/vue.jsp").forward(request, response);
}
```

JEE

Récupération de données par la JSP

```
<%@ page language="java" 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>Projet JEE</title>
  </head>
  <body>
    <%
      String notreVille = (String) request.getAttribute("
         maVille");
      out.println("Bienvenue à " + notreVille);
    %>
  </body>
</html>
```

Envoi d'un objet

```
protected void doGet(HttpServletRequest request,
   HttpServletResponse response) throws
   ServletException, IOException {
     Personne perso = new Personne();
     perso.setNom("Wick");
     perso.setPrenom("John");
     perso.setNum(100);
     request.setAttribute("perso",perso);
     this.getServletContext().getRequestDispatcher("/
        WEB-INF/vue.jsp").forward(request, response);
}
```

JEE

Récupération de l'objet

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ page import = "org.eclipse.model.*" %>
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset
       =UTF-8">
    <title>Projet JEE</title>
  </head>
  <body>
    <%
        Personne p = (Personne) request.getAttribute("perso");
        out.print("Hello " + p.getPrenom() + " " + p.getNom());
    %>
  </body>
</html>
```

JEE

Exercice

Modifiez `CalculServlet` pour qu'elle réalise le traitement précédent et affiche le résultat dans une vue `calcul.jsp`

JEE

Quatre portées pour les variables

- `page` : la variable est accessible seulement dans cette page
- `request` : la variable est accessible seulement entre la servlet et la vue appelée
- `session` : la variable est accessible dans toutes les pages de l'application pour un utilisateur donné
- `application` : la variable est accessible dans toutes les pages de l'application et est partagée par tous les utilisateurs

JEE

Et si on a besoin de créer un objet dans la page JSP

```
<jsp:useBean id="per" scope="page" class="org.eclipse.model.
   Personne" >
</jsp:useBean>
```

JEE

Et si on a besoin de créer un objet dans la page JSP

```
<jsp:useBean id="per" scope="page" class="org.eclipse.model.
   Personne" >
</jsp:useBean>
```

Explication

- La balise précédente est équivalente en **Java** à `Personne perso = new Personne();`
- Notre objet est accessible seulement dans cette page **JSP** (`scope="page"`)
- Il faut que notre classe `Personne` soit un `JavaBean` : obligatoirement un constructeur sans paramètre

JEE

Et si on a besoin de créer un objet dans la page JSP et affecter des valeurs aux attributs

```
<jsp:useBean id="per" scope="page" class="org.eclipse.model.Personne">
  <jsp:setProperty name="perso" property="nom" value="wick"/>
  <jsp:setProperty name="perso" property="prenom" value="john"/>
</jsp:useBean>
```

JEE

Et si on a besoin de créer un objet dans la page JSP et affecter des valeurs aux attributs

```
<jsp:useBean id="per" scope="page" class="org.eclipse.model.Personne">
  <jsp:setProperty name="perso" property="nom" value="wick"/>
  <jsp:setProperty name="perso" property="prenom" value="john"/>
</jsp:useBean>
```

Ou aussi

```
<jsp:useBean id="per" scope="page" class="org.eclipse.model.Personne">
</jsp:useBean>

<%
  perso.setNom("wick");
  perso.setPrenom("wick");
%>
```

JEE

EL : Expression Language

- Proposée par **JSTL** (Java Standard Tag Library)
- Disponible depuis la version 2.4 de l'API Servlet
- Permettant d'optimiser les pages **JSP** (simplifier le code)
- Forme générale : `${ expression }`

JEE

Rôle

- Réaliser des tests, des opérations arithmétiques
- Manipuler des objets, des collections,
- ...

JEE

Les EL supportent plusieurs types du langage Java

- `Long`
- `Double`
- `String` : entouré par `"..."` ou `'...'`
- `Boolean`
- ...

JEE

Les EL supportent plusieurs types du langage Java

- `Long`
- `Double`
- `String` : entouré par `"..."` ou `'...'`
- `Boolean`
- ...

Les EL permettent d'évaluer une expression arithmétique

```
${ 5 } <!-- affiche 5 -->
${ 5.2 } <!-- affiche 5.2 -->
${ "bonjour" } <!-- affiche bonjour -->
${ 'bonjour' } <!-- affiche bonjour -->
${ true } <!-- affiche true -->
```

JEE

Les EL permettent d'évaluer une expression arithmétique

```
${ 4 * 3 + 5 } <!-- affiche 17 -->
${ 8 % 2 } <!-- affiche 0 -->
```

JEE

Les EL permettent d'évaluer une expression arithmétique

```
${ 4 * 3 + 5 } <!-- affiche 17 -->
${ 8 % 2 } <!-- affiche 0 -->
```


Les opérateurs arithmétiques

- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `/` ou `div` : division
- `%` ou `mod` : reste de la division

JEE

On peut réaliser des tests en utilisant les opérateurs de comparaison

```
${ 'e' < 'f' } <!-- affiche true -->
${ 5 + 5 == 25 } <!-- affiche false -->
```

JEE

On peut réaliser des tests en utilisant les opérateurs de comparaison

```
${ 'e' < 'f' } <!-- affiche true -->
${ 5 + 5 == 25 } <!-- affiche false -->
```

Opérateurs de comparaison

- == ou `eq` : pour tester l'égalité
- != ou `ne` : pour tester l'inégalité
- > ou `gt` : supérieur à
- < ou `lt` : inférieur à
- >= ou `ge` : supérieur ou égal à
- <= ou `le` : inférieur ou égal à

On peut aussi enchaîner les tests en utilisant les opérateurs logiques

```
${ 2 == 5 || 3 == 4 } <!-- affiche false -->
${ 2 < 5 && 5 >= 3 } <!-- affiche true -->
```

JEE

On peut aussi enchaîner les tests en utilisant les opérateurs logiques

```
${ 2 == 5 || 3 == 4 } <!-- affiche false -->
${ 2 < 5 && 5 >= 3 } <!-- affiche true -->
```

Opérateurs logiques

- `&&` ou `and` : et
- `||` ou `or` : ou
- `!` ou `not` : non

JEE

Pour les chaînes de caractères, on peut utiliser l'opérateur `empty`

```
${ empty 'chaine' } <!-- affiche false -->
${ !empty 'chaine' } <!-- affiche true -->
${ !empty 'chaine' ? true : false  } <!-- test
   ternaire affichant true -->
```

JEE

Pour les chaînes de caractères, on peut utiliser l'opérateur `empty`

```
${ empty 'chaine' } <!-- affiche false -->
${ !empty 'chaine' } <!-- affiche true -->
${ !empty 'chaine' ? true : false  } <!-- test
   ternaire affichant true -->
```

JEE

Pour les chaînes de caractères, on peut utiliser l'opérateur `empty`

```
${ empty 'chaine' } <!-- affiche false -->
${ !empty 'chaine' } <!-- affiche true -->
${ !empty 'chaine' ? true : false  } <!-- test
   ternaire affichant true -->
```


Les résultats sont affichés là où l'EL est appelée

```
<div> 7 < 5 : ${ 7 < 5 } </div>
<div> 7 < 5 : false </div>
```

JEE

EL simplifie la récupération des attributs ajoutés depuis la Servlet dans l'objet `request`

```
${ nom } <!-- affiche la valeur de la variable nom définie dans la
   Servlet appelante -->
```

JEE

EL simplifie la récupération des attributs ajoutés depuis la Servlet dans l'objet `request`

```
${ nom } <!-- affiche la valeur de la variable nom définie dans la
   Servlet appelante -->
```

Avec les scriptlets, pour récupérer un objet

```
<%@ page import = "org.eclipse.model.*" %>
<%
  Personne p = (Personne) request.getAttribute("perso");
  out.print("Hello " + p.getPrenom() + " " + p.getNom());
%>
```

JEE

EL simplifie la récupération des attributs ajoutés depuis la Servlet dans l'objet `request`

```
${ nom } <!-- affiche la valeur de la variable nom définie dans la
   Servlet appelante -->
```

Avec les scriptlets, pour récupérer un objet

```
<%@ page import = "org.eclipse.model.*" %>
<%
  Personne p = (Personne) request.getAttribute("perso");
  out.print("Hello " + p.getPrenom() + " " + p.getNom());
%>
```

Avec EL, l'écriture a été simplifiée

```
${ perso.nom } <!-- affiche Wick -->
${ perso.getPrenom() } <!-- affiche John -->
```

JEE

Explication

- `perso` est le nom d'objet qui a été ajouté à la requête comme attribut (avec `request.setAttribute()`
- `${ perso.nom }` est équivalent à `${ perso.getNom() }`

JEE

Explication

- `perso` est le nom d'objet qui a été ajouté à la requête comme attribut (avec `request.setAttribute()`
- `${ perso.nom }` est équivalent à `${ perso.getNom() }`

Même si l'objet ou un de ses attributs n'existe pas, `null` ne sera jamais affiché.

JEE

Considérons la liste suivante définie dans la Servlet

```
ArrayList<String> sport = new ArrayList<String>();
sport.add( "football" );
sport.add( "tennis" );
sport.add( "rugby");
sport.add( "basketball" );
request.setAttribute( "sport" , sport );
```

JEE

Pour récupérer l'élément d'indice `i` dans la vue

```
sport.get(i);
sport[i];
sport['i'];
sport["i"];
```

JEE

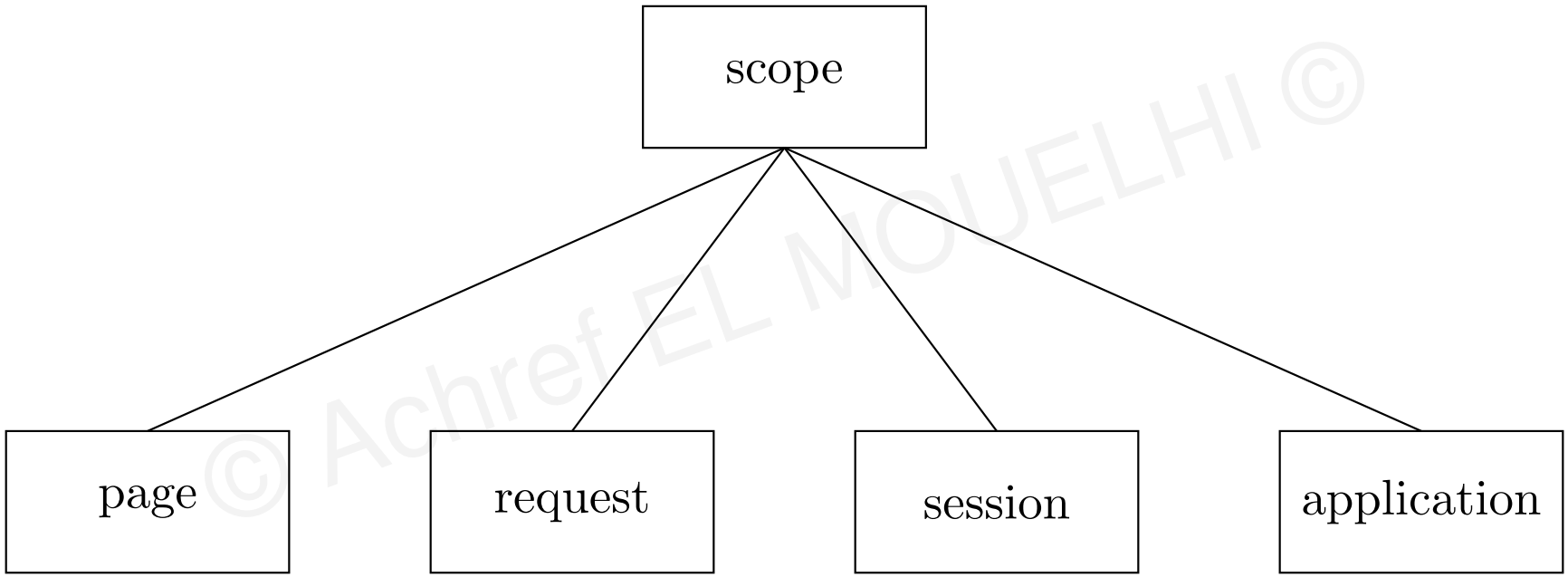
Pour récupérer l'élément d'indice `i` dans la vue

```
sport.get(i);
sport[i];
sport['i'];
sport["i"];
```


Exemple

```
J'aime le ${ sport.get(0) } et le ${ sport[3] }.
Je deteste le ${ sport['1'] } et le ${ sport["3"] }.
```

JEE



JEE

Dans les exemples précédents

- on a utilisé des objets (implicites) sans les instancier.
  - `out` : pour afficher un message
  - `request` : pour récupérer des attributs et/ou des paramètres
- ces objets (et certains autres) ont déjà été instanciés dans la Servlet qui correspond à notre page JSP

JEE

Autres objets implicites

- `session` : permet de récupérer/écrire des données relatives à l'utilisateur courant
- `application` : permet d'obtenir/modifier des informations relatives à l'application dans laquelle elle est exécutée.
- `response` : permet de modifier des données relatives à la réponse (encodage...)
- `exception` : pour récupérer des informations sur l'exception capturée
- `...`

JEE

Les objets implicites de EL sont des `Map`

- `sessionScope` : une `Map` qui permet de récupérer/écrire des données relatives à l'utilisateur courant
- `param` : une `Map` qui permet de récupérer/écrire les noms et valeurs des paramètres de la requête.
- `cookie` : une `Map` qui permet d'associer les noms et instances des cookies.
- ...

JEE

Le code JSP permettant de récupérer les paramètres d'une requête

```
<%
  String nom = request.getParameter("nom");
  String prenom = request.getParameter("prenom");
  out.println("<br/>Hello " + nom + " " + prenom);
%>
```


On peut le remplacer par

```
  Hello ${param.prenom} ${param.nom}
```

JEE

Considérant le code suivant (contenant une division par zéro)

```
<%
    int x = 3 / 0;
%>
```

JEE

Considérant le code suivant (contenant une division par zéro)

```
<%
    int x = 3 / 0;
%>
```

À l'exécution, une exception est affichée

```
org.apache.jasper.JasperException: An exception
   occurred processing JSP page [/WEB-INF/vue.jsp]
   at line [11]

10:      <%
11:              int x = 3 / 0;
12:      %>
```

JEE

Il faut capturer l'exception

```
<%
  try {
    int x = 3 / 0;
  }
  catch(Exception e){
    out.print("Erreur " + e.getMessage());
  }
%>
```

Et le résultat est :

```
Erreur / by zero
```

JEE

Une deuxième solution consiste à

- créer une vue d'erreur
- rediriger vers cette page chaque fois qu'une exception est levée

JEE

La page `erreur.jsp`

```
<%@ page language="java" 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> Page d'erreur </title>
  </head>
  <body>
        Erreur
  </body>
</html>
```

JEE

Faisons référence à `erreur.jsp` dans `vue.jsp` (en ajoutant la ligne `errorPage="erreur.jsp"`) et supprimons le bloc `try ... catch`

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" errorPage="erreur.jsp" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset
       =UTF-8">
    <title>First Page</title>
  </head>
  <body>
    <%
      int x = 3 / 0;
    %>
  </body>
</html>
```

En exécutant, la redirection a eu lieu mais le message d'erreur a disparu

JEE

Pour afficher le message d'erreur, il faut modifier `erreur.jsp` et déclarer la page comme page d'erreur `isErrorPage="true"`

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset
       =UTF-8">
<title>Page d'erreur</title>
</head>
<body>
        Erreur
        <%=exception.getMessage() %>
</body>
</html>
```

Ne pas utiliser le navigateur d'**Eclipse** pour tester.

JEE

Extension **Eclipse** pour **JSP**

- **Emmet** : extension d'auto-complétion pour les langages Web
- Pour l'utiliser, allez dans `Help > Eclipse Marketplace...` et installez l'extension `Emmet (ex-Zen Coding)`
- Pour l'activer, allez dans `Window > Preferences`, cherchez `Emmet` et ajoutez `.jsp` à la liste des extensions supportées.