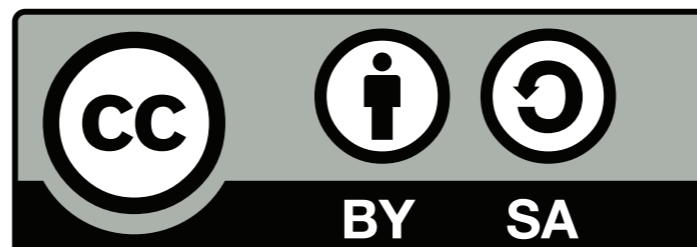


Tecnologia e Applicazioni Internet 2010/11

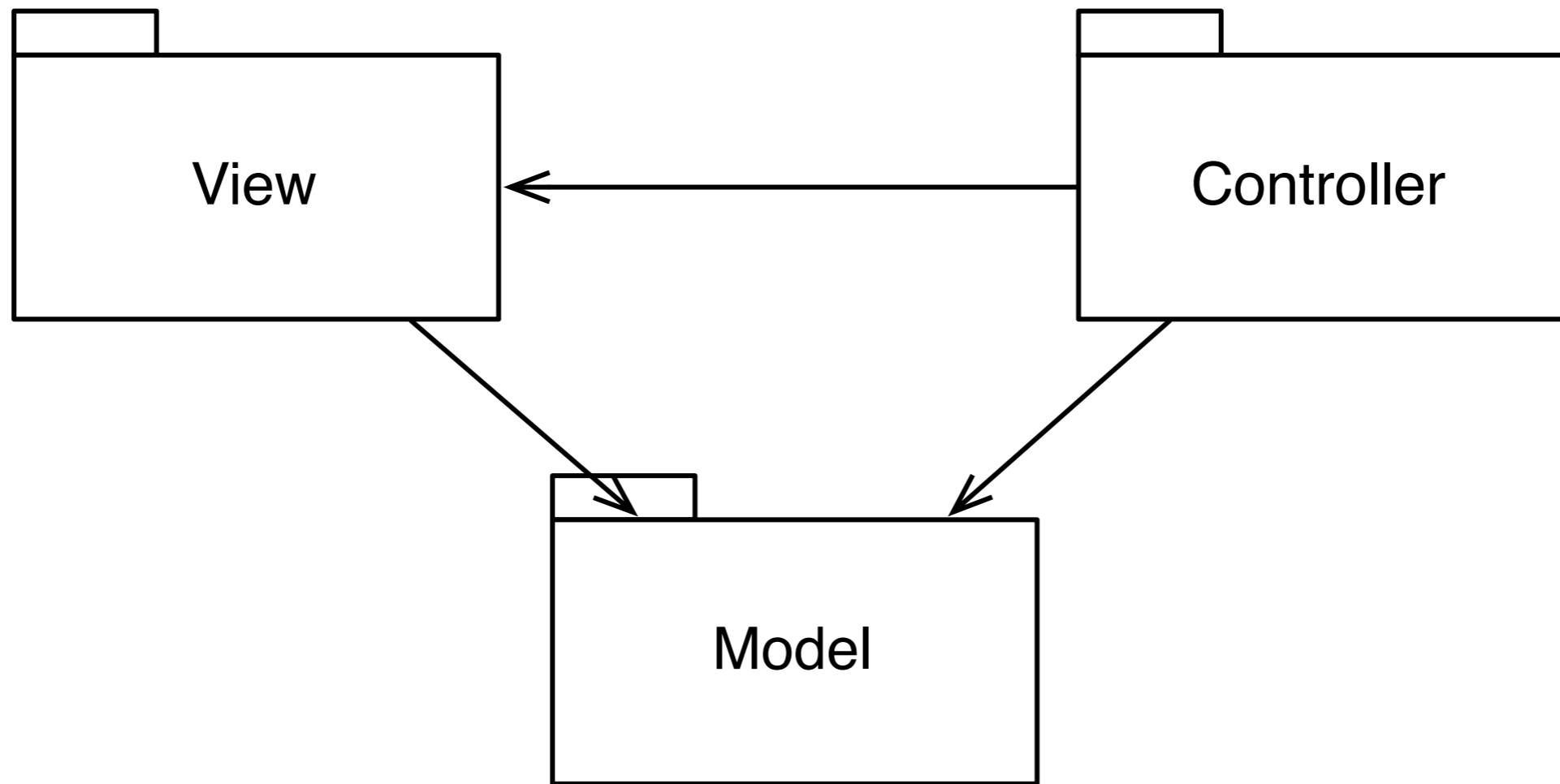
Lezione 3 - View templates and XPath

Matteo Vaccari

<http://matteo.vaccari.name/>
matteo.vaccari@uninsubria.it



Model, view, controller



Mixing controller and view

```
public String doGet(String string) {  
    String heading = "<h1>People Directory</h1>";  
    String tableStart = "<table>";  
    String rows = "";  
    for (Person person : repository.getPeople(0, 10)) {  
        rows += convertPersonToTableRow(person);  
    }  
    String tableEnd = "</table>";  
    return heading + tableStart + rows + tableEnd ;  
}
```

View code in blue, data access in red

Come realizzare la view?

```
public String toHtml() {
    String result = "" +
        "<html>" +
        "<head><title>" + title + "</title></head>" +
        "<body>" +
        "<div id='picture'>" +
        "  <h3>" + title + "</h3>" +
        "  <img src='" + pictureUrl + "' alt='some picture' />" +
        "  <p>" + caption + "</p>" +
        "</div>" +
        "</body>" +
        "</html>";
    return result ;
}
```

Come realizzare la view?

- Templates
 - ◆ Freemarker, Velocity, StringTemplate...
 - ◆ Java Server Pages (JSP)
- Builder
 - ◆ Pattern poco usato in Java ma utile

Realizzare le view con FreeMarker

Template + data-model = output

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome ${user}!</h1>
  <p>Our latest product:
  <a href="${latestProduct.url}">
    ${latestProduct.name}</a>!
</body>
</html>
```

+

```
(root)
|
+- user = "Big Joe"
|
+- latestProduct
  |
  +- url = "products/greenmouse.html"
  |
  +- name = "green mouse"
```

=

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome Big Joe!</h1>
  <p>Our latest product:
  <a href="products/greenmouse.html">
    green mouse</a>!
</body>
</html>
```

Examples from the Freemarker manual - <http://freemarker.sourceforge.net/>

The data model

- Is a simple map names to values
- The values can be any standard Java Maps, Lists, Arrays, or beans


```

(root)
|
+- animals
| |
| | +- (1st)
| | |
| | | +- name = "mouse"
| | | |
| | | +- size = "small"
| | | |
| | | +- price = 50
| | |
| | +- (2nd)
| | |
| | | +- name = "elephant"
| | | |
| | | +- size = "large"
| | | |
| | | +- price = 5000
| | |
| | +- (3rd)
| | |
| | | +- name = "python"
| | | |
| | | +- size = "medium"
| | | |
| | | +- price = 4999
| |
+- whatnot
|
+- fruits
|
+- (1st) = "orange"
|
+- (2nd) = "banana"

```

```

class Animal {
    public String getName() {...}
    public String getSize() {...}
    public int getPrice() {...}
}

```

```
List other = Arrays.asList("orange", "banana");
```

```

Animal[] myAnimals = new Animal [] {
    new Animal("mouse", "small", 50),
    new Animal("elephant", "large", 5000),
    new Animal("python", "medium", 4999),
};

```

```

Map modelRoot = new HashMap();
modelRoot.put("animals", myAnimals);
modelRoot.put("whatnot", other);

```

- The data model is a tree
- Scalars: numbers, strings, dates, boolean
- Non-scalars: arrays, lists, maps, any Java class with getters

Templates directives

```
<#if animals.python.price < animals.elephant.price>  
  Pythons are cheaper than elephants today.  
<#else>  
  Pythons are not cheaper than elephants today.  
</#if>
```

```
<p>We have these animals: <#include "/copyright_footer.html">  
<table border=1>  
  <tr><th>Name</th><th>Price</th></tr>  
  <#list animals as animal>  
  <tr><td>${animal.name}<td>&euro; ${animal.price}</td></tr>  
  </#list>  
</table>
```

```
<p>We have these animals:  
<table border=1>  
  <tr><th>Name</th><th>Price</th>  
  <tr><td>mouse<td>&euro; 50</td></tr>  
  <tr><td>elephant<td>&euro; 5000</td></tr>  
  <tr><td>python<td>&euro; 4999</td></tr>  
</table>
```

Checking for null

```
<h1>Welcome ${user!"Anonymous"}!</h1>
```

Print value of “user” if not null, otherwise “Anonymous”

```
<#if user??><h1>Welcome ${user}!</h1></#if>
```

Print message only if value of “user” is not null

Programming Freemarker

```
// configure where templates are loaded from
Configuration configuration = new Configuration();
configuration.setDirectoryForTemplateLoading(
    new File("/where/you/store/templates"));

// load the template from file
// /where/you/store/templates/test.ftl
Template template = configuration.getTemplate("test.ftl");

// create the root model
Map root = new HashMap();
root.put("user", "Big Joe");

// merge data and template
Writer out = new OutputStreamWriter(System.out);
template.process(root, out);
out.flush();
```

Testing views

- Voglio testare la *logica della view* e solo quella
- Voglio passare informazioni al template e vedere che le renderizzi senza eccezioni
- *Non voglio* dover lanciare un web server per eseguire i test

Simple test

```
@Test
public void testFreemarker() throws Exception {
    String templateString = "Hello, ${user}";
    Template template = new Template("my template",
        new StringReader(templateString), new Configuration());

    Map model = new HashMap();
    model.put("user", "Tizius");

    StringWriter writer = new StringWriter();
    template.process(model, writer);
    assertEquals("Hello, Tizius", writer.toString());
}
```

Not so simple test

```
@Test
public void bigTest() throws Exception {
    Template template = configuration.getTemplate("bigjoe.ftl");

    Map model = new HashMap();
    model.put("user", "Big Joe");

    StringWriter writer = new StringWriter();
    template.process(model, writer);

    String expected = "<html>\n" +
        "<head>\n" +
        "  <title>Welcome!</title>\n" +
        "</head>\n" +
        "<body>\n" +
        "  <h1>Welcome Big Joe!</h1>\n" +
        "  <p>Our latest product:\n" +
        "  <a href=\"products/greenmouse.html\">\n" +
        "green mouse</a>!\n" +
        "</body>\n" +
        "</html>\n";
    assertEquals(expected, writer.toString());
}
```

Testare l'html come stringa non conviene

- Test fragile: basta aggiungere uno spazio e si rompe
- Test rigido: modifiche non significative (es. cambio il titolo della pagina) rompono il test

Testare la view con *XPath*

```
@Test
public void betterTest() throws Exception {
    ...
    Document document = xmlDocumentFromString(writer.toString());
    assertEquals("Welcome!", getNodeContent(document, "/html/head/title"));
    assertEquals("green mouse",
        getNodeContent(document, "//a[@href='greenmouse.html']"));
}
```

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome Big Joe!</h1>
  <p>Our latest product:
    <a href="greenmouse.html">green mouse</a>
  </p>
</body>
</html>
```

Test con XPath

```
@Test
public void betterTest() throws Exception {
    ...
    Document document = xmlDocumentFromString(writer.toString());
    assertEquals("Welcome!", getNodeContent(document, "/html/head/title"));
    assertEquals("green mouse",
        getNodeContent(document, "//a[@href='greenmouse.html']"));
}
```

`"//a[@href='x']"`

In XPath significa “un elemento *a* con *href*='x'”

Introduzione a XPath

Estratto da una presentazione di Roberto Liverani

<http://www.slideshare.net/robertosl81/xpath-injection-3547860>

An XML document from XPath perspective (1/2)

■ XPath Nodes:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<users>
```

```
<user>
```




```
<username id="1">root</username>
```

```
<password>OAhgg</password>
```

```
<account>root</account>
```

```
</user>
```

```
</users>
```

	document node
	element node
	attribute node
	atomic value / item



An XML document from Xpath perspective (2/2)

Relationships of Nodes:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username ="1">root</username>
    <password>OAhhgg</password>
    <account>root</account>
  </user>
</users>
```

Relationships:

<user> is the **parent** node of <username> , <password> , <account>
<username> , <password> , <account> are **children** nodes of the element <user>
<username> , <password> , <account> are all **siblings** (they have the same parent)
<users> and <user> are **ancestors** of <username>, <password>, <account>
<username>, <password>, <account> are **descendants** of the element <users>

XPath Syntax (1/3)

- XPath uses path expressions to select nodes or node-sets in an XML document.
- Path expressions is very similar to URI syntax and file path syntax.
- Selecting Nodes:

Expression	Description
nodename	Selects all child nodes of the named node
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node

XPath Syntax (2/3)

■ Example:

XPath Query: `/users/user/username`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>root</username>
    <password>0Ahhgg</password>
    <account>root</account>
  </user>
  <user>
    <username>admin</username>
    <password>9salkl</password>
    <account>admin</account>
  </user>
</users>
```

XPath Syntax – other query examples (3/3)

XPath Query: `/users/user/username`

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<users>
  <user>
    <username>root</username>
    <password>OAhhg</password>
    <account>root</account>
  </user>
  <user>
    <username>admin</username>
    <password>9salk</password>
    <account>admin</account>
  </user>
</users>
```

Expression	Result
<code>users</code>	Selects all the child nodes of the users element
<code>/users</code>	Selects the root element users
<code>users/user</code>	Selects all user elements that are children of users
<code>//users</code>	Selects all users elements no matter where they are in the document
<code>users//user</code>	Selects all user elements that are descendant of the users element, no matter where they are under the users element

XPath Predicates

- Predicates are used to find a specific node or a node that contains a specific value. Predicates can use XPath operators.
- Predicates are always embedded in square brackets.

Expression	Result
<code>/users/user[1]</code>	Selects the first user element that is the child of the users element.
<code>/users/user[last()]</code>	Selects the last user element that is the child of the users element
<code>/users/user[position()<3]</code>	Selects the first two user elements that are children of the users element
<code>//username[@id='1']</code>	Selects all the username elements that have an attribute named id with a value of '1'

XPath operators are shown in red.

XPath Location Path (1/2)

- Location path is a special case of XPath Expression.
- Two types: absolute and relative location path
 - Absolute Location Path starts with a (forward) slash
 - Relative Location Path starts without a slash
- In both cases the location path consists of one or more **steps**, each separated by a slash. Example: Absolute Location Path: /users/user/username
- A step is composed by:
 - an axis (defines the tree-relationship between the selected nodes and the current node)
 - a node-test (identifies a node within an axis)
 - zero or more predicates (to further refine the selected node-set)
- The syntax for a location step is: axisname::nodetest[predicate]
- There are several axisname that can be used. Most common are: ancestor, attribute, descendant, child

XPath Location Path – Examples (2/2)

Example	Result
<u>child</u> ::user	Selects all user nodes that are children of the current node
<u>attribute</u> ::id	Selects the id attribute of the current node
<u>child</u> ::*	Selects all children of the current node
<u>attribute</u> ::*	Selects all attributes of the current node
<u>child</u> :: text()	Selects all text child nodes of the current node
<u>child</u> :: node()	Selects all child nodes of the current node
<u>descendant</u> ::users	Selects all users descendants of the current node

XPath Wildcards are bolded in red.
XPath Axisname are underlined.

XPath Functions

- Functions specified for XSLT and Xquery can also be used for XPath.
- Functions are related to strings, boolean, date/time, error and trace, numeric, node, sequence, QName, anyURI, context.
- Short list of the most important functions:

Function Name	Description
<code>substring(string,start,len)</code>	Returns the substring from the start position to the specified length. Index of the first character is 1. If length is omitted it returns the substring from the start position to the end
<code>string-length(string)</code>	Returns the length of the specified string.
<code>count((item,item,...))</code>	Returns the count of nodes
<code>starts-with(string1,string2)</code>	Returns true if string1 starts with string2, otherwise it returns false
<code>contains(string1,string2)</code>	Returns true if string1 contains string2, otherwise it returns false
<code>number(arg)</code>	Returns the numeric value of the argument. The argument could be a boolean, string, or node-set
<code>string(arg)</code>	Returns the string value of the argument. The argument could be a number, boolean, or node-set

Perché non JSP?

- Perché è difficile scrivere test *unitari*
- Occorre un compilatore Java
- Si può fare (Lasse Koskela ha scritto una libreria) ma vale la pena solo per progetti *legacy*

Builder style

```
html do
  head do
    title 'Big products!'
    stylesheet_link_tag 'scaffold'
  end

  body do
    h1 "Big products!", :style => "color: green"
    p "first paragraph"
  end
end
```

```
new Html().add(
  new Head().add(
    new Title("My Title!"),
    new StylesheetLinkTag("scaffold")
  ),
  new Body().add(
    new H1("Big Products!"),
    new Paragraph("first paragraph").with("style", "color: green")
  )
).build();
```

Come funziona?

```
public interface HtmlBuilder {  
    public HtmlBuilder add(HtmlBuilder ... children);  
    public String build();  
}
```

```
public class Title implements HtmlBuilder {  
  
    private final String content;  
  
    public Title(String content) {  
        this.content = content;  
    }  
  
    @Override  
    public String build() {  
        return "<title>" + content + "</title>";  
    }  
}
```

Come funziona?

```
public interface HtmlBuilder {  
    public HtmlBuilder add(HtmlBuilder ... children);  
    public String build();  
}
```

```
public class Head implements HtmlBuilder {  
    private List<HtmlBuilder> children = new ArrayList<HtmlBuilder>();
```

```
    public HtmlBuilder add(HtmlBuilder ... moreChildren) {  
        this.children.addAll(Arrays.asList(moreChildren));  
        return this;  
    }
```

```
    private String processChildren() {  
        String result = "";  
        for (HtmlBuilder child : children) {  
            result += child.build();  
        }  
        return result;  
    }
```

```
}
```


Vantaggi del *builder*

- L'html è valido *per costruzione*
- Posso generare diversi linguaggi (html 4.0, xhtml 1.0,...) cambiando un opzione