weave: control flow

What is weave doing when a HTTP Request comes in?

What is weave?

weave is a Framework written in Java to ease the development of web-based applications. In a way it is something like Ruby on rails and similar Frameworks, i.e. it makes several implications on how to write a web-based applications. This document describes the control flow of weave, that is, what happens when weave handles an incoming request.

Terms

The INTERNET is a network consisting of several networks (interconnected networks), i.e. it is some kind of infrastructure. Certain SERVICES are built on top of this infrastructure, backed by PROTOCOLS which describe how the infrastructure provided by the internet should be utilized to provide the desired functionality. Some of these services are email, the WORLD WIDE WEB and instant messaging. The protocol used by the Web is HTTP[1], the Hypertext Transfer Protocol. While other protocols, like Jabber or Oscar (the protocol used by the services ICQ and AIM), imply that a connection is created between client and server (the author assumes you are familiar with the Client/Server architecture) HTTP does not. Basically requesting a certain resource works as follows:

- Open a TCP connection
- Send a REQUEST
- Receive the RESPONSE
- Close the TCP connection

Note: Newer versions of the HTTP protocol utilize a feature called "keep-alive"[2] to reduce the overhead which occurs by opening a connection again and again (for example when loading images contained in a hypertext document from the same server). However, it does not change the way HTTP requests are processed, since no guarantees are made about a keep-alive connection and there is no session management.

Since no permanent connection is established, creating a SESSION is rather difficult. How do you know that two request origin from the same user? (comparing IP-addresses is rather dumb, since IP-addresses may change during a session and an IP-address can be shared by multiple users).

What is weave doing for me?

weave handles requests and responses. It provides you with session management, including login and logout procedures. In short: it provides an environment in which you only have to care about business logic, not about technical stuff and "how the web works". In fact, using weave you could drop weave, implement weaves API and port your application to a different service (like a standalone GUI application). However, weave is for the web, and we'll look at some of its features by tracing a common situation: A HTTP Request.

Furthermore weave provides utilities for user management (privileges), templating (in an MVC fashion) and database access (using a utility called *meta*).

When a HTTP request comes in

Applications written using weave feature an MVC-like architecture (*Model, View, Controller*). weave is about the C in MVC, while your application provides the Model and the View. So, when a HTTP request comes in it is first handled by weave. Based on the request is determined what logic should be executed, so if your application is running at `localhost/helloworld` the request

`localhost/helloworld/wom/bat`

would issue the weave controller to execute the action “bat” of the module “wom”. Modules are Java-Classes which contain business logic, they are provided by your application (in fact: they **are** your application).

Some special requests are handled by weave directly, notably those which are prefixed by a tilde (~):

`localhost/helloworld/~login`

Let’s have a look at the module “wom”:

```
@ModuleInfo(name="wom")
public class MyFirstModule extends de.fu.weave.impl.GenericModule<MyORManager> {

	@Action(template = "wom.xsl")
	public void _default(String[] path) { }

	@Action(name = "bat", template="wom.bat.xsl")
	public void myFirstAction(
		String[] path,
		@Arg(name = "op1", intDefault = 0) int firstOperand,
		@Arg(name = "op2”, intDefault = 0) int secondOperand
	) {
		put("sum", (firstOperand + secondOperand) * 2);
	}
}
```

It is a Java-class extending `de.fu.weave.impl.GenericModule<M>` (weave only requires you to implement `de.fu.weave.Module<M>`, but `GenericModule` already features some useful methods for creating weave modules). In there are two methods:

- `_default(String[] path)`
- `myFirstAction(String[] path, int firstOperand, int secondOperand).`

Virtually everything inside this class is annotated. Due to the information contained in these annotations the weave controller knows what to do. In the example of calling `localhost/helloworld/wom/bat` the request would issue the weave controller to execute `myFirstAction` and prepare the template `wom.bat.xsl`. Since the request contained no arguments, weave will automatically fill `intDefault` into `firstOperand` and `secondOperand`, i.e. 0 in this example.

`myFirstAction` on the other hand uses `Module.put(String, int)` to bind the result of the expression `(firstOperand + secondOperand) * 2` to to the template variable `sum`.

After the execution of `myFirstAction` the weave controller will take over again, i.e. it will produce a nicely formatted web-page based on the rules found in `wom.bat.xsl` and send it back as response to the client who made the request.

A closer look

The default implementations of `weave.Module` and `weave.Controller` used in this example are `weave.impl.GenericModule` and `weave.impl.GenericController`. They provide a template mechanism based on XSL-T[3]. In a default setup the XSL-Transformation rules are to be found inside the directory `xsl`, which is residing in the root-directory of your web application. So the template `wom.bat.xsl` is to be found there.

When a modules action is requested the controller will instantiate a new object of this modules class. The module will be informed about the current session (which includes information about the user being logged in or not) and setup an XML-DOM[4]-Object.

The controller then invokes the method associated with this action. This method manipulates the DOM and returns, thereby handling control over to the weave controller again. While you are generally free to modify the DOM to your liking (it is *your* template that has do deal with it) the predefined `put(...)`-Methods create a pre-defined structure. The document created by `wom/bat` may look like this:

```
<n0:meta xmlns:n0="http://technodrom.scravy.de/2010/data"
         xmlns:n1="http://technodrom.scravy.de/2010/item"
         xmlns:n2="http://technodrom.scravy.de/2010/request"
         action="bat" context-path="/" language="en" logged-in="no"
         module="wom" module-path="/helloworld/wom/bat"
         servlet-path="/helloworld" style="default.css" type="xsl">
  <n0:sum>0</n0:sum>
  <n0:user>
    <n1:user>de.fu.weave.AnonymousUser@6d62f58e</n1:user>
  </n0:user>
  <n2:req method="get" />
</n0:meta>
```

The weave Controller will transform this into the actual data being sent as response using the template which is specified by the requested action, `wom.bat.xsl` in this example. Given `wom.bat.xsl` as

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/1999/xhtml" xmlns:vars="http://technodrom.scravy.de/
2010/data" xmlns:item="http://technodrom.scravy.de/2010/item">

  <xsl:output omit-xml-declaration="yes" />
  <xsl:template match="/meta:meta">

  <html>
    <head>
      <title>Hello, world!</title>
    </head>
    <body>
      <h1>wombat sum</h1>
      <p>The wombat sum is <xsl:value-of select="vars:sum" />.</p>
    </body>
  </html>

  </xsl:template>
</xsl:transform>
```

the response might look something like this:

```
<html>
  <head><title>Hello, world!</title></head>
  <body>
    <h1>wombat sum</h1>
    <p>The wombat sum is 0.</p>
  </body>
</html>
```

References

1 **HTTP 1.1**
http://www.w3.org/Protocols/rfc2616/rfc2616.html

2 **HTTP 1.1 – Connections**
http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html

3 **XSL-T: eXtensible Style Language – Transformations**
http://www.w3.org/TR/xslt

4 **DOM: Document Object Model**
http://www.w3.org/DOM/
W3C DOM in Java: http://download.oracle.com/javase/6/docs/api/org/w3c/dom/package-summary.html