weave: annotations

A description of the Annotations provided by weave

This is a description of the annotations found in the package de.fu.weave and in its sub packages. If you're not familiar with annotations you should have a glance at the Java Annotations Tutorial[1] . However, here a quickstart:

Annotations are like javadoc-tags, but they are parsed by the compiler and can be defined like classes (they are types, like class or enum). Java knows three types of retention policies for annotations, that is, where annotations are available: *runtime*, *class* and *source*. The first means that the annotated information will be available at runtime, i.e. it is maintained by the JVM and can be obtained using Reflection[2]. It implies the second.

The second means that the annotated information will be available in the compiled class file (which is useful if you want to add static information to your binaries, like a signature or something, which should be parsed prior to executing the code). It implies the third, which is the default.

Source means, that the annotated information is neither available at runtime nor in the binary class file, i.e. it is only available in source. However, `javac` as well as `javadoc`, can be extended using the Annotation Processing Tool[3] so that it statically validates the special meaning of annotations. For example you could create an annotation for annotating annotations (@Documented[4] is an example for such an annotation) which states what parameters a method annotated with the annotated annotation must have. A compile error could be thrown if the annotated annotation is misused:

```
@Target(ElementType.ANNOTATION_TYPE)
public @interface TargetMethod {
      Class<?>[] value;
}
```

A sample annotated annotation could look like this:

```
@Target(ElementType.METHOD)
@TargetMethod({int.class, int.class})
public @interface SampleAnnotation {
      ...
}
```

And some sample annotated method like this:

```
...
      @SampleAnnotation
      public void one(int a, int b) { ... }

      @SampleAnnotation
      public void two(String a, int b) { ... }
...
```

The first is correctly annotated, the second not. The `apt` stated above could issue a compile error here.

Base annotations

@de.fu.weave.annotation.ModuleInfo

Status: implemented

Retention policy: runtime

Denotes a class as a module of a weave controller. Modules contain business logic, they are the model part of the MVC architecture in weave. A module consists of several actions and MUST have a **name**. A module MUST feature a default-action named _default, annotated with @Action and no name for that action (which will make it the default-action).

A module MAY impose certain requirements using **requires**. Possible requirements at the time of this writing are `DATABASE` and `LOGGED_IN`. Allowed values are those from the enum `de.fu.weave.annotation.Requirement`. The actual logic is implemented in the controller. Currently `DATABASE` makes the controller expose a Database connection via a `de.fu.weave.orm.ConnectionManager` to the module (if you forget to mention this requirement your module will not be able to talk to the database). Another possible requirement is `LOGGED_IN`, which will make the controller refuse to grant access to anonymous users for the annotated module.

Other values are purely informational and do export traditional meta-data only, like **author**, **description**, **version** and **license**.

@de.fu.weave.annotation.Action

Status: implemented

Retention policy: runtime

Denotes a method as action which is accessible via web. A method which is annotated via Action must have the first parameter of the type String[]. Information about the path being called is inserted at this point, e.g. `localhost/context/servlet/module/action/some/special` will result in `{"some", "special"}` being passed as first parameter (assuming that your action is named "action" in the module "module").

The value **name** is the name exposed to the web (i.e. `localhost/context/servlet/module/exposed-action-name`). If no name is given the method is assumed to be the default action and must have the name `_default`. The default action is the action being called if the module is called with no action given (i.e. `localhost/context/servlet/module`).

The optional value **requiresPrivilege** denotes, that a certain privilege is needed to request the action. weave automatically checks whether the current user does have the required privilege or not and denies access to the action if. The checking is done prior to invoking the action, i.e. not a single bit of the method is executed if the user lacks the required privilege.

template is the only required value. If specified the name of the template which is to be used by weave for rendering the resulting DOM.

If an exception occurs in an Action it is handled by weave. Currently this means that weave wraps the exception in to a template called “error.xsl” and displays the result, discarding any results of the requested Action. Using **handles** one or more exceptions MAY be specified which are natively handled by the action. If one of these exceptions is thrown by the annotated method they will be passed to the template.

@de.fu.weave.annotation.Arg

Status: implemented

Retention policy: runtime

Denotes a parameter for a method which is annotated via @ACTION to be an argument which can be passed in via HTTP (either in the query string using `GET` or in a `POST` request). The weave controller will automatically check if an argument of the annotated **name** is given in the current request and cast it to the value of the parameter. If such an argument does not exist the default value as specified by `<type>`**Default** will be used. Currently **booleanDefault**, **intDefault**, **longDefault**, **floatDefault**, **doubleDefault**, and **stringDefault** are supported.

isTarget is used in conjunction with **requiresPrivilege** and it MUST NOT be used without. If given the controller will automatically check for the required privilege for the target taken form the argument which is annotated via isTarget.

@de.fu.weave.annotation.Commit

Status: lacks implementation (ticket #30[5])

Retention policy: runtime

(still to come)

XML Annotations

Example:

```
@XmlElement("sample-object")
class XmlSerializableObject {

      @XmlAttribute("sample-value")
      public String getSampleValue() { ... }

      @XmlCollection("sample-sequence")
      public java.util.Collection<AnotherObject> getMoreObjects() { ... }
}

@XmlElement("another-sample”)
class AnotherObject {

      @XmlAttribute("some-property")
      public int getSomeProperty() { ... }

      @XmlAttribute(“just-another-value”)
      public double getAnother() { ... }
}
```

@de.fu.weave.xml.XmlAttribute

Status: implemented

Retention policy: runtime

Denotes a method as a getter whose value should be used as an attribute with the given name (via **value**) when serializing the object as XML.

@de.fu.weave.xml.XmlCollection

Status: implemented

Retention policy: runtime

Denotes a method as a getter which returns multiple values which should be serialized as a sequence of elements of the given name (via **value**) when serializing the object as XML.

@de.fu.weave.xml.XmlElement

Status: not implemented

Retention policy: runtime

Denotes the name (via **value**) which should be used as local name for the element if this object is serialized as an element when serializing it as XML.

ORM Annotations

The annotations defined in `de.fu.weave.orm.annotation` are purely informational, i.e. they are created by meta when auto generating the DAOs. They should suffice to regenerate a Relations.xml from a given set of DAO-Classes.

@de.fu.weave.orm.annotation.Entity

Status: informational

Retention policy: class

Denotes an Entity.

@de.fu.weave.orm.annotation.Relationship

Status: informational

Retention policy: class

Denotes a Relationship.

Future thoughts

An annotation processor for the annotation processing tool (`apt`) which statically checks at compile time the requirements (marked using SMALLCAPS) imposed within this document can be built[6].

XML annotations can be used to automatically create XML Schema files (using apt or reflection).

ORM annotations can be used to automatically reconstruct a Relations.xml from existing auto generated code.

Since IPC relies on XML, XML annotations may better be suited for serializing objects.

Using javalx

I today (night 2010-10-18 to 2010-10-19) implemented a Lexer which is able to tokenize Java- as well as PHP-Code (I guess it will also be able to cope with various other languages, like JavaScript or Google Go). It is written in Haskell and my be compiled using the Makefile in trunk/meta, it outputs XML. The only available input method is from stdin, in other words: pipe the output of cat.

The resulting XML can be processed using XSL-T (in fact, writing your own parser, java compiler or whatever *is* possible). That way the functionality mentioned above can be implemented, too.

1 **The Java Tutorials – Annotations**
http://download.oracle.com/javase/tutorial/java/javaOO/annotations.html

2 **Package java.lang.reflect (Java Platform SE6)**
http://download.oracle.com/javase/6/docs/api/java/lang/reflect/package-summary.html

3 **Getting Started with the Annotation Processing Tool (apt)**
http://download.oracle.com/javase/1.5.0/docs/guide/apt/GettingStarted.html

4 **Annotation Type Documented (java.lang.annotation)**
http://download.oracle.com/javase/6/docs/api/java/lang/annotation/Documented.html

5 **Ticket #30 – Add Commit-Functionality**
https://dev.spline.de/trac/score-myCourses/ticket/30

6 **Ticket #33 – Create Annotation Processor for weave**
https://dev.spline.de/trac/score-myCourses/ticket/33