Control where HTML files are loaded from

Table of contents

- In Wicket 1.1
- In Wicket 1.2
- In Wicket 1.3
- In Wicket 1.4
- Simple Partitioning in Maven Projects

In Wicket 1.1

Disclaimer: I'm a Wicket Newbie, and this has only been tested on Wicket 1.1 B2. YMMV.

I needed a way to load up HTML files that are all inside the same directory. That directory happened to be /WEB-INF/html. Essentially, I needed to chop off the fully qualified class name and turn `com/example/wicket/app/HomePage.html` into simply `/WEB-INF/html/HomePage.html`.

Wicket allows you to define multiple directories to look for HTML files, but they all assume the HTML file is in a directory structure mirroring your classes.

To generate a new strategy for turning FQCN HTML files into simple names, you have to override the `getResourceStreamLocator()` method of the `Application` class.

```
public ResourceStreamLocator getResourceStreamLocator() {
        final ResourceStreamLocator defaultLocator = super.getResourceStreamLocator();

        return new ResourceStreamLocator(new AbstractResourceStreamLocator() {

            public IResourceStream locate(final String path, final String style, final Locale locale,
                    final String extension) {
                IResourceStream stream = super.locate(path, style, locale, extension);
                if (stream == null) {
                    return defaultLocator.locate(path, style, locale, extension);
                }
                return stream;
            }

            protected IResourceStream locate(String path) {
                String fullPath = "/WEB-INF/html"+path.substring(
                        path.lastIndexOf('/'), path.length());
                try {
                    final URL url = getWicketServlet().getServletContext()
                        .getResource(fullPath);

                    if (url != null) {
                        return new UrlResourceStream(url);
                    }

                } catch(MalformedURLException e) { }

                return null;
            }
        });
    }
```

Of course, you can change the path prefix, `/WEB-INF/html` to whatever you like.

There may be a better way to do this. If not, I suggest the ResourceStreamLocator implement the Chain pattern, so that in the future I could simple change the method to this:

```
public ResourceStreamLocator getResourceStreamLocator() {
        ResourceStreamLocator defaultLocator = super.getResourceStreamLocator();
        ResourceStreamLocator myCustomLocator = new MyCustomResourceLocator();
        myCustomLocator.setNextInChain(defaultLocator);

        return myCustomLocator;
    }
```

But of course, I'm combining two different actions here. The first is converting a class name to its HTML file representation. The second action is I'm locating that file in some set of locations.

Both strategies should be pluggable, thus reducing the complexity.

Again, maybe this is already implemented somewhere?

Hope this all helps,
Seth

In Wicket 1.2

Firstly, define the new IResourceStreamLocator class:

```
public class WebPageResourceStreamLocator extends AbstractResourceStreamLocator {
    private IResourceFinder finder;

    public WebPageResourceStreamLocator(IResourceFinder finder) {
        this.finder = finder;
    }

    protected IResourceStream locate(final Class clazz, final String path) {
        final URL file = finder.find(trimFolders(path));
        if (file != null) {
            return new UrlResourceStream(file);
        }
        return null;
    }

    private String trimFolders(String path) {
        return path.substring(path.lastIndexOf("/") + 1);
    }
}
```

Secondly, override Application.init() method to add the new IResourceStreamLocator:

```
public class CizelgemApplication extends AuthDataApplication {
    @Override
    protected void init() {
        super.init();
        CompoundResourceStreamLocator locator =
            (CompoundResourceStreamLocator)getResourceSettings().getResourceStreamLocator();
        WebApplicationPath resourceFinder = (WebApplicationPath) getResourceSettings().getResourceFinder();
        resourceFinder.add("src/main/webapp"); //this path should be changed

        locator.add(0, new WebPageResourceStreamLocator(resourceFinder));
    }
```

Mert Nuhoglu

In Wicket 1.3

1.2 instructions updated for 1.3:

Firstly, define the new IResourceStreamLocator class:

```
public class PathStripperLocator extends ResourceStreamLocator {

    public PathStripperLocator() {
    }

    public IResourceStream locate(final Class clazz, final String path) {
        IResourceStream located = super.locate(clazz, trimFolders(path));
        if (located != null) {
            return located;
        }
        return super.locate(clazz, path);
    }

    private String trimFolders(String path) {
        return path.substring(path.lastIndexOf("/") + 1);
    }
}
```

Secondly, override Application.init() method to add the new IResourceStreamLocator:

```
public class CizelgemApplication extends AuthDataApplication {
    @Override
    protected void init() {
        super.init();
        IResourceSettings resourceSettings = getResourceSettings();
        resourceSettings.addResourceFolder("src/main/webapp"); //this path should be changed
        resourceSettings.setResourceStreamLocator(new PathStripperLocator());
    }
```

David Rosenstrauch

In Wicket 1.4

Two lines in your Wicket init are enough so you can put the HTML in your Maven `webapp` directory:

```
        IResourceSettings resourceSettings = getResourceSettings();
        resourceSettings.addResourceFolder("");
```

This works because the default IResourceSettings looks on the servlet context root if the path does not start with a "/". This resource resolution happens **in addition** to the classpath resolution, so you don't lose any functionality this way.

Brian Topping

Simple Partitioning in Maven Projects

With a simple change in your Maven pom.xml file, you can get clean separation of the Java and HTML source directories. (The Java class and HTML markup are still in the same package, though, unless you also use one of the strategies mentioned above.

```
<project>
[...]
 <build>
   <resources>
     <resource>
       <directory>src/main/resources</directory>
     </resource>
     <resource>
       <directory>src/main/html</directory>
     </resource>
   </resources>
[...]
 </build>
[...]
</project>
```

Then, put all the HTML template files in:

```
<your-project>/src/main/html
```

Sualeh Fatehi