GeoffLongmanSandbox

NOTE: This is outdated information that applies only to Tapestry 4.

I think there is a bug in T4 that is going to rear it's ugly head at some point. (and since wiki's are great for formatting things I'm going to lay out my case here and then point a JIRA issue to this page. Although JIRA will be the permanent record so I guess this will be an expanded discussion on the issue).

Depending on how a page is referenced in a link, Tapestry may or may not find the right page class. And, once it chooses the wrong page class, the app will probably break and stay broken until the servlet container is restarted.

If a page spec explicity specifies a class there is no problem, the problem arises when Tapestry goes searching for the class

In fact it's worse than this. There could be the case where an app had two links, one in a form (not an HTML form!) that would cause Tapestry to find the correct class and one that would not. When this is possible then the stability of the application is suspect because all depends on whether the first user clicked the right link. If the first click by the first user was on the bad link, the whole app is broken until the servlet container is restarted.

Currently the ComponentClassProvider uses a Hivemind chain to search for the page class. Passed into this chain is an instance of ComponentClassProviderContext which contains:

- the page name
- the INamespace for the page
- the page specification

One member of the chain, NamespaceClassSearchComponentClassProvider, gets a list of packages, say `com.myfoo`, by checking a property in the Namespace and then checks each package for a class with a fully qualified name of the package with the page name appended. If the NamespaceClassSearchComponentClassProvider does not locate a class it falls to other members of the chain and in most cases the resulting class would be `org.apache.tapestry.html.BasePage`.

The problem arises because the page name may be a path like

```
pages/MyPage
```

In the case above there is no problem, Tapestry would look for and probably find the class `com.myfoo.pages.MyPage`. However, Tapestry uses the same page name to locate the page spec (if one exists) and looks in various places like:

```
/
/WEB-INF
/WEB-INF/servletname
```

and there are several valid ways to reference a page in these places and each way would result in Tapestry looking for the page class in a different place. So there is the distinct possibility that Tapestry will not find the page class intended by the developer and fall back to the BasePage. Hence the app might break or not depending on who clicked what first.

Example

the following appears in test.application

```
  <meta key="org.apache.tapestry.page-class-packages" value="com.myfoo"/>
  <page name="Page" specification-path="/WEB-INF/pages/MyPage.page"/>
```

and of course the files are located at

```
/WEB-INF/test.application
/WEB-INF/pages/MyPage.page
```

Sidenote: If a page is specless things can go wrong too. But, that's another story

The class I've created for this page is `com.myfoo.pages.MyPage`.

Immediately I see 3 ways to reference MyPage that will result in Tapestry sucessfully locating the specification and then go off looking for the page class:

```
 <span jwcid="@PageLink" page="Page"/>
 <span jwcid="@PageLink" page="pages/MyPage"/>
 <span jwcid="@PageLink" page="/WEB-INF/pages/Mypage"/>
```

Some of them links might not be pretty, but all are valid. We know that our class is `com.myfoo.pages.MyPage` and that the name used by NamespaceClassSearchComponentClassProvider searched for will be, for each link, respectively:

```
com.myfoo.Page
com.myfoo.pages.MyPage
com.myfoo.WEB-INF.pages.MyPage
```

HowardLewisShip: The third option is something I've never considered and would consider banning outright. Also, I believe that page "Page" and page "pages/MyPage" are *distinct* pages, because they have distinct page names ("Page" vs. "pages/MyPage"). The expectation is that page "Page" will be class "com.myfoo.Page" and page "pages/MyPage" will be class "com.myfoo.pages.MyPage".

In all cases but the second, the page class Tapestry is searching for is not found and usually BasePage ends up being used. And if `com.myfoo.pages.MyPage` has some properties that are refenced by an OGNL expression in the template, the page will break.

I think this problem also, in some cases, affects template lookup, but I have not delved into that.

The Fix

It seems to me that the way this can be fixed so that Tapestry always looks for and finds `com.myfoo.pages.MyPage` is to do a bit more processing on the name passed in before handing things off to the PageClassProvider. In every case above, Tapestry sucessfully locates the specification `/WEB-INF/pages/MyPage.page`. I think the fix would be to examine the path part of the page name and lop off the leading parts that would cause the class lookup to fail.

In particular I think that the locations Tapestry uses to find undeclared pages is a good place to start.
*Compare with the location of the namespace specification, if that is a prefix of the page name path, lop it off and STOP.
*Compare with `/WEB-INF/servletname` , if that is a prefix, lop it off and STOP.
*Compare with `/WEB-INF`, if that is a prefix, lop it off and STOP.
*If the above didn't result in a lopping, then the path is good.

HowardLewisShip: Still digesting this.

There would appear to be a special case. That is

```
<span jwcid="@PageLink" page="Page"/>
```

But it is not a special case, the lopping happens after the specification has been resolved and thus the declared name does not come into the picture.

```
And I think that is a good thing. The declared name is much more likely to change than the name of the
specification.
```

but what about Specless pages? There is no spec! Well, really there is a spec. It's just that Tapestry create a stand-in spec and that spec is given a location that can be used in the lopping too. Interestingly, the name given to the stand-in spec is the name in the path, so if one were to reference a specless page thus:

```
<span jwcid="@PageLink" page="pages/MySpeclessPage"/>
```

and Tapestry would go looking for `com.myfoo.pages.MySpeclessPage`, which is what I think users would expect.

There is a catch to that though, the location given to the stand-in spec is relative to the location of the namespace. In the example above our specless page's stand-in would have a location of

```
/WEB-INF/pages/MySpeclessPage.page
```

Which looks good, if we take the lopping appraoch, the stand-in's location will work fine too.

Implementing the Fix

I'm about 1/2 way through the fix. Since the lopping described above so closely parallels the logic in PageSpecificationResolverImpl and since it it true that once a page class has been found for a page there is no way to change it short of restarting the container, I think the whole kit and kaboodle should be moved out of PageLoader and into PageSpecificationResolverImpl.

Out of time for today, will pick this up again tomorrow night

last note. I have an example application that displays the broken behaviour I have described.

http://static.flickr.com/29/54487210_e699a93095.jpg