Bookmarkable pages and links

Bookmarkable vs non-bookmarkable pages

Introduction

Pages can be constructed with any constructor when they are being used in a Wicket session, but if you wish to link to a Page using a URL that is "bookmarkable" (which implies that the URL will not have any session information encoded in it, and that you can call this page directly without having a session first directly from your browser), you need to implement your Page with a no-arg constructor or with a constructor that accepts a PageParameters argument (which wraps any query string parameters for a request). In case the page has both constructors, the constructor with PageParameters will be used.

If you have a Page with a default constructor and not a PageParameter constructor then it will default to one without any warning or error. Because wicket doesn't check if there are really request parameters for it and wicket can't know if they are really meant for this page or not.

Non-bookmarkable pages have default constructors and constructors with the `PageParameters` argument hidden (protected/private), or none at all. It could have any other constructor, like `MyPage(FooClass bar)` etc. The net effect of your page being non-bookmarkable is that there is no way a user can directly access your page. So, by making a page non-bookmarkable, you make it a 'safe' page.

A recommended approach is that bookmarkable pages do nothing in their default behaviour (construction time) that can harm you (like deleting records etc), and that everything you want to hide, should be either put in protected pages, or in handlers such as links and forms on the pages. Note that those handlers are protected too, as they can never be called directly without having created the page instance first.

Examples

Linking to pages can be done in multiple ways:

First one is by just using the Link class directly:

```
    new Link()
    {
       public void onClick()
       {
          setResponsePage(new MyPage(...));
       }
    }
```

This way you instantiate the page and set it as response page when the link is clicked on.

You can also use `PageLink` class, in three ways:

```
public PageLink(final String id, final Class c)
```

This constructor just makes a new instance of that Page when the link is clicked by its default constructor, which it must have.

Second is with a Page directly in the constructor so that you have to make your place when you create the link:

```
public PageLink(final String id, final Page page)
```

You could also use the constructor with the IPageLink interface so that you can lazy initialize the page:

```
public PageLink(final String id, final IPageLink pageLink)
```

And a IPageLink implementation could be something like this:

```
new IPageLink()
{
    public Page getPage()
    {
        return new MyPage(argument1,argument2);
    }

    public Class getPageIdentity()
    {
        return MyPage.class;
    }
};
```

So you control the page creation and which constructor is called.

The `linksTo` of a `Link` just returns false and for a `PageLink` it just checks for the page class. But if your page has state then you should implement that method yourself and check if the state matches:

For a page that has state the defaults `linksTo(Page page)` of a `PageLink` doesn't fit you should override this method and use an implementation that checks the state of that page:

```
Link customize = new Link("customize")
{
  public void onClick()
  {
    setResponsePage(new CustomizeProduct(imageID));
  }

  public boolean linksTo(Page page)
  {
    if (!(page instanceof CustomizeProduct))
      return false;
    CustomizeProduct typedPage = (CustomizeProduct) page;
    return typedPage.getImageID()==imageID();
  }
};
```

Lastly, if your page has a default constructor or if you must push state to it that can be mapped into strings, then you could also use the `BookmarkablePageLink` with `PageParameters`. This results in pushing a bit more state to the client, and the URL's looks a bit different.