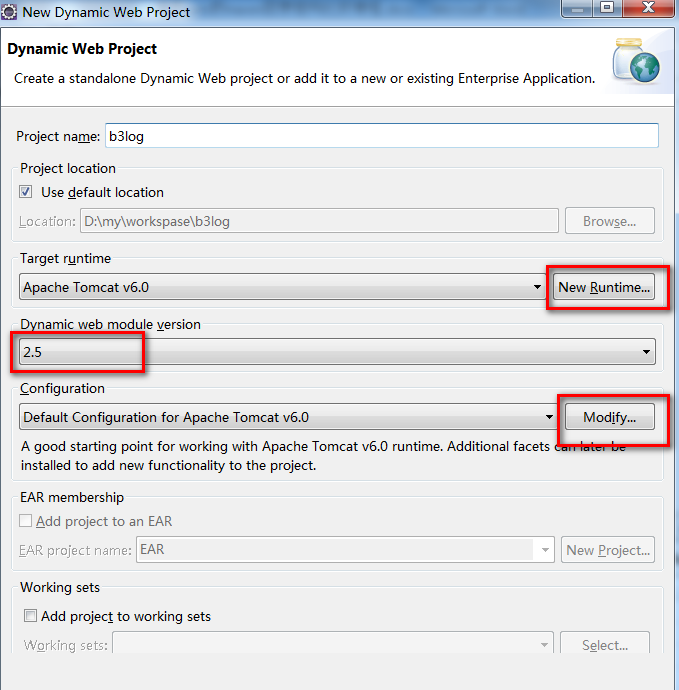
由于solo的maven目录结构比较奇怪，无法直接呈现出标准的web目录结果，使得直接热部署开发有点困难，导致很多时间浪费在重启上。虽然我本人觉得热部署不重启服务器的开发方式并非是正确的开发流程（我们需要用完整单元测试进行底层代码的校验而并非是直接开游览器看结果），但是现实中过多的注意单元测试会极大的拖累开发效率，既然存在即合理，所以大家还是需要这样的方式去开发，在这篇文档的进行简单的说明。  
  
开发环境：eclipse和插件

eclipse版本：示例使用eclipse jee(eclipse-jee-indigo-SR2-win32-x86\_64)版，主要用到wtp的server模块, 你也可以装个eclipse classic版本，安装下wtp模块，因为eclipse classic版本自带git和maven插件。

插件：eclipse的git插件Egit和maven插件m2e，Help-->Marketpalce可以搜到。

准备  
  
eclipse下Dynamic Web Project的说明：

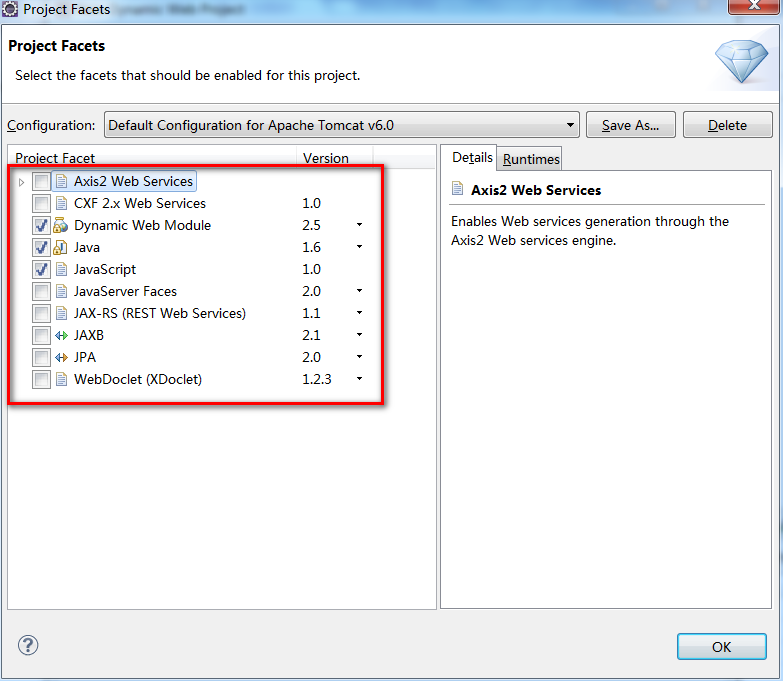
新建Dynamic Web Project工程：



1 配置你自己需要的服务器，按照new Runtime的对话框新建

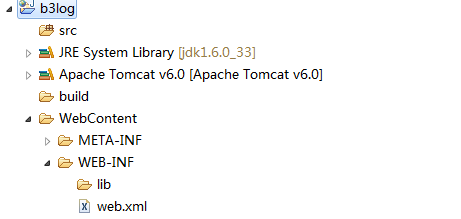
2 web module version 应该就是servlet的版本

3 关于这个web module的相关体系模块配置，点击Modify：

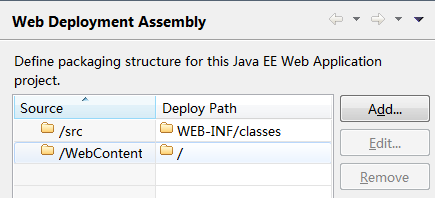


这里有一些eclipse提供的常用模块的插件选择，一般默认即可。

最后生成出来的项目结果如下：



点击项目属性里的Web Deployment Assembly：



这块的配置非常的重要：它配置了我们的项目资源发布到标准web项目的位置（映射）。

在标准的web容器下，

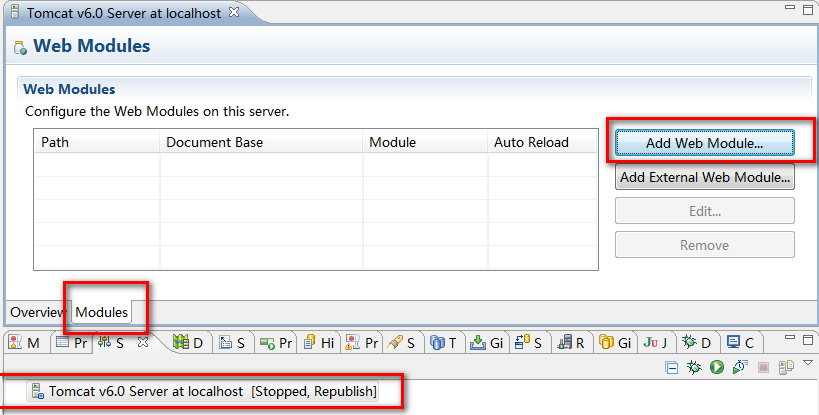
项目的根目录： 我们工程的web资源的根目录（我们这里是WebContent目录）映 射到项目的根目录。

项目的class文件： 我们这里的源码包是’src’,于是我们生成的classes文件会映射到WEB-INF/classes 目录下，eclipse插件 对这块产出的class可以进行热替换（在不破坏class类结构的前提下，我理解为不是修改类结构，方法结构之类的修改）。

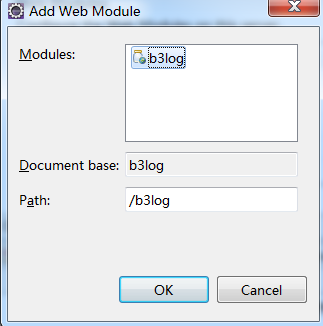
项目的lib文件，这里没有列出，我们可以指定相关的lib，它会被映射到WEB-INF/lib目录中。

以上知识其实是非常基础的J2ee web项目的搭建过程，对于工程目录和项目目录不理解为什么会这样的同学请仔细再去学习下J2ee项目，鉴于篇幅，会省去很多基础的东西。

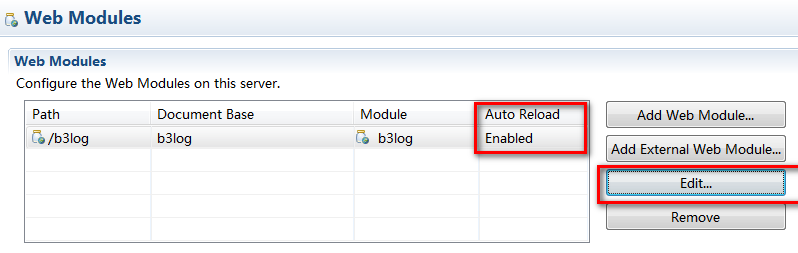
Winodw-->open view中打开Servers视图：



把刚才我们的项目添加进Modules里



然后切记把Auto Reload改成false，点击Edit去改。



Server的一些其他配置在OverView里面，个人建议需要修改的地方

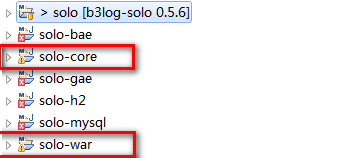
1 Sever Locations 选择第二个，完全控制Tomcat。

2 TimeOuts 里面的start 的时间写长一点，有时候需要断定调试启动。

右键可以选择start，debug等，默认资源会自动publish过去。

## **solo的目录说明：**

从git上clone solo代码，然后import Existing Maven Projects，选择拉下来的solo代码位 置的根目录即可，team-->share project里关联下git ， 然后你会看到以下的代码结构：



虽然会产生的项目很多，但我们主要关心solo-core和solo-war。

Solo项目 ： 可以无视，elipse已经根据子pom建立了相关的子项目映射，不要去动它。但你可以选择在它这里进行git控制。

Solo-core： solo的后台逻辑代码

Sole-war：sole的web资源代码

四个子项目，每个子项目有不同的依赖，至少目前没有包含代码。

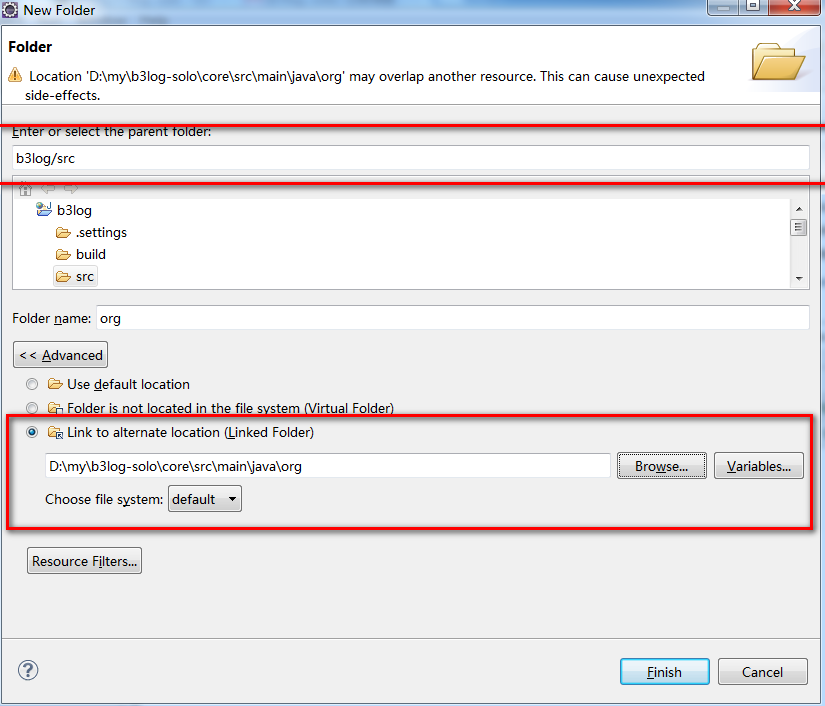
注意：1. eclipse四个子项目前会有红叉，无视即可，没搞清楚怎么消掉。

2. 可以在solo项目上进行git控制，那些多余的eclipse配置文件还没完全忽略，有时候会出现莫名其妙的>，可以同步看下是哪些多余的文件，我会尽快消除哪些可以忽略的文件。

# 整合

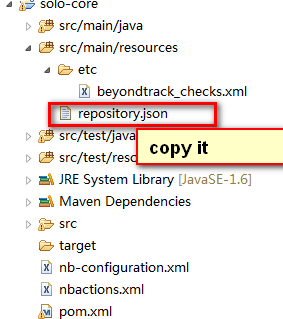
最后，我们将solo整合到前面所建立的Dynamic Web Project工程中！（激动吧）

1. solo后台代码关联，在前面的Dynamic Web Project b3log的src中，建立link文件夹，关联solo-core的代码。

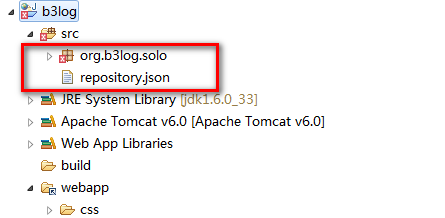


eclipse这种link应该是硬链接，改动文件所有映射过的都会生效。

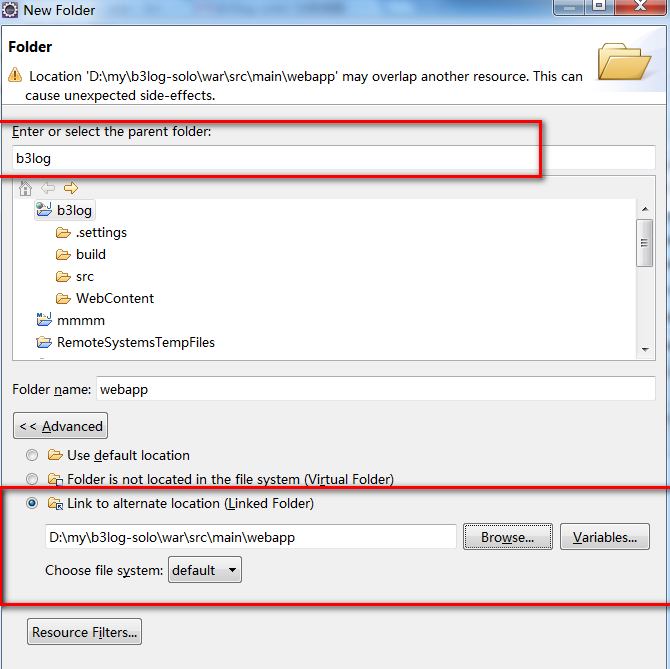
注意到solo-core资源文件资源文件夹下还有一个repository.json，方便起见直接把它复制到b3log下的src里（当然你也可以直接映射solo-core的resource文件夹，然后做成源码包）。



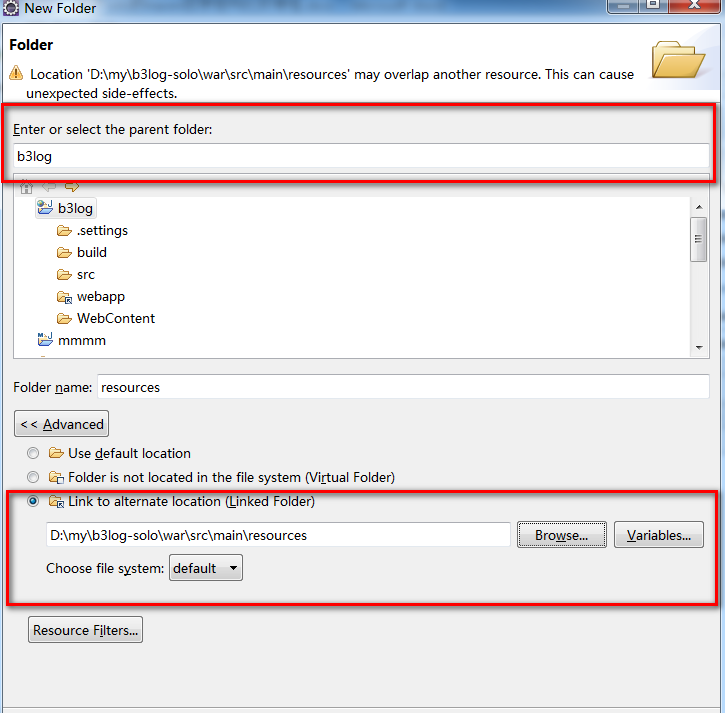
然后在你的b3log下会看到solo-core的代码和它的资源。



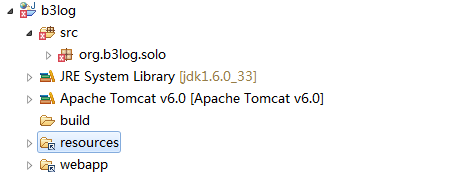
1. sole的web资源代码，在solo下的webapp里，我们仍旧在b3log项目下进行文件夹映射：



这样可以把原来的WebContent那个目录给删了，注意到solo-war的webapp同级还有一个resource目录，那个目录也需要一起link过来。

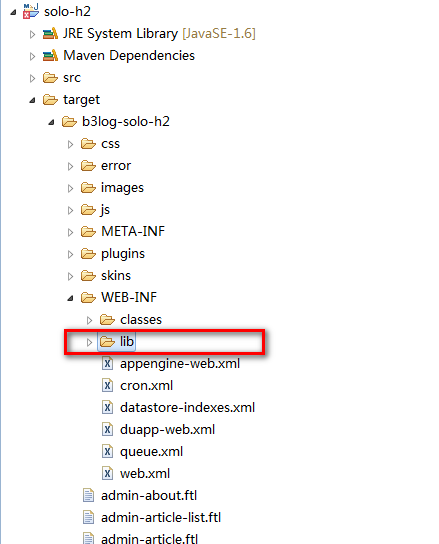


于是，你看到b3log目录，如下：

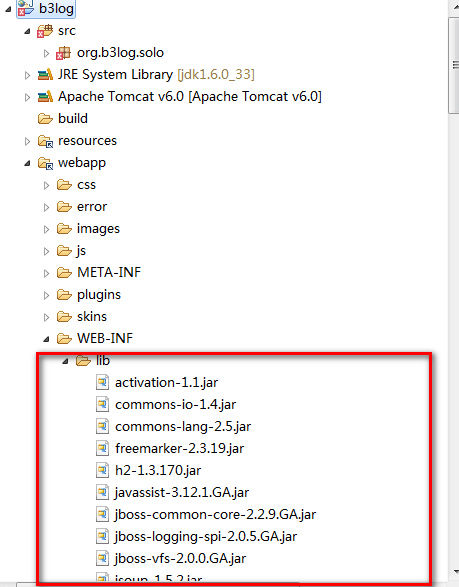


1. 有了src和wabapp，那么最后需要的就是lib了，关联lib这一步比较不自动化，也没有想出来比较自动化的方法。

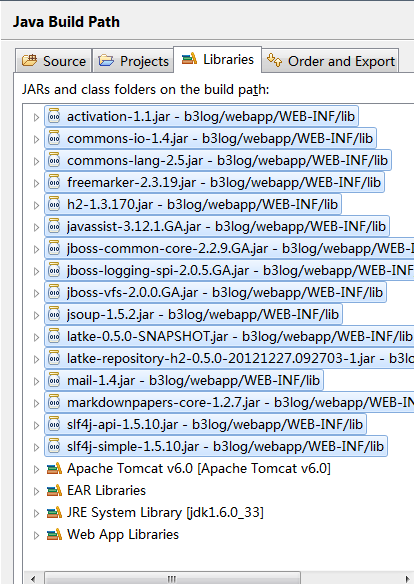
首先，在想要你开发的本地版本的项目名上，进行install，比如我要在本地开发H2版本，那我先点击solo-h2，右键run as—install，然后经过漫长的等待，去solo-h2找到



这个lib中的依赖就是你这个h2项目所依赖所有lib，把这个lib的下所有jar（**除了solo-coreXX.jar,我们已经在src导入了它的代码**），全部复制到b3log的wabapp的WEB-INF下的lib（新建下lib文件夹）

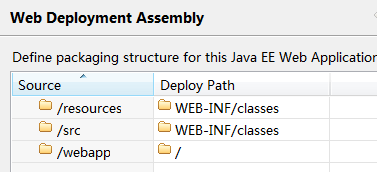


在b3log的项目属性里，把这些jar添加到build path里。



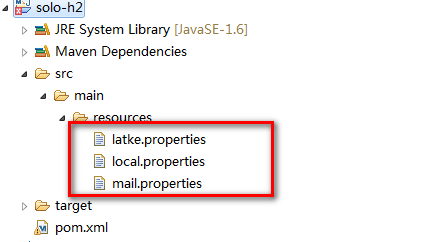
然后你会发现项目的红叉叉都没了。

1. 然后，调整下我们的发布映射，还是在项目属性的Web Deployment Assembly里配置：

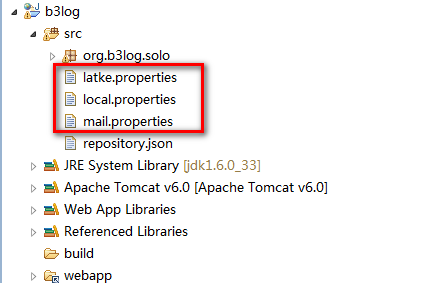


然后你会发现在b3log项目目录里resources隐藏了。

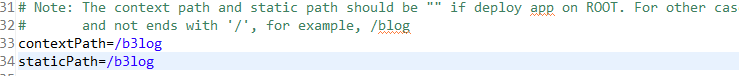
5 最后 就是项目的用户配置文件了，把你需要本地开发的那个版本的项目配置文件复制一份到b3log的src下，比如solo-h2的三个用户配置文件在：

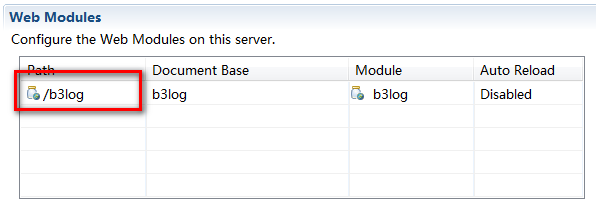


这三个文件复制一份到b3log的src下去：



然后根据你本地开发的实际情况，修改用户配置文件，数据库配置，路径配置，特别注意latke.properties中的contextPath和taticPath 需要和您的Server的发布路径一样！！





整个过程就算是完成了！ 当你在Server视图里启动容器后，修改src下的java文件（非结构性的大改动）和webapp下的文件，插件会自动进行热部署，不用重启容器可以提高开发效率。同时这个结构又保持eclipse的git支持，可以直接在solo项目里进行git的提交和同步。

这文档只是提供一个思路，方便大家进行开发，只要大家明白这样的做法的思想，如果你要修改其他的项目，比如latke，比如想用gae的eclipse plugin去开发solo，都可以按照这样的想法完成！也希望可以提供给大家一个思路，对大家的日常的开发工作也起到一定的帮助。

By mainlove 2013.1.17