[**用Eclipse+Maven+Jetty构建Java Web开发环境(综合几篇教程整合2014年版)**](http://www.itnose.net/detail/6077288.html)

2014-07-23 12:53

工作需要用到Jetty作为web容器，得知Eclipse+Maven+Jetty的组合很不错，因此到网上查找了很多教程，但是或多或少写的不详细或者有过时的内容在里面导致最后配置失败，为了以后同事配置方便，写下这篇配置教程，顺便来方便一下大家。

|  |
| --- |
| 用到的参考资料如下：  **【用Maven构建Java Web开发环境（Jetty容器）】（以此教程为主）http://sarin.iteye.com/blog/784275 以及 http://sarin.iteye.com/blog/784697**  【在Eclipse中编写servlet时出现"The import javax.servlet cannot be resolved" 问题解决办法】http://blog.163.com/y\_love\_qq/blog/static/27637235201211141422583/ 【使用maven加载项目中缺少的jar包】http://blog.csdn.net/daven\_java/article/details/8775809  【Stackoverflow- maven jetty - org.mortbay.jetty vs org.eclipse.jetty】http://stackoverflow.com/questions/15386461/maven-jetty-org-mortbay-jetty-vs-org-eclipse-jetty |

在此首先感谢这些作者的心血，把这些结合在一起我才终于配置成功。

下面开始正题。

=======================软件环境=======================

『系统』Windows 7 x64

『JAVA』JDK 1.7.0\_65

『Eclipse』 Eclipse for JavaEE

『Maven』 Apache Maven 3.2.2

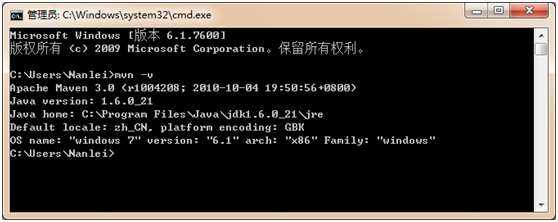
『Jetty』 jetty-distribution-8.1.15

=======================配置过程=======================

**一、JAVA和Maven的配置（如果都配置好了而且清楚maven的用法可以直接跳过看第二部分）**

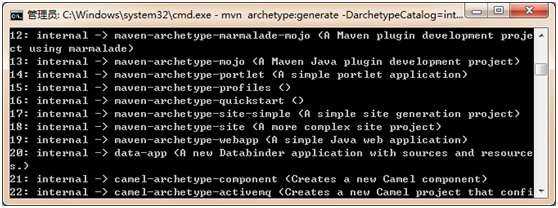
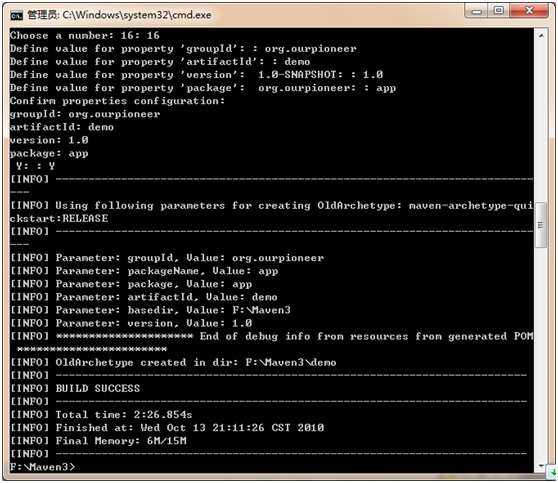
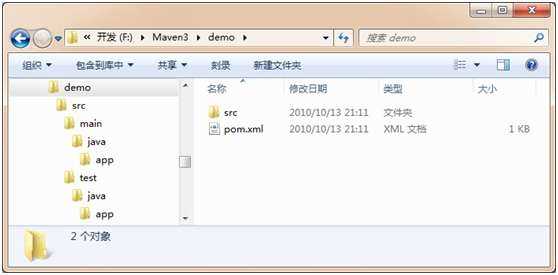
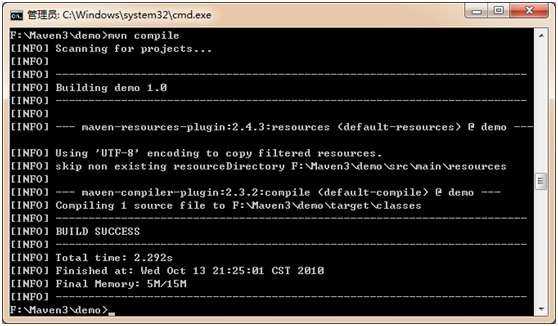
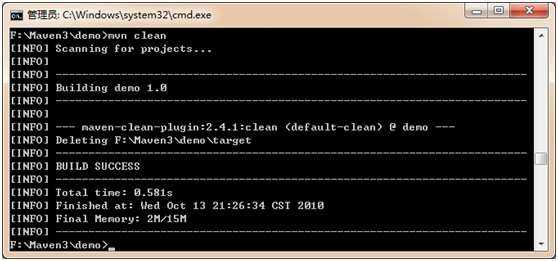
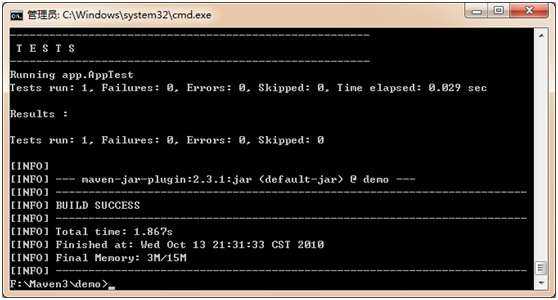
这部分我就直接使用参考资料中的教程了，因为写得比较详细也没有错误。

（源地址：http://sarin.iteye.com/blog/784275）

    Maven是构建Java应用的优秀管理工具，是Apache软件基金会的顶级项目。其具有开发目录构建和管理，统一管理第三方依赖，编译，测试，打包，发布等功能。最大的特点要数第三方依赖管理了，因为其它功能都能从Ant中找到踪影。以往每个项目独立管理依赖，使得每个项目打包后都是巨大无比的，而真正的程序文件却没有多少，Maven接管第三方依赖后，我们的应用就只有程序文件了，没有了jar包轻巧很多，而统一的管理，使得我们可以随意获取jar资源。   
    初学者使用Maven总是感觉很难上手，其中也有一部分人是被Maven下载资源那疯狂的做法吓退的，这就是Maven的精巧之处。从Apache官方网站下载的Maven工具仅仅2M大小，其中是不包含我们要使用的依赖的，那么就要从网络中获取这些依赖，就是那个疯狂下载的过程。下面我们从头开始构建Maven环境，直到运行Jetty容器执行Web应用，这一切都很简单。   
    下载Maven的开发包之后，解压到一个固定的文件夹，建议不要使用汉字并在磁盘的根目录下，这是比较好的实践。那么本文以Maven 3为例来说明，Maven的位置是F:\Maven 3目录。解压后就要添加系统环境变量，这里要说明，因为Maven也是Java应用，所以JDK就必须要提前安装并设置环境遍历。将JAVA\_HOME和M3\_HOME单独设置，防止其使用时不能用Path变量下发现所需路径。下面是JAVA\_HOME和M3\_HOME的配置位置，这个因人而异，但是最好固定不要随意修改。   
   
   
    在Path中，这样来添加即可，非常简单。   
   
    至此Maven已经安装完成了，下面我们来验证安装，在CMD中输入mvn ?v   
   
    可以看到，已经识别mvn命令了。做点简单的设置，就是第三方依赖下载的时候的存放位置，默认是在C盘用户目录的当前用户文件夹下（如C:\用户\Admin，这里是Windows 7的命名方式），在这里Maven会创建.m2文件夹，这里面就会存放下载的类库。当然我不想这些类库放在C盘下，不是特别安全，我想让它们存放在Maven安装目录下，那么就可以这样进行：在Maven安装目录的conf下打开settings.xml，这是对Maven进行设置的XML文件，这里我们简单设置第三方依赖的位置，如下设置：

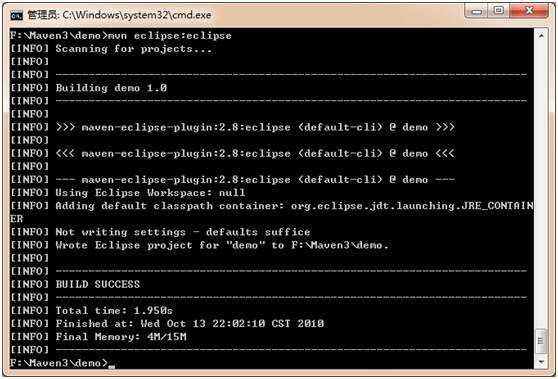
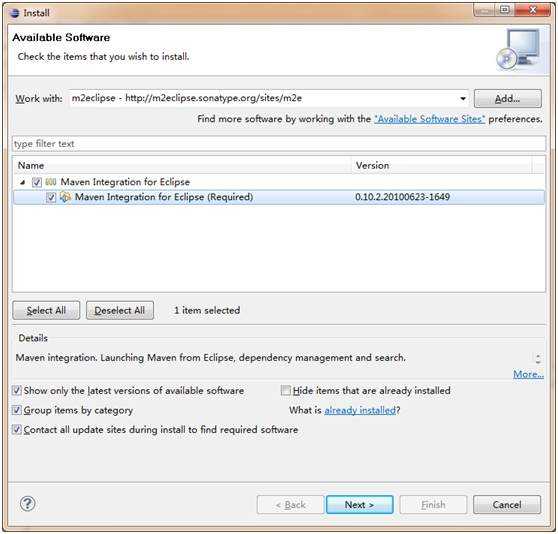
Xml代码   收藏代码

1. <localRepository>F:/Maven3/repository</localRepository>

    这样，那些疯狂下载的依赖就在这个位置下了，要留给足够大的空间，因为依赖下载很可能占用很大的一块空间。为什么？下载后去看看它的真实面目就知道了。    
    下面我们就来创建应用目录，使用mvn archetype:generate命令一步一步根据提示，选择项目类型，添加groupId和artifactId就可以创建项目了，我们来看一下，这里执行mvn archetype:generate -DarchetypeCatalog=internal命令，要手工设置一下项目类型分类属性，可能因为类型太多导致屏幕显示不全，那么我们就简单来看几个常用的就行，就是internal类型。执行该命令后，在控制台我们看到有如下的内容。（切记这个操作位置就是生成项目目录的位置）    
     
    这里我们先选择16是maven-archetype-quickstart，就是基本的JavaSE项目结构的构建，19是maven-archetype-webapp，用来创建简单的Java Web应用，我们先选择16来看看。    
     
    选择16之后，Maven提示输出groupId，就是我们的项目命名组成之一，没什么过多解释的东西，artifactId用来标记项目名称，它们两个是联合使用的。而package定义的是包名，我们在文件夹内就发现了demo命名的项目文件夹。看看这个目录的结构，这是Maven为我们创建的，就是一个标准的格式：    
     
    根目录下有一个pom.xml，这就是项目的Maven配置文件，暂时不介绍其强大的功能，仅仅简单看一下实际应用，这是基础的目录结构，Maven为我们创建了一个Hello World的小程序，我们来运行它。想要运行这个程序，那么就要编译，mvn的编译命令是mvn compile，这样就为我们编译整个项目，这个操作要在demo目录下完成。    
     
    回头看看demo文件夹，多出一个target目录，这就是我们的类路径了，class文件就在下面。这个操作的逆过程是mvn clean，顾名思义，就是清理，清理生成的可执行部分，仅留下源码部分，如下：    
     
    Maven执行了Deleting操作，删除了target目录。如果想执行程序，是需要class文件或jar文件的，前面的mvn compile命令仅仅是编译，而没有对应用进行打包发布，而mvn package命令可以将应用打包（jar或war等），那么我们就来为项目打包发布：    
     
    这个过程，Maven执行了很多操作，有日志可以细看，其中还有测试过程，这是JUnit的测试。其实我们创建项目时，Maven为我们引入的第一个依赖就是JUnit。下面我们来执行jar文件，运行程序。（当然执行class是一样的效果）    
     
    这样就得到执行效果了，Hello World!

**二、Eclipse+Maven+Jetty配置过程**

   这一部分同样以sarin博主的教程为主，其中添加、修改了部分不详细的或是过时的内容（会以 **暗红色**标出以示区别）。

   目前为止我们还是手工命令行方式执行程序的，没有和IDE结合，其实Maven天生就对Eclipse做了集成，我们使用mvn eclipse:eclipse就得到了一个Eclipse的项目结构，在Eclipse中使用import功能就能直接导入到IDE中了。我们来看一下这个过程：    
     
    此时的demo就是Eclipse项目格式的了，出现了.project和.classpath文件。我们在Eclipse中引入这个项目，此时的Eclipse没有安装Maven插件，不能自动运行Maven命令，我们来安装Maven的Eclipse插件M2E。    


**如果由于网络原因Eclipse无法联网安装M2E插件，可以到这里下载我上传的离线包（安装方式为解压后合并同名文件夹并重启eclipse）：**

**http://download.csdn.net/detail/u012510614/7666949**

同样的方法我们来安装一下run-Jetty-run插件：

地址改成  <http://run-jetty-run.googlecode.com/svn/trunk/updatesite> 即可

**如果由于网络原因Eclipse无法联网安装run-jetty-run插件，可以到这里下载我上传的离线包，然后在Install New Software的窗口点击Add，然后选择Archive，选中下载的压缩包安装即可，压缩包地址：http://download.csdn.net/detail/u012510614/7667397**

 在Eclipse的Install New Software中直接选择安装即可，非常简单。下面我们来创建Web项目并导入Eclipse中，在Jetty容器中运行程序。如果你看了第一部分的教程，那么请删除第一部分建立的项目文件夹，或者另外新建一个maven项目，过程如下：

1、cmd呼出命令行窗口

2、mvn archetype:generate -DgroupId=com.cwbeta -DartifactId=demo-DarchetypeArtifactId=maven-archetype-webapp -Dversion=1.0（请将“com.cwbeta”替换成你的项目开发组，“demo”换成你的项目名称）

3、cd到你的项目目录

4、mvn eclipse:eclipse（使用这条指令使得项目成为eclipse能够导入的格式）

5、用Eclipse的File->Import导入该项目根目录

上面几部如果记不住可以写一个批处理的脚本方便以后建立项目：

@echo off

set /p project\_name=Name of the new project:

set /p producer\_name=Name of the producer:

mvn archetype:generate -DgroupId=%producer\_name% -DartifactId=%project\_name% -DarchetypeArtifactId=maven-archetype-webapp -Dversion=1.0

cd %project\_name%

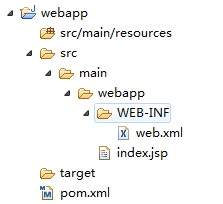
mvn eclipse:eclipse

echo [Press any key to exit]

pause>nul

exit

写入记事本保存成bat格式文件粘贴到workspace的目录运行即可。

 然后我们可以看到如下的Eclipse项目格式   
 

 可以看到，刚创建的web项目结构包含了resources目录，而没有java代码目录，我们需要手工创建，在Eclipse中创建source folder，路径为src/main/java/src，创建方法如下：

在你的项目上右键 -> New -> Source Folder

Folder name填上路径：src/main/java/src

勾上Update exclusion filters...选项（一定要勾否则不能创建）

Finish！

现在我们得到如下一个项目结构，新建一个Servlet用于测试，新建方法如下：

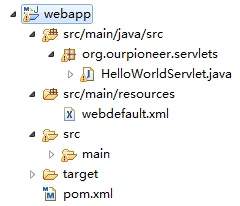
在你新建立的src/main/java/src的source file上右键 -> New -> Servlet

Java package填org.ourpioneer.servlets（源教程这么写，不知道是否能改，姑且先这么写着吧）

Class name填HelloWorldServlet

Finish！

然后你就会看到如下的文件脉络。

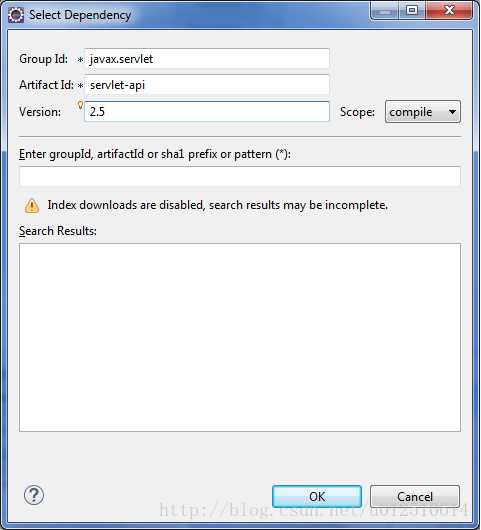


**如果在这里你发现HelloWorldServlet.java有报错说“The import javax.servlet cannot be resolved”，可以这么解决：**

在Eclipse中，右击项目，选择Build Path -> Configure Build Path... -> Libraries->Add External JARs，找到你计算机中Jetty(tomcat也适用)的安装路径，在lib文件夹下选中"servlet-api-3.0.jar"，添加点击“确定”，ok! 

 此时，项目中没有Servlet的依赖，需要添加，我们使用m2eclipse插件来直接添加依赖，如下所示：

双击项目根目录的pom.xml -> 选择Dependencies -> 左边的add -> 如下图配置



也可以手动编辑pom.xml

插入这一段：

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<type>jar</type>

<scope>compile</scope>

</dependency>

 下面就可以编写Servlet了，很简单，就输出HelloWorld吧（编辑HelloWorldServlet.java）。

package org.ourpioneer.servlets;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HelloWorldServlet extends HttpServlet {

protected void doGet(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

this.process(request, response);

}

protected void doPost(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

this.process(request, response);

}

private void process(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html;charset=utf-8");

response.setCharacterEncoding("utf-8");

PrintWriter out = response.getWriter();

String title="Webapp Demo";

out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">");

out.println("<html xmlns=\"http://www.w3.org/1999/xhtml\">");

out.println("<head>");

out.println("<meta http-equiv=\"Content-Type\" content=\"text/html;charset=utf-8\" />");

out.println("<title>" + title + "</title>");

out.println("<body>");

out.println("<h1>Hello World!</h1>");

out.println("</body>");

out.println("</html>");

}

}

源教程说需要再配置web.xml，而我用的eclipse自动配置完成了，所以这里跳过。

 程序都有了，剩下就是运行了，Maven既然天生和Jetty是一对儿，这里我们就使用Jetty吧，在Maven中配置Jetty，首先是webdefault.xml要准备好，它是配置Jetty的，这个可以从Jetty的包中找到，并复制到resources下，这里多说一点，默认Jetty运行时是锁定JS/CSS等静态文件的，如果想在Jetty运行时也能修改它们，要在webdefault.xml中修改如下设置：  

<init-param>

<param-name>useFileMappedBuffer</param-name>

<param-value>false</param-value>

</init-param>

 Jetty也准备了，运行命令是jetty:run，这要在Maven中设置，那么需要在pom.xml中加入Jetty的插件的设置信息。这里直接贴出其整体构建信息。

这部分注意，由于源教程贴出的构建信息中org.apache.maven.plugins这个plugin已不能使用（可以参考Jetty-Maven-Plugin的发展史：http://www.eclipse.org/jetty/documentation/current/jetty-maven-plugin.html），因此我做了一些修改，换成了org.eclipse.jetty的jetty插件，版本为9.0.0.v20130308（官网上给出的最新版本我这里测试不可用，而此2013年的版本可以使用，因此使用该版本）。修改后的内容如下，修改方法也是编辑项目根目录的pom.xml(双击然后选择pom.xml选项卡即可编辑)：

<build>

<finalName>demo</finalName>

<sourceDirectory>src/main/java/src</sourceDirectory>

<testSourceDirectory>src/test</testSourceDirectory>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>2.0.2</version>

<configuration>

<source>1.6</source>

<target>1.6</target>

<encoding>utf-8</encoding>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-resources-plugin</artifactId>

<configuration>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<plugin>

<groupId>org.eclipse.jetty</groupId>

<artifactId>jetty-maven-plugin</artifactId>

<version>9.0.0.v20130308</version> <!-- latest at time of writing -->

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-eclipse-plugin</artifactId>

<version>2.7</version>

<configuration>

<addVersionToProjectName>false</addVersionToProjectName>

<useProjectReferences>false</useProjectReferences>

<encoding>UTF-8</encoding>

<wtpmanifest>false</wtpmanifest>

<wtpapplicationxml>true</wtpapplicationxml>

<wtpversion>1.5</wtpversion>

<additionalBuildcommands>

<buildcommand>org.eclipse.jdt.core.javabuilder</buildcommand>

<buildcommand>org.eclipse.wst.common.project.facet.core.builder</buildcommand>

<buildcommand>org.eclipse.wst.validation.validationbuilder</buildcommand>

</additionalBuildcommands>

<additionalProjectnatures>

<nature>org.springframework.ide.eclipse.core.springnature</nature>

<nature>org.maven.ide.eclipse.maven2Nature</nature>

<nature>org.eclipse.wst.common.project.facet.core.nature</nature>

<nature>org.eclipse.jdt.core.javanature</nature>

<nature>org.eclipse.wst.common.modulecore.ModuleCoreNature</nature>

</additionalProjectnatures>

<classpathContainers>

<classpathContainer>org.eclipse.jdt.launching.JRE\_CONTAINER</classpathContainer>

</classpathContainers>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.1-beta-1</version>

<configuration>

<warName>webapp</warName>

</configuration>

</plugin>

</plugins>

</build>

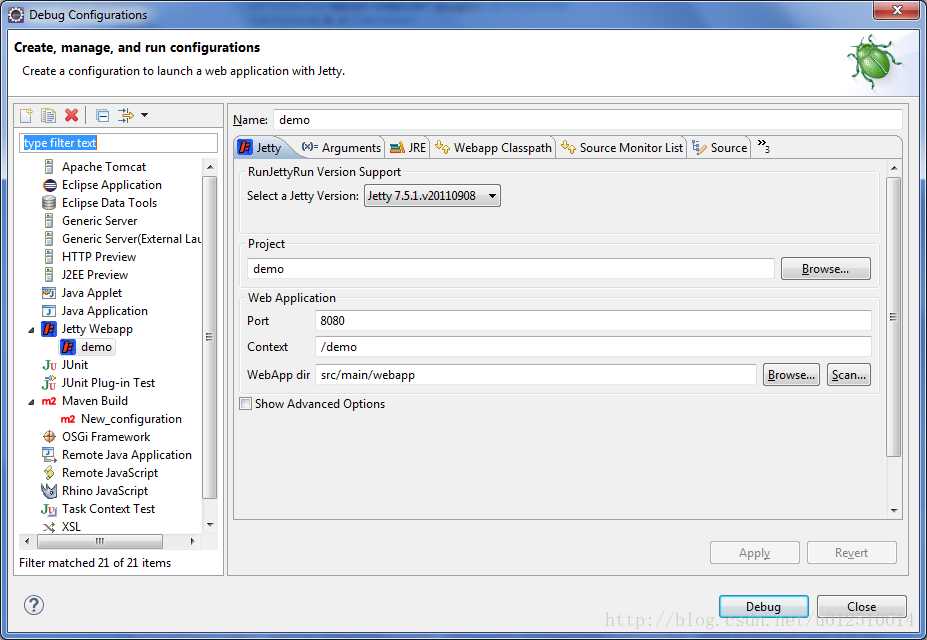
接着让我们更新一下maven的依赖，用cmd命令行工具cd到项目的根目录，执行mvn dependency:copy-dependencies。

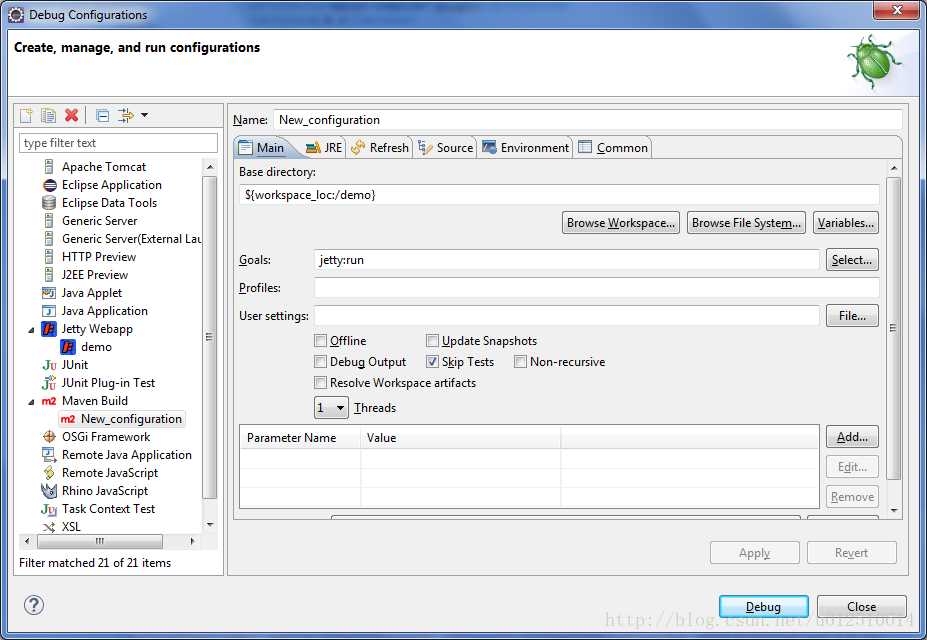
需要更新一段时间，不过整个配置就快完成啦，可以站起来活动活动，放松一下~

更新完成后，我们就可以在Eclipse中配置Debug运行了。配置很简单，如下：

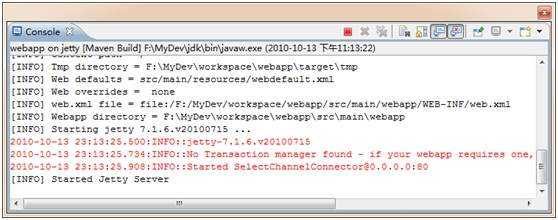
Run -> Debug Configurations...

因为安装了run-jetty-run插件，Jetty运行的端口可以在这里配置：

   
而Maven如下配置即可：



    这是Debug模式运行，Run模式下是一样的，用Debug模式可以在Eclipse中断点运行程序，非常便于调试。下面我们就让它跑起来吧。运行命令是jetty:run，Base directory配置是：${workspace\_loc:/应用名}，启动调试，看到如下信息，Jetty就成功启动了。

    
    如果你使用了80端口，在浏览器中访问地址如下：http://localhost/你的项目名称，之后，我们就看到了效果。    


**可能感兴趣**

* [run-jetty-run - Jetty Eclipse Plugin! Best plugin for running Jetty in Eclipse! Jetty,Eclipse,Maven,Web - Google Project Hosting](http://www.itnose.net/detail/480501.html)
* [利用Eclipse中的Maven构建Web项目（二）](http://www.itnose.net/detail/6048738.html)
* [利用Eclipse中的Maven构建Web项目（三）](http://www.itnose.net/detail/6048737.html)
* [利用Eclipse中的Maven构建Web项目（一）](http://www.itnose.net/detail/6048731.html)
* [利用Eclipse中的Maven构建Web项目报错（二）](http://www.itnose.net/detail/6048806.html)
* [利用Eclipse中的Maven构建Web项目报错](http://www.itnose.net/detail/6048743.html)
* [Eclipse maven构建springmvc项目](http://www.itnose.net/detail/6074086.html)
* [Eclipse maven构建springmvc项目](http://www.itnose.net/detail/514822.html)

   From : http://blog.csdn.net/u012510614/article/details/38059319