之前的Opendoc中没有涉及过此部分的内容，maven又是现在非常流行的java的工具，再加上到目前为止搭建OSGi Maven开发和部署的环境还是比较的麻烦，觉得有必要写篇这样的blog，:)，在这篇blog中来看下如何搭建一个比较好用的OSGi Maven开发和部署环境，看看我在搭建一个这样的环境中的痛苦历程。

　　首先说下我期望的OSGi Maven开发/部署的环境：

**1、META-INF中的MANIFEST.MF文件可以自己控制, 打包的时候需要打入自己控制的这个MANIFEST.MF**

　　Eclipse对插件工程的开发支持的很好了，在IDE中可以很方便的去修改这个manifest.mf，所以还是自己控制更爽，当然，**打包的时候需要打入自己控制的这个manifest.mf**。

**2、.classpath文件中，能够不包含所依赖的bundle的jar包。包括在mvn eclipse:eclipse生成的**

　　因为在OSGi环境中，已经不再通过直接在project的classpath中依赖其他bundle的jar了来调用其他bundle中的package，**而是通过在manifest.mf中增加import-package这样的方式，所以不能再把依赖的bundle的jar打到classpath里了，否则会很奇怪，**当然，这也源于eclipse有个很好的插件开发环境，让你可以在不依赖bundle jar的情况下直接写依赖其他bundle的package的代码。

**3、在mvn clean package的时候能够把bundle需要依赖的jar打到bundle中，并和META-INF/Manifest.mf文件中的Bundle-Classpath是匹配的；**

　　在某些bundle中可能会依赖一些jar，在META-INF中通常会去指定依赖的这些jar，放入bundle-classpath中，**因此要求在打包的时候能够把这些依赖的jar打入相应的路径下**。

说完想法后，首先想到的是在OSGi界中支持maven环境的大名鼎鼎的**maven-bundle-plugin**（http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html），maven-bundle-plugin基于Peter写的bnd实现，不说废话了，按照自己期望的环境来使用maven-bundle-plugin进行搭建：

　步骤一

　　按照上面页面的指导，**在pom.xml中增加maven-bundle-plugin先**，接着按照自己的想法，要自己控制manifest.mf，于是在plugin的configuration中增加：

　　<include>META-INF/MANIFEST.MF</include>

　　满心欢喜的等待着完美的结果，可惜....不如人意呀，打包出来的jar里面的MANIFEST.MF已经物是人非了，完全不是自己控制的那个，插件给你自动的加上了一堆的import-package、private-package、export-package，我知道这个插件是基于bnd来写的，但没想到竟然连自己控制的权力都不给我了，完全仍然是通过bnd来计算出import-package、private-package什么的；

　　步骤二

　　好，在伤心过后接着仔细看，还好，**在plugin的configuration中可以自己指定export-package、private-package这些**，于是继续欣喜的使用，这两个倒是控制住了，但...**.import-package自己是不能控制的，这个是不行的，这样就导致了必须同时自己维护pom.xml以及project中的META-INF/MANIFEST.MF，让它们保持一致，**否则可能导致打出来的包和你在project中运行的表现不一致，并且bnd计算出来的import-package并不是我想要的，有点太复杂了，还是自己控制比较好；

　　步骤三

　　伤心到极点了，其实到目前为止，已经可以确定**maven-bundle-plugin，也是OSGi maven中唯一的插件，不能满足我的需求**，不过还是继续看看这个插件其他方面的表现，惊喜的发现有一点倒是做的不错的，它支持一个<Embed-Dependency>\*;scope=compile|runtime</Embed-Dependency>，有了这个标签后，它可以直接把依赖的jar打入bundle jar包中，并且相应的自动在bundle-classpath中加上了，这点倒是不错的，看起来与我期望的环境的第3点是比较匹配的，可惜了。

有就是，很当然的，它没法做到控制mvn eclipse:eclipse时生成的.classpath不包含bundle jar的引用。

　　按照上面的三个步骤，总结下，有些时候智能是好事，但maven-bundle-plugin就是过于智能了，为什么不给点权力给使用者呢，因此这个插件要提升到完全可用的情况的话，还需要提供下让使用者自己控制MANIFEST.MF的权力，相信这点要做到并不困难，而且做到这点后基本也就可以使用了。

　　继续寻找，于是静心分析了下自己的需求，貌似可以自己通过maven现有的几个插件来达成自己的愿望，于是开始了组合拳：

**1、MANIFEST.MF文件自己控制**

　　不就是要自己控制这个文件嘛，OK，干脆，就只用**maven-jar-plugin**，这个插件允许指定所使用的MANIFEST.MF文件，于是，尝试着在这个plugin的configuration中增加：

　　<archive>

　　 <manifestFile>META-INF/MANIFEST.MF</manifestFile>

　　</archive>

　　恩，很顺利，开门红呀，**打出来的jar包中的MANIFEST.MF文件就是自己的那个。**

**2、mvn eclipse:eclipse生成的.classpath中要去掉bundle jar的依赖**

　　对于我这么一个对maven不是那么熟悉的人来讲，这个有点复杂，于是不断的google，甚至是翻看了maven-eclipse-plugin的源码...

　　最终终于功夫不负有心人，找到一个简单的办法：

　　首先将工程方式指定为pde，也就是eclipse插件工程，**在maven-eclipse-plugin的configuration配置中增加<pde>true</pde>；**

**然后在pom.xml中将不希望生成到.classpath中依赖的scope指定为provided**；

　　心惊胆战的开始运行mvn eclipse:eclipse，OH YEAH!，成功！

　　ps: 另外也可以通过在maven-eclipse-plugin的configuration中增加exclude配置，来将某些依赖从.classpath中去掉，当然，这方法没有上面的易用。

**3、在mvn clean package的时候能够把需要依赖的jar打到bundle jar中，并和META-INF/Manifest.mf文件中的Bundle-Classpath是匹配的**；

　　恩，这点，印象中貌似maven是有支持的，于是继续开始找，终于找到了maven-dependency-plugin（之前还找到了一个**maven-shade-plugin**，也很帅，不过不满足需求），**通过这个插件可以把需要的依赖的jar都复制到某个指定的目录中去，但记得把这个指定的目录加入到maven-jar-plugin的resources目录里面去，否则这些jar文件是不会出现在你的bundle jar里的**。

　　OK，通过上面这套组合拳，终于达成了目的，看来有必要找个时间写个好用点的maven的OSGi插件，否则真的忒折腾了。