

NJU

Conup 使用简介

如何编译源码及利用 `conup` 进行开发

陈栋

2013-6-24

1.简介

本文档详细说明 conup 的安装与配置，以及如何基于 conup 开发例子并实现动态更新。备注：需结合 conup 官方文档进行阅读，在 conup 官方文档阅读之后尚未动手实验之前阅读效果最佳，可以避免走很多弯路。

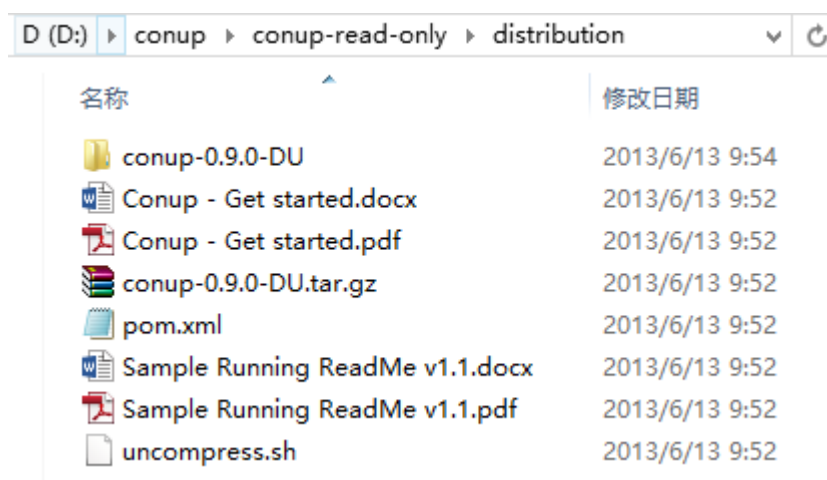
2.Conup 下载与安装

去 google code 上搜索 conup，进入后通过 SVN 下载代码即可。代码比较大，不翻墙的话可能很慢，下载下来的文档目录如下。不妨设此目录的路径是 CONUP_ROOTPATH。



名称	修改日期	类型
.svn	2013/6/13 10:16	文件夹
distribution	2013/6/13 9:54	文件夹
docs	2013/6/13 9:53	文件夹
maven	2013/6/13 9:53	文件夹
modules	2013/6/13 9:56	文件夹
samples	2013/6/13 10:11	文件夹
pom.xml	2013/6/13 9:53	XML 文件

其中有个 distribution 目录，进入 CONUP_ROOTPATH/distribution 之后目录结构如下：



名称	修改日期
conup-0.9.0-DU	2013/6/13 9:54
Conup - Get started.docx	2013/6/13 9:52
Conup - Get started.pdf	2013/6/13 9:52
conup-0.9.0-DU.tar.gz	2013/6/13 9:52
pom.xml	2013/6/13 9:52
Sample Running ReadMe v1.1.docx	2013/6/13 9:52
Sample Running ReadMe v1.1.pdf	2013/6/13 9:52
uncompress.sh	2013/6/13 9:52

其中有两份文档，大致说明了安装和配置以及测试用例，但是很遗憾，文档比较言简意赅，有些地方不是很详细，本文会做进一步说明。

CONUP_ROOTPATH/modules 中可以看做是 conup 的源码，即修改后的 tuscanyc 源码，如果开发者有需要的话可以进一步修改。

CONUP_ROOTPATH/distribution 目录下有 conup-0.9.0-DU.tar.gz 文件，这是 binary 版本。解压缩后得到 conup-0.9.0-DU 目录，其目录结构是

distribution ▶ conup-0.9.0-DU ▶ conup-0.9.0-DU			搜索 cc
名称	修改日期	类型	
bin	2013/6/13 20:45	文件夹	
features	2013/6/13 9:54	文件夹	
lib	2013/6/13 9:54	文件夹	
modules	2013/6/14 14:49	文件夹	
samples	2013/6/14 23:10	文件夹	
INSTALL	2013/1/7 21:14	文件	

我们可以将其拷贝到别处，如果不是太爱折腾的话还是不要动它位置好了。假设它的目录是 CONUP_BINARY_PATH,设置环境变量 TUSCANY_HOME=CONUP_BINARY_PATH，并将其子目录下的 bin 目录添加到 PATH 中，我们就可以在终端中直接执行命令 tuscanyc 了。

3.编译源码

Conup 和 tuscanyc 一样，也是个 maven 工程，maven 版本 3.0.4。

先在 CONUP_ROOTPATH 下执行 mvn install，这步生成的 jar 包后续编译都有可能用到。

为了使 tuscanyc 的 binary version 保持最新，且在 maven 本地库中生成开发项目时用到的 jar 包，我们需要编译源码，在 CONUP_ROOTPATH/modules 目录下执行 mvn install，不出意外会报如下错误：

- 1) Jdk 版本要求 1.7，不解释了，conup 开发时用的是 1.7，乖乖换成 1.7 吧
- 2) 测试用例报错，mvn install -DskipTests=true 跳过测试。
- 3) 缺少对...的依赖，这是因为源码中引用了一个非官方 pom 依赖，叫做 deus-0.4.8.jar，获得它，然后安装到本地 maven 库中。

```
Mvn install :install-file -Dfile=deus-0.4.8.jar -DgroupId=it.unipr.ce.dsg -DartifactId=deus -Dversion=0.4.8 -Dpackaging=jar。
```

编译过程中会将代码打包拷贝到 TUSCANY_HOME 下的 modules 等目录下。

4.测试例子

CONUP_ROOTPATH/samples 中自带了几个例子，其中 authUpdate 就是论文中的例子。

(D:) > conup > conup-read-only > samples		
名称	修改日期	类型
authUpdate	2013/6/13 9:53	文件夹
conup-travel-sample	2013/6/13 9:53	文件夹
target	2013/6/13 10:11	文件夹
pom.xml	2013/6/13 9:53	XML 文件

在 authUpdate 目录下执行 `mvn install`，会将生成的 jar 包拷贝到 TUSCANY_HOME/samples 对应的目录下。当然也可以直接在 samples 目录下编译。

Conup 运行组件时需要显式定义构件之间的依赖关系及替换算法，参照文档，这是在 TUSCANY_HOME/bin 目录下的 conup.xml 中定义。确保所运行的所有构件之间的关系都在其中进行了声明。

按照文档，修改 ip 之后编译，然后在 terminal 中启动 4 个节点，`tuscany.sh xxxjar`。

千万注意，无论运行什么组件都得通过这种方式执行，试图通过 `mvn Tuscany:run` 来运行都是妄想，具体原因得问开发者。大概是因为 conup 修改了原 tuscany 的一些东西，在脚本中添加了一些额外的动作，例如代码插入，通信模块等，通过其他方式执行会跳过这些动作之类。而且，在 windows 下运行 `tuscany.bat` 也是不行的，报的是路径错误。。。

此外，conup 对原 tuscany jsonrpc 的通信格式也进行了修改，如果要自己开发能够动态更新的例子的话，构件之间务必采用 jsonrpc 的通信方式。还有一点需要强调的是，加入自己的例子中加入了别的包的依赖，如 `mysql-connector` 之类，一定要将它的 jar 包放入到 TUSCANY_HOME/modules 目录下，这样 classpath 中才有，tuscany 才能够找到。

例如，我在自己的例子中引用了

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.0.3</version>
</dependency>
```

则在 modules 目录下创建 `mysql-connector-java-5.0.3`，并将 `mysql-connector-java-5.0.3.jar` 放入目录中。

mysql-connector-java-5.0.3	2013/5/27 23:39	文件夹
ode-agents-1.3.2	2012/6/18 13:19	文件夹

5.构件更新

文档中关于更新遗漏了一点，就是更新前先要做代码插入。前面已经提到，通过在终端中执行 `tuscany.sh xxx.jar` 命令启动组件，这个过程中终端会输出

```

Try to preprocess source code...
Jun 24, 2013 9:05:28 PM cn.edu.nju.moon.conup.apppre.TuscanyProgramAnalyzer addTxLifecycleManager
INFO: There has already a _txLifecycleMgr in the cn/edu/nju/moon/service/DBServiceImpl
Jun 24, 2013 9:05:28 PM cn.edu.nju.moon.conup.apppre.TuscanyProgramAnalyzer transform
INFO: The insert didn't work because the injection of _txLifecycleMgr fails!
Preprocessing is done...

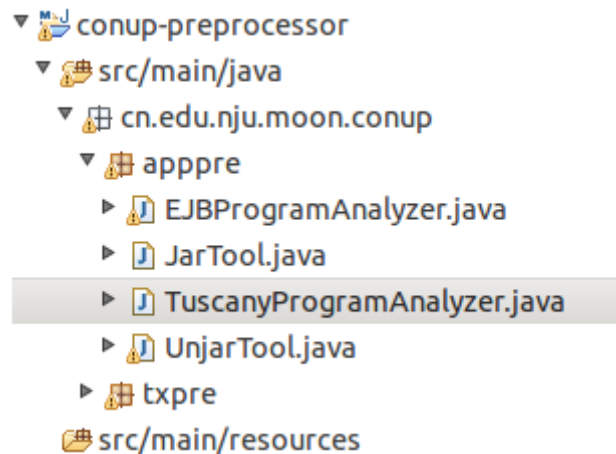
```

INFO: The insert didn't work because the injection of _txLifecycleMgr fails!

Preprocessing is done...

以上输出是因为我已经运行过这个 Jar 包，即代码插入已经做过了，初次执行的话会输出 insert successfully 之类。

那么，在我们要更新一个组件时，也要对用来替换的代码做插入操作。找到 CONUP_ROOTPATH/modules/conup-preprocessor 工程。



修改其中的 TuscanyProgramAnalyzer 类中的 main 方法，修改 baseDir 和需要进行代码插入的工程。例如我用来替换的构件位置在

CONUP_ROOTPATH/samples/hydrology-pondage-update，那么 main 函数将会是这样：

```

public static void main(String args[]) {
    try {
        TuscanyProgramAnalyzer analyse = new TuscanyProgramAnalyzer();

        List<String> targetProjs = new ArrayList<String>();
        targetProjs.add("hydrology-pondage-update");
        targetProjs.add("fullapp-bespoketrip");
        targetProjs.add("fullapp-coordination");
        targetProjs.add("fullapp-currency");
        targetProjs.add("fullapp-packagedtrip");
        targetProjs.add("fullapp-shoppingcart");
        targetProjs.add("payment-java");
        //targetProjs.add("fullapp-bank");

        String baseDir = "/home/deerstalker/conup/conup-read-only/samples/";
        for(String projLoc : targetProjs){
            projLoc = baseDir + projLoc + "/target/classes";
            analyse.analyzeApplication(projLoc, "");
        }
    }
}

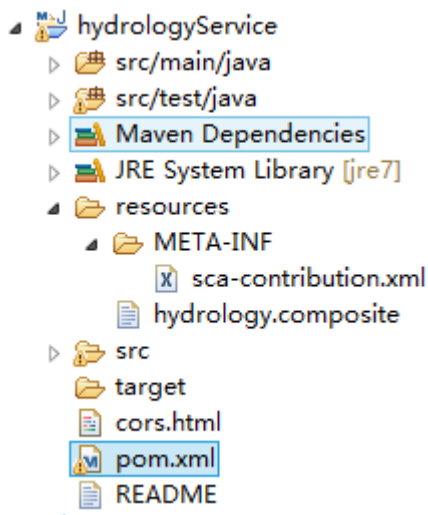
```

文档中关于更新还有一点没有说明清楚的是，更新是通过显式调用另外一个组件的服务执行的，即 `authUpdate/conup-sample-configuration-client`，修改 `ConfService` 类加入对你要替换的组件的逻辑判断。

```
port = 18084;  
} else if (compIdentifier.equals("NodePondageComponent")){  
    classFilePath = "cn.edu.nju.moon.node.pondage.PondageServiceImpl";  
    contributionUri = "hydrology-pondage";  
    compsiteUri = "App.composite";  
    port = 18082;  
}
```

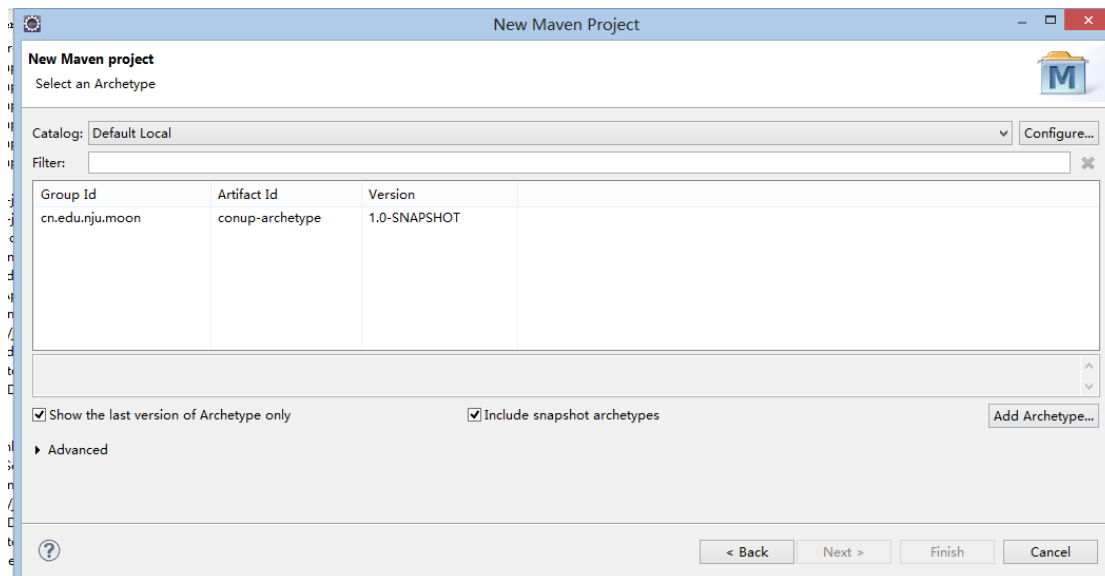
6.创建合适的 maven archetype

在 eclipse 创建 tuscanry 工程的时候，没有找到合适的原型，我们需要的是如下的目录结构，如果有个原型刚好帮我们自动生成这样的结构则最好，不需要我们每次手动创建和添加。



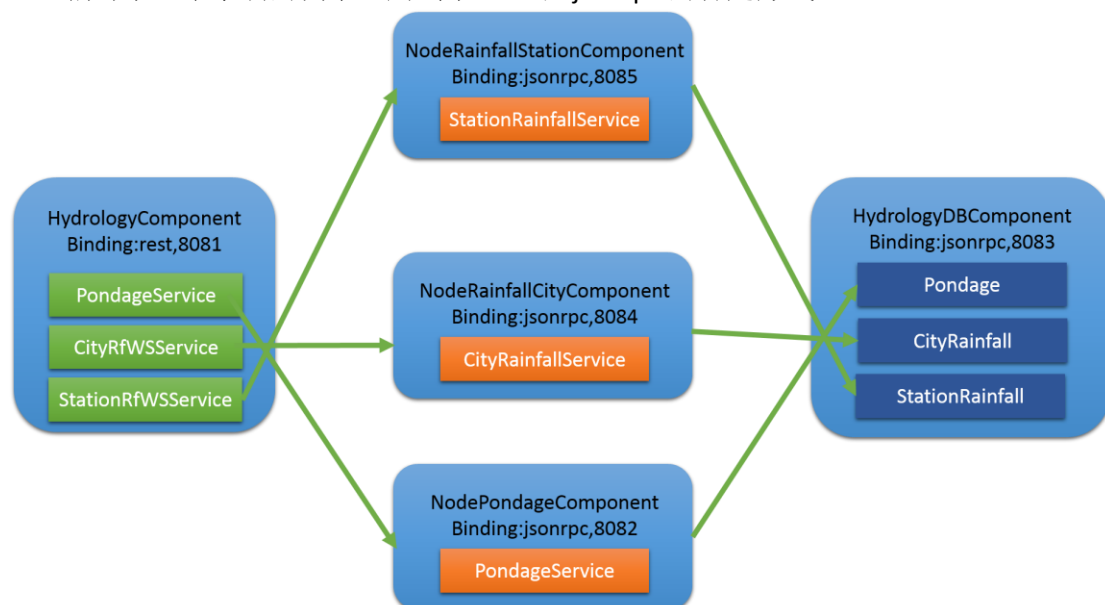
为了便于开发，我们先创建适合 conup 的 maven archetype，参照 <http://maven.apache.org/guides/mini/guide-creating-archetypes.html>

当然，为了方便，我们可以让 eclipse 查找到我们刚新建的 archetype，那么继续参照 <http://maven.apache.org/archetype/maven-archetype-plugin/specification/archetype-catalog.html>，这样在 eclipse 新建 maven 工程的时候，就可以使用自己的原型。(如果 eclipse 找不到，请点击 configure 或者 add archetype 手动添加)



7.编写用例

编写了一个水利的例子，用到了 `rest` 和 `jsonrpc` 的绑定方式。



8.eclipse 中运行例子

上面提到的都是在终端中执行例子，其实 `eclipse` 中也可以运行例子，参考 `authUpdate` 中各个子工程的源码中的 `launcher` 包，额外强调的是在运行前需要对其进行代码插入，自己写的每个工程都要进行代码插入。区别于在终端中执行的方式，在终端中执行 `tuscany.sh xxx.jar` 会自动进行代码插入。

此外，很有可能出现 pom 中各种依赖找不到的情况，那么你就要回顾下编译源码那一节，看是否跳过了些许步骤。例如，假设没有在 CONUP_ROOTPATH 下执行 mvn install，虽然 CONUP_ROOTPATH/modules 模块也可编译，但是 conup 自带的例子会报各种找不到 jar 包，然后你会惊讶地发现这些 jar 包很多都在 maven 的本地库中（编译 modules 模块生成的）。。。

9.补充

在运行例子之前需要做代码的预处理， conup-preprocessor 修改 main 方法，主要是增加新构件项目的路径。

如果通过 conup-sample-configuration-client 更新，conup-sample-configuration-client 中要修改 update 方法，加入对更新构件的判断，端口号为需要被替换的构件的绑定地址+10000。

composite 中 service 的 name 一定要和服务接口类的名字一致

需要用到 conup-core 的 jar 包

引用的 duy 的 jar 包：需要将 RemoteClient.jar 放到 classpath 中去

对服务的引用 set 方法前加 reference。