#开发环境配置

##相关软件安装

为了运行Eclipse Che实例，需要安装Docker；为了编译打包项目，需要安装Maven。

Docker官网在明显的地方只提供了Docker for Windows和Docker for Mac下载，若要在Linux环境下安装Docker，需要在Docker Home Page->Resources->Docs->Get Docker->Docker CE->Linux寻找对应版本的安装文档。值得一提的是，Docker在17.03之后分化为企业版的Docker EE和社区版的Docker CE，考虑到经济因素本文选择Docker CE。在Win 8.1以及其他某些版本的Windows，Docker会要求在BIOS中开启Hyper-V，但之后有仍有可能出现一些系统环境问题，因此强烈不建议在Windows上编译部署此项目。

Maven的安装比较简单，在已经安装Java的情况下，下载安装包解压，配置MAVEN\_HOME，并将%MAVEN\_HOME%/bin添加到Path中即可。因为本文不涉及Dashboard部分的编译，因此可以跳过Eclipse Che指南中的GO语言安装。

##项目配置

Eclipse Che有多种方法可以达到生成带插件的产品这一目的。其中包括fork完整源代码的项目，在运行的Eclipse Che中fork插件项目和在本地fork插件项目方法等。经过多次尝试，发现通过完整源代码部署存在配置复杂，部分依赖插件不易找到等问题；在运行的Eclipse Che中fork插件项目存在资源开销大，系统相应慢和版本不一致等问题（Che环境会自动拉取最新版本的che-dev的Docker镜像，而非当前版本）。因此选用本地fork插件项目，再进行修改的开发方式。该方式具有资源开销小，项目结构简单，同时版本容易控制等优点。

根据设计，本课题要对Eclipse Che的IDE端输入的数据和Workspace中存储的数据进行编辑，因此选用ide-server-extension的模板进行开发。Fork “che-samples/che-ide-server-extension”项目到个人仓库下并克隆到本地。克隆的模板项目有两个模块：plugins和assembly。plugins模块下包含我们开发的各个插件，当前只有一个插件；assembly模块中包含打包生成产品的控制代码，项目模板已经将对插件的添加到对应的包中。

接下来需要调整项目POM对象属性。原来POM对象的groupId为Eclipse Che项目模板Id：org.eclipse.che.sample。Maven 依靠groupId，artifactId和版本号来标识确定一个对象。在执行Maven命令“mvn clean install”时，可以将含有自己的扩展的对象发布到本地Maven仓库中，后续的打包过程会先从本地仓库中获取对应的扩展，可以获取到我开发的扩展对象并进行打包。但若代码编写出现问题部分编译失败，Maven则会尝试从远程仓库获取扩展，并继续打包。这样在实例有可能仍然运行成功，但却不包含我的扩展。因此，我将groupId改为自己的标识，这样在扩展没有成功生成的情况下，后续打包会因找不到Maven对象而中断，方便错误的识别。

在此过程中，除了要将各个文件夹下的pom.xml文件的根节点下的“<groupId>”标签和“<dependency>”标签下的“<groupId>”标签中内容替换为自己的命名空间外，还需将assembly-main下的assembly.xml中的“<dependencySets><dependencySet><includes><include>”标签下的assembly-ide-war和assembly-wsagent-server的groupId换成自己的命名空间。同时对assembly-wsagent-server下的assembly.xml也做类似替换。

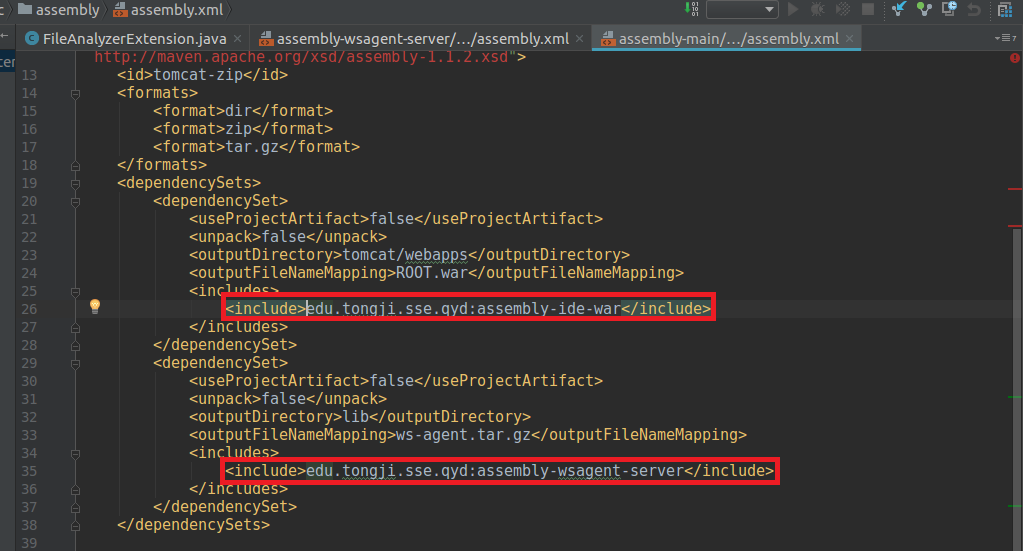


图 assembly-main中assembly.xml中需要修改的命名空间

之后可以在项目跟目录下执行“mvn clean install”命令，可能出现license不匹配，pom文件没有排序，java文件不符合规范等报错，原因是Eclipse Che上层pom配置了在build过程中相应检查。可以通过执行“mvn license:format”，“mvn sortpom:sort”以及“mvn fmt：format命令”来格式化相关代码。除此之外，还可以直接在pom.xml文件的<builder>标签下配置相关plugin，具体配置如下图：