



实训报告

作业名称	Vi, Java, Ant, JUnit 自学报告				
院系	软件学院	班级	教务 2 班	姓名	李明宽
学号	11331173	完成日期	2013 年 07 月 02 日		
QQ	736459905	E-mail	limkuan@mail2.sysu.edu.cn		

今天是中级实训的第二天，大概总结一下这两天做了什么。由于使用过 Linux，对 Linux 不是很陌生，因此使用起来不是很困难，然后学习了神之编辑器 Vi 的一些基本用法，用 Vi 编辑了 helloWorld.java，然后使用 ant 实现了 helloWorld.java 的自动编译（总觉得这个词不能这么用，应该是解释？）运行，以及打包成 jar 文件，最后学会了如何使用 JUnit 来测试部分的函数功能是否正确，检查环境无误后，成功编译运行了 BugRunner.java。

Vi 的学习心得

以前用的都是另一个神级编辑器 Emacs，因为有现成的配置文件，然后根据自己的习惯又对配置文件修改了写，然后让 Emacs 使用起来非常方便。今天第一次使用 Vim，体会到了大神常说的用 Vim 多了后 Esc 键会被按坏的意思了 o(╯□╰)o

虽然说对于没用过 Vim 的人来说，编辑方式太复杂，经常要在命令模式与插入模式之间切换，而且一些常用的快捷键不按熟就会感觉很奇怪，例如使用 h、j、k、l 这四个键来进行光标移动，很明显好处就是能够手指不离开键盘进行光标移动，但是对于不熟练的使用者来说可能就经常会按错，而且在插入的时候如果打错会习惯输入光标上下左右，然后就会出现神奇的字母 ABCD，然后又要回到命令模式里删除。虽然说 Vim 不支持鼠标定位，但是可以直接在命令模式里使用一些组合按键快速定位到后几个单词或者第几行等等这些位置，对于代码编辑来说是很灵活的。

```
vim helloWorld.java
public class helloWorld
{
    public static void main(String[] args)
    {
        System.out.println("helloWorld!!");
    }
}

~
~
~
~
"helloWorld.java" 10L, 130C 1,1 全部
```

在我今天使用的 Vim，是没有经过配置的 Vi，没有语法高亮，也没有自动补全之类的插件或者配置文件，再加上我个人以前对 Vi 都不是很熟悉，在看完 Wiki 上提供的帮助文档后，以及阅读了 Vim 的帮助文档 Vimtutor，勉勉强强能在 Vim 上写出了 HelloWorld.java，如上图所示（没使用虚拟机，使用了 putty 连接 Linux 主机），然后使用“:!{command}”这种方法执行带个 shell 命令，在 Vim 中对 java 源码进行解释运行，而不用再次退出 Vim 进入 shell 命令界面运行，这对编程效率又是一个很大的提高，如果能完善 Vim 的配置文件，让它支持语法高亮以及自动缩进自动补全，然后还要熟练掌握 Vim 的快捷键，这样，Vim 一定会变成帮助我们写代码的利器。

Java 学习心得

首先要安装 JDK 完成 Java 环境的配置，在 Ubuntu 下，我选择了使用包管理软件 apt-get 来安装，在终端中输入命令：sudo apt-get install openjdk-7-jdk，然后包管理软件就会在软件源里找到相应的软件包并下载完成安装，不用再次麻烦配置 Linux 下的环境变量。

对于 Java 这门语言，由于有学习过 Android 的开发，当时粗略地看了一下 Java 的相关书籍，就是感觉语法跟 C++ 很像，对 C++ 比较熟悉的话是比较好学习的，但是它是解释性语言，而且所有的东西都是对象，每一个文件都包含一个类，类名要求与文件名相同。自我感觉 Java 对类掌握要求较高，一些继承，多态等特性会经常使用。

通过编写了简单的 HelloWorld 程序，大致了解了 Java 的一些特性，包括使用命令 javac 命令对源代码进行解释，使用 java 命令来运行 Java 程序。但是由于对 Java 的一些特性不熟悉，在后来对 BugRunner 的编译运行中出现了很多奇怪的问题还没解决，可能要等到以后慢慢熟悉整个 GridWorld 的框架，这样才好找出解决问题的方法。

在接下来的学习过程中，我们要继续深入学习 Java，并了解在 Eclipse IDE 下对 Java 工程进行编译运行的方法，继续深入学习对 GridWorld 的具体实现。

Ant 学习心得

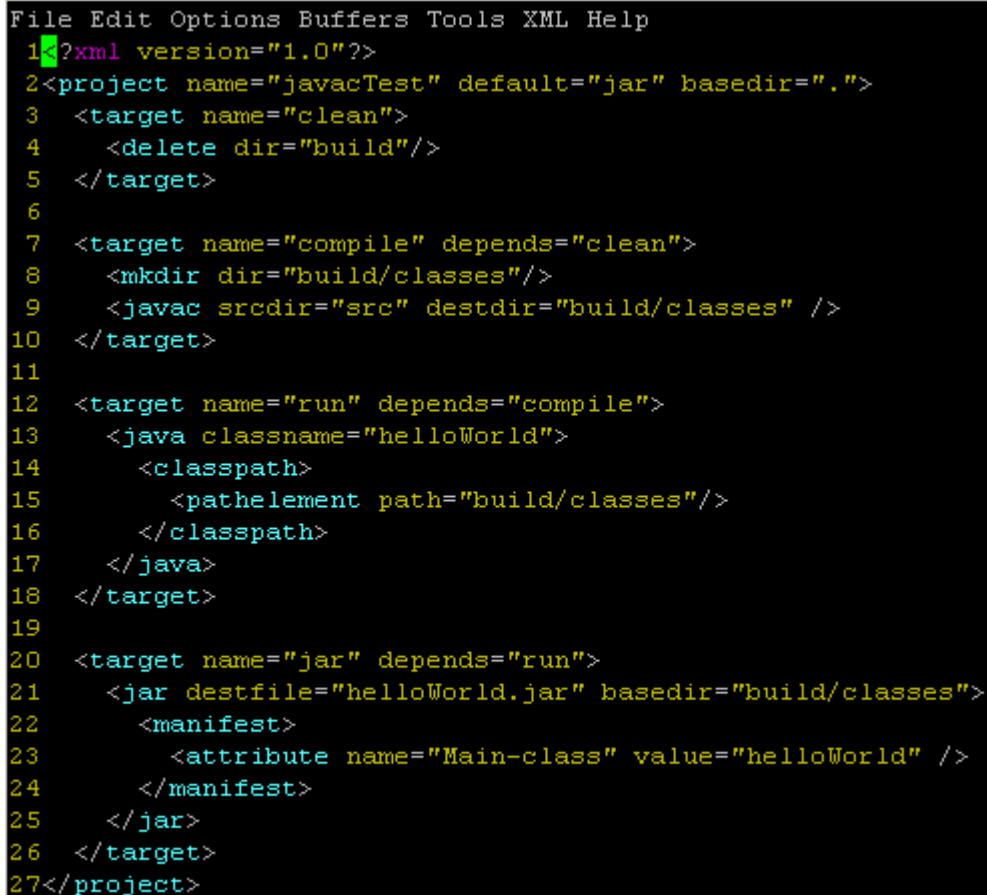
第一次听说 Ant 这个软件，以前对它的一些用途用法都不是很熟悉，经过学习了帮助文档，渐渐感觉到了 Ant 的强大。Ant 是一个基于 Java 的生成工具，可以实现对 Java 工程的自动编译打包运行，如果一个代码项目很大了之后，每次对代码的重新编译，打包测试都很复杂，但是使用了 Ant 后，只要编写好这个项目的 build.xml，就可以很方便对项目进行自动编译运行。

了解了 Ant 的用法后，感觉 Ant 长得很像 C/C++ 里面的 Make，功能都差不多，都是使用脚本对源代码进行批量编译或者运行，但是 Ant 使用的脚本是基于 XML 的，因此书写维护起来都比较方便。而且 Ant 还有诸如跨平台，操作简单，结构清晰等优点是 Make 无法达到的。

使用 Ant 首先要编写好 build.xml 的运行脚本。脚本里大概包含 3 大部分：

- 首先是一些定义的 property 元素，它包含 name 和 value 属性，一般是一些工程里使用到的目录，文件名等等，例如源代码目录 src 等。在后面使用到的时候可以使用 \${name} 的方法来使用在这里定义好的 value。这个部分在写 HelloWorld 的 build.xml 的时候没有用到。

下图是 HelloWorld 的 build.xml，包括了编译，运行，以及打包成 jar。



```
File Edit Options Buffers Tools XML Help
1<?xml version="1.0"?>
2<project name="javacTest" default="jar" basedir=".">
3  <target name="clean">
4    <delete dir="build"/>
5  </target>
6
7  <target name="compile" depends="clean">
8    <mkdir dir="build/classes"/>
9    <javac srcdir="src" destdir="build/classes" />
10  </target>
11
12  <target name="run" depends="compile">
13    <java classname="helloWorld">
14      <classpath>
15        <pathelement path="build/classes"/>
16      </classpath>
17    </java>
18  </target>
19
20  <target name="jar" depends="run">
21    <jar destfile="helloWorld.jar" basedir="build/classes">
22      <manifest>
23        <attribute name="Main-class" value="helloWorld" />
24      </manifest>
25    </jar>
26  </target>
27</project>
```

- 第二大部分是 project 元素，它定义了这个构建文件的属性，每个构建文件都有一个 project 元素。这个元素有下面一些属性：**default** 表示默认的运行目标，这个属性是必不可少的，**basedir** 表示项目的基准目录，默认是当前目录，**name** 表示项目名称，**description** 表示项目的描述。
- 第三部分是 target 元素，一个 project 元素下可以有一个或多个 target 标签。target 有以下的一些属性，**name** 表示这个 target 的名字，这个属性是必须的，**depends** 表示这个 target 依赖的目标，这要求运行这个 target 前一定要运行它依赖的目标，**description** 表示项目的描述。

然后在运行 Ant 的时候，一般是在终端运行 `ant+target.name`，如果不加 target 的名字，那运行的就是 project 中 default 中默认的目标。

在学习 Ant 后，写了 HelloWorld 的自动运行脚本，在配置完 GridWorld 的运行环境后，尝试使用 ant 来实现 BugRunner.java 的自动编译运行，但是过程中遇到了很多问题，主要是编译 BugRunner 需要使用外部的 jar 包 gridworld.jar，在 ant 中要引用这个包，教程中没有提到这种方法，通过查询网络上的资料，终于找到了解决办法。

左图所示的目录树是 BugRunner 所在的 Project 的目录，其中包括 build.xml，lib 与 src 子目录，lib 里包含 gridworld.jar，src 中包含 BugRunner.java。运行后的目录变化如右图所示，会在添加子目录 build，这里是解释出来的 BugRunner 类，然后子目录 dist 是可执行的 BugRunner 的 jar 包，可以直接使用 java -jar BugRunner.jar 运行。

```
➔ firstProject tree
.
├── build.xml
├── lib
│   └── gridworld.jar
└── src
    └── BugRunner.java
2 directories, 3 files
```

```
➔ firstProject tree
.
├── build
│   ├── classes
│   │   └── BugRunner.class
│   └── build.xml
├── dist
│   └── BugRunner.jar
├── lib
│   └── gridworld.jar
└── src
    └── BugRunner.java
5 directories, 5 files
➔ firstProject
```

下图是 build.xml 的代码，包括了 4 个 target，分别是 clean，compile，jar，run。

```
File Edit Options Buffers Tools XML Help
1<?xml version="1.0"?>
2<project name="BugRunner" default="run" basedir=".">
3
4  <path id="project.class.path">
5    <fileset dir="${basedir}/lib">
6      <include name="*.jar"/>
7    </fileset>
8    <pathelement location="${basedir}/build/classes" />
9    <pathelement path="${java.class.path}" />
10  </path>
11
12  <target name="clean">
13    <delete dir="build"/>
14    <delete dir="dist"/>
15  </target>
16
17  <target name="compile" depends="clean">
18    <mkdir dir="build/classes"/>
19    <javac srcdir="src" destdir="build/classes"
20      fork="true">
21      <classpath refid="project.class.path"/>
22    </javac>
23  </target>
24
25  <target name="jar" depends="clean, compile">
26    <mkdir dir="dist" />
27    <echo message="${ant.project.name}"/>
28    <jar jarfile="dist/${ant.project.name}.jar"
29      basedir="build/classes">
30      <manifest>
31        <attribute name="Main-Class" value="BugRunner"/>
32        <attribute name="Class-Path" value="${basedir}/lib/gridworld.jar"/>
33      </manifest>
34      <zipfileset excludes="META-INF/*.SF" src="${basedir}/lib/gridworld.jar"/>
35    </jar>
36  </target>
37
38  <target name="run" depends="jar">
39    <java jar="${basedir}/dist/${ant.project.name}.jar"
40      fork="true">
41    </java>
42  </target>
43
44</project>
```

这个实现的要点是创建 `project.class.path`，包含 `lib` 里的文件夹所有要用到的 `jar` 文件，在编译的时候使用 `classpath` 包含进去。然后要先打包然后使用这个 `jar` 包来运行。打包的时候 `manifest` 中 `attribute` 中，使用 `Class-Path` 包含外部的 `jar` 包，要具体写每一个包的名字，而不能使用 `*.jar` 代替。

完成了这个 `build.xml` 的编写，感觉对 `ant` 的脚本文件的编写又有了进一步的理解，更加深入理解了 `ant` 的用法。

JUnit 学习心得

JUnit 是 `java` 的单元测试工具，用来测试项目中的文件的部分代码的正确性，这对于大工程是很有帮助的，因为项目里有许多函数，如果一些函数的错误会给后面的开发带来很大的不方便。而如果要确认自己的函数是否正确，传统的方法就是写主函数然后编译运行，看输出的结果是否与正确的结果吻合，这样子显然很麻烦。有了 Junit 的帮助，就可以使用 `@before`，`@after`，`@test` 等元数据编写一个测试代码，从而检验自己的函数功能是否正确。

在例子中，我们都是使用命令行进行 JUnit 的测试，就要在命令行中输入：

```
javac -classpath .:junit-4.9.jar HelloWorldTest.java
```

```
java -classpath .:junit-4.9.jar -ea org.junit.runner.JUnitCore HelloWorldTest
```

使用这两句命令进行编译执行。运行后，如果函数的功能都正确，那么会返回测试通过的信息，并且会显示做出了几个测试。如果测试不通过，那会在错误提示中显示出函数运行结果与预想的不同的地方。

以后我们可以在 `Eclipse` 等 IDE 中进行 JUnit 测试，这样极大提高了效率以及代码的准确性，对写 `Java` 项目是很有帮助的。

实训刚刚开始，继续加油！O(∩_∩)O