自学报告

13331371 周吉昊

VI

vim 是 vi 的高级版本，使用更加广泛，学习 vim 需要了解常用的快捷键，如下所示：
h,j,k,l 上，下，左，右
dd 删除光标所在行
dw 删除一个字
D 删除到行末
yy 复制一行
p 粘贴粘贴板的内容到当前行的下面
i 从当前光标处进入插入模式
Esc 退出插入模式
u 撤销 ctrl+r 重做
:w 将缓冲区写入文件，即保存修改
:wq 保存修改并退出
:q! 强制退出，放弃修改

JAVA

Java 的环境配置不是很难，下载 jdk 解压即可，最后在/etc/profile 中添加环境变量就行了。
在我的 Java 学习过程中，主要针对了以下几点。
1. Java 面向对象的特性。java 是以类和对象为载体的编程语言，具有封装，继承，多态等特性。与 C++不同的是，Java 还定义了接口 Interface ，只能被执行，不能被继承。其中方法有重载和重写的特性，重载是要求函数原型不能相同，而重写则要求函数原型一定相同，一般出现在子类中。static 关键字表明属性和方法为类所有，具有一定的全局性。而 final 关键字则表示变量或方法不能被修改，是一种最终形态。
2. Java 的集合框架，java 有类似 C++的 STL 数据结构，这便是集合框架，提供了 Map，
set，list，queue 等容器以及相应的迭代操作，这样，在使用数据结构的时候，就可以直接用了，而不用自己定义。
3. Java 异常处理，语法形式和 C++类似，都是 try， throw， catch， finally，其中 finally 语句表示异常处理之后最后要进行的动作。异常处理能够解决很多运行时发生的问题。
关于一些 Java 的语法，多数和 C++类似，就没有深入研究了，至于 Java 的高级特性，如多线程，发射，网络编程等问题，我相信随着学习的深入，也会漫漫弄懂的。

Ant

Ant 可以看作 JAVA 的 makefile，它能够编写一系列 target 用于完成 java 源程序的自动编译，运行，打包等一系列任务，从而简化构建，部署等操作。
第一步，下载并解压 Ant 程序，添加环境变量:

```
export ANT_HOME=/home/zhoujihao/Software/ant
export PATH=$PATH:$ANT_HOME/bin
```

输入 ant 命令之后就可以检测是否安装成功。
第二步，学习 Ant 的基本语法
Ant 是基于 XML 的语法进行编写的，默认名称为 build.xml，主标签为 preject，规定项目的名字，默认的 target，和 basedir 等等，target 标签定义一系列任务，echo 标签输出一些 message，property 规定一些列键值对，可以用来规定目录的结构。
第三步，为 gridworld 项目编写 build.xml 文件，实现构建过程的自动化。

```xml
<?xml version="1.0"?>
<!--Compile all .java files in ${src} and put the .class files to ${classes}
Then package the .class files to package.jar in ${lib}. Finally run the package.jar
2015-08-19 ZhouJihao-->
<!-- Project Name and default target-->
<project name="HelloWorld" default="run" basedir=".">

    <property name="base" value="${basedir}"></property>
    <property name="src" value="${base}/boxBug"></property>
    <property name="classes" value="${base}/classes"></property>
    <property name="lib" value="${base}/lib"></property>
    <property name="outputjar" value="${base}/lib/package.jar"></property>

    <!--Classpath-->
    <path id="gridworld.classpath">
        <pathelement location="../../gridworld.jar"/>
    </path>

    <!-- <target name="all" depends="clean,init,compile,jar,run"></target>-->
    <target name="init" depends="clean">
        <mkdir dir="${classes}"></mkdir>
        <mkdir dir="${lib}"></mkdir>
    </target>

    <target name="compile" depends="init">
        <javac srcdir="${src}" destdir="${classes}"><classpath refid="gridworld.classpath"/></javac>
    </target>

    <target name="jar" depends="compile">
        <jar jarfile="${outputjar}">
            <fileset dir="${classes}"/>
            <manifest>
                <attribute name="Built-By" value="${user.name}"/>
                <attribute name="Main-Class" value="BoxBugRunner"/>
            </manifest>
        </jar>
    </target>
```

```xml
    <target name="run" depends="jar">
        <java classname="BoxBugRunner" fork="true">
            <classpath>
                <fileset dir="${lib}">
                    <include name="package.jar"/>
                </fileset>
            </classpath>

            <classpath>
                <fileset dir="${base}/../..">
                    <include name="gridworld.jar"/>
                </fileset>
            </classpath>
        </java>
    </target>

    <target name="clean">
        <delete dir="${classes}"/>
        <delete dir="${lib}"/>
    </target>
</project>
```

在上面的文件中，设置了 clean init compile jar run 这些 target，从而实现了项目的自动编译，打包和运行。难点在于对 gridworld.jar 的引用，也即如何设置 classpath。后面通过直接引用和简介引用就可以解决了。

在使用的时候，要修改 project name，src 目录和主类的名字。

Junit

Junit 也是以 jar 包形式存在的，所以在编译单元测试文件的时候，classpath 引用这个 jar 包即可。
编译时候的命令如下所示：

```
javac -cp .:./../junit-4.9.jar HelloWorldTest.java
java -cp .:./../junit-4.9.jar -ea org.junit.runner.JUnitCore HelloWorldTest
```

即可进行单元测试。

至于 Junit 的语法，则有以下几个要点。
首先是使用 Annotation 关键字，@Before 表示执行每个测试之前都要进行的动作，@Test 表示一个测试单元,@After 表示在每个单元测试完成之后要执行的动作,@Test(timeout=xxx)表示测试要在规定时间内完成，否则也算失败，@Ignore 表示下面的测试方法会被忽略，说明该方法还没有完成，或者在等待其他操作的完成。其次，常用的测试语句有 assertEquals(expected_value, getValue())等等。
下面是一个计算器程序的单元测试结果：

```
JUnit version 4.9
I.E...E
Time: 1.019
There were 2 failures:
1) testSquareRoot(CalculatorTest)
java.lang.Exception: test timed out after 1000 milliseconds

2) testSubstract(CalculatorTest)
java.lang.AssertionError: expected:<8> but was:<9>

FAILURES!!!
Tests run: 4,  Failures: 2
```