阶段一自学报告

Vi 部分

Vi 部分的内容绝逼是令人蛋疼的，一开始不熟悉就使用 vi 进行环境配置，总是会出问题，比如输入的空格多了，当使用 x 键删除内容之后再添加新的东西时总是会添加到光标之前。还有就是之前一直把 vi 当成是 vim，事实证明我还是 too naïve.

而我初次接触，只是基本掌握了一点点的的 vi 的知识。比如命令行模式和插入模式的互换。一开始使用 **vi** 命令打开文件是进入命令行模式的，按 i 键则切换到插入模式。这时候 就能够进行编辑了。如果说想要删除文字，那么就得回退到命令行模式，在命令行模式下可以移动光标的位置，也可以按 x 键删除光标所在位置的后面的一个字符，还可以按 dd 删除光标所在的一整行的内容。
对于碰到的那个“使用 x 键删除内容之后再添加新的东西时总是会添加到光标之前。”这个问题，那就是在删除内容之后想进入插入模式，按 a 键，那么之后插入的内容将会出现在光标后面。

最后就是退出 vi 编辑器啦。在命令行模式下，按冒号键，之后输入 q!就可以退出啦。当然注意在退出前要对修改的内容进行保存，那么就应该输入 wq，先保存，再退出。

Ubantu 一般自带 vi 编辑器，如果想使用 vim 编辑器而刚好系统又没有的话，可以在 shell 中输入 apt-get install vim 来安装 vim。

Java 部分

一、Java 感觉和 C++确实挺像的，都是注重封装，继承和多态。当然，由于现在学的只是一点的基础和皮毛。
个人在初级入门阶段觉得特别需要注意的地方主要有：
1. Java 程序对于大小写的敏感性。
2. Public static void main(String args[])：java 程序都是从 main()方法开始处理的。
3. 程序文件名一定要和类的名称保持一致。

二、对了，对于 java 程序的编译语句是 javac **Name_Of_Your_Programe**.java 这样的话将会在当前目录将你的 java 程序编译成 **Name_Of_Your_Programe**.Class.
对于 java 程序的运行语句是 java **Name_Of_Your_Programe** 它将会将运行你之前编译出来的类。

三、对于 java 和 jdk 的安装一直以来对我们来说都至为关键，环境变量写错，那么实验根本就没法继续进行下去。
看一下内容的时候请特别注意，部分命令需要 root 权限。本人默认你对 ubantu 或 linux 系统有一点初级知识，于此不进行过多说明。

1.我安装 jdk 版本的是 jdk-6u45-linux-x64.bin。(系统是 64 位的)
(1)在安装 jdk 之前需要先增加该文件的可执行权限,所以使用命令行 chmod 755 jdk-6545-linux-x64.bin
(2)之后就是执行 jdk-6u45-linux-x86.bin
使用命令行 ./jdk-6u45-linux-x86.bin
(3) 将生成的 jdk1.6.0_13 目录移动到/opt 目录下
使用命令行 mv ./jdk1.6.0_45 /opt　　注意空格

2.之后就是配置环境变量
1.使用 vi /etc/profile 配置 profile（电脑每次运行 ubantu 系统的时候就会自动利用 profile 文件将系统配置一次）
在文件下面添加这么一段内容：

```
JAVA_HOME=/opt/jdk1.6.0_45
PATH=$JAVA_HOME/bin:$PATH
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export JAVA_HOME ANT_HOME PATH CLASSPATH
```

之后回到 shell，先 source /etc/profile，之后重启电脑

2.修改.bashrc 文件（只对个人用户有效）

```
JAVA_HOME=/opt/jdk1.6.0_45
PATH=$JAVA_HOME/bin:$PATH
CLASSPATH=.$JAVA_HOME/lib/dt.jar:JAVA_HOME/lib/tools.jar
export JAVA_HOME
export PATH
export CLASSPATH
                                                    120,1         Bot
```

之后就可以喽
当然，最简单是使用包管理进行安装，命令行是 sudo apt-get install openjdk-6-jdk

Ant 部分

Ant 为什么我觉得很像是之前学的 Makefile 呢。ant 中的 depends 功能能够有效地将不同的指令联系在一起，同时多个 target，可以完成诸如编译运行的操作，在唯一的 project 中通过设置 default，可以控制程序的执行顺序，同时在终端中，可以看到指令执行的顺序和执行结果。除此之外，ant 具有类似于 tar 的压缩功能，能够将多个 classses 压缩并创建.jar 文件。

不管那么多了，首先我下载的是 apache-ant-1.7.1-bin.tar.gz 的版本，之后就是解压和安装啦。
我是把它解压并安装到目录/usr/local/ant,
之后再添加系统环境变量

```
export ANT_HOME=/usr/local/ant
export PATH=$PATH:$ANT_HOME/bin
```

最后再重新登录系统
测试

```
lucas@lucas-virtual-machine:~$ ant
Buildfile: build.xml does not exist!
Build failed
lucas@lucas-virtual-machine:~$
```

发现 ant 已经成功安装和配置。

之后我尝试使用 Ant 的 javac 任务来编译 java 程序。
首先我们建立名为 antstudy 的 Java 工程，在工程根目录下建立 src 目录为源代码目录，在 src 目录下建立 HelloWorld.java 这个类文件。该类文件的内容如下：

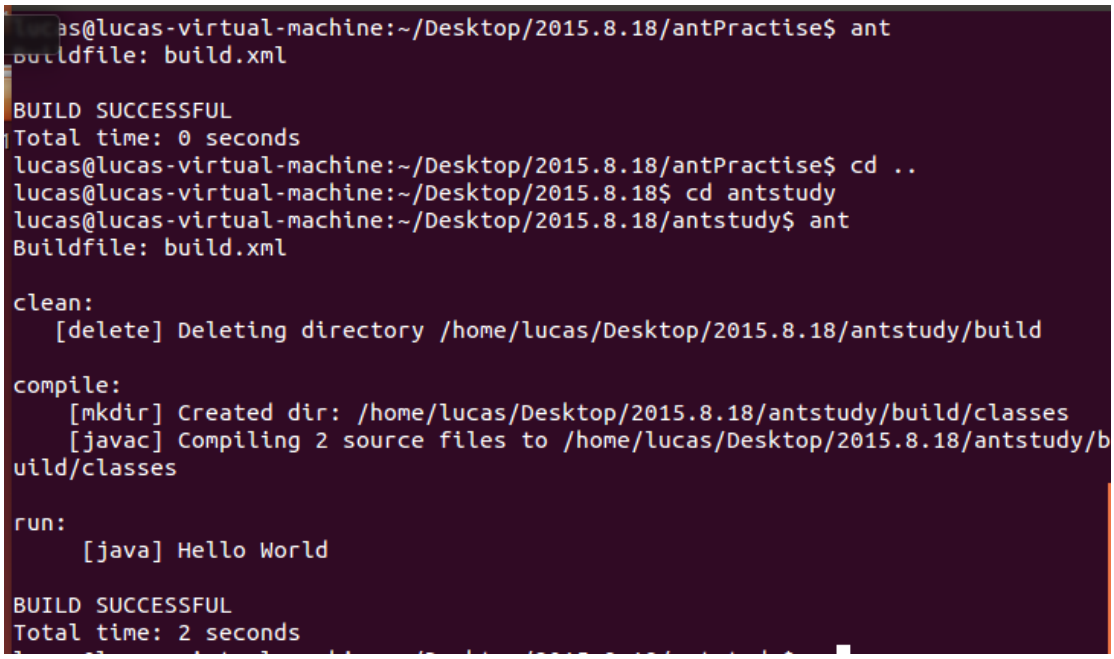
```
public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

同时在 antstudy 工程的根目录下建立 build.xml 文件，在编译前，会自动清除 classes 目录。之后在该文件中编译 src 目录下的 java 文件，并将编译后的 class 文件放入 build/classes 目录中，之后再运行项目。该文件的内容如下：

```
<?xml version="1.0"?>

<project name="javacTest" default="run" basedir=".">


    <target name="clean">

       <delete dir="build"/>

    </target>

    <target name="compile" depends="clean">

       <mkdir dir="build/classes"/>

    <javac srcdir="src" destdir="build/classes"/>

    </target>

   <target name="run" depends="compile">

       <java classname="HelloWorld">

           <classpath>

              <pathelement path="build/classes"/>
```

```
            </classpath>
        </java>
    </target>
</project>
```

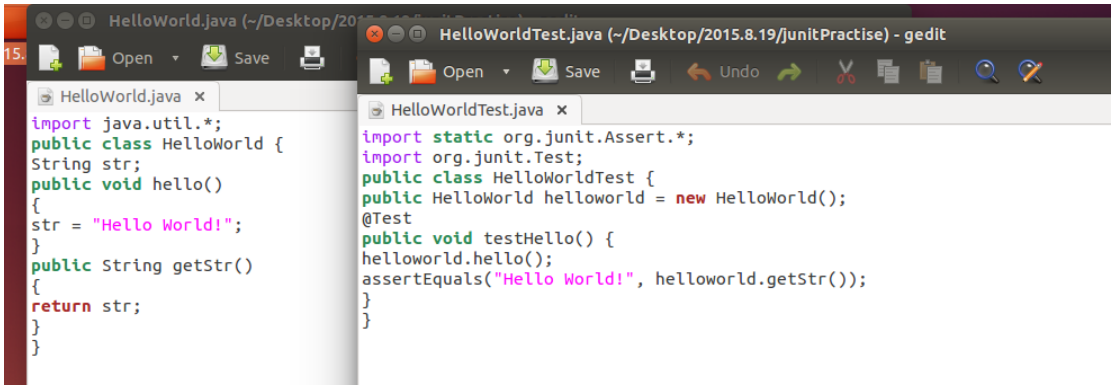
由下图可知成功运行



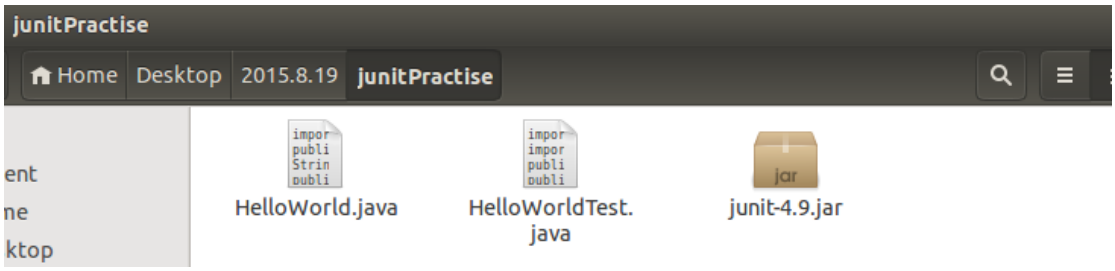
Junit 部分

JUnit 的主用用途在于对程序的单元测试，正如资料上所言，当我们有成百上千个函数时，使用 JUnit,有助于我们对每个函数进行测试，从而避免整个程序出现难以查找的 bug.对于今后的代码编写，JUnit 的出现和使用，能够便于代码的检测与维护。

我为了验证环境是否配置正确，编写一个类 HelloWorld.java 以及一个测试类 HelloWorldTest.java。代码如下：



然后将这两个类以及下载的 junit 中的 junit-4.9.jar 放在同个文件夹中



最后进入该文件夹中，然后使用命令行

javac –classpath .:junit-4.9.jar HelloWorldTest.java 进行编译

java –classpath .:junit-4.9.jar –ea org.junit.runner.JUnitCore HelloWorldTest 进行运行

成功测试截图如下：