Ant
再目录下执行 ant 默认寻找的 build.xml
但是也可以指定文件路径 ant -file build2.xml
ant 可以用来编译和运行 java 文件 类似 c/c++的 makefile 文件
看 ant 首先关注的是 project 标签 然后找 target 标签 再看 project 是否有 fefault 属性，然后找对应的 target 来理解 整个 build.xml 的运行过程
声明属性就相当于申明变量用$()来引用变量，可以简化代码

```
<?xml version="1.0"?>
//申明任务 可以用 description 来注释
<project name="GridWorld" default="make-zip" basedir=".">
   <property file="build.properties"/>
//申明很多的属性 以便使用
   <property name="framework.dir" value="framework"/>
   <property name="projects.dir" value="projects"/>
   <property name="build.dir" value="build"/>
   <property name="dist.dir" value="dist"/>
   <property name="zip.name" value="GridWorldCode"/>
   <property name="dist.zip.dir" value="${dist.dir}/${zip.name}"/>
   <property name="package.name" value="info.gridworld"/>
   <property name="framework.resources"
             value="${framework.dir}/info/gridworld/gui/WorldFrameResources.properties" />
   //一个 init 目标
   <target name="init">
   //时间戳
      <tstamp>
         <format property="version.date" pattern="yyyy-MM-dd" locale="en"/>
      </tstamp>
      //创建三个目录
      <mkdir dir="${build.dir}"/>
      <mkdir dir="${dist.dir}"/>
      <mkdir dir="${dist.zip.dir}"/>
   </target>

   //这里是正则
   <target name="props" depends="init" description="Set version-specific properties.">
      <replaceregexp file="${framework.resources}" byline="true">
         <regexp pattern="(version.id\s*=\s*).*"/>
         <substitution expression="\1${version.id}"/>
      </replaceregexp>
      <replaceregexp file="${framework.resources}" byline="true">
         <regexp pattern="(version.date\s*=\s*).*"/>
         <substitution expression="\1${version.date}"/>
      </replaceregexp>
   </target>
  // 编译 java 源文件
   <target name="compile" depends="props">
      <javac srcdir="${framework.dir}" destdir="${build.dir}" debug="true" target="1.5">
         <compilerarg value="-Xlint:unchecked"/>
      </javac>
   </target>

   // 生成 jar 文件
   <target name="build-jar" depends="compile">
      <copy todir="${build.dir}">
         <fileset dir="${framework.dir}">
            <include name="**/*.gif"/>
```

```
            <include name="**/*.properties"/>
            <include name="**/*.html"/>
         </fileset>
      </copy>

      <jar destfile="${dist.zip.dir}/gridworld.jar" basedir="${build.dir}"/>
   </target>
// 生成文本当
   <target name="javadoc" depends="build-jar">
      <javadoc
            destdir="${dist.zip.dir}/javadoc"
            packagenames="${package.name}.*"
            sourcepath="${framework.dir}"
            excludepackagenames="${package.name}.gui.*"
            link="${java.api.url}"/>
   </target>
//制作压缩包
   <target name="make-zip" depends="javadoc">
      <copy todir="${dist.zip.dir}/projects">
         <fileset dir="${projects.dir}">
            <include name="**/*.java"/>
            <include name="**/*.gif"/>
         </fileset>
      </copy>
      <copy todir="${dist.zip.dir}/framework">
         <fileset dir="${framework.dir}">
            <include name="**/*.java"/>
            <include name="**/*.gif"/>
            <include name="**/*.properties"/>
         </fileset>
      </copy>
      <copy todir="${dist.zip.dir}">
         <fileset dir="${basedir}">
            <include name="build.xml"/>
            <include name="build.properties"/>
         </fileset>
      </copy>
      <delete file="${dist.dir}/${zip.name}.zip" />
      <zip destfile="${dist.dir}/${zip.name}.zip" basedir="${dist.dir}">
      </zip>
   </target>
// 清楚目录
   <target name="clean">
      <delete dir="${build.dir}"/>
      <delete dir="${dist.dir}"/>
   </target>
</project>
```

java

java 有类似 c++的封装和继承
java 种大写开头的数据类型 都是类，
类是引用赋值
数组的初始化 不太一样
java 有点类似 web 种的 html css js 的集合，基本上能变种实现 web 的网页
java 有很多库 也可以自己生成库 这样能有效的避免冲突

junit

junit 用来测试一个 java 的类的方法是否正确

一个例子解析

```
//下面这个用于断言函数的调用
import static org.junit.Assert.*;
// 下面这两个包 是对与自己选择运行器 准备的
//import org.junit.internal.runners.TestClass; 发现过时
import org.junit.runner.RunWith;

//把若干种情况变成参数一次性测试，因为一个测试只有一个断言可以使用
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import java.util.Arrays;
import java.util.Collection;

// 就是那些标志
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;

// 调用运行器 这里可省 和不加是一样的效果 加了这里 也要加对应的 package
/////@RunWith(TestClassRunner.class)

// 使用参数的方法时添加的 因为添加了特殊功能嘛
@RunWith(Parameterized.class)
public class CalculatorTest {
    private static Calculator calculator = new Calculator();
    private int param;
    private int result;
        //每次测试都要预先执行这个函数
    @Before
    public void setUp() throws Exception {
        calculator.clear();
    }

    //参量形式 最后会以每一组数据测试一次  所以就是相当于测试 test 数乘以3
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][]{
            {2, 4},
            {0, 0},
            {-3, 9},
        });
    }
   //构造函数,对变量进行初始化 顺序和参量的顺序相同 是相当于
    public CalculatorTest(int param, int result) {
        this.param = param;
        this.result = result;
    }
```

```
    @Test
    public void testAdd() {
        calculator.add(2);
        calculator.add(3);
        assertEquals(5, calculator.getResult());
    }

    @Test
    public void testSubstract() {
        calculator.add(10);
        calculator.substract(2);
        assertEquals(8, calculator.getResult());
    }

    // 可以忽略测试   并且用于提示
    @Ignore("Multiply() Not yet implemented")
    @Test
    public void testMultiply() {
    }

    @Test
    public void testDivide() {
        calculator.add(8);
        calculator.divide(2);
        assertEquals(4, calculator.getResult());
    }

    @Test(timeout = 1000)
    public void squareRoot(){
        calculator.squareRoot(4);
        assertEquals(2, calculator.getResult());
    }

    @Test(expected = ArithmeticException.class)
    public void divideByZero() {
        calculator.divide(0);
    }

    @Test
    public void square() {
        calculator.square(param);
        assertEquals(result, calculator.getResult());
    }
}
```

用这个 junit 测试还是比较智能，不过对于检测的质量 还是依赖你给定的例子，毕竟你给定的例子不能覆盖所有值，所以有些时候感觉不是很准确

vim

对于 vi 的学习,也算是水到渠成,基本的操作都会

:wq 保存退出
:sp 水平分割窗口
:vs 垂直分割窗口
shift+ g 跳到文件最后一行
shift+ G 跳刀文件第一行
i 进入插入
ctrl+v 一片区域的删除或者复制
n+g 跳到第 n 行
dd 删除当前行
num+dd 删除 n 行
num+yy 复制 n 行

配置文件 带有 IDE 的功能自动补全 和 增加字典

/home/zhoutengteng/.vimrc

```
set sw=4
set ts=4
set et
set smarttab
set smartindent
set lbr
set fo+=mB
set sm
set selection=inclusive
set wildmenu
set mousemodel=popup
set nu

au FileType php setlocal dict+=~/.vim/dict/php_funclist.dict
au FileType css setlocal dict+=~/.vim/dict/css.dict
au FileType c setlocal dict+=~/.vim/dict/c.dict
au FileType cpp setlocal dict+=~/.vim/dict/cpp.dict
au FileType scale setlocal dict+=~/.vim/dict/scale.dict
au FileType javascript setlocal dict+=~/.vim/dict/javascript.dict
au FileType html setlocal dict+=~/.vim/dict/javascript.dict
au FileType html setlocal dict+=~/.vim/dict/css.dict
au FileType java setlocal dict+=~/.vim/dict/java.dict
au FileType java setlocal dict+=~/Desktop/java_test/api/*

autocmd FileType python set omnifunc=pythoncomplete#Complete
autocmd FileType javascript set omnifunc=javascriptcomplete#CompleteJS
```

```
autocmd FileType html set omnifunc=htmlcomplete#CompleteTags
autocmd FileType css set omnifunc=csscomplete#CompleteCSS
autocmd FileType xml set omnifunc=xmlcomplete#CompleteTags
autocmd FileType php set omnifunc=phpcomplete#CompletePHP
autocmd FileType c set omnifunc=ccomplete#Complete
```