## Day01\_基础加强

### 0.1学习目标

1、能够掌握junit单元测试

2、能够掌握反射原理

3、能够掌握注解的应用

### 0.2 学习指南

* Junit单元测试（重要）
* 反射（重要）
* 注解（重点）

### 第一堂课

本节知识点

1、今日内容

2、Junit\_测试概述

3、Junit\_使用步骤

4、Junit\_@Before&@After

本节目标

Junit的使用

#### 今日内容

##### 1.1.1知识概述

今天主要是基础加强

##### 1.1.2视频详情



##### 1.1.3总结与补充

无

##### 1.1.4课堂提问与练习

##### 1.1.5习题答案

##### 1.1.6视频缺陷

无

##### 1.1.7视频扩展

无

#### Junit\_测试概述

##### 1.2.1知识概述

 测试分类：  
        1. 黑盒测试：不需要写代码，给输入值，看程序是否能够输出期望的值。  
        2. 白盒测试：需要写代码的。关注程序具体的执行流程。

##### 1.2.2视频详情



##### 1.2.3总结与补充

测试就是对所完成功能的校验，查看功能是否有缺陷有漏洞。在工作中，每次做完功能后都要进行测试，测试通过才可以结束该功能的编写。测试是开发中很重要的一部分。

##### 1.2.4课堂提问与练习

什么是黑盒和白盒测试？

##### 1.2.5习题答案

黑盒测试：不需要写代码，给输入值，看程序是否能够输出期望的值。  
        2. 白盒测试：需要写代码的。关注程序具体的执行流程。

##### 1.2.6视频缺陷

无

##### 1.2.7视频扩展

无

#### Junit\_使用步骤

##### 1.3.1知识概述

 步骤：  
            1. 定义一个测试类(测试用例)  
                \* 建议：  
                    \* 测试类名：被测试的类名Test        CalculatorTest  
                    \* 包名：xxx.xxx.xx.test        cn.itcast.test  
  
            2. 定义测试方法：可以独立运行  
                \* 建议：  
                    \* 方法名：test测试的方法名        testAdd()    
                    \* 返回值：void  
                    \* 参数列表：空参  
  
            3. 给方法加@Test  
            4. 导入junit依赖环境  
  
        \* 判定结果：  
            \* 红色：失败  
            \* 绿色：成功  
            \* 一般我们会使用断言操作来处理结果  
                \* Assert.assertEquals(期望的结果,运算的结果);

##### 1.3.2视频详情



##### 1.3.3总结与补充

进行单元测试时，一定先要引入Junit包才可以使用。并且该注解必须在方法名上。

##### 1.3.4课堂提问与练习

* Junit\_使用步骤

##### 1.3.5习题答案

@Test

public void testAdd(){

// System.out.println("我被执行了");

//1.创建计算器对象

System.out.println("testAdd...");

Calculator c = new Calculator();

//2.调用add方法

int result = c.add(1, 2);

//System.out.println(result);

//3.断言 我断言这个结果是3

Assert.assertEquals(3,result);

}

##### 1.3.6视频缺陷

无

##### 1.3.7视频扩展

无

#### Junit\_@Before&@After

##### 1.4.1知识概述

 @Before:  
                \* 修饰的方法会在测试方法之前被自动执行  
            \* @After:  
                \* 修饰的方法会在测试方法执行之后自动被执行

##### 1.4.2视频详情



##### 1.1.3总结与补充

@BeforeClass：针对所有测试，只执行一次，且必须为static void

@Before：初始化方法

@Test：测试方法，在这里可以测试期望异常和超时时间

@After：释放资源

@AfterClass：针对所有测试，只执行一次，且必须为static void

@Ignore：忽略的测试方法

一个单元测试用例执行顺序为： @BeforeClass –> @Before –> @Test –> @After –> @AfterClass

每一个测试方法的调用顺序为： @Before –> @Test –> @After

##### 1.4.4课堂提问与练习

* 练习Junit\_@Before&@After

##### 1.4.5习题答案

/\*\*

\* 初始化方法：

\* 用于资源申请，所有测试方法在执行之前都会先执行该方法

\*/

@Before

public void init(){

System.out.println("init...");

}

/\*\*

\* 释放资源方法：

\* 在所有测试方法执行完后，都会自动执行该方法

\*/

@After

public void close(){

System.out.println("close...");

}

##### 1.4.6视频缺陷

无

##### 1.4.7视频扩展

无

### 第二堂课

本节知识点：

1、反射\_概述

2、反射\_获取字节码Class对象的三种方式

3、反射\_Class对象功能概述

本节目标 ：

1、知道反射的相关概念

2、掌握反射获取class对象的三种方式（重点）

3、知道class对象的功能

#### 反射\_概述

##### 2.1.1知识概述

框架：半成品软件。可以在框架的基础上进行软件开发，简化编码  
    \* 反射：将类的各个组成部分封装为其他对象，这就是反射机制  
        \* 好处：  
            1. 可以在程序运行过程中，操作这些对象。  
            2. 可以解耦，提高程序的可扩展性。



##### 2.1.2视频详情



##### 2.1.3总结与补充

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

##### 2.1.4课堂提问与练习

* 反射的好处有哪些？

##### 2.1.5习题答案

好处：  
            1. 可以在程序运行过程中，操作这些对象。  
            2. 可以解耦，提高程序的可扩展性。

##### 2.1.6视频缺陷

无

##### 1.1.7视频扩展

无

#### 反射\_获取字节码Class对象的三种方式

##### 2.2.1知识概述

获取Class对象的方式：  
        1. Class.forName("全类名")：将字节码文件加载进内存，返回Class对象  
            \* 多用于配置文件，将类名定义在配置文件中。读取文件，加载类  
        2. 类名.class：通过类名的属性class获取  
            \* 多用于参数的传递  
        3. 对象.getClass()：getClass()方法在Object类中定义着。  
            \* 多用于对象的获取字节码的方式  
  
        \* 结论：  
            同一个字节码文件(\*.class)在一次程序运行过程中，只会被加载一次，不论通过哪一种方式获取的Class对象都是同一个。

##### 2.2.2视频详情



##### 2.2.3总结与补充

因为所有类都继承Object类。从而调用Object类来获取class字节码对象。

三种方式常用第三种，第一种对象都有了还要反射干什么。第二种需要导入类的包，依赖太强，不导包就抛编译错误。一般都第三种，一个字符串可以传入也可写在配置文件中等多种方法。

##### 2.2.4课堂提问与练习

* 练习获取字节码Class对象的三种方式

##### 2.2.5习题答案

Class cl1=Class.forName("cn.itcast.test.ClassTest01");

System.out.println(cl1);

Class cl2=ClassTest01.class;

System.out.println("cl2 "+cl2);

Class cl3=new ClassTest01().getClass();

System.out.println("cl3 "+cl3);

##### 2.2.6视频缺陷

无

##### 2.2.7视频扩展

无

#### 反射\_Class对象功能概述

##### 2.3.1知识概述

获取功能：

1. 获取成员变量们

\* Field[] getFields() ：获取所有public修饰的成员变量

\* Field getField(String name) 获取指定名称的 public修饰的成员变量

\* Field[] getDeclaredFields() 获取所有的成员变量，不考虑修饰符

\* Field getDeclaredField(String name)

2. 获取构造方法们

\* Constructor<?>[] getConstructors()

\* Constructor<T> getConstructor(类<?>... parameterTypes)

\* Constructor<T> getDeclaredConstructor(类<?>... parameterTypes)

\* Constructor<?>[] getDeclaredConstructors()

3. 获取成员方法们：

\* Method[] getMethods()

\* Method getMethod(String name, 类<?>... parameterTypes)

\* Method[] getDeclaredMethods()

\* Method getDeclaredMethod(String name, 类<?>... parameterTypes)

4. 获取全类名

\* String getName()

##### 2.3.2视频详情



##### 2.3.3总结与补充

通过class字节码对象可以获取该类的方法以及属性

##### 2.3.4课堂提问与练习

无

##### 2.3.5习题答案

无

##### 2.3.6视频缺陷

无

##### 2.3.7视频扩展

无

### 第三堂课

本节知识点

1、反射\_Class对象功能\_获取Field

2、反射\_Class对象功能\_获取Constructor

3、反射\_Class对象功能\_获取Method

本节目标

1、能够完成class对象的获取功能

#### 反射\_Class对象功能\_获取Field

##### 3.1.1知识概述

 操作：  
            1. 设置值  
                \* void set(Object obj, Object value)    
            2. 获取值  
                \* get(Object obj)   
  
            3. 忽略访问权限修饰符的安全检查  
                \* setAccessible(true):暴力反射

可以获取该类的成员变量

##### 3.1.2视频详情



##### 3.1.3总结与补充

1. 1.批量的
2. \*      1).Field[] getFields():获取所有的"公有字段"
3. \*      2).Field[] getDeclaredFields():获取所有字段，包括：私有、受保护、默认、公有；
4. \* 2.获取单个的：
5. \*      1).public Field getField(String fieldName):获取某个"公有的"字段；
6. \*      2).public Field getDeclaredField(String fieldName):获取某个字段(可以是私有的)

##### 3.1.4课堂提问与练习

* 练习Class对象获取Field

##### 3.1.5习题答案

Class personClass=Person.class;

Field[] fields=personClass.getFields();

for (Field field : fields) {//iter

System.out.println("getFileds(): "+field);

}

Field field=personClass.getField("a");

System.out.println("getField(): "+field);

System.out.println("=====================================================");

Field a=personClass.getField("a");

Person p=new Person();

Object value=a.get(p);

System.out.println(p);

a.set(p,1);

System.out.println(p);

System.out.println("=====================================================");

Field[] fiels2=personClass.getDeclaredFields();

for (Field field2 : fiels2) {

System.out.println("getDeclaredFields(): "+field2);

}

Field field3=personClass.getDeclaredField("name");

System.out.println("getDeclaredField(): "+field3);

##### 3.1.6视频缺陷

无

##### 3.1.7视频扩展

无

#### 反射\_Class对象功能\_获取Constructor

##### 3.2.1知识概述

 Constructor:构造方法  
        \* 创建对象：  
            \* T newInstance(Object... initargs)    
  
            \* 如果使用空参数构造方法创建对象，操作可以简化：Class对象的newInstance方法

获取构造方法们  
                \* Constructor<?>[] getConstructors()    
                \* Constructor<T> getConstructor(类<?>... parameterTypes)    
  
                \* Constructor<T> getDeclaredConstructor(类<?>... parameterTypes)    
                \* Constructor<?>[] getDeclaredConstructors()

##### 3.2.2视频详情



##### 3.2.3总结与补充

构造器可以提供许多特殊的方法，构造器作为一种方法，负责类中成员变量（域）的初始化。实例构造器分为缺省构造器和非缺省构造器。

构造器最大的用处就是在创建对象时执行初始化，当创建一个对象时，系统会为这个对象的实例进行默认的初始化。如果想改变这种默认的初始化，就可以通过自定义构造器来实现。

##### 3.2.4课堂提问与练习

* 练习Class对象功能\_获取Constructor

##### 3.2.5习题答案

Class personClass=Person.class;

Constructor constructor=personClass.getConstructor();

Object p1=constructor.newInstance();

System.out.println(p1);

System.out.println("=========================================");

Constructor constructor1=personClass.getConstructor(String.class,String.class);

Object person1=constructor1.newInstance("aaa","bbb");

System.out.println(person1);

System.out.println("=========================================");

Object person=personClass.newInstance();

System.out.println(person);

##### 3.2.6视频缺陷

无

##### 3.2.7视频扩展

无

#### 反射\_Class对象功能\_获取Method

##### 3.3.1知识概述

Method：方法对象  
        \* 执行方法：  
            \* Object invoke(Object obj, Object... args)    
  
        \* 获取方法名称：  
            \* String getName:获取方法名

\* Method[] getMethods()    
         \* Method getMethod(String name, 类<?>... parameterTypes)    
  
          \* Method[] getDeclaredMethods()    
          \* Method getDeclaredMethod(String name, 类<?>... parameterTypes)

##### 3.3.2视频详情



##### 3.3.3总结与补充

获取该类的方法，并且通过反射来执行该方法。

使用method.getParameterTypes();可以获取该方法的参数

##### 3.3.4课堂提问与练习

* 练习class对象获取Method

##### 3.3.5习题答案

Class personClass=Person.class;

Method[] methods=personClass.getMethods();

for (Method method : methods) {

System.out.println(method);

}

System.out.println("=============================");

Method method01=personClass.getMethod("eat");

Person person=new Person();

method01.invoke(person);

System.out.println("==============================");

Method method02=personClass.getMethod("eat",String.class);

method02.invoke(person,"zhangsan")

##### 3.3.6视频缺陷

无

##### 3.3.7视频扩展

无

### 第四堂课

本节知识点：

1、反射\_案例

2、注解\_概念

3、注解\_JDK内置注解

本节目标 ：

1. 完成反射案例
2. 知道注解的相关概念
3. 能够使用JDK内置注解

#### 反射\_案例

##### 4.1.1知识概述

\* 需求：写一个"框架"，不能改变该类的任何代码的前提下，可以帮我们创建任意类的对象，并且执行其中任意方法  
            \* 实现：  
                1. 配置文件  
                2. 反射  
            \* 步骤：  
                1. 将需要创建的对象的全类名和需要执行的方法定义在配置文件中  
                2. 在程序中加载读取配置文件  
                3. 使用反射技术来加载类文件进内存  
                4. 创建对象  
                5. 执行方法

##### 4.1.2视频详情



##### 4.1.3总结与补充

首先要从配置文件中读取该类的全限定类名，然后用class.forName()，来获取该类的class对象，再通过class对象获取并执行成员方法。

##### 4.1.4课堂提问与练习

* 练习反射案例

##### 4.1.5习题答案

Properties properties=new Properties();

ClassLoader cl=ClassTest05.class.getClassLoader();

InputStream is=cl.getResourceAsStream("source.properties");

properties.load(is);

String className=properties.getProperty("className");

String method=properties.getProperty("methodName");

Class personClass=Class.forName(className);

Object person=personClass.newInstance();

Method method1=personClass.getMethod("eat");

method1.invoke(person);

##### 4.1.6视频缺陷

ClassLoader以前没有讲过在这个内容当中直接使用，学员很难理解。

##### 4.1.7视频扩展

类加载器，详情见资料

#### 注解\_概念

##### 4.2.1知识概述

概念：说明程序的。给计算机看的

\* 注释：用文字描述程序的。给程序员看的

\* 定义：注解（Annotation），也叫元数据。一种代码级别的说明。它是JDK1.5及以后版本引入的一个特性，与类、接口、枚举是在同一个层次。它可以声明在包、类、字段、方法、局部变量、方法参数等的前面，用来对这些元素进行说明，注释。

\* 概念描述：

\* JDK1.5之后的新特性

\* 说明程序的

\* 使用注解：@注解名称

\* 作用分类：

①编写文档：通过代码里标识的注解生成文档【生成文档doc文档】

②代码分析：通过代码里标识的注解对代码进行分析【使用反射】

③编译检查：通过代码里标识的注解让编译器能够实现基本的编译检查【Override】

##### 4.2.2视频详情



##### 4.2.3总结与补充

注解就是为了简化开发，避免写过多的代码，不利于程序的扩展以及维护

##### 4.2.4课堂提问与练习

* 注解的作用有哪些分类

##### 4.2.5习题答案

作用分类：  
        ①编写文档：通过代码里标识的注解生成文档【生成文档doc文档】  
        ②代码分析：通过代码里标识的注解对代码进行分析【使用反射】  
        ③编译检查：通过代码里标识的注解让编译器能够实现基本的编译检查【Override】

#### 注解\_JDK内置注解

##### 4.3.1知识概述

JDK中预定义的一些注解  
        \* @Override    ：检测被该注解标注的方法是否是继承自父类(接口)的  
        \* @Deprecated：该注解标注的内容，表示已过时  
        \* @SuppressWarnings：压制警告  
            \* 一般传递参数all  @SuppressWarnings("all")

##### 4.3.2视频详情



##### 4.3.3总结与补充

@Override

限定重写父类方法。对于子类中被@Override 修饰的方法，如果存在对应的被重写的父类方法，则正确；如果不存在，则报错。@Override 只能作用于方法，不能作用于其他程序元素。

@Deprecated

用于表示某个程序元素（类、方法等）已过时。如果使用了被@Deprecated修饰的类或方法等，编译器会发出警告。

@SuppressWarnings

抑制编译器警告。指示被@SuppressWarnings修饰的程序元素（以及该程序元素中的所有子元素，例如类以及该类中的方法.....）取消显示指定的编译器警告。例如，常见的@SuppressWarnings（value="unchecked"）

SuppressWarnings注解的常见参数值的简单说明：

1.deprecation：使用了不赞成使用的类或方法时的警告(使用@Deprecated使得编译器产生的警告)；

　　　　2.unchecked：执行了未检查的转换时的警告，例如当使用集合时没有用泛型 (Generics) 来指定集合保存的类型; 关闭编译器警告

　　　　3.fallthrough：当 Switch 程序块直接通往下一种情况而没有 Break 时的警告;

　　　　4.path：在类路径、源文件路径等中有不存在的路径时的警告;

　　　　5.serial：当在可序列化的类上缺少 serialVersionUID 定义时的警告;

　　　　6.finally：任何 finally 子句不能正常完成时的警告;

　　　　7.all：关于以上所有情况的警告。

##### 4.3.4课堂提问与练习

* 练习JDK内置注解的使用

##### 4.2.5习题答案

@SuppressWarnings("all")

public class AnnoDemo2 {

@Override

public String toString() {

return super.toString();

}

@Deprecated

public void show1(){

//有缺陷

}

public void show2(){

//替代show1方法

}

public void demo(){

show1();

Date date = new Date();

}

}

##### 4.2.6视频缺陷

无

##### 4.2.7视频扩展

无

### 第五堂课

本节知识点：

1、注解\_自定义注解\_格式&本质

2、注解\_自定义注解\_属性定义

3、注解\_自定义注解\_元注解

本节目标 ：

* 1. 掌握自定义注解的格式
  2. 能够完成自定义注解属性定义
  3. 知道元注解的作用

#### 注解\_自定义注解\_格式&本质

##### 5.1.1知识概述

\* 格式：

元注解

public @interface 注解名称{

属性列表;

}

\* 本质：注解本质上就是一个接口，该接口默认继承Annotation接口

\* public interface MyAnno extends java.lang.annotation.Annotation {}

##### 5.1.2视频详情



##### 5.1.3总结与补充

自定义注解本质是一个接口类。它的实现类在jvm运行时会自动帮我们创建它的实现类。

##### 5.1.4课堂提问与练习

* 编写代码实现自定义注解

##### 5.1.5习题答案

public @interface MyAnno {

int value();

Person per();

MyAnno2 anno2();

String[] strs();

/\*String name() default "张三";\*/

/\*String show2();

Person per();

MyAnno2 anno2();

String[] strs();\*/

}

#### 注解\_自定义注解\_属性定义

##### 5.2.1知识概述

属性：接口中的抽象方法  
            \* 要求：  
                1. 属性的返回值类型有下列取值  
                    \* 基本数据类型  
                    \* String  
                    \* 枚举  
                    \* 注解  
                    \* 以上类型的数组  
  
                2. 定义了属性，在使用时需要给属性赋值  
                    1. 如果定义属性时，使用default关键字给属性默认初始化值，则使用注解时，可以不进行属性的赋值。  
                    2. 如果只有一个属性需要赋值，并且属性的名称是value，则value可以省略，直接定义值即可。  
                    3. 数组赋值时，值使用{}包裹。如果数组中只有一个值，则{}可以省略

##### 5.2.2视频详情



##### 5.2.3总结与补充

自定义注解中的属性定义，可以在注解使用时给属性赋值，或者在编写自定义注解时给属性一个默认值。

##### 5.2.4课堂提问与练习

* 编写代码实现自定义注解

##### 5.2.5习题答案

无

public @interface MyAnno {

int value();

Person per();

MyAnno2 anno2();

String[] strs();

/\*String name() default "张三";\*/

/\*String show2();

Person per();

MyAnno2 anno2();

String[] strs();\*/

@MyAnno(value=12,per = Person.P1,anno2 = @MyAnno2,strs="bbb")

public class Worker {

public String name = "aaa";

public void show(){

}

}

}

##### 5.2.6视频缺陷

无

##### 5.2.7视频扩展

无

#### 注解\_自定义注解\_元注解

##### 5.3.1知识概述

 元注解：用于描述注解的注解  
            \* @Target：描述注解能够作用的位置  
                \* ElementType取值：  
                    \* TYPE：可以作用于类上  
                    \* METHOD：可以作用于方法上  
                    \* FIELD：可以作用于成员变量上  
            \* @Retention：描述注解被保留的阶段  
                \* @Retention(RetentionPolicy.RUNTIME)：当前被描述的注解，会保留到class字节码文件中，并被JVM读取到  
            \* @Documented：描述注解是否被抽取到api文档中  
            \* @Inherited：描述注解是否被子类继承

##### 5.3.2视频详情



##### 5.3.3总结与补充

1.@Retention: 定义注解的保留策略

@Retention(RetentionPolicy.SOURCE) //注解仅存在于源码中，在class字节码文件中不包含

@Retention(RetentionPolicy.CLASS) // 默认的保留策略，注解会在class字节码文件中存在，但运行时无法获得，

@Retention(RetentionPolicy.RUNTIME) // 注解会在class字节码文件中存在，在运行时可以通过反射获取到

首 先要明确生命周期长度 SOURCE < CLASS < RUNTIME ，所以前者能作用的地方后者一定也能作用。一般如果需要在运行时去动态获取注解信息，那只能用 RUNTIME 注解；如果要在编译时进行一些预处理操作，比如生成一些辅助代码（如 ButterKnife），就用 CLASS注解；如果只是做一些检查性的操作，比如 @Override 和 @SuppressWarnings，则可选用 SOURCE 注解。

2.@Target：定义注解的作用目标

源码为：

@Documented

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.ANNOTATION\_TYPE)

public @interface Target {

ElementType[] value();

}

@Target(ElementType.TYPE) //接口、类、枚举、注解

@Target(ElementType.FIELD) //字段、枚举的常量

@Target(ElementType.METHOD) //方法

@Target(ElementType.PARAMETER) //方法参数

@Target(ElementType.CONSTRUCTOR) //构造函数

@Target(ElementType.LOCAL\_VARIABLE)//局部变量

@Target(ElementType.ANNOTATION\_TYPE)//注解

@Target(ElementType.PACKAGE) ///包

3.@Document：说明该注解将被包含在javadoc中

4.@Inherited：说明子类可以继承父类中的该注解

##### 5.3.4课堂提问与练习

编写代码实现自定义注解元注解

##### 5.3.5习题答案

@Target({ElementType.***TYPE***,ElementType.***METHOD***,ElementType.***FIELD***})  
@Retention(RetentionPolicy.***SOURCE***)  
@Documented  
@Inherited  
**public** @**interface** MyAnno3 {  
}

##### 5.3.6视频缺陷

无

##### 5.3.7视频扩展

无

### 第六堂课

本节知识点：

1. 注解\_解析注解.
2. 注解\_案例\_简单的测试框架

本节目标 ：

1. 能够完成注解的解析以及案例

#### 注解\_解析注解

##### 6.1.1知识概述

在程序使用(解析)注解：获取注解中定义的属性值

1. 获取注解定义的位置的对象 （Class，Method,Field）

2. 获取指定的注解

\* getAnnotation(Class)

//其实就是在内存中生成了一个该注解接口的子类实现对象

public class ProImpl implements Pro{

public String className(){

return "cn.itcast.annotation.Demo1";

}

public String methodName(){

return "show";

}

}

3. 调用注解中的抽象方法获取配置的属性值

##### 6.1.2视频详情



##### 6.1.3总结与补充

首先要获取注解中属性的值，再根据获取到的值通过反射来创建对象，并调用对象的方法来实现功能。

##### 6.1.4课堂提问与练习

编写代码实现解析注解

##### 6.1.5习题答案

public static void main(String[] args) throws Exception {

Class<ReflectTest03> reflectTest03Class=ReflectTest03.class;

Pro an=reflectTest03Class.getAnnotation(Pro.class);

String className=an.className();

String methodName=an.methodName();

System.out.println(className);

System.out.println(methodName);

Class cls=Class.forName(className);

Object person=cls.newInstance();

Method method=cls.getMethod(methodName);

method.invoke(person);

}

##### 6.1.6视频缺陷

无

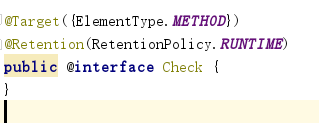
##### 6.1.7视频扩展

无

#### 注解\_案例\_简单的测试框架

##### 6.2.1知识概述

自定义注解：



定义的测试类：



执行测试：



##### 6.2.2视频详情



##### 6.2.3总结与补充

无

##### 6.2.4课堂提问与练习

* 编写代码实现测试框架案例

##### 6.2.5习题答

参考知识概述

##### 6.2.6视频缺陷

无

##### 6.2.7视频扩展

无