单元测试、反射、注解、动态代理

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

今日课程同学们需要学会什么

开发好的系统中存在很多的方法，如何对这些方法的正确性进行测试。

如何在程序运行时去得到Class对象，然后去获取Class中的每个成分。

注解是什么，具体是如何在Java程序中解决问题的？

框架技术的底层会用到的。

目录

Contents

- 单元测试
  - 单元测试概述
  - 单元测试快速入门
  - 单元测试常用注解
- 反射
- 注解
- 动态代理

单元测试

- 单元测试就是针对最小的功能单元编写测试代码，Java程序最小的功能单元是方法，因此，单元测试就是针对Java方法的测试，进而检查方法的正确性。

目前测试方法是怎么进行的，存在什么问题

- 只有一个main方法，如果一个方法的测试失败了，其他方法测试会受到影响。
- 无法得到测试的结果报告，需要程序员自己去观察测试是否成功。
- 无法实现自动化测试。

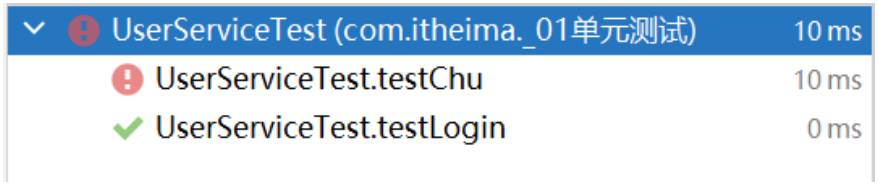
```
public class Test {
    public static void main(String[] args) {
        // 查询所有的方法测试
        findAllStudent();
        // 添加学生
        addStudent() ;
        // 修改学生
        updateStudent();
        // 删除学生
        deleteStudent() ;
    }
    // 测试删除学生
    private static void deleteStudent() {...}
    // 测试修改学生
    private static void updateStudent() {...}
    // 测试添加学生的方法
    private static void addStudent() {...}
    // 查询所有的学生数据
    private static void findAllStudent() {...}
}
```

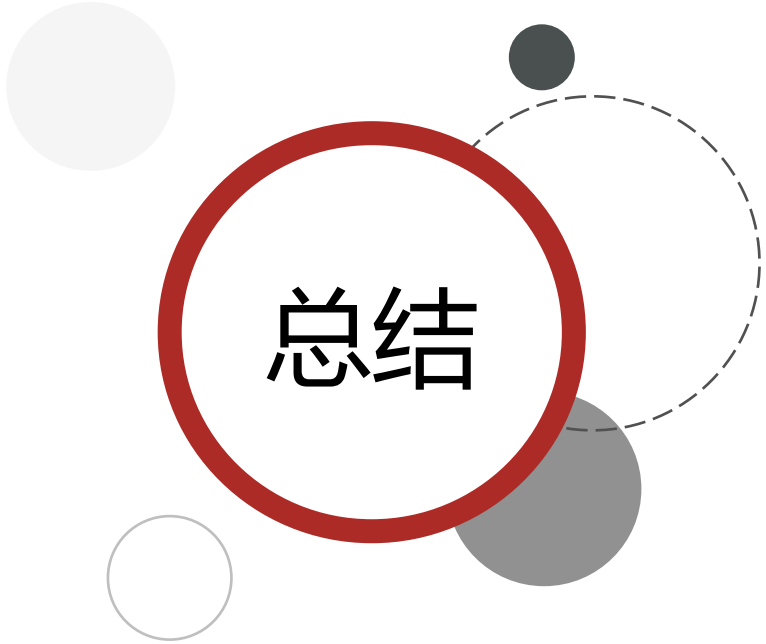
Junit单元测试框架

- JUnit是使用Java语言实现的单元测试框架，它是开源的，Java开发者都应当学习并使用JUnit编写单元测试。
- 此外，几乎所有的IDE工具都集成了JUnit，这样我们就可以直接在IDE中编写并运行JUnit测试，JUnit目前最新版本是5。

JUnit优点

- JUnit可以灵活的选择执行哪些测试方法，可以一键执行全部测试方法。
- Junit可以生成全部方法的测试报告。
- 单元测试中的某个方法测试失败了，不会影响其他测试方法的测试。





1. Junit单元测试是做什么的？

- **测试类中方法的正确性的。**

2. Junit单元测试的优点是什么？

- **JUnit可以选择执行哪些测试方法，可以一键执行全部测试方法的测试。**
- **JUnit可以生测试报告，如果测试良好则是绿色；如果测试失败，则是红色**

  。
- **单元测试中的某个方法测试失败了，不会影响其他测试方法的测试。**

目录

Contents

- **单元测试**
  - 单元测试概述
  - 单元测试快速入门
  - 单元测试常用注解
- **反射**
- **注解**
- **动态代理**

步骤

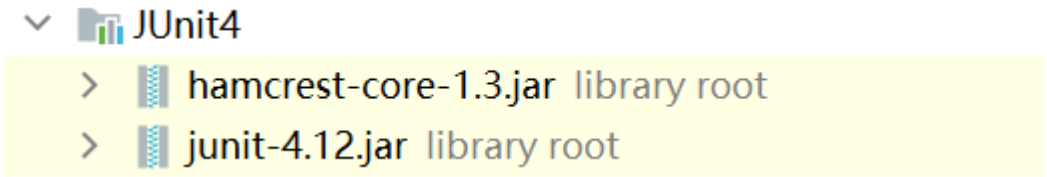
单元测试快速入门

需求：使用单元测试进行业务方法预期结果、正确性测试的快速入门

分析：

① 将JUnit的jar包导入到项目中

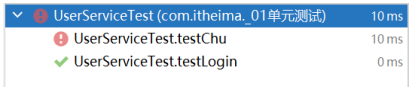
- IDEA通常整合好了Junit框架，一般不需要导入。
- 如果IDEA没有整合好，需要自己手工导入如下2个JUnit的jar包到模块



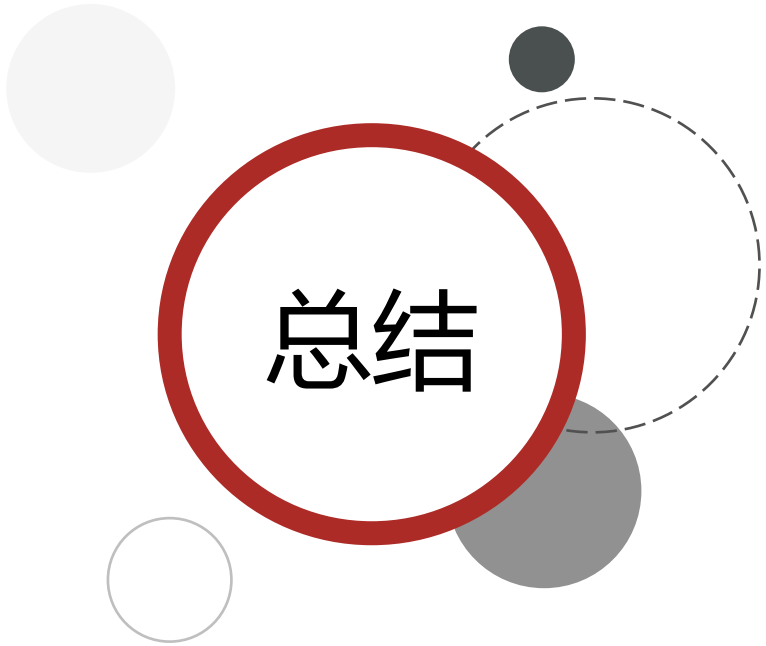
② 编写测试方法：该测试方法必须是公共的无参数无返回值的非静态方法。

③ 在测试方法上使用@Test注解：标注该方法是一个测试方法

④ 在测试方法中完成被测试方法的预期正确性测试。



⑤ 选中测试方法，选择“JUnit运行”，如果**测试良好**则是**绿色**；如果**测试失败**，则是**红色**



1. JUnit单元测试的实现过程是什么样的？

- **必须导入Junit框架的jar包。**
- **定义的测试方法必须是无参数无返回值，且公开的方法。**
- **测试方法使用@Test注解标记。**

2. JUnit测试某个方法，测试全部方法怎么处理？成功的标志是什么

- **测试某个方法直接右键该方法启动测试。**
- **测试全部方法，可以选择类或者模块启动。**
- **红色失败，绿色通过。**

目录

Contents

- 单元测试
  - 单元测试概述
  - 单元测试快速入门
  - 单元测试常用注解
- 反射
- 注解
- 动态代理

Junit常用注解(Junit 4.xxxx版本)

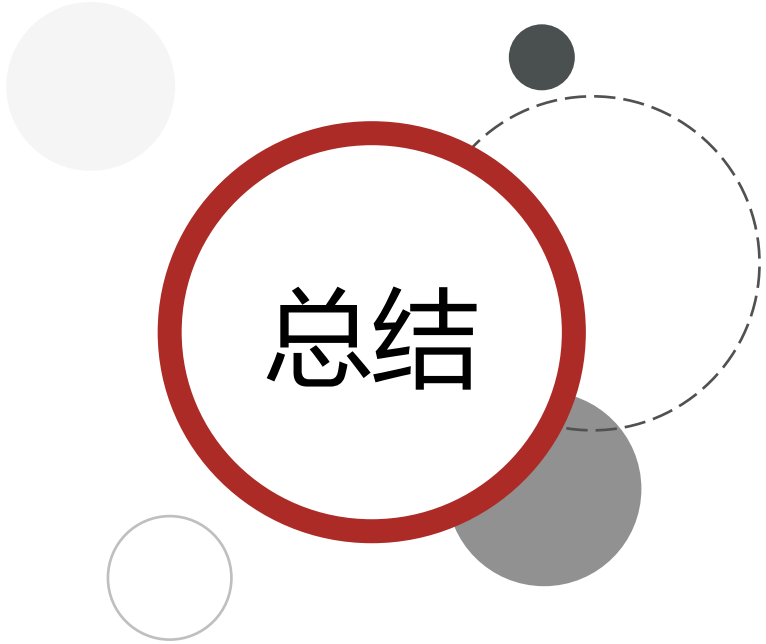
注解	说明
@Test	测试方法
@Before	用来修饰实例方法，该方法会在每一个测试方法执行之前执行一次。
@After	用来修饰实例方法，该方法会在每一个测试方法执行之后执行一次。
@BeforeClass	用来静态修饰方法，该方法会在所有测试方法之前只执行一次。
@AfterClass	用来静态修饰方法，该方法会在所有测试方法之后只执行一次。

- 开始执行的方法:初始化资源。
- 执行完之后的方法:释放资源。

Junit常用注解(Junit 5.xxxx版本)

注解	说明
@Test	测试方法
@BeforeEach	用来修饰实例方法，该方法会在每一个测试方法执行之前执行一次。
@AfterEach	用来修饰实例方法，该方法会在每一个测试方法执行之后执行一次。
@BeforeAll	用来静态修饰方法，该方法会在所有测试方法之前只执行一次。
@AfterAll	用来静态修饰方法，该方法会在所有测试方法之后只执行一次。

- 开始执行的方法:初始化资源。
- 执行完之后的方法:释放资源。



1. JUnit4的注解有哪些？

注解	说明
@Test	测试方法
@Before	用来修饰实例方法，该方法会在每一个测试方法执行之前执行一次。
@After	用来修饰实例方法，该方法会在每一个测试方法执行之后执行一次。
@BeforeClass	用来静态修饰方法，该方法会在所有测试方法之前只执行一次。
@AfterClass	用来静态修饰方法，该方法会在所有测试方法之后只执行一次。

目录

Contents

- 单元测试
- 反射
  - 反射概述
  - 反射获取类对象
  - 反射获取构造器对象
  - 反射获取成员变量对象
  - 反射获取方法对象
  - 反射的作用-绕过编译阶段为集合添加数据
  - 反射的作用-通用框架的底层原理
- 注解
- 动态代理

反射概述

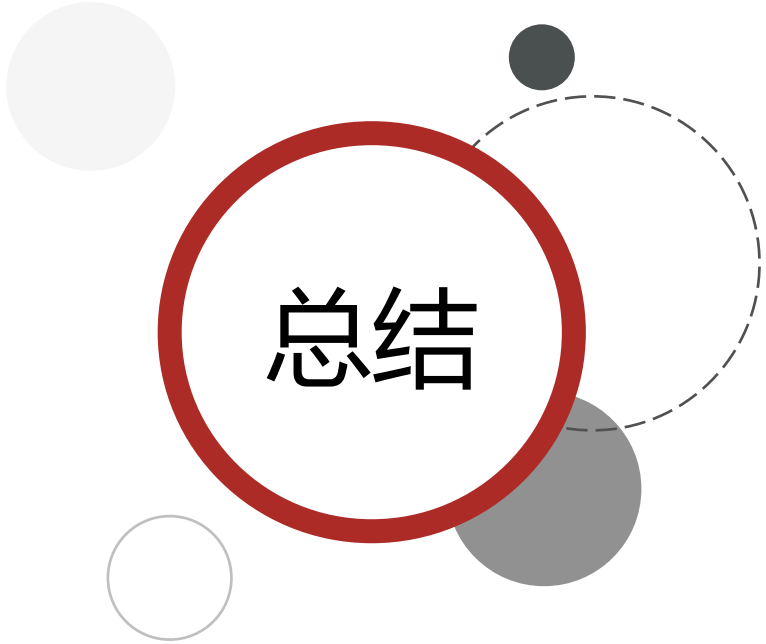
- 反射是指对于任何一个Class类，在"运行的时候"都可以直接得到这个类全部成分。
- 在运行时,可以直接得到这个类的构造器对象：Constructor
- 在运行时,可以直接得到这个类的成员变量对象：Field
- 在运行时,可以直接得到这个类的成员方法对象：Method
- 这种运行时动态获取类信息以及动态调用类中成分的能力称为Java语言的反射机制。

反射的关键：

- 反射的第一步都是先得到编译后的Class类对象，然后就可以得到Class的全部成分。

```
HelloWorld.java -> javac -> HelloWorld.class

Class c = HelloWorld.class;
```



1. 反射的基本作用、关键？

- **反射是在运行时获取类的字节码文件对象：然后可以解析类中的全部成分**

  。

- **反射的核心思想和关键就是:得到编译以后的class文件对象。**

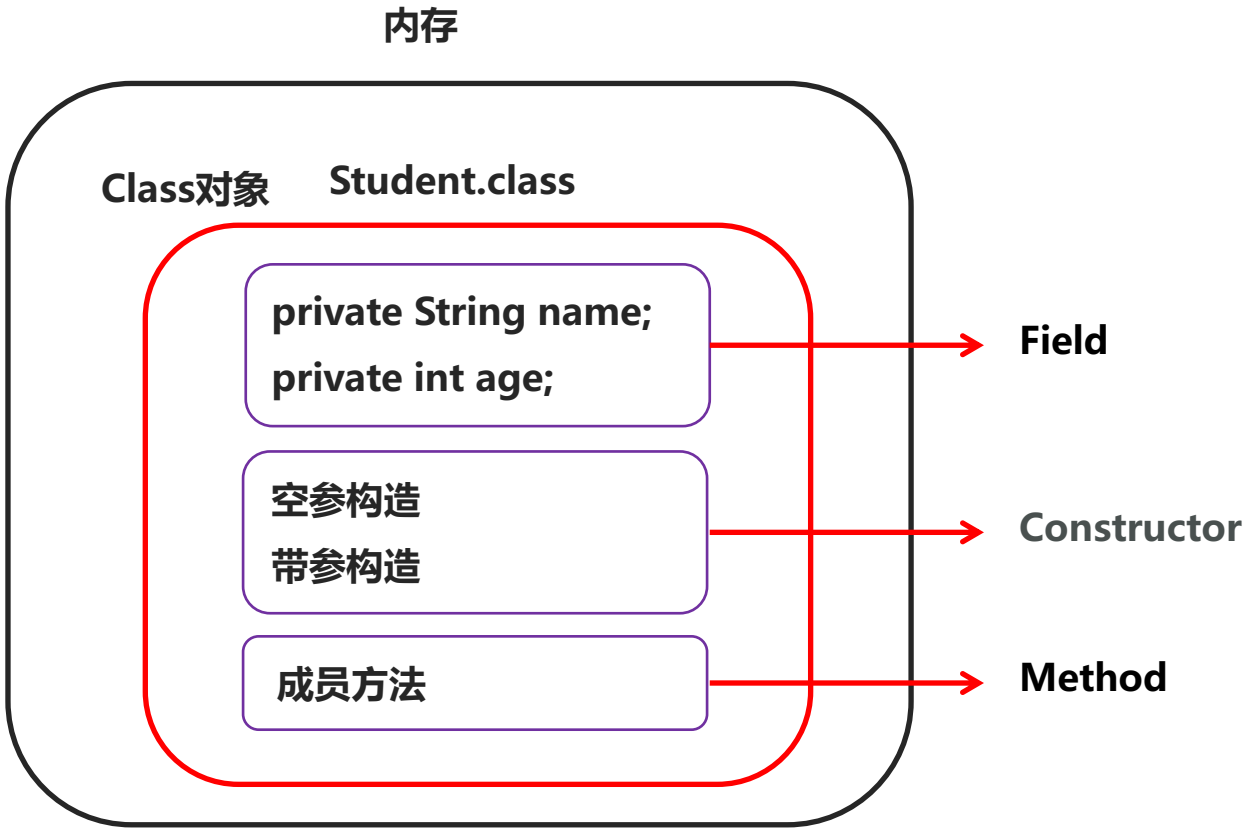
目录

Contents

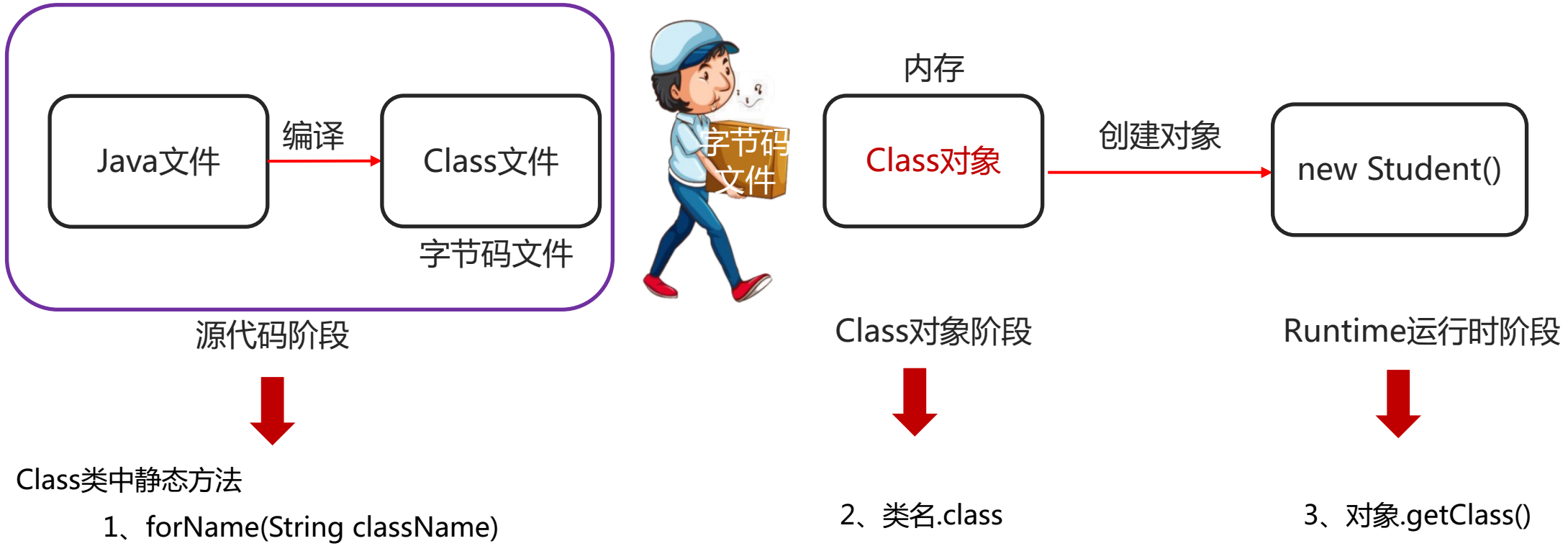
- 单元测试
- 反射
  - 反射概述
  - 反射获取类对象
  - 反射获取构造器对象
  - 反射获取成员变量对象
  - 反射获取方法对象
  - 反射的作用-绕过编译阶段为集合添加数据
  - 反射的作用-通用框架的底层原理
- 注解
- 动态代理

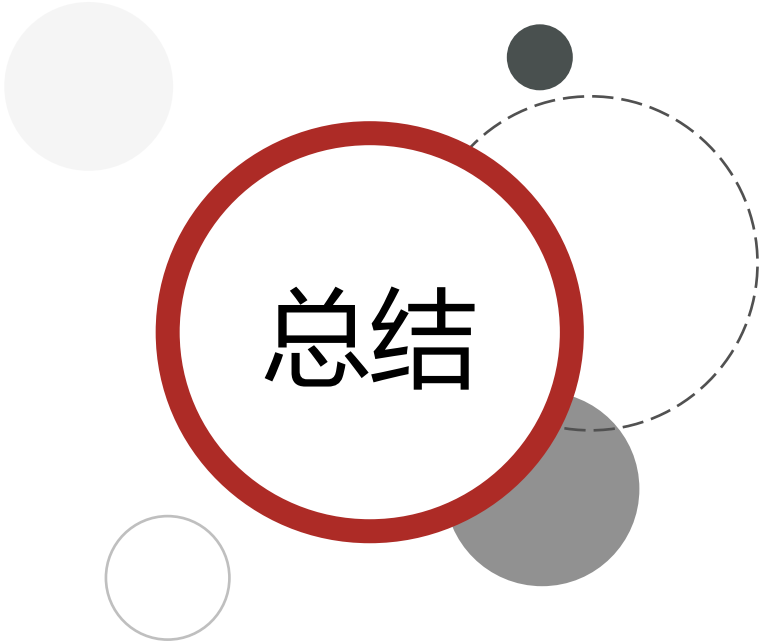
反射的第一步：获取Class类的对象

反射的第一步：获取Class类的对象



反射的第一步：获取Class类的对象





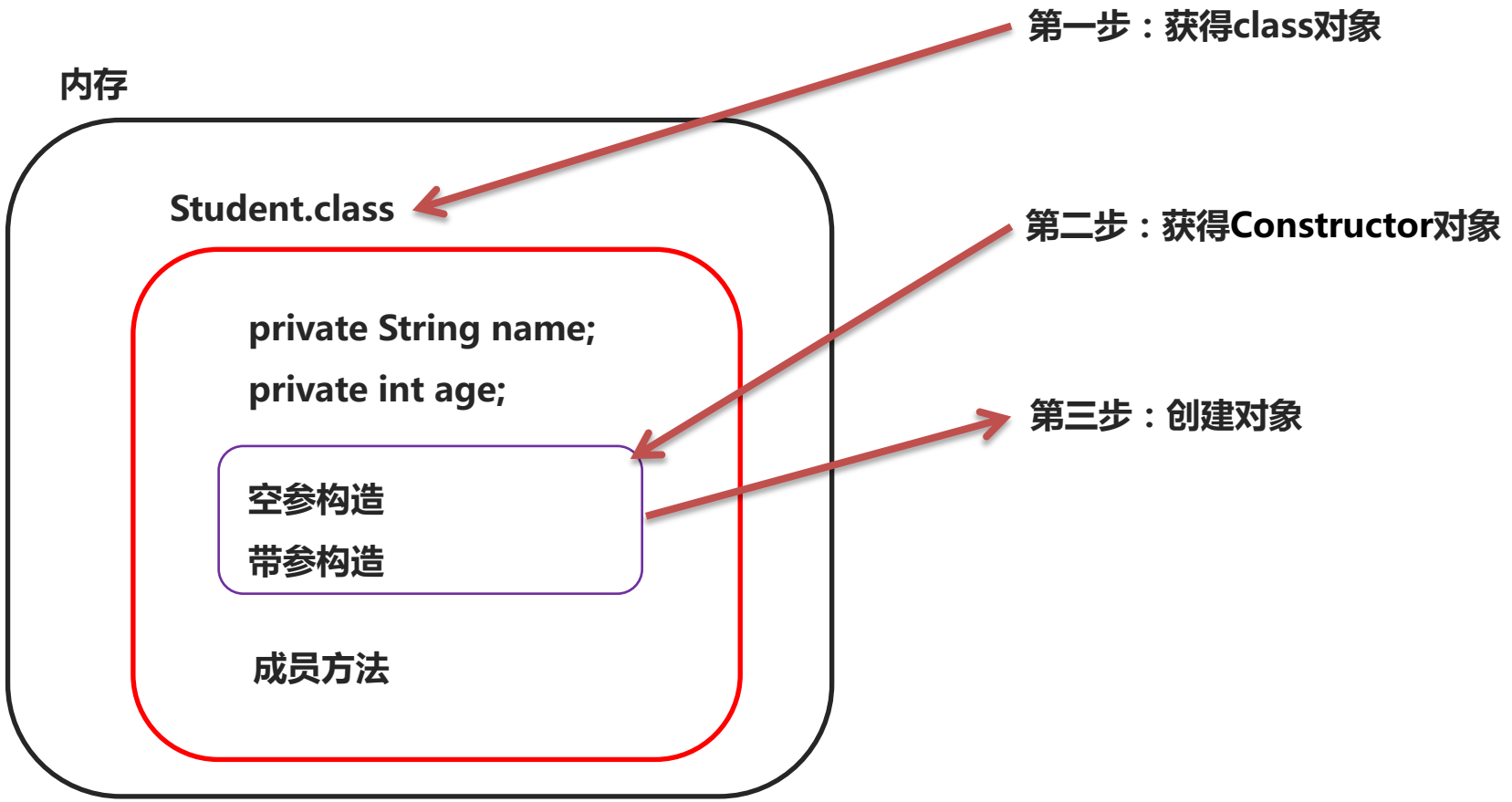
1. 反射的第一步是什么？
   - **获取Class类对象，如此才可以解析类的全部成分**
2. 获取Class类的对象的三种方式
   - **方式一：Class c1 = Class.forName(“全类名”);**
   - **方式二：Class c2 = 类名.class**
   - **方式三：Class c3 = 对象.getClass();**

目录

Contents

- 单元测试
- 反射
  - 反射概述
  - 反射获取类对象
  - 反射获取构造器对象
  - 反射获取成员变量对象
  - 反射获取方法对象
  - 反射的作用-绕过编译阶段为集合添加数据
  - 反射的作用-通用框架的底层原理
- 注解
- 动态代理

使用反射技术获取构造器对象并使用



使用反射技术获取构造器对象并使用

- 反射的第一步是先得到类对象，然后从类对象中获取类的成分对象。
- Class类中用于获取构造器的方法

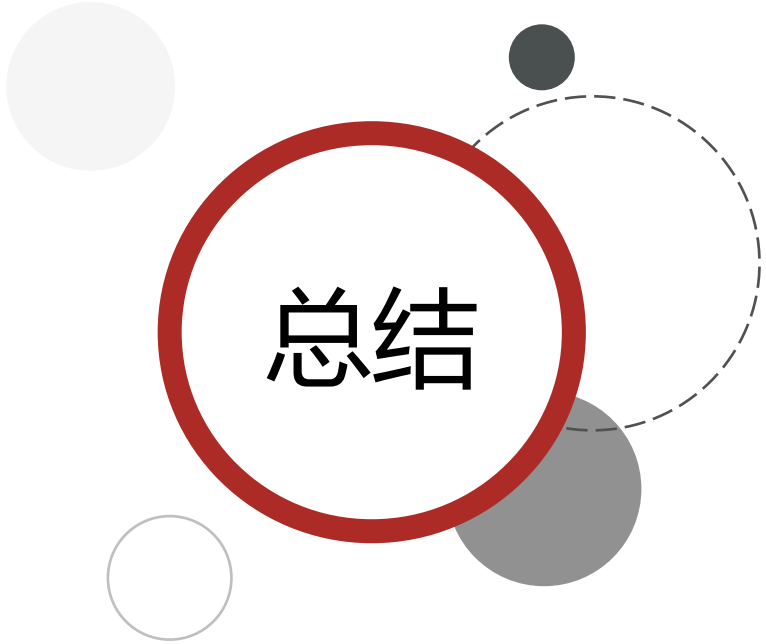
方法	说明
`Constructor<?>[] getConstructors ()`	返回所有构造器对象的数组（只能拿public的）
`Constructor<?>[] getDeclaredConstructors ()`	返回所有构造器对象的数组，存在就能拿到
`Constructor<T> getConstructor (Class<?>... parameterTypes)`	返回单个构造器对象（只能拿public的）
`Constructor<T> getDeclaredConstructor (Class<?>... parameterTypes)`	返回单个构造器对象，存在就能拿到

使用反射技术获取构造器对象并使用

- 获取构造器的作用依然是初始化一个对象返回。

Constructor类中用于创建对象的方法

符号	说明
T newInstance(Object... initargs)	根据指定的构造器创建对象
public void setAccessible(boolean flag)	设置为true,表示取消访问检查，进行暴力反射



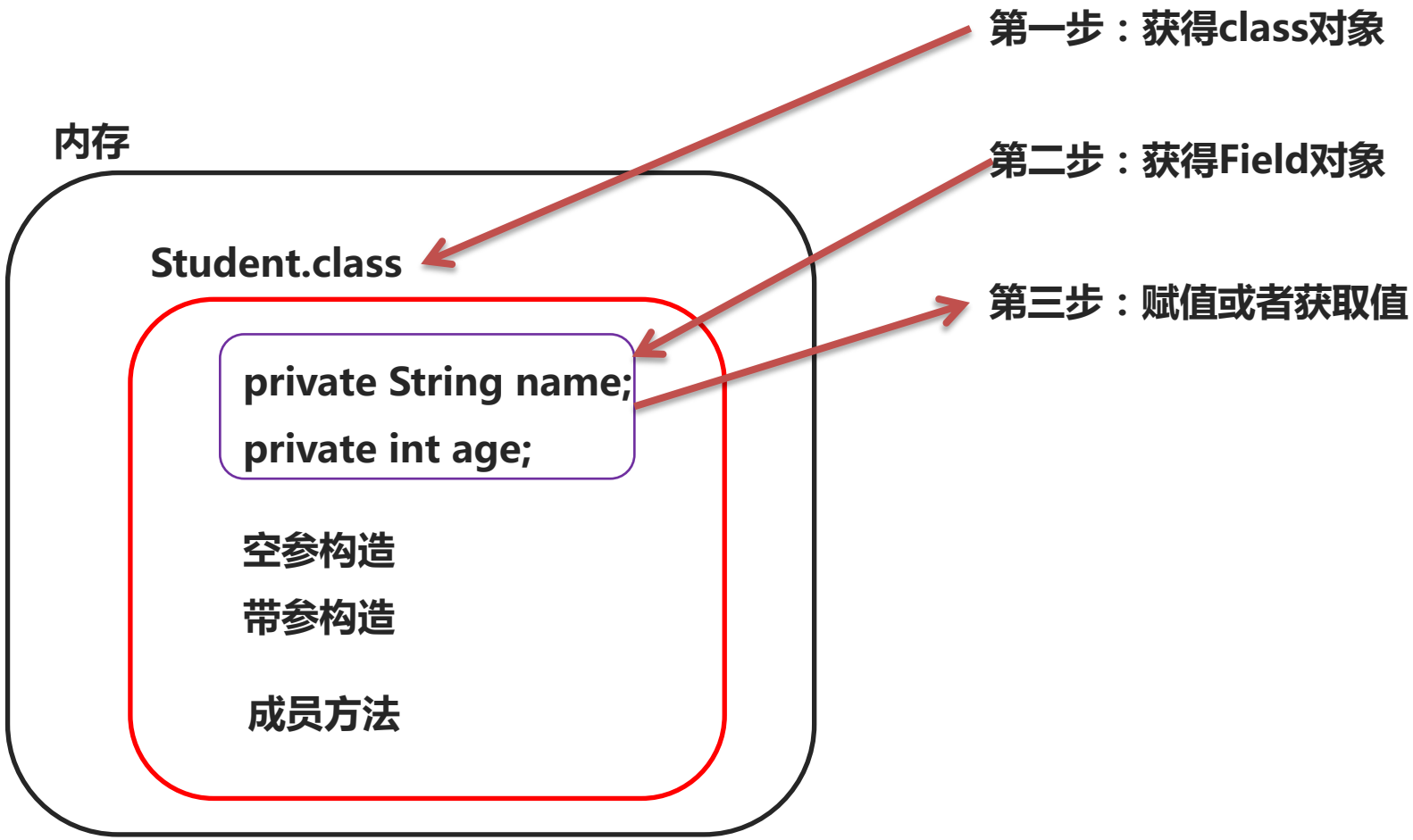
1. 利用反射技术获取构造器对象的方式
   - **getDeclaredConstructors()**
   - **getDeclaredConstructor (Class<?>... parameterTypes)**
2. 反射得到的构造器可以做什么？
   - **依然是创建对象的**
     - **public newInstance(Object... initargs)**
   - **如果是非public的构造器，需要打开权限（暴力反射），然后再创建对象**
     - **setAccessible(boolean)**
     - **反射可以破坏封装性，私有的也可以执行了。**

目录

Contents

- 单元测试
- 反射
  - 反射概述
  - 反射获取类对象
  - 反射获取构造器对象
  - 反射获取成员变量对象
  - 反射获取方法对象
  - 反射的作用-绕过编译阶段为集合添加数据
  - 反射的作用-通用框架的底层原理
- 注解
- 动态代理

使用反射技术获取成员变量对象并使用



使用反射技术获取成员变量对象并使用

- 反射的第一步是先得到类对象，然后从类对象中获取类的成分对象。
- Class类中用于获取成员变量的方法

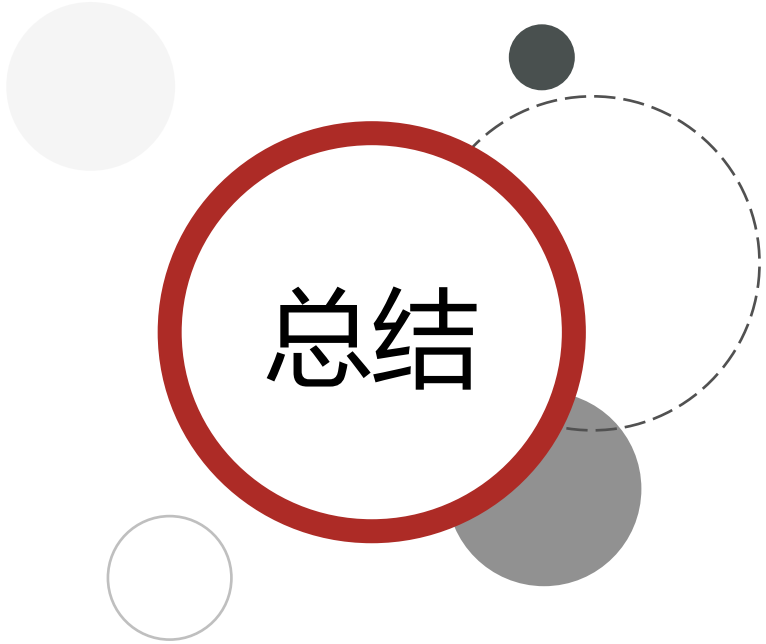
方法	说明
Field[] getFields()	返回所有成员变量对象的数组（只能拿public的）
Field[] getDeclaredFields()	返回所有成员变量对象的数组，存在就能拿到
Field getField(String name)	返回单个成员变量对象（只能拿public的）
Field getDeclaredField(String name)	返回单个成员变量对象，存在就能拿到

使用反射技术获取成员变量对象并使用

- 获取成员变量的作用依然是在某个对象中取值、赋值

Field类中用于取值、赋值的方法

符号	说明
void set(Object obj, Object value)：	赋值
Object get(Object obj)	获取值。



1. 利用反射技术获取成员变量的方式

- **获取类中成员变量对象的方法**
  - `getDeclaredFields()`
  - `getDeclaredField (String name)`

2. 反射得到成员变量可以做什么？

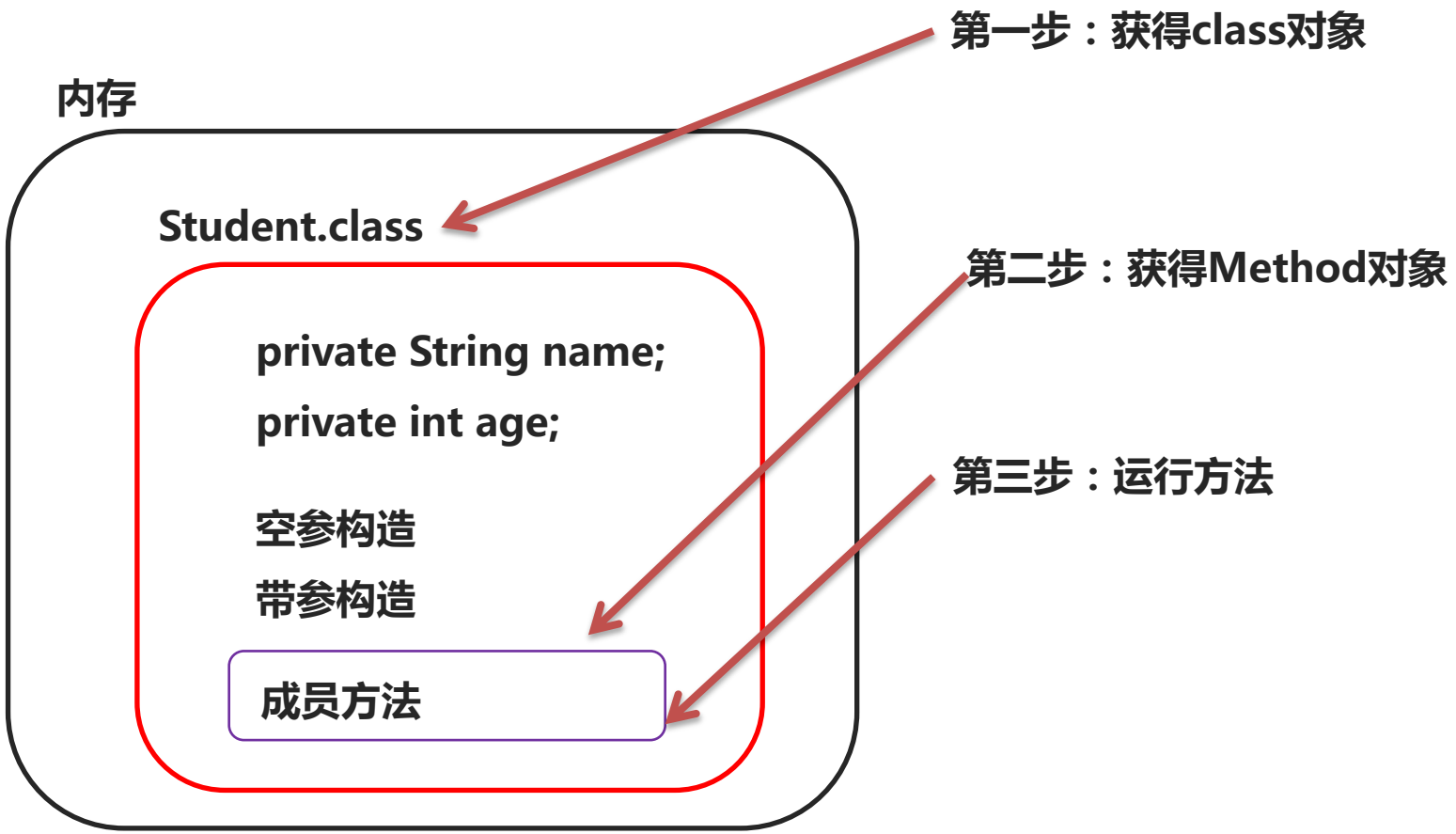
- **依然是在某个对象中取值和赋值。**
  - `void set (Object obj, Object value)：`
  - `Object get (Object obj)`
- **如果某成员变量是非public的，需要打开权限（暴力反射），然后再取值、赋值**
  - `setAccessible(boolean)`

目录

Contents

- 单元测试
- 反射
  - 反射概述
  - 反射获取类对象
  - 反射获取构造器对象
  - 反射获取成员变量对象
  - 反射获取方法对象
  - 反射的作用-绕过编译阶段为集合添加数据
  - 反射的作用-通用框架的底层原理
- 注解
- 动态代理

使用反射技术获取方法对象并使用



使用反射技术获取方法对象并使用

- 反射的第一步是先得到类对象，然后从类对象中获取类的成分对象。
- Class类中用于获取成员方法的方法

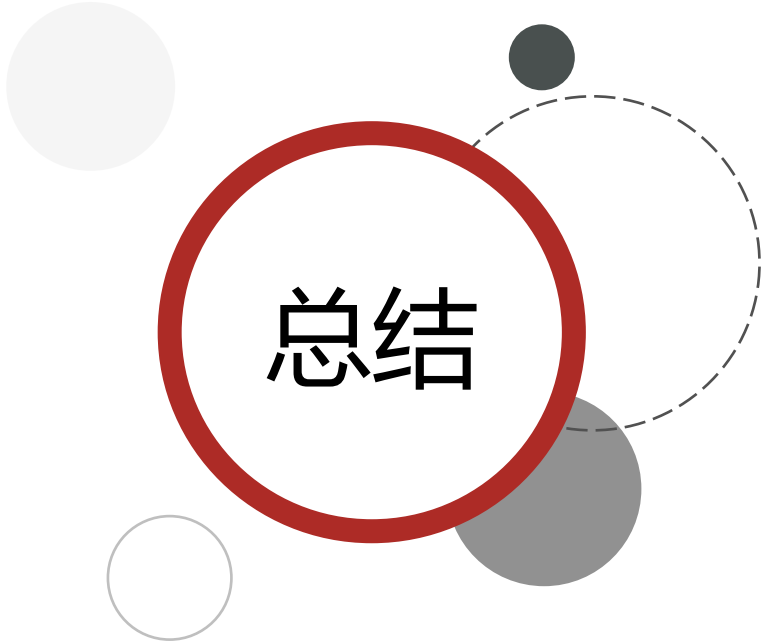
方法	说明
`Method[] getMethods ()`	返回所有成员方法对象的数组（只能拿public的）
`Method[] getDeclaredMethods ()`	返回所有成员方法对象的数组，存在就能拿到
`Method getMethod (String name, Class<?>... parameterTypes)`	返回单个成员方法对象（只能拿public的）
`Method getDeclaredMethod (String name, Class<?>... parameterTypes)`	返回单个成员方法对象，存在就能拿到

使用反射技术获取方法对象并使用

- 获取成员方法的作用依然是在某个对象中进行执行此方法

Method类中用于触发执行的方法

符号	说明
`Object invoke (Object obj, Object... args)`	运行方法 参数一：用obj对象调用该方法 参数二：调用方法的传递的参数（如果没有就不写） 返回值：方法的返回值（如果没有就不写）



1. 利用反射技术获取成员方法对象的方式

- **获取类中成员方法对象**
  - **getDeclaredMethods()**
  - **getDeclaredMethod (String name, Class<?>... parameterTypes)**

2. 反射得到成员方法可以做什么？

- **依然是在某个对象中触发该方法执行。**
  - **Object invoke(Object obj, Object... args)**
- **如果某成员方法是非public的，需要打开权限（暴力反射），然后再触发执行**
  - **setAccessible(boolean)**

目录

Contents

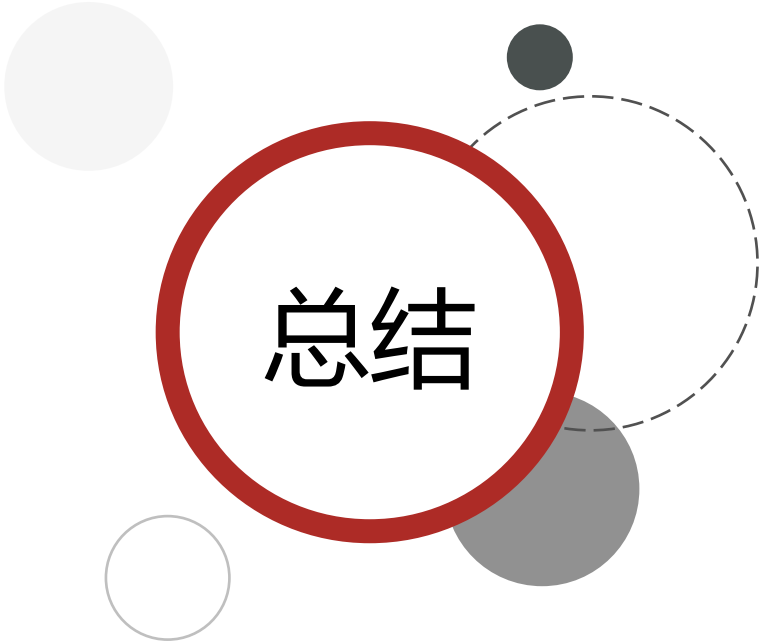
- 单元测试
- 反射
  - 反射概述
  - 反射获取类对象
  - 反射获取构造器对象
  - 反射获取成员变量对象
  - 反射获取方法对象
  - 反射的作用-绕过编译阶段为集合添加数据
  - 反射的作用-通用框架的底层原理
- 注解
- 动态代理

反射的作用-绕过编译阶段为集合添加数据

- 反射是作用在运行时的技术，此时集合的泛型将不能产生约束了，此时是可以为集合存入其他任意类型的元素的。

```
ArrayList<Integer> list = new ArrayList<>();
list.add(100);
// list.add( “黑马"); // 报错
list.add(99);
```

- 泛型只是在编译阶段可以约束集合只能操作某种数据类型，在编译成Class文件进入运行阶段的时候，其真实类型都是ArrayList了，泛型相当于被擦除了。



1. 反射为何可以给约定了泛型的集合存入其他类型的元素？

- **编译成Class文件进入运行阶段的时候，泛型会自动擦除。**
- **反射是作用在运行时的技术，此时已经不存在泛型了。**

目录

Contents

- 单元测试
- 反射
  - 反射概述
  - 反射获取类对象
  - 反射获取构造器对象
  - 反射获取成员变量对象
  - 反射获取方法对象
  - 反射的作用-绕过编译阶段为集合添加数据
  - 反射的作用-通用框架的底层原理
- 注解
- 动态代理

案例 反射做通用框架

需求：给你任意一个对象，在不清楚对象字段的情况可以，可以把对象的字段名称和对应值存储到文件中去。

```
public class Student {
    private String name;
    private int age;
    private char sex;
    private double height;
    private String hobby;
}
```

```
public class Teacher {
    private String name;
    private double salary;
}
```

```
Student s = new Student(“柳岩”, 40 , ‘女’ , 167.5 , “女星" );
```

```
Teacher t = new Teacher("波妞" , 6000);
```

```
==============Student================
name=柳岩
age=40
sex=女
height=167.5
hobby=女星
==============Teacher================
name=波妞
salary=6000.0
```

案例

反射做通用框架

需求

- 给你任意一个对象，在不清楚对象字段的情况可以，可以把对象的字段名称和对应值存储到文件中去。

分析

① 定义一个方法，可以接收任意类的对象。

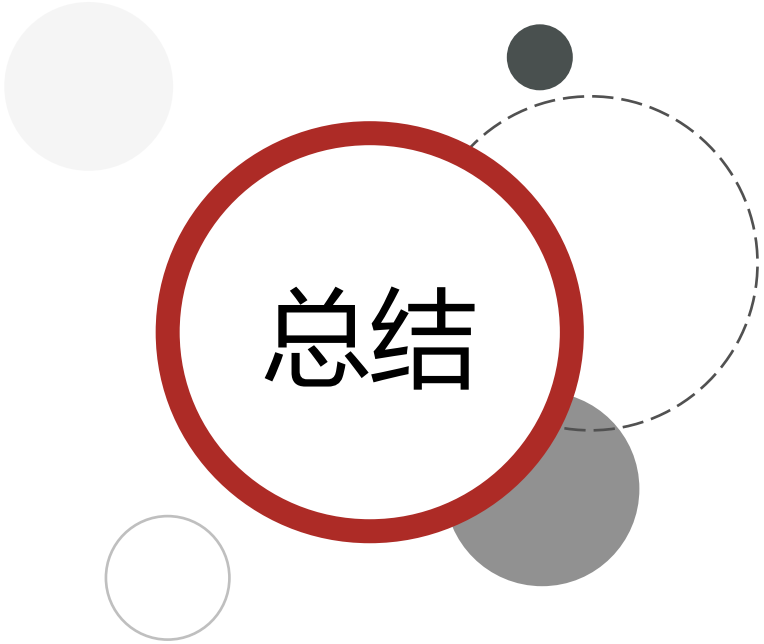
② 每次收到一个对象后，需要解析这个对象的全部成员变量名称。

③ 这个对象可能是任意的，那么怎么样才可以知道这个对象的全部成员变量名称呢？

④ 使用反射获取对象的Class类对象，然后获取全部成员变量信息。

⑤ 遍历成员变量信息，然后提取本成员变量在对象中的具体值

⑥ 存入成员变量名称和值到文件中去即可。



1. 反射的作用？

- **可以在运行时得到一个类的全部成分然后操作。**
- **可以破坏封装性。（很突出）**
- **也可以破坏泛型的约束性。（很突出）**
- **更重要的用途是适合：做Java高级框架**
- **基本上主流框架都会基于反射设计一些通用技术功能。**

目录

Contents

- 单元测试
- 反射
- 注解
  - 注解概述
  - 自定义注解
  - 元注解
  - 注解解析
  - 注解的应用场景一：junit框架
- 动态代理

注解概述、作用

- Java 注解（Annotation）又称 Java 标注，是 JDK5.0 引入的一种注释机制。
- Java 语言中的类、构造器、方法、成员变量、参数等都可以被注解进行标注。

```
public class UserServiceTest {
    @Test
    public void testLogin(){
    }
    @Test
    public void testChu(){

    }
}
```

注解（Annotation）

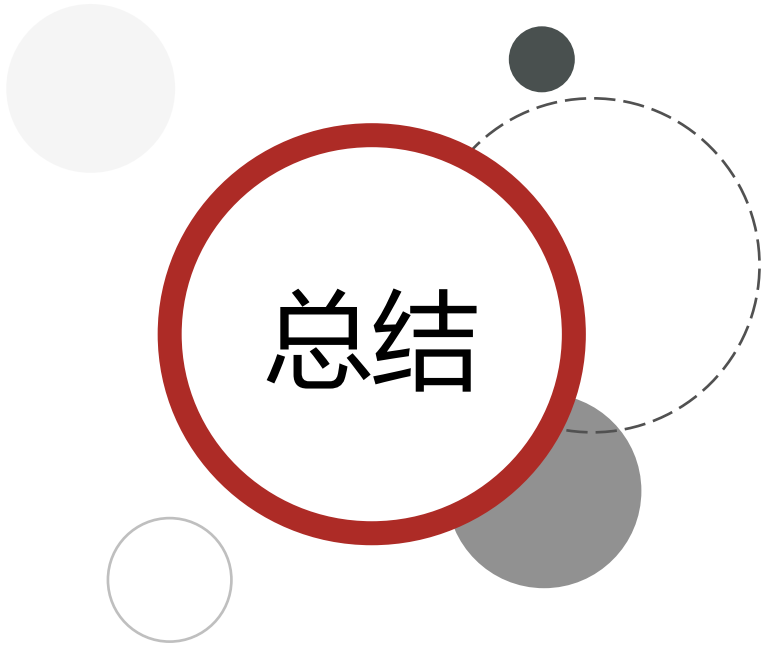
注解（Annotation）

注解的作用是什么呢？

- 对Java中类、方法、成员变量做标记，然后进行特殊处理，至于到底做何种处理由业务需求来决定。
- 例如：JUnit框架中，标记了注解@Test的方法就可以被当成测试方法执行，而没有标记的就不能当成测试方法执行。







1. 注解的作用

- **对Java中类、方法、成员变量做标记，然后进行特殊处理。**
- **例如：JUnit框架中，标记了注解@Test的方法就可以被当成测试方法执行，而没有标记的就不能当成测试方法执行**

目录

Contents

- 单元测试
- 反射
- 注解
  - 注解概述
  - 自定义注解
  - 元注解
  - 注解解析
  - 注解的应用场景一：junit框架
- 动态代理

自定义注解 --- 格式

- 自定义注解就是自己做一个注解来使用。

```
public @interface 注解名称 {

    public 属性类型 属性名() default 默认值 ;

}
```

Java支持的数据类型
基本上都支持

特殊属性

- value属性，如果只有一个value属性的情况下，使用value属性的时候可以省略value名称不写!!
- 但是如果有多个属性，且多个属性没有默认值，那么value名称是不能省略的。

总结

1. 自定义注解

```
public @interface 注解名称 {

    public 属性类型 属性名() default 默认值 ;

}
```

目录

Contents

- 单元测试
- 反射
- 注解
  - 注解概述
  - 自定义注解
  - 元注解
  - 注解解析
  - 注解的应用场景一：junit框架
- 动态代理

元注解

- 元注解：注解注解的注解。

元注解有两个：

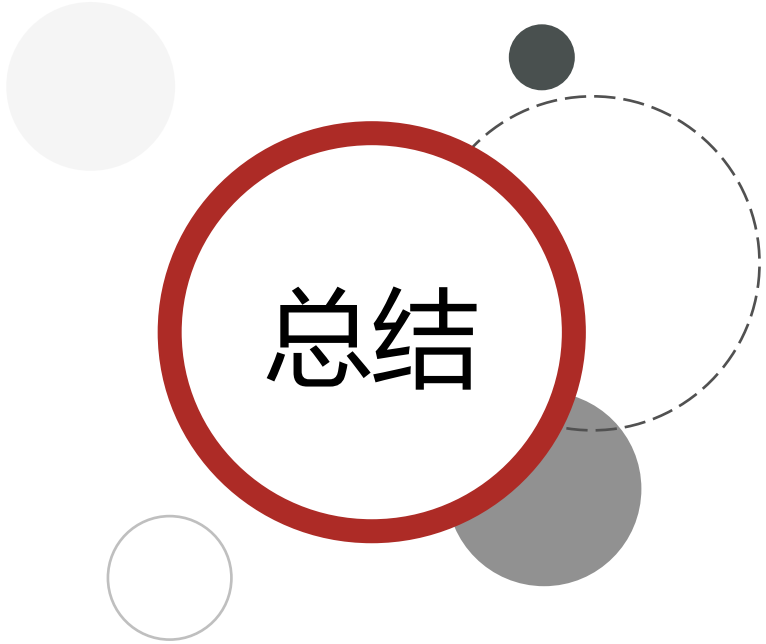
- **@Target**: 约束自定义注解只能在哪些地方使用，
- **@Retention**：申明注解的生命周期

@Target中可使用的值定义在ElementType枚举类中，常用值如下

- TYPE，类，接口
- FIELD, 成员变量
- METHOD, 成员方法
- PARAMETER, 方法参数
- CONSTRUCTOR, 构造器
- LOCAL_VARIABLE, 局部变量

@Retention中可使用的值定义在RetentionPolicy枚举类中，常用值如下

- SOURCE： 注解只作用在源码阶段，生成的字节码文件中不存在
- CLASS： 注解作用在源码阶段，字节码文件阶段，运行阶段不存在，默认值.
- RUNTIME：注解作用在源码阶段，字节码文件阶段，运行阶段（开发常用）



1. 元注解是什么？

- **注解注解的注解**
- **@Target约束自定义注解可以标记的范围。**

  @Retention用来约束自定义注解的存活范围。

目录

Contents

- 单元测试
- 反射
- 注解
  - 注解概述
  - 自定义注解
  - 元注解
  - 注解解析
  - 注解的应用场景一：junit框架
- 动态代理

注解的解析

- 注解的操作中经常需要进行解析，注解的解析就是判断是否存在注解，存在注解就解析出内容。

与注解解析相关的接口

- Annotation: 注解的顶级接口，注解都是Annotation类型的对象
- AnnotatedElement:该接口定义了与注解解析相关的解析方法

方法	说明
Annotation[] getDeclaredAnnotations()	获得当前对象上使用的所有注解，返回注解数组。
T getDeclaredAnnotation(Class<T> annotationClass)	根据注解类型获得对应注解对象
boolean isAnnotationPresent(Class<Annotation> annotationClass)	判断当前对象是否使用了指定的注解，如果使用了则返回true，否则false

- 所有的类成分Class, Method，Field，Constructor，都实现了AnnotatedElement接口他们都拥有解析注解的能力：

解析注解的技巧

- 注解在哪个成分上，我们就先拿哪个成分对象。
- 比如注解作用成员方法，则要获得该成员方法对应的Method对象，再来拿上面的注解
- 比如注解作用在类上，则要该类的Class对象，再来拿上面的注解
- 比如注解作用在成员变量上，则要获得该成员变量对应的Field对象，再来拿上面的注解

案例

注解解析的案例

需求：注解解析的案例

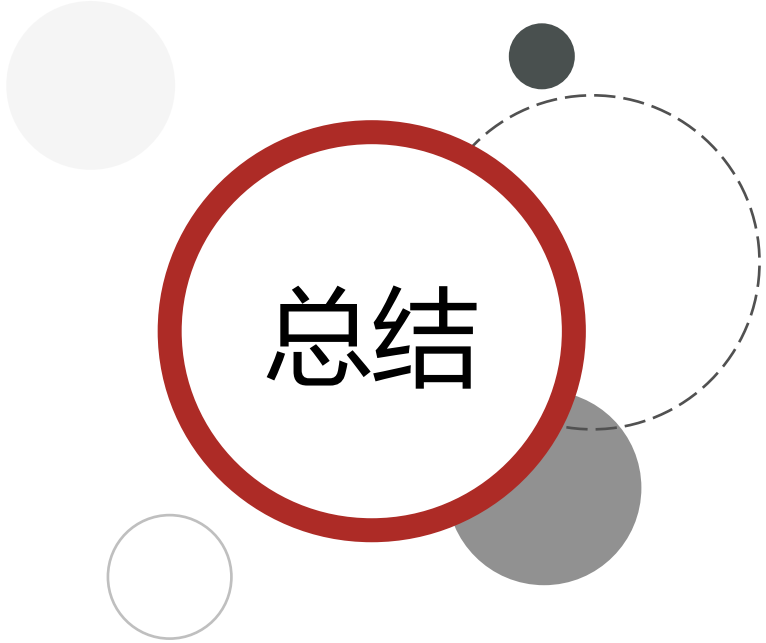
分析

① 定义注解Book，要求如下：

- 包含属性：String value() 书名
- 包含属性：double price() 价格，默认值为 100
- 包含属性：String[] authors() 多位作者
- 限制注解使用的位置：类和成员方法上
- 指定注解的有效范围：RUNTIME

② 定义BookStore类，在类和成员方法上使用Book注解

③ 定义AnnotationDemo01测试类获取Book注解上的数据



1. 注解解析的方式

方法
Annotation[] getDeclaredAnnotations()
T getDeclaredAnnotation(Class<T> annotationClass)
boolean isAnnotationPresent(Class<Annotation> annotationClass)

目录

Contents

- 单元测试
- 反射
- 注解
  - 注解概述
  - 自定义注解
  - 元注解
  - 注解解析
  - 注解的应用场景一：junit框架
- 动态代理

案例

模拟Junit框架

需求

- 定义若干个方法，只要加了MyTest注解，就可以在启动时被触发执行

分析

① 定义一个自定义注解MyTest，只能注解方法，存活范围是一直都在。

② 定义若干个方法，只要有@MyTest注解的方法就能在启动时被触发执行，没有这个注解的方法不能执行。

目录

Contents

- 单元测试
- 反射
- 注解
- 动态代理
  - 准备案例、提出问题
  - 使用动态代理解决问题

案例

模拟企业业务功能开发，并完成每个功能的性能统计

需求

- 模拟某企业用户管理业务，需包含用户登录，用户删除，用户查询功能，并要统计每个功能的耗时

。

分析

① 定义一个UserService表示用户业务接口，规定必须完成用户登录，用户删除，用户查询功能。

② 定义一个实现类UserServiceImpl实现UserService，并完成相关功能，且统计每个功能的耗时。

③ 定义测试类，创建实现类对象，调用方法。

本案例存在哪些问题？

答：业务对象的的每个方法都要进行性能统计，存在大量重复的代码。

目录

Contents

- 单元测试
- 反射
- 注解
- 动态代理

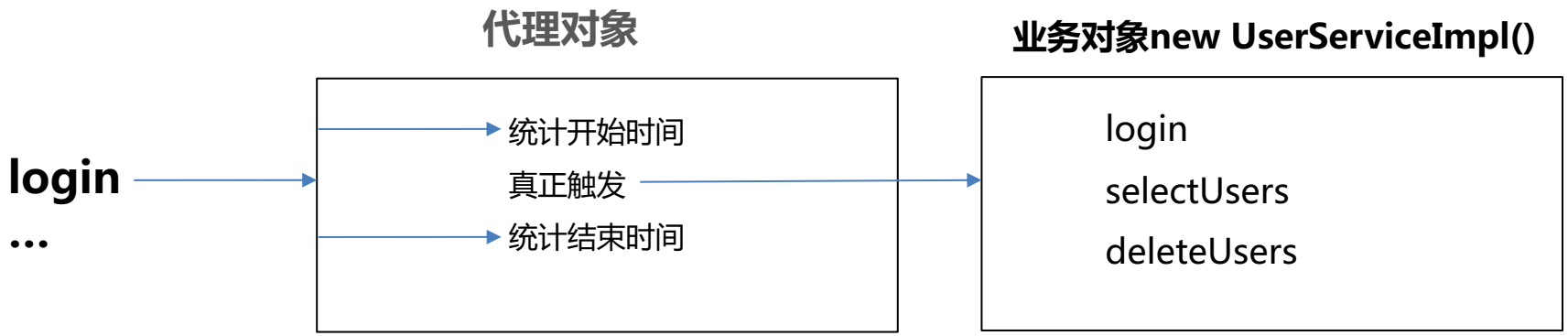
动态代理

- 代理就是被代理者没有能力或者不愿意去完成某件事情，需要找个人代替自己去完成这件事，动态代理就是用来对业务功能（方法）进行代理的。

关键步骤

1.必须有接口，实现类要实现接口（代理通常是基于接口实现的）。

3.创建一个实现类的对象，该对象为业务对象，紧接着为业务对象做一个代理对象。



动态代理的优点

- 非常的灵活，支持任意接口类型的实现类对象做代理，也可以直接为接本身做代理。
- 可以为被代理对象的所有方法做代理。
- 可以在不改变方法源码的情况下，实现对方法功能的增强。
- 不仅简化了编程工作、提高了软件系统的可扩展性，同时也提高了开发效率。

传智教育旗下高端IT教育品牌