* 能够说出反射的概念
* 能够说出获取一个类的class文件对象的三个方式
* 能够实现反射构造方法
* 能够实现反射方法
* 能够使用BeanUtils工具类方法setProperty
* 能够使用BeanUtils工具类方法getProperty
* 能够完成BeanUtils工具类的综合案例

# 反射

## 类的加载

当程序要使用某个类时，如果该类还未被加载到内存中，则系统会通过加载，连接，初始化三步来实现对这个类进行初始化。

* 加载

就是指将class文件读入内存，并为之创建一个Class对象。

任何类被使用时系统都会建立一个Class对象

* 连接

验证是否有正确的内部结构，并和其他类协调一致

准备负责为类的静态成员分配内存，并设置默认初始化值

解析将类的二进制数据中的符号引用替换为直接引用

* 初始化

就是我们以前讲过的初始化步骤

## 类的初始化

1. 创建类的实例

2. 类的静态变量，或者为静态变量赋值

3. 类的静态方法

4. 使用反射方式来强制创建某个类或接口对应的java.lang.Class对象

5. 初始化某个类的子类

6. 直接使用java.exe命令来运行某个主类

到目前为止我们已经知道把class文件加载到内存做了哪些事情了，那么，如果我们仅仅站在这些class文件的角度，我们如何来使用这些class文件中的内容呢？那就是我们反射将要学习的内容了。

## 反射概述

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。

要想解剖一个类，必须先要获取到该类的字节码文件对象。而解剖这个类，使用的就是Class类中的方法，所以先要获取到每一个字节码文件所对应的Class类型对象。

## 准备数据



publicclass Bean {

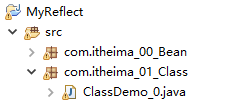
private String id;

private String className;

## 知识储备

* 需要掌握6个单词
  + Class类
  + Constructor 构造
  + Method 方法
  + Filed 字段
  + instance 实例
  + invoke 执行

## Class获得方式



//1 通过类型获得

// 语法：类名.class

// 应用场景：确定类型等

Classclazz1 = Bean.**class**;

//2 通过实例对象获得

// 语法：变量.getClass()

// 应用场景：在方法内部通过参数获得类型等

Bean bean = **new**Bean();

Classclazz2 = bean.getClass();

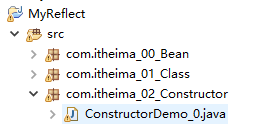
//3 通过字符串获得

// 语法：Class.forName("全限定类名")

// 应用场景：通过配置获得字符串等

Classclazz3 = Class.*forName*("com.itheima\_00\_Bean.Bean");

## 构造方法与实例



### 添加构造

* 修改Bean ，添加构造方法

**public**Bean() {

System.***out***.println("无参构造");

}

**public**Bean(String id, String className) {

**this**.id = id;

**this**.className = className;

System.***out***.print("有参构造：" + id);

System.***out***.println(" ，" + className);

}

### 无参构造

* 无参构造，并获得实例

**publicvoid**testDefaultCons() **throws** Exception{

//无参构造 , 并实例化

//1获得Class

ClassbeanClass = Bean.**class**;

//2获得构造 -- 没有形参

Constructorconstructor = beanClass.getConstructor();

//3 实例对象，没有实参

Object bean = constructor.newInstance();

System.***out***.println(bean);

/\* 结果：

\* 无参构造

\* com.itheima\_00\_Bean.Bean@8c5488

\*/

}

### 有参构造

* 有参构造，并获得实例

**publicvoid**testParamCons() **throws** Exception{

//有参构造 , 并实例化

//1获得Class

ClassbeanClass = Bean.**class**;

//2获得构造 -- 两个字符串形参 -- Bean(String id, String className)

Constructorconstructor = beanClass.getConstructor(String.**class**,String.**class**);

//3 实例对象，两个字符串实参

Object bean = constructor.newInstance("ArrayListId","java.util.ArrayList");

System.***out***.println(bean);

/\* 结果：

\* 有参构造：ArrayListId，java.util.ArrayList

\* com.itheima\_00\_Bean.Bean@101acff

\*/

}

### 无参构造，简化版获得实例

* 无参构造，简化版获得实例

**publicvoid**testDefaultSimple() **throws** Exception{

//无参构造 , 简化版

//1获得Class

ClassbeanClass = Bean.**class**;

//2 直接获得实例对象，两个字符串实参

Object bean = beanClass.newInstance();

System.***out***.println(bean);

/\* 结果：

\* 无参构造

\* com.itheima\_00\_Bean.Bean@101acff

\*/

}

### 扩展：私有构造(暴力反射)

* + 修改Bean添加私有构造

**private**Bean(String id) {

**this**.id = id;

System.***out***.println("有参构造：" + id);

}

* + getConstructor() 使用该方法将无法获得私有方法，程序运行抛异常



* + 没有使用setAccessible(true)，将抛异常





**publicvoid**testPrivateCons() **throws** Exception{

//私有构造

//1获得Class

Class beanClass = Bean.**class**;

//2获得构造 -- 两个字符串形参 -- Bean(String id, String className)

// \* getConstructor() 将抛异常java.lang.NoSuchMethodException

// \* getDeclaredConstructor可以获得私有构造

Constructor constructor = beanClass.getDeclaredConstructor(String.**class**);

//暴力访问

constructor.setAccessible(**true**);

//3 实例对象，两个字符串实参

Object bean = constructor.newInstance("userId");

System.***out***.println(bean);

/\* 结果：

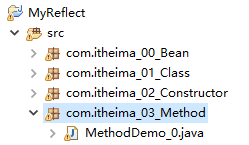
\* 有参构造：userId

\* com.itheima\_00\_Bean.Bean@8c5488

\*/

}

## 方法与执行



### 添加set、get方法

* Bean类提供getter和setter方法

public String getId() {

System.out.println("getId方法执行");

return id;

}

public void setId(String id) {

System.out.println("setId方法执行：" + id);

this.id = id;

}

### public方法

* 获得方法并设置

**publicvoid**testMethod() **throws** Exception{

//1 获得Class

Class clazz = Class.*forName*("com.itheima\_00\_Bean.Bean");

//2 获得实例 ,相当于 Object obj = new Bean();

Object obj = clazz.newInstance();

//3 操作setId方法

//3.1 获得的方法，一个形参

// \* 格式：getMethod(方法名,形成列表)

Method setMethod = clazz.getMethod("setId", String.**class**);

//3.2 执行方法，一个实参

Object setReturnObj = setMethod.invoke(obj, "我是参数");

System.***out***.println("set方法返回值：" + setReturnObj);

/\* 运行结果：

\* 无参构造

\* setId方法执行：我是参数

\* set方法返回值：null

\*/

System.***out***.println("---------------");

//4 操作getId方法 (巩固)

// 3.1 获得方法，没有形参

Method getMethod = clazz.getMethod("getId");

// 3.2 执行方法，没有实参

Object getReturnObj = getMethod.invoke(obj);

System.***out***.println("get方法返回值：" + getReturnObj);

/\* 运行结果：

\* getId方法执行

\* get方法返回值：我是参数

\*/

}

### 扩展：私有方法(暴力反射)

* + 修改bean，添加方法：

//私有方法

**private** String show(){

System.***out***.println("私有方法执行");

**return**"Bean["+id+", "+ className +"]";

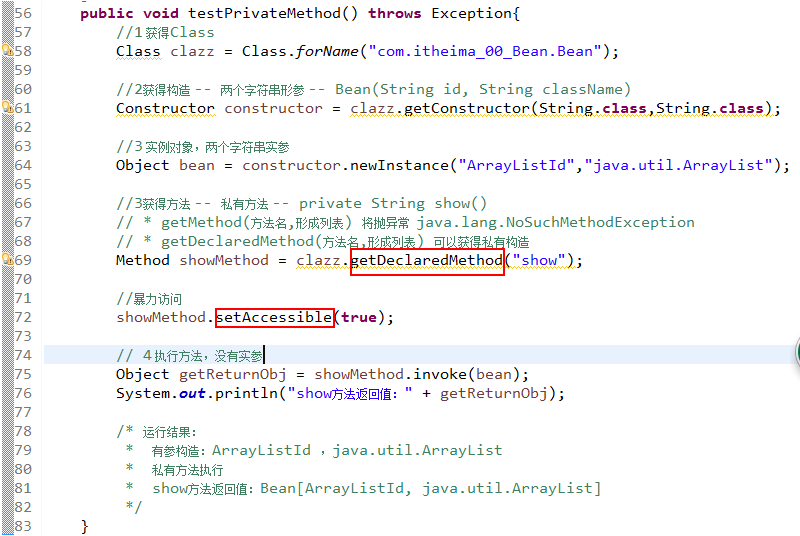
}

* + getMethod() 使用该方法将无法获得私有方法，程序运行抛异常



* + 没有使用setAccessible(true)，将抛异常





**publicvoid**testPrivateMethod() **throws** Exception{

//1 获得Class

Class clazz = Class.*forName*("com.itheima\_00\_Bean.Bean");

//2获得构造 -- 两个字符串形参 -- Bean(String id, String className)

Constructor constructor = clazz.getConstructor(String.**class**,String.**class**);

//3 实例对象，两个字符串实参

Object bean = constructor.newInstance("ArrayListId","java.util.ArrayList");

//3获得方法 -- 私有方法 -- private String show()

// \* getMethod(方法名,形成列表) 将抛异常java.lang.NoSuchMethodException

// \* getDeclaredMethod(方法名,形成列表) 可以获得私有构造

Method showMethod = clazz.getDeclaredMethod("show");

//暴力访问

showMethod.setAccessible(**true**);

// 4 执行方法，没有实参

Object getReturnObj = showMethod.invoke(bean);

System.***out***.println("show方法返回值：" + getReturnObj);

/\* 运行结果：

\* 有参构造：ArrayListId，java.util.ArrayList

\* 私有方法执行

\* show方法返回值：Bean[ArrayListId, java.util.ArrayList]

\*/

}

## 字段(成员变量)与数据操作(了解)

### 添加public字段

* Bean类提供成员变量 address

publicclass Bean {

private String id;

private String className;

public String description;

### public字段的操作

**publicvoid**testField() **throws** Exception{

/\* 获得实例，为public字段赋值、获取值

\* public String description;

\*/

//1 获得Class

Class clazz = Class.*forName*("com.itheima\_00\_Bean.Bean");

//2 获得实例 ,相当于 Object obj = new Bean();

Object bean= clazz.newInstance();

//3 操作字段，进行赋值，public String description;

//3.1 获得的字段，一个形参

// \* 格式：getField(字段名)

Field descriptionField = clazz.getField("description");

//3.2 为对象的字段赋值

descriptionField.set(bean, "Bean的描述");

//3.3 获取对象的字段值

Object fieldReturnObj = descriptionField.get(bean);

System.***out***.println("description字段返回值："+fieldReturnObj);

}

## 综合案例

* 提供一个User对象，并提供getter和setter方法，并复写toString()

public class User {

private String id;

private String username;

private String pwd;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

//....省略

@Override

public String toString() {

return "User [id=" + id + ", username=" + username + ", pwd=" + pwd + "]";

}

* 读取properties文件，将文件内容设置到User中

@

/\*

\* 读取配置文件(IO+集合),键值对存储到集合中

\* 键获取集合中的所有值

\* 反射,创建出User类的对象,调用方法setXX,数据存储

\*/

public void test()throws Exception{

Properties pro = new Properties();

FileInputStream fis = new FileInputStream("data.properties");

pro.load(fis);

fis.close();

Class clazz = Class.forName("cn.itcast.domain.User");

Object obj = clazz.newInstance();

Set<String> keys = pro.stringPropertyNames();

for(String key : keys){

String methodName = "set"+key.substring(0,1).toUpperCase()+key.substring(1);

Method method = clazz.getMethod(methodName, String.class);

method.invoke(obj, pro.getProperty(key));

}

System.out.println(obj);

}

配置文件data.properties

uid=001

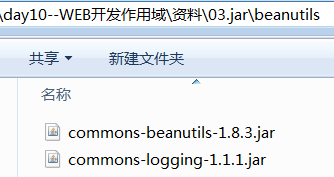
username=jack

password=123

# BeanUtils工具类

BeanUtils 是 Apache commons组件的成员之一，主要用于简化JavaBean封装数据的操作。它可以给JavaBean封装一个字符串数据，也可以将一个表单提交的所有数据封装到JavaBean中。

使用第三方工具，需要导入jar包：



BeanUtils工具常用工具类有两个：BeanUtils、ConvertUtils。BeanUtils用于封装数据，ConvertUtils用于处理类型转换，常用API如下

|  |  |  |
| --- | --- | --- |
|  | 方法 | 描述 |
| BeanUtils对象 | populate(Object bean, Map<String,String[]> properties) | 将Map数据封装到指定Javabean中，一般用于将表单的所有数据封装到javabean |
| setProperty(Object obj,String name,Object value) | 设置属性值 |
| getProperty(Object obj,String name) | 获得属性值 |

* 提供JavaBean User ，并提供对应的构造方法

public class User {

private String id;

private String username;

private String pwd;

//省略setter和getter

@Override

public String toString() {

return "User [id=" + id + ", username=" + username + ", pwd=" + pwd + "]";

}

* 功能1：设置属性

@Test

public void demo01() throws Exception{

User user = new User();

//设置属性

BeanUtils.setProperty(user, "id", "u001");

BeanUtils.setProperty(user, "username", "jack");

BeanUtils.setProperty(user, "pwd", "1234");

System.out.println(user);

/\* 结果：

\* User [id=u001, username=jack, pwd=1234]

\*/

}

* 功能2：获得属性

@Test

public void demo02() throws Exception{

User user = new User("u001","jack","1234");

//设置属性

String id = BeanUtils.getProperty(user, "id");

String username = BeanUtils.getProperty(user, "username");

System.out.println(id);

System.out.println(username);

/\* 结果：

\* u001

\* jack

\*/

}

* 功能3：封装表单数据，populate方法

@Test

public void demo03() throws Exception{

Map<String, String[]> map = new HashMap<>();

map.put("id", new String[]{"u001"});

map.put("username", new String[]{"jack"});

map.put("pwd", new String[]{"1234"});

//将数据封装到javabean中

User user = new User();

BeanUtils.populate(user, map);

System.out.println(user);

/\* 结果：

\* User [id=u001, username=jack, pwd=1234]

\*/

}

## 综合案例

读取XML中的配置文件信息,使用BeanUtils工具类创建JavaBean对象,将XML中的数据保存到JavaBean类的属性中

@Test

**publicvoid** demo01()**throws** Exception{

SAXReader sax = **new** SAXReader();

//读取XML文档,获取Document对象

Document document = sax.read("bean.xml");

//获取根标签 beans

Element element = document.getRootElement();

//获取beans根标签下的子标签 <bean>

List<Element>beanElement = element.elements();

//遍历集合,获取集合中的bean标签

**for**(Element beanEle : beanElement){

//获取bean标签中的属性className的属性值

String className = beanEle.attributeValue("className");

Classclazz = Class.*forName*(className);

Object obj = clazz.newInstance();

//获取bean标签子标签property

List<Element>proElement = beanEle.elements();

//遍历集合,获取出子标签

**for**(Element proEle : proElement){

//获取子标签的属性name,value的值

String name = proEle.attributeValue("name");

String value = proEle.attributeValue("value");

BeanUtils.*setProperty*(obj, name, value);

}

System.***out***.println(obj);

}

}

