第33天 反射

今日内容介绍

类加载器

反射构造方法

反射成员变量

反射成员方法

反射配置文件运行类中的方法

# 类加载器

## 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

## 类初始化时机

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

## 类加载器

负责将.class文件加载到内在中，并为之生成对应的Class对象。

虽然我们不需要关心类加载机制，但是了解这个机制我们就能更好的理解程序的运行

## 类加载器的组成

Bootstrap ClassLoader 根类加载器

也被称为引导类加载器，负责Java核心类的加载

比如System,String等。在JDK中JRE的lib目录下rt.jar文件中

Extension ClassLoader 扩展类加载器

负责JRE的扩展目录中jar包的加载。

在JDK中JRE的lib目录下ext目录

System ClassLoader 系统类加载器

负责在JVM启动时加载来自java命令的class文件，以及classpath环境变量所指定的jar包和类路径。

通过这些描述就可以知道我们常用的类，都是由谁来加载完成的。

到目前为止我们已经知道把class文件加载到内存了，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?

这就是我们反射要研究的内容。

# 反射

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对应的Class类型的对象。

## Class类

阅读API的Class类得知，Class 没有公共构造方法。Class 对象是在加载类时由 Java 虚拟机以及通过调用类加载器中的 defineClass 方法自动构造的

获取Class对象的三种方式

方式一: 通过Object类中的getObject()方法

Person p = new Person();

Class c = p.getClass();

方式二: 通过 类名.class 获取到字节码文件对象（任意数据类型都具备一个class静态属性,看上去要比第一种方式简单）。

Class c2 = Person.class;

方式三: 通过Class类中的方法（将类名作为字符串传递给Class类中的静态方法forName即可）。

Class c3 = Class.forName("Person");

注意：第三种和前两种的区别

前两种你必须明确Person类型.

后面是指定这种类型的字符串就行.这种扩展更强.我不需要知道你的类.我只提供字符串,按照配置文件加载就可以了

代码演示

/\*

\* 获取.class字节码文件对象的方式

\* 1：通过Object类中的getObject()方法

\* 2: 通过 类名.class 获取到字节码文件对象

\* 3: 反射中的方法,

\* public static Class<?> forName(String className) throws ClassNotFoundException

\* 返回与带有给定字符串名的类或接口相关联的 Class 对象

\*/

**public** **class** ReflectDemo {

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

// 1： 通过Object类中的getObject()方法

// Person p1 = new Person();

// Class c1 = p1.getClass();

// System.out.println("c1 = "+ c1);

// 2: 通过 类名.class 获取到字节码文件对象

// Class c2 = Person.class;

// System.out.println("c2 = "+ c2);

// 3: 反射中的方法

Class c3 = Class.*forName*("cn.itcast\_01\_Reflect.Person");// 包名.类名

System.***out***.println("c3 = " + c3);

}

}

Person类

package cn.itcast\_01\_Reflect;

public class Person {

//成员变量

public String name;

public int age;

private String address;

//构造方法

public Person() {

System.*out*.println("空参数构造方法");

}

public Person(String name) {

this.name = name;

System.*out*.println("带有String的构造方法");

}

//私有的构造方法

private Person(String name, int age){

this.name = name;

this.age = age;

System.*out*.println("带有String，int的构造方法");

}

public Person(String name, int age, String address){

this.name = name;

this.age = age;

this.address = address;

System.*out*.println("带有String, int, String的构造方法");

}

//成员方法

//没有返回值没有参数的方法

public void method1(){

System.*out*.println("没有返回值没有参数的方法");

}

//没有返回值，有参数的方法

public void method2(String name){

System.*out*.println("没有返回值，有参数的方法 name= "+ name);

}

//有返回值，没有参数

public int method3(){

System.*out*.println("有返回值，没有参数的方法");

return 123;

}

//有返回值，有参数的方法

public String method4(String name){

System.*out*.println("有返回值，有参数的方法");

return "哈哈" + name;

}

//私有方法

private void method5(){

System.*out*.println("私有方法");

}

@Override

public String toString() {

return "Person [name=" + name + ", age=" + age + ", address=" + address+ "]";

}

}

## 通过反射获取构造方法并使用

在反射机制中，把类中的成员（构造方法、成员方法、成员变量）都封装成了对应的类进行表示。其中，构造方法使用类Constructor表示。可通过Class类中提供的方法获取构造方法：

返回一个构造方法

* + public Constructor<T> getConstructor(Class<?>... parameterTypes) 获取public修饰, 指定参数类型所对应的构造方法
  + public Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes) 获取指定参数类型所对应的构造方法(包含私有的)

返回多个构造方法

* + public Constructor<?>[] getConstructors() 获取所有的public 修饰的构造方法
  + public Constructor<?>[] getDeclaredConstructors() 获取所有的构造方法(包含私有的)

获取构造方法的代码演示：

**public** **class** ReflectDemo {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException {

//获取Class对象

Class c = Class.*forName*("cn.itcast\_01\_Reflect.Person");//包名.类名

//获取所有的构造方法

//Constructor[] cons = c.getConstructors();

Constructor[] cons = c.getDeclaredConstructors();

**for** (Constructor con : cons) {

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

}

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

//获取一个构造方法

//public Person()

Constructor con1 = c.getConstructor(**null**);

System.***out***.println(con1);

//public Person(String name)

Constructor con2 = c.getConstructor(String.**class**);

System.***out***.println(con2);

//private Person(String name, int age)

Constructor con3 = c.getDeclaredConstructor(String.**class**, **int**.**class**);

System.***out***.println(con3);

//public Person(String name, int age, String address)

Constructor con4 = c.getDeclaredConstructor(String.**class**, **int**.**class**, String.**class**);

System.***out***.println(con4);

}

}

### 通过反射方式，获取构造方法，创建对象

获取构造方法，步骤如下：

1. 获取到Class对象

2. 获取指定的构造方法

3. 通过构造方法类Constructor中的方法，创建对象

public T newInstance(Object... initargs)

代码演示

**public** **class** ConstructorDemo {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException, InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetException {

//1,获取到Class对象

Class c = Class.*forName*("cn.itcast\_01\_Reflect.Person");//包名.类名

//2,获取指定的构造方法

//public Person()

//Constructor con = c.getConstructor(null);

//public Person(String name, int age, String address)

Constructor con = c.getConstructor(String.**class**, **int**.**class**, String.**class**);

//3,通过构造方法类中Constructor的方法，创建对象

//Object obj = con.newInstance(null);

Object obj = con.newInstance("小明", 22, "哈尔滨");

//显示

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

}

}

### 通过反射方式，获取私有构造方法，创建对象

AccessibleObject 类是 Field、Method 和 Constructor 对象的父类。它提供了将反射的对象标记为在使用时取消默认 Java 语言访问控制检查的能力。

对于公共成员、默认（打包）访问成员、受保护成员和私有成员，在分别使用 Field、Method 或 Constructor 对象来设置或获取字段、调用方法，或者创建和初始化类的新实例的时候，会执行访问检查。常用方法如下：

public void **setAccessible**(boolean flag) throws [SecurityException](mk:@MSITStore:D:\传智基础班\JDK_API_1_6_zh_CN.CHM::/java/lang/SecurityException.html)

参数值为 true 则指示反射的对象在使用时应该取消 Java 语言访问检查。参数值为 false 则指示反射的对象应该实施 Java 语言访问检查。

获取私有构造方法，步骤如下：

1. 获取到Class对象

2. 获取指定的构造方法

3. 暴力访问, 通过**setAccessible**(boolean flag)方法

4. 通过构造方法类Constructor中的方法，创建对象

public T newInstance(Object... initargs)

代码演示：

**public** **class** ConstructorDemo2 {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException, InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetException {

//1,获取到Class对象

Class c = Class.*forName*("cn.itcast\_01\_Reflect.Person");//包名.类名

//2,获取指定的构造方法

//private Person(String name, int age)

Constructor con = c.getDeclaredConstructor(String.**class**, **int**.**class**);

//3,暴力反射

con.setAccessible(**true**);//取消 Java 语言访问检查

//4,通过构造方法类中的功能，创建对象

Object obj = con.newInstance("小明", 23);

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

}

}

## 通过反射获取成员变量并使用

在反射机制中，把类中的成员变量使用类Field表示。可通过Class类中提供的方法获取成员变量：

返回一个成员变量

* + public Field getField(String name) 获取指定的 public修饰的变量
  + public Field getDeclaredField(String name) 获取指定的任意变量

返回多个成员变量

* + public Field[] getFields() 获取所有public 修饰的变量
  + public Field[] getDeclaredFields() 获取所有的 变量 (包含私有)

获取成员变量的代码演示：

**public** **class** FieldDemo {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchFieldException, SecurityException {

//获取Class对象

Class c = Class.*forName*("cn.itcast\_01\_Reflect.Person");

//获取成员变量

//多个变量

//Field[] fields = c.getFields();

Field[] fields = c.getDeclaredFields();

**for** (Field field : fields) {

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

}

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

//一个变量

//public int age;

Field ageField = c.getField("age");

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

//private String address

Field addressField = c.getDeclaredField("address");

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

}

}

### 通过反射，创建对象，获取指定的成员变量，进行赋值与获取值操作

获取成员变量，步骤如下：

1. 获取Class对象

2. 获取构造方法

3. 通过构造方法，创建对象

4. 获取指定的成员变量（私有成员变量，通过**setAccessible**(boolean flag)方法暴力访问）

5. 通过方法，给指定对象的指定成员变量赋值或者获取值

* + - public void set(Object obj, Object value)

在指定对象obj中，将此 Field 对象表示的成员变量设置为指定的新值

* + - public Object get(Object obj)

返回指定对象obj中，此 Field 对象表示的成员变量的值

代码演示：

**public** **class** FieldDemo2 {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException, InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetException, NoSuchFieldException {

//1,获取Class对象

Class c = Class.*forName*("cn.itcast\_01\_Reflect.Person");

//2，获取构造方法

//public Person(String name)

Constructor con = c.getConstructor(String.**class**);

//3，通过构造方法，创建对象

Object obj = con.newInstance("小明");

//4，获取指定的成员变量

//public String name;

Field nameField = c.getField("name");

//public int age;

Field ageField = c.getField("age");

//private String address;

Field addressField = c.getDeclaredField("address");

addressField.setAccessible(**true**); //取消 Java 语言访问检查

//5，通过方法，给指定对象的指定成员变量赋值或者获取值

System.***out***.println("name = "+ nameField.get(obj));

System.***out***.println("age = "+ ageField.get(obj));

System.***out***.println("address = "+ addressField.get(obj));

//赋值

ageField.set(obj, 23);

addressField.set(obj, "凯利广场");

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

System.***out***.println("name = "+ nameField.get(obj));

System.***out***.println("age = "+ ageField.get(obj));

System.***out***.println("address = "+ addressField.get(obj));

}

}

## 通过反射获取成员方法并使用

在反射机制中，把类中的成员方法使用类Method表示。可通过Class类中提供的方法获取成员方法：

返回获取一个方法：

* + public Method getMethod(String name, Class<?>... parameterTypes)

获取public 修饰的方法

* + public Method getDeclaredMethod(String name, Class<?>... parameterTypes)

获取任意的方法，包含私有的

参数1: name 要查找的方法名称； 参数2： parameterTypes 该方法的参数类型

返回获取多个方法：

* + public Method[] getMethods() 获取本类与父类中所有public 修饰的方法

public Method[] getDeclaredMethods() 获取本类中所有的方法(包含私有的)

获取成员方法的代码演示：

**public** **class** MethodDemo {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException {

//获取Class对象

Class c = Class.*forName*("cn.itcast\_01\_Reflect.Person");

//获取多个方法

//Method[] methods = c.getMethods();

Method[] methods = c.getDeclaredMethods();

**for** (Method method : methods) {

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

}

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

//获取一个方法：

//public void method1()

Method method = c.getMethod("method1", **null**);

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

//public String method4(String name){

method = c.getMethod("method4", String.**class**);

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

//私有方法

//private void method5()

method = c.getDeclaredMethod("method5", **null**);

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

}

}

### 通过反射，创建对象，调用指定的方法

获取成员方法，步骤如下：

1. 获取Class对象

2. 获取构造方法

3. 通过构造方法，创建对象

4. 获取指定的方法

5. 执行找到的方法

* + - public Object invoke(Object obj, Object... args)

执行指定对象obj中，当前Method对象所代表的方法，方法要传入的参数通过args指定。

代码演示：

**public** **class** MethodDemo2 {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException, InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetException {

//1， 获取Class对象

Class c = Class.*forName*("cn.itcast\_01\_Reflect.Person");

//2,获取构造方法

//public Person(String name, int age, String address){

Constructor con = c.getConstructor(String.**class**, **int**.**class**, String.**class**);

//3，通过构造方法，创建对象

Object obj = con.newInstance("小明", 23, "哈尔滨");

//4，获取指定的方法

//public void method1() 没有返回值没有参数的方法

//Method m1 = c.getMethod("method1", null);

//public String method4(String name)

Method m4 = c.getMethod("method4", String.**class**);

//5，执行找到的方法

//m1.invoke(obj, null);

Object result = m4.invoke(obj, "itcast");

System.***out***.println("result = " + result);

}

}

### 通过反射，创建对象，调用指定的private 方法

获取私有成员方法，步骤如下：

1. 获取Class对象

2. 获取构造方法

3. 通过构造方法，创建对象

4. 获取指定的方法

5. 开启暴力访问

6. 执行找到的方法

* + - public Object invoke(Object obj, Object... args)

执行指定对象obj中，当前Method对象所代表的方法，方法要传入的参数通过args指定。

代码演示：

**public** **class** MethodDemo3 {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException, InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetException {

//1， 获取Class对象

Class c = Class.*forName*("cn.itcast\_01\_Reflect.Person");

//2,获取构造方法

//public Person(String name, int age, String address){

Constructor con = c.getConstructor(String.**class**, **int**.**class**, String.**class**);

//3，通过构造方法，创建对象

Object obj = con.newInstance("小明", 23, "哈尔滨");

//4，获取指定的方法

//private void method5(){

Method m5 = c.getDeclaredMethod("method5", **null**);

//5,开启暴力访问

m5.setAccessible(**true**);

//6，执行找到的方法

m5.invoke(obj, **null**);

}

}

# 反射练习

## 泛型擦除

思考，将已存在的ArrayList<Integer>集合中添加一个字符串数据，如何实现呢？

我来告诉大家，其实程序编译后产生的.class文件中是没有泛型约束的，这种现象我们称为泛型的擦除。那么，我们可以通过反射技术，来完成向有泛型约束的集合中，添加任意类型的元素

代码如下：

**public** **class** ReflectTest {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException, NoSuchMethodException, SecurityException, IllegalAccessException, IllegalArgumentException, InvocationTargetException {

ArrayList<Integer> list = **new** ArrayList<Integer>();

//添加元素到集合

list.add(**new** Integer(30));

list.add(**new** Integer("12345"));

list.add(123);

//list.add("哈哈");//因为有泛型类型的约束

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

//通过反射技术，实现添加任意类型的元素

//1, 获取字节码文件对象

//Class c = list.getClass();

//Class c = ArrayList.class;

Class c = Class.*forName*("java.util.ArrayList");

//2, 找到add()方法

// public boolean add(E e)

Method addMethod = c.getMethod("add", Object.**class**);

//3， 执行add()方法

addMethod.invoke(list, "哈哈");// list.add("哈哈");

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

}

}

## 反射配置文件

* 通过反射配置文件，运行配置文件中指定类的对应方法

读取Peoperties.txt文件中的数据，通过反射技术，来完成Person对象的创建

Peoperties.txt文件内容如下：

className=cn.itcast\_01\_Reflect.Person

methodName=method5

读取配置文件，调用指定类中的对应方法

**public** **class** ReflectTest2 {

**public** **static** **void** main(String[] args)

**throws** FileNotFoundException, IOException, ClassNotFoundException, NoSuchMethodException, SecurityException,

InstantiationException, IllegalAccessException, IllegalArgumentException, InvocationTargetException {

// 通过Properties集合从文件中读取数据

Properties prop = **new** Properties();

// 读取文件中的数据到集合中

prop.load(**new** FileInputStream("properties.txt"));

// 获取键所对应的值

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

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

// 1，获取Person.class 字节码文件对象

Class c = Class.*forName*(className);

// 2，获取构造方法

// public Person(String name, int age, String address)

Constructor con = c.getConstructor(String.**class**, **int**.**class**, String.**class**);

// 3,创建对象

Object obj = con.newInstance("小明", 20, "中国");

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

// 4，获取指定的方法

// private void method5(){}

String methodName = prop.getProperty("methodName");

Method m5 = c.getDeclaredMethod(methodName, **null**);

// 5,开启暴力访问

m5.setAccessible(**true**);

// 6，执行找到的方法

m5.invoke(obj, **null**);

}

}

# 总结

## 知识点总结

如何获取.Class文件对象

1, 通过Object类 getClass()方法获取 Class对象

2, 通过类名.class 方式 获取 Class对象

3, 通过反射的方式, Class.forName(String classname) 获取Class对象

public static Class<?> forName(String className)throws ClassNotFoundException

返回与带有给定字符串名的类或接口相关联的 Class 对象

通过反射， 获取类中的构造方法，并完成对象的创建

获取指定的构造方法

public Constructor<T> getConstructor(Class<?>... parameterTypes)

获取指定的public修饰的构造方法

public Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes)

获取指定的构造方法，包含私有的

获取所有的构造方法

public Constructor<?>[] getConstructors() 获取所有的public 修饰的构造方法

public Constructor<?>[] getDeclaredConstructors() 获取所有的构造方法，包含私有的

通过反射， 获取类中的构造方法，并完成对象的创建

步骤：

1，获取字节码文件对象

2,通过字节码文件对象 ，获取到指定的构造方法

getConstructor(参数);

3,通过构造方法，创建对象

public T newInstance(Object... initargs)

私有构造方法，创建对象

1，获取字节码文件对象

2,通过字节码文件对象 ，获取到指定的构造方法

getDeclaredConstructor (参数);

3,暴力访问

con.setAccessible(true);

4,通过构造方法，创建对象

public T newInstance(Object... initargs)

通过反射，获取Class文件中的方法

获取指定的方法

public Method getMethod(String name, Class<?>... parameterTypes)

获取指定的public方法

public Method getDeclaredMethod(String name, Class<?>... parameterTypes)

获取指定的任意方法，包含私有的

获取所有的方法

public Method[] getMethods() 获取本类与父类中所有public 修饰的方法

ublic Method[] getDeclaredMethods()获取本类中所有的方法，包含私有的

通过反射，调用方法

步骤：

1，获取Class对象

2,获取构造方法，创建对象

3,获取指定的public方法

4,执行方法

public Object invoke(Object obj, Object... args)

私有方法的调用：

1，获取Class对象

2,获取构造方法，创建对象

3,获取指定的private方法

4,开启暴力访问

m5.setAccessible(true);

5,执行方法

public Object invoke(Object obj, Object... args)

通过反射，获取成员变量(字段)

获取指定的成员变量

public Field getField(String name) 获取public修饰的成员变量

public Field getDeclaredField(String name) 获取任意的成员变量，包含私有

获取所有的成员变量

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

public Field[] getDeclaredFields() 获取司所有的成员变量，包含私有

通过反射，获取成员 变量，并赋值使用

步骤：

1，获取字节码文件对象

2,获取构造方法，创建对象

3,获取指定的成员变量

4,对成员变量赋值\获取值操作

public void set(Object obj, Object value) 赋值

public Object get(Object obj) 获取值

私有成员变量的使用

步骤：

1，获取字节码文件对象

2,获取构造方法，创建对象

3,获取指定的成员变量

4,开启暴力访问

5,对成员变量赋值\获取值操作

public void set(Object obj, Object value) 赋值

public Object get(Object obj) 获取值