# 反射机制

## 简介

获得类的类型信息Class<?>实例对象，通过该对象来使用类。

获得Class<?>实例对象的三种方式：

1.类名.class;

2.object.getClass();

3.Class.forName(“类的完整路径名”);该方法使用它类无需import语句，但是可能发生ClassNotFoundException。

## 反射的应用

### 反射与工厂设计模式

通过反射和泛型的配合，可以设计不局限于接口类型的工厂设计模式。

|  |
| --- |
| **package** com.pcq.react;  **import** java.lang.reflect.InvocationTargetException;  */\*\*  \* 反射和工厂设计模式  \* 工厂设计模式应不限于具体的类，甚至于具体的接口类型。  \* 前者需要反射，后者需要泛型配合  \*/* **public class** ReactAndFactory {  **public static void** main(String[] args) {  IMessageService msgService = Factory.*getInstance*(**"com.pcq.react.NetMessage"**, IMessageService.**class**);  msgService.send(**"net message"**);  ILifeService lifeService = Factory.*getInstance*(**"com.pcq.react.HomeService"**, ILifeService.**class**);  lifeService.apply(**"home live"**);  } }  **class** Factory {  **private** Factory() {}   */\*\*  \*  \** ***@param className*** *实例化对象的类路径  \** ***@param clazz*** *实例化对象的接口类型  \** ***@param <T>*** *\** ***@return*** *\*/* **public static** <T> T getInstance(String className, Class<T> clazz) {  T instance = **null**;  **try** {  instance = (T)Class.*forName*(className).getDeclaredConstructor().newInstance();  } **catch** (Exception e) {  e.printStackTrace();  }  **return** instance;  } }  */\*\*  \* 消息服务接口  \*/* **interface** IMessageService {  **void** send(String msg); }  */\*\*  \* 生活服务接口  \*/* **interface** ILifeService {  **void** apply(String service); }  **class** NetMessage **implements** IMessageService {   @Override  **public void** send(String msg) {  System.***out***.println(**"send："** + msg);  } }  **class** HomeService **implements** ILifeService {   @Override  **public void** apply(String service) {  System.***out***.println(**"support："** + service);  } } |

### 反射与单例设计模式（懒汉式）

在多线程情况下，懒汉式并不安全。此时需要通过同步类型对象来解决单例问题。

|  |
| --- |
| **package** com.pcq.react; */\*\*  \* 反射与单例模式（懒汉式）  \*/* **public class** ReactAndSingleton {  **public static void** main(String[] args) {  **for**(**int** i = 0; i < 5; i++) {  **new** Thread(  ()->{  Singleton.*getInstance*();  }  ).start();  }  } }  */\*\*  \* 标准的多线程下饿汉式单例模式  \*/* **class** Singleton {  **private static volatile** Singleton *instance* = **null**;*//与主内存中保持一致* **private** Singleton() {  System.***out***.println(Thread.*currentThread*().getName() + **": exe getInstance"**);  }  **public static** Singleton getInstance() {  **if**(*instance* == **null**) {  **synchronized** (Singleton.**class**) {  **if**(*instance* == **null**) {*//同步逻辑中需要再次判断  instance* = **new** Singleton();  }  }  }  **return** *instance*;  } } |