转 3 1

a通过反射创建对象

2012 26日 15:50:01

java 行时动态获取某个类的类信息，这就是java的反射。

一、 射创建不带参数的对象

这个 单，不需要获取这个类的构造方法类，不需要指明构造参数的参数列表。下面是要使用的类和方法，使用步骤

Clas

Cla 实例表示正在运行的 Java 应用程序中的类和接口。枚举是一种类，注释是一种接口。每个数组属于被映射为 型 数组都共享该Class 对象。基本的 Java 类型（boolean、byte、char、short、int、long、float 和 double）

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

以下示例使用 Class 对象来显示对象的类名：

```
    void printClassName(Object obj) {          System.out.println("The class of " + obj +
().getName());       }
```

这个类就是类对象，是具体某个类对象，通常我们说说的对象是，某个类的对象，而Class是类级别的对象，描述的类的信息。

例如String a=new String();

这个a指的是类String的对象，那么什么是类对象呢，看这个例子：

Class c=String.class;这个c就是String的类对象，描述的是String的信息。不是对象信息。

Class forName(String className):

返回与带有给定字符串名的类或接口相关联的 Class 对象。调用此方法等效于：

```
Class.forName(className, true, currentLoader)
```

其中 currentLoader 表示当前类的定义类加载器。
例如，以下代码片段返回命名为 java.lang.Thread 的类的运行时Class 描述符。

```
Class t = Class.forName("java.lang.Thread")
```

调用 forName("X") 将导致命名为X 的类被初始化。

参数：

className - 所需类的完全限定名。

返回：

具有指定名的类的 Class 对象。

抛出：

LinkageError - 如果链接失败

ExceptionInInitializerError - 如果此方法所激发的初始化失败

ClassNotFoundException - 如果无法定位该类

通过指明类所在的具体位置来生成这个类的类对象。注意：**className**指明的是具体的这个类所在的位置，例如**java**内置的**String**类，那么它的位置是 **className=j**
String;如果自己创建的类，当加载到项目中后请也指明其具体的位置，例如有一个自己创建的类**GameCanvas**在 **com.jijing.gameDemo**中，**className=com.jijin**
o.GameCanvas;

```
public T newInstance();
```

创建此 Class 对象所表示的类的一个新实例。如同用一个带有一个空参数列表的new 表达式实例化该类。如果该类尚未初始化，则初始化这个类。
注意，此方法传播 null 构造方法所抛出的任何异常，包括已检查的异常。使用此方法可以有效地绕过编译时的异常检查，而在其他情况下都会执行该检查。 Constructor.newInstance 方法将该构造方法所抛出的任何异常包装在一个（已检查的）InvocationTargetException 中，避免了这一问题。

返回：

此对象所表示的类的一个新分配的实例。

抛出：

IllegalAccessException - 如果该类或其 null 构造方法是不可访问的。

InstantiationException - 如果此Class 表示一个抽象类、接口、数组类、基本类型或 void； 或者该类没有 null 构造方法； 或者由于其他某种原因实例化失败。

ExceptionInInitializerError - 如果该方法引发的初始化失败。

SecurityException - 如果存在安全管理器*s*，并满足下列任一条件：

- 调用 s.checkMemberAccess(this, Member.PUBLIC) 拒绝创建该类的新实例
- 调用者的类加载器不同于也不是当前类的类加载器的一个祖先，并且对 s.checkPackageAccess() 的调用拒绝访问该类的包

这个方法是 Class类中的方法，且只能创建不带任何参数的对象形式(构造函数没有参数列表)。直接返回T模版类型的对象，自己要把它转换为实

下面是实际例子：

```
Class c=Class.forName("java.lang.String");
```

String s=(String)c.newInstance();

二、创建带参数的对象

创建带参数的对象就比较复杂 ，由于构造函数不止一个，且参数列表参数类型不一要，所以自己要指明构造函数中的参数列表，自己也必须明确参数
否则会出现异常的。下面是一些类和方法的说明：

```
public Constructor<T> getConstructor(Class<?>... parameterTypes)                    throws NoSuchMethodException,
                SecurityException
```

返回一个 Constructor 对象，它反映此Class 对象所表示的类的指定公共构造方法。parameterTypes 参数是Class 对象的一个数组，这些 Class 对象按声明顺序标
的形参类型。 如果此Class 对象表示非静态上下文中声明的内部类，则形参类型作为第一个参数包括显示封闭的实例。
要反映的构造方法是此 Class 对象所表示的类的公共构造方法，其形参类型与parameterTypes 所指定的参数类型相匹配。

参数：

parameterTypes - 参数数组

返回：

与指定的 parameterTypes 相匹配的公共构造方法的Constructor 对象

抛出：

NoSuchMethodException - 如果找不到匹配的方法。

SecurityException - 如果存在安全管理器s，并满足下列任一条件：

- 调用 s.checkMemberAccess(this, Member.PUBLIC) 拒绝访问构造方法
- 调用者的类加载器不同于也不是当前类的类加载器的一个祖先，并且对 s.checkPackageAccess() 的调用拒绝访问该类的包

从以下版本开始：

JDK1.1

里面要注意的就是parametrrTypes这个Class类型的参数列表，用来指明当前你要创建的类的某个构造方法的参数列表，且顺序要是当前这个类的构造方法中声明的参数列表是Class类型，指明一些类信息就可以了。可以按照下面的规则来传入相应的类信息参数列表：例如：**Integer类，它有两个构造函数，**

```
public Integer(int value)
```

构造一个新分配的 Integer 对象，它表示指定的int 值。

参数：

value - Integer 对象表示的值。

这个构造函数指明的Class参数列表是一个Interger Class类型的类对象 这么写：Interger.**TYPE：返回的是其Class类信息**

另一个构造函数：

```
public Integer(String s)          throws NumberFormatException
```

构造一个新分配的 Integer 对象，它表示String 参数所指示的int 值。使用与parseInt 方法（对基数为 10 的值）相同的方式将该字符串转换成int 值。

参数：

s - 要转换为Integer 的String。

抛出：

NumberFormatException - 如果String 不包含可解析的整数。

另请参见：

parseInt(java.lang.String, int)

其参数类型是这样创建：String.class

class参数列表创建总结：如果是基本类型请使用它们的Type字段，如果是非基本类型使用class字段来返回其CLass类信息。

现在就是如何创建对象了，对于带参数的对象，你要传入参数，可以使用newInstance(参数列表)来创建对象,这个Constructor对象的方法。

```
public T newInstance(Object... initargs)              throws InstantiationException,                    IllegalAccessException,
        IllegalArgumentException,                     InvocationTargetException
```

使用此 Constructor 对象表示的构造方法来创建该构造方法的声明类的新实例，并用指定的初始化参数初始化该实例。个别参数会自动解包，以匹配基本形参，必本参数和引用参数都要进行方法调用转换。
如果底层构造方法所需形参数为 0，则所提供的 initargs 数组的长度可能为 0 或 null。

如果构造方法的声明类是非静态上下文的内部类，则构造方法的第一个参数需要是封闭实例；请参阅*Java 语言规范* 第 15.9.3 节。

如果所需的访问检查和参数检查获得成功并且实例化继续进行，这时构造方法的声明类尚未初始化，则初始化这个类。

如果构造方法正常完成，则返回新创建且已初始化的实例。

参数：

initargs - 将作为变量传递给构造方法调用的对象数组；基本类型的值被包装在适当类型的包装器对象（如 Float 中的 float）中。

返回：

通过调用此对象表示的构造方法来创建的新对象

抛出：

IllegalAccessException - 如果此Constructor 对象实施 Java 语言访问控制并且底层构造方法是不可访问的。

IllegalArgumentException - 如果实参和形参的数量不同；如果基本参数的解包转换失败；如果在可能的解包后，无法通过方法调用转换将参数值转换的形参类型；如果此构造方法属于枚举类型。

InstantiationException - 如果声明底层构造方法的类表示抽象类。

InvocationTargetException - 如果底层构造方法抛出异常。

ExceptionInInitializerError - 如果此方法引发的初始化失败。

好了，下面就是一个实例，包括构造函数中包括自定义的类，基本数据类型，和非基本数据类型。

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package com.jijing.classDemo;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
    rt java.util.logging.Level;
    rt java.util.logging.Logger;


/**
 *
 * @author Administrator
 * 用来演示通过反射来创建对象，带参数的构造方法
 */
public class ClassMain {

    public ClassMain(){
    }
    public static void main(String args[]){
        //创建不带参数的对象
        //ReflectClass rc1=(ReflectClass) ClassMain.getInstance("com.jijing.classDemo.ReflectClass");
        //System.out.println("ReflectClass111="+rc1);
        System.out.println("*****************************");
        ReflectClass rc2=(ReflectClass) ClassMain.getInstance("com.jijing.classDemo.ReflectClass",
                                                              new Class[]{Integer.TYPE,String.class,MyClass.class},
                                                              new Object[]{20,"我是ReflectClass",new MyClass("我是MyClass")});
        System.out.println("ReflectClass222="+rc2);
    }


    /**
     *
     * @param className 类路劲的名字
     * @return 返回根据className指明的类信息
     */
    public static Class getclass(String className){
        Class c=null;
        try {
            c=Class.forName(className);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        }
        return c;
    }

    /**
     *
     * @param name 类路劲
     * @param classParas Class类信息参数列表
     *  如果是基本数据类型是可以使用其Tpye类型，如果用class字段是无效的
     *  如果是非数据类型可以使用的class字段来创建其Class类信息对象，这些都要遵守。
     * @param paras       实际参数列表数据
     * @return            返回Object引用的对象，实际实际创建出来的对象，如果要使用可以强制转换为自己想要的对象
     *
     * 带参数的反射创建对象
     */
    public static Object getInstance(String name,Class classParas[],Object paras[]){
        Object o=null;
        try {
            Class c=getclass(name);
            Constructor con=c.getConstructor(classParas);//获取使用当前构造方法来创建对象的Constructor对象，用它来获取构造函数的一些
```

```
            try { 65                    //信息
                o=con.newInstance(paras);//传入当前构造函数要的参数列表
            } catch (InstantiationException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IllegalAccessException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IllegalArgumentException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (InvocationTargetException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            }
        } catch (NoSuchMethodException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SecurityException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        }

        return o;//返回这个用Object引用的对象
    }

    /**
     *
     * @param name 类路劲
     * @return 不带参数的反射创建对象
     */
    public static Object getInstance(String name){
        Class c=getclass(name);
        Object o=null;
        try {
             o=c.newInstance();
        } catch (InstantiationException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        }
        return o;
    }

}
/**
 *
 * @author Administrator
 *   自定义一个类型
 */
class MyClass{
    private String name="";//名字显示,用来表明创建成功
    MyClass(String name){
        this.name=name;
        show();//显示
    }
    public void show(){
        System.out.println(name);
    }
    @Override
    public String toString(){
        return "MyClass="+name;
    }
}

/**
 *
 * @author Administrator
 *  通过反射创建对象
 */
class ReflectClass{
    private String name="ReflectClass";
    public ReflectClass(int age,String name,MyClass my){
         this.name=name;
         show(age,name,my);
    }
//    ReflectClass(){//构造函数重载，使用不同的参数列表创建对象
```

```
//        //没有带参数的构造方法
//    }
    /**
     *
     * @param age
     * @param name
     * @param my
     */
    public final void show(int age,String name,MyClass my){
        System.out.println("age="+age+" name="+name+" my="+my);
    }
    @Override
    public String toString(){
        return "ReflectClass="+name;
    }
}
```

上面有三个方法自己可以拿去用，已经封装好了，就是这三个方法。

```
/**
     *
     * @param className 类路劲的名字
     * @return 返回根据className指明的类信息
     */
    public static Class getclass(String className){
        Class c=null;
        try {
            c=Class.forName(className);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        }
        return c;
    }



/**
     *
     * @param name 类路劲
     * @param classParas Class类信息参数列表
     *  如果是基本数据类型是可以使用其Tpye类型，如果用class字段是无效的
     *  如果是非数据类型可以使用的class字段来创建其Class类信息对象，这些都要遵守。
     * @param paras       实际参数列表数据
     * @return            返回Object引用的对象，实际实际创建出来的对象，如果要使用可以强制转换为自己想要的对象
     *
     * 带参数的反射创建对象
     */
    public static Object getInstance(String name,Class classParas[],Object paras[]){
        Object o=null;
        try {
            Class c=getclass(name);
            Constructor con=c.getConstructor(classParas);//获取使用当前构造方法来创建对象的Constructor对象，用它来获取构造函数的一些
            try {
                //信息
                o=con.newInstance(paras);//传入当前构造函数要的参数列表
            } catch (InstantiationException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IllegalAccessException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IllegalArgumentException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (InvocationTargetException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            }
        } catch (NoSuchMethodException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SecurityException ex) {
```

```
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        }

        return o;//返回这个用Object引用的对象
    }



/**
     *
     * @param name 类路劲
     * @return 不带参数的反射创建对象
     */
    public static Object getInstance(String name){
        Class c=getclass(name);
        Object o=null;
        try {
             o=c.newInstance();
        } catch (InstantiationException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        }
        return o;
    }



像上面的带参数的getInstance反射可以该写一下：



/**
     *
     * @param name 类路劲
     * @param paras     实际参数列表数据
     * @return          返回Object引用的对象，实际实际创建出来的对象，如果要使用可以强制转换为自己想要的对象
     *
     * 带参数的反射创建对象
     */
    public static Object getInstance(String name,Object paras[]){
        Object o=null;
        try {
            Class c=getclass(name);
            int len=paras.length;
            Class classParas[]=new Class[len];
            for(int i=0;i<len;++i){
                classParas[i]=paras[i].getClass();//返回类信息
                System.out.println("classParas[i]="+classParas[i]);
            }
            Constructor con=c.getConstructor(classParas);//获取使用当前构造方法来创建对象的Constructor对象，用它来获取构造函数的一些
            try {
                //信息
                o=con.newInstance(paras);//传入当前构造函数要的参数列表
            } catch (InstantiationException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IllegalAccessException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IllegalArgumentException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            } catch (InvocationTargetException ex) {
                Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
            }
```

```
        } catch (NoSuchMethodException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SecurityException ex) {
            Logger.getLogger(ClassMain.class.getName()).log(Level.SEVERE, null, ex);
        }

        return o;//返回这个用Object引用的对象
    }
```

但是上面会出现一个问题，如果在编写类时没有注意会出现java.lang.NoSuchMethodException异常，因为基本类型要转换为类的形式才可以通过。 例如这个类：

```
class ReflectClass{
    private String name="ReflectClass";
    public ReflectClass(Integer age,String name,MyClass my){
        this.name=name;
        show(age,name,my);
    }
//    ReflectClass(){//构造函数重载，使用不同的参数列表创建对象
//        //没有带参数的构造方法
//    }
    /**
     *
     * @param age
     * @param name
     * @param my
     */
    public final void show(Integer age,String name,MyClass my){
        System.out.println("age="+age+" name="+name+" my="+my);
    }
    @Override
    public String toString(){
        return "ReflectClass="+name;
    }
}
```

如果要getClass()的话一定要以类的形式存在，请注意这点。

Java 反射方法的运用（通过反射创建对象） 笑 235

java 中通过反射创建对象

java使用反射创建对象 - CSDN博客

Class对象中包括构造器(Constructor)、属性(Field)、方法(Method)。下面要讲的是通过反射来构造对应类的实…

反射创建对象

反射进行对象的创建,并在反射的时候,进行赋值... 反射进行对象的创建,并在反射的时候,进行赋值 综合评分:0 收…

创 几种方式 2208

在ja ，所有对象的创建都是基于原型的。在js中任意的对象都有一个内部属性[[Prototype]]。这个属性…

Java - 如何通过反射创建对象？ 142

- 方法1：通过类对象调用newInstance()方法，例如：String.class.newInstance() - 方法2：通过类对象的getCon…

Java通过反射创建对象(带参数例子) - CSDN博客

java.lang.Class类和反射机制创建对象 java.lang.Class类 Java程序在运行时,Java运行时系统一直对所有的对…

通过反射创建对象? - CSDN博客

通过反射创建对象?2017年05月14日 17:21:59 阅读数:294 - 方法1:通过类对象调用newInstance()方法,例如:Strin…

Java反射机制（创建Class对象的三种方式） 28

Java反射机制（创建Class对象的三种方式） 1：了解什么是反射机制？ 在通常情况下，如果有一个类，可以通…

JAVA反射机制创建对象 - CSDN博客

在运行的时候来构造无参数构造器或有参数构造器的对象,实现运行时创建使用类对象的方式,这就是通过JAVA的…

通过字符串创建对象并访问类中的方法(利用java的反射) - CSDN博客

首先,通过字符串创建对象,也就是说同一个包下有好多xxxclass.java文件,这些文件中都有相同的属性和方法,那么…

单例防止暴力反射和反序列化创建对象 776

单例防止暴力反射和反序列化创建对象 public class SigletonDemo06 implements Serializable { private sta…

番外 01：Spring IoC 实现原理简析，Java的反射机制，通过类名创建… 588

转载请注明来源 赖赖的博客前景概要在 01 走进Spring，Context、Bean和IoC 中，我们看到了强大的Spring通…

Java通过反射创建对象 - CSDN博客

通过反射创建对象 创建对象之前,我们必须先知道要为哪个类创建对象。我们需要拿到这个类的全路径名。类似j…

通过反射,创建类的实例 - CSDN博客

sysytem库中有一个Activator类,可以用此类的CreateInstance方法来创对象实例; 1.创建目标程序集: 2.主入口程…

关于java 反射 创建对象 调用 有参数 的 构造函数 499

反射创建对象 当我们反射创建对象，一般采用 Class clazz = Class.forName("java.lang.String"); Object instance…

Java之反射类的构造函数，通过单元测试反射创建类的对象 1985

需要测试的Person类的源代码如下： package cn.itcast.reflect; import java.util.List; public class Person { publ...

个人资料

freecodenow

原创 6 粉丝 12

等级： 博客 3 访问

积分： 903 排名

最新文章

JDWP Transport dt_socke
e, TRANSPORT_INIT(51(

Spring MVC 实现一个控制

用JS阻止事件冒泡

spring如何使用多个xml配

Js中 关于top、clientTop、
Top的用法

个人分类

spring

hibernate

html+css

javascript

java

展开

归档

2016年1月

2013年4月

2013年3月

2013年2月

2013年1月

2012年12月

2012年9月

2012年8月

2012年7月

热门文章

Java通过反射创建对象
阅读量：34001

window.XMLHttpRequest
阅读量：26808

spring如何使用多个xml配
阅读量：10115

反向AJAX
阅读量：8699

exe4j使用教程，inno setu
阅读量：5477

最新评论

从SVN上更新项目后，My
lccone：[reply]liujici200860[/re

redhat5.4下安装jdk和t...