反射

什么是反射

> 正常情况下，如果已经存在一个类，就可以通过这个类进行对象的实例化操作，这属于正常情况。如果现在需要通过一个对象获取类的详细信息，这时就需要反射。

例子

示例代码

```
package com.oaec.classtype;

import com.oaec.entity.User;

public class ClassType {
    public static void main(String[] args) {
        User user = new User();
        System.out.println(user.getClass().getName());
    }
}
```

执行结果

```
com.oaec.entity.User
```

分析

此时，我们发现，输出了`User`类的全类名，也就是说，通过反射机制，反向获取到了类的详细信息。

其中`getClass()`方法是`Object`类中定义的方法，其方法声明如下：

```
public final Class<?> getClass()
```

此方法返回的是`Class`类型的实例，所有，`Class`是一切反射的根源

认识Class类

概述

`Class`类是一切反射的根源，此类的定义如下：

```
public final class Class<T>
extends Object
implements Serializable, GenericDeclaration, Type, AnnotatedElement
```

通过API文档可以发现：

1. `Class`类被定义为最终类，是不允许有子类的，而且此类声明的时候时候使用了泛型声明，也就是说说使用的时候要指明类型参数。
2. `Class`类没有 构造方法，想获取该类的实例一般使用以下三种方法：
    - 第一种：通过`Object`类中的`getClass()`方法获取
    - 第二种：通过`类.class`获取
    - 第三种：通过`Class`类中的静态方法`forName(String className)`获取

第一种：getClass();

示例代码

```
import java.util.Date;

public class ClassType {
    public static void main(String[] args) {
        Date date = new Date();
        Class<? extends Date> class1 = date.getClass();
        String name = class1.getName();
        System.out.println(name);//java.util.Date
    }
}
```

执行结果

```
java.util.Date
```

第二种：类.class

示例代码

```
import java.util.Date;

public class ClassType {
    public static void main(String[] args) {
        String name = Date.class.getName();
        System.out.println(name);
    }
}
```

执行结果

```
java.util.Date
```

第三种：Class.forName(String name)[重点]

示例代码

```
public class ClassType {
    public static void main(String[] args) throws ClassNotFoundException {
        String name = Class.forName("java.util.Date").getName();
        System.out.println(name);
    }
}
```

执行结果

```
java.util.Date
```

Java.lang.reflect库

1. `Field` 类：代表类的成员变量（成员变量也称为类的属性）。
2. `Method` 类：代表类的方法。
3. `Constructor` 类：代表类的构造方法。

使用Class类进行对象的实例化

调用无参构造方法

例子

示例代码

```
try {
    Class<?> forName = Class.forName("com.oaec.entity.User");
    Object newInstance = forName.newInstance();
    System.out.println(newInstance);
} catch (Exception e) {
    e.printStackTrace();
}
/*****************************/
try {
    Class<?> forName = Class.forName("com.oaec.entity.User");
    Object newInstance = forName.getConstructor(new Class[]{}).newInstance(new Object[]{});
    System.out.println(newInstance);
} catch (Exception e) {
    e.printStackTrace();
}
```

执行结果

```
User [id=0, userName=null, password=null]
```

调用有参构造方法

例子

示例代码

```
try {
    Class<?> forName = Class.forName("com.oaec.entity.User");
    Object newInstance = forName.getConstructor(new Class[]
{int.class,String.class,String.class}).newInstance(new Object[] {1,"张三","12345"});
    System.out.println(newInstance);
} catch (Exception e) {
    e.printStackTrace();
}
```

执行结果

```
User [id=1, userName=张三, password=12345]
```

例子：简单工厂模式的改进

之前我们设计的简单工厂模式：

```
public class FruitFactory {
    public static IFruit createFruit(String className){
        if(className.equals("苹果")){
            return new Apple();
        }else if(className.equals("橘子")){
            return new Orange();
        }else if(className.equals("香蕉")){
            return new Banana();
        }
        return null;
    }
}
```

分析：通过该工厂类我们发现，只有添加一种水果，就需要对工厂类进行改写，这样很不好，需要改进。

改进之后：

```
public class FruitFactory {
    public static IFruit createFruit(String className){
        IFruit fruit = null;
        try {
            fruit = (IFruit) Class.forName(className).newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return fruit;
    }
}
```

分析：此时，我们通过反射机制创建对象，调用方法时传入全类名就可以，大大降低了程序的耦合性

使用Class类获取类的信息

示例类User

接下来的所有反射操作都操作此类：

```java
/**
 * User实体类
 * @author Kevin
 */
public class User {
    //公共属性
    public String data;
    //1.私有化的属性
    private int id;
    private String userName;
    private String password;
    //2.无参构造方法
    public User() {
        super();
        System.out.println("无参构造方法");
    }
    //3.有参构造方法
    public User(int id, String userName, String password) {
        super();
        this.id = id;
        this.userName = userName;
        this.password = password;
    }
    //4.为私有化的属性提供set和get方法
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }

    //5.重写toString()
    @Override
    public String toString() {
        return "User [id=" + id + ", userName=" + userName + ", password=" + password + "]";
    }
    //私有方法，测试暴力反射使用
    private void show(String str){
        System.out.println(str);
    }
}
```

获取构造方法

根据参数类型获取指定的构造方法

方法声明

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

示例代码

```
//获取User类的Class对象
Class<?> cls = Class.forName("com.oaec.entity.User");
//获取三个参数的构造方法
Constructor<?> constructor = cls.getConstructor(new Class[]
{int.class,String.class,String.class});
System.out.println(constructor);
```

执行结果

```
public com.oaec.entity.User(int,java.lang.String,java.lang.String)
```

获取全部构造方法

方法声明

```
public Constructor<?>[] getConstructors()
                                 throws SecurityException
```

示例代码

```
// 获取User类的Class对象
Class<?> cls = Class.forName("com.oaec.entity.User");
// 获取全部的构造方法，返回数组
Constructor<?>[] constructors = cls.getConstructors();
// 遍历输出
for (Constructor<?> constructor : constructors) {
    System.out.println(constructor);
}
```

执行结果

```
public com.oaec.entity.User()
public com.oaec.entity.User(int,java.lang.String,java.lang.String)
```

获取方法和动态调用方法

获得当前类以及父类的公共方法

方法声明

```
public Method[] getMethods()throws SecurityException
```

示例代码

```
// 获取User类的Class对象
Class<?> cls = Class.forName("com.oaec.entity.User");
// 获得当前类以及父类的公共方法
Method[] methods = cls.getMethods();
//遍历输出
for (Method method : methods) {
    System.out.println(method);
}
```

执行结果

```
public java.lang.String com.oaec.entity.User.toString()
public int com.oaec.entity.User.getId()
public void com.oaec.entity.User.setId(int)
public java.lang.String com.oaec.entity.User.getUserName()
public void com.oaec.entity.User.setUserName(java.lang.String)
public java.lang.String com.oaec.entity.User.getPassword()
public void com.oaec.entity.User.setPassword(java.lang.String)
public final void java.lang.Object.wait() throws java.lang.InterruptedException
public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
public final native void java.lang.Object.wait(long) throws java.lang.InterruptedException
public boolean java.lang.Object.equals(java.lang.Object)
public native int java.lang.Object.hashCode()
public final native java.lang.Class java.lang.Object.getClass()
public final native void java.lang.Object.notify()
public final native void java.lang.Object.notifyAll()
```

通过执行结果可以发现，所输出的方法包括当前类和父类的公共方法

获得当前类申明的所有方法

方法声明

```
public Method[] getDeclaredMethods()throws SecurityException
```

示例代码

```
// 获取User类的Class对象
Class<?> cls = Class.forName("com.oaec.entity.User");
// 获得当前类申明的所有方法
Method[] declaredMethods = cls.getDeclaredMethods();
//遍历输出
for (Method method : declaredMethods) {
    System.out.println(method);

}
```

执行结果

```
public java.lang.String com.oaec.entity.User.toString()
public int com.oaec.entity.User.getId()
public void com.oaec.entity.User.setId(int)
public java.lang.String com.oaec.entity.User.getUserName()
public void com.oaec.entity.User.setUserName(java.lang.String)
public java.lang.String com.oaec.entity.User.getPassword()
public void com.oaec.entity.User.setPassword(java.lang.String)
private void com.oaec.entity.User.show(java.lang.String)
```

通过执行结果可以看出，获得了当前类定义的所有方法，包括私有方法

获得当前类以及父类指定的公共方法

方法声明

```
public Method getMethod(String name,
                        Class<?>... parameterTypes)
                 throws NoSuchMethodException,
                        SecurityException
```

示例代码

```
// 获取User类的Class对象
Class<?> cls = Class.forName("com.oaec.entity.User");
// 获得User类申明的setPassword方法
Method method = cls.getMethod("setPassword", String.class);
// 输出方法
System.out.println(method);
```

执行结果

```
public void com.oaec.entity.User.setPassword(java.lang.String)
```

此时是获取当前类以及父类声明的指定公共方法，如果尝试获取私有的show方法，则会抛出以下异常

```
java.lang.NoSuchMethodException: com.oaec.entity.User.show(java.lang.String)
        at java.lang.Class.getMethod(Unknown Source)
        at com.oaec.classtype.ClassType.main(ClassType.java:11)
```

获得当前类声明的指定方法

方法声明

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

                                SecurityException
```

示例代码

```
// 获取User类的Class对象
Class<?> cls = Class.forName("com.oaec.entity.User");
// 获得User类申明的私有show方法
Method method = cls.getDeclaredMethod("show", String.class);
// 输出方法
System.out.println(method);
```

执行结果

```
private void com.oaec.entity.User.show(java.lang.String)
```

此时成功获取到私有的show方法

通过反射动态运行指定

方法声明

```
public Object invoke(Object obj,
                     Object... args)
              throws IllegalAccessException,
                     IllegalArgumentException,
                     InvocationTargetException
```

示例代码

1. 通过反射调用无参构造方法创建对象
2. 通过反射为对象的userName属性赋值
3. 通过反射获取userName的值

```
// 获取User类的Class对象
Class<?> cls = Class.forName("com.oaec.entity.User");
// 调用无参构造方法创建User对象
Object instance = cls.newInstance();
// 获取setUserName方法
Method setMethod = cls.getMethod("setUserName", String.class);
// 执行方法，为userName赋值为张三
setMethod.invoke(instance, "张三");
// 获取getUserName方法
Method getMethod = cls.getMethod("getUserName");
// 执行方法，获取userName的值
Object value = getMethod.invoke(instance);
// 输出获取到的userName
System.out.println(value);
```

执行结果

```
张三
```

此时在执行方法的时候，需要注意方法参数

获取属性

常用方法

1. 获得当前类以及父类的公共属性
    - `Field[] arrFields = classType. getFields();`
2. 获得当前类申明的所有Field
    - `Field[] arrFields = classType. getDeclaredFields();`
3. 获得当前类以及超类指定的public Field
    - `Field field = classType. getField(String name);`
4. 获得当前类申明的指定的Field
    - `Field field = classType. getDeclaredField(String name);`
5. 通过反射动态设定和获取Field的值
    - `field.set(Object obj, Object value);`
    - `Object obj = field. get(Object obj);`

示例代码

```
// 获取User类的Class对象
Class<?> cls = Class.forName("com.oaec.entity.User");
System.out.println("***获取当前类和父类公共的属性***");
Field[] fields = cls.getFields();
for (Field field : fields) {
    System.out.println(field);
}
System.out.println("***获取当前类声明的所有属性***");
Field[] declaredFields = cls.getDeclaredFields();
for (Field field : declaredFields) {
    System.out.println(field);
}
System.out.println("***获取当前类以及父类的指定公共属性***");
Field field = cls.getField("data");
//Field publicField = cls.getField("id");//id是私有属性，获取不到
System.out.println(field);
System.out.println("***获取当前类声明的指定属性***");
Field declaredField = cls.getDeclaredField("id");//此时id可以获取
System.out.println(declaredField);
System.out.println("***属性赋值***");
Object instance = cls.newInstance();
field.set(instance, "字符串");
System.out.println("***属性取值***");
Object value = field.get(instance);
System.out.println(value);
```

执行结果

```
***获取当前类和父类公共的属性***
public java.lang.String com.oaec.entity.User.data
***获取当前类声明的所有属性***
public java.lang.String com.oaec.entity.User.data
private int com.oaec.entity.User.id
private java.lang.String com.oaec.entity.User.userName
private java.lang.String com.oaec.entity.User.password
***获取当前类以及父类的指定公共属性***
public java.lang.String com.oaec.entity.User.data
***获取当前类声明的指定属性***
private int com.oaec.entity.User.id
***属性赋值***
无参构造方法
***属性取值***
字符串
```

暴力反射（不推荐使用）

> - 通常情况即使是当前类，private属性或方法也是不能访问的，你需要设置压制权限`setAccessible(true)`来取得private的访问权。
> - 但这已经破坏了面向对象的规则，所以除非万不得已，请尽量少用。

示例代码

```java
// 获取User类的Class对象
Class<?> cls = Class.forName("com.oaec.entity.User");
// 实例化对象
Object instance = cls.newInstance();
// 获取私有属性id
Field declaredField = cls.getDeclaredField("id");
// 压制权限（私有属性不可有直接赋值/取值，需要压制权限）
declaredField.setAccessible(true);
// 为对象的私有属性id直接赋值
declaredField.set(instance, 2);
// 直接获取私有属性id的值
Object value = declaredField.get(instance);
System.out.println(value);
```

执行结果

```
2
```

如果不使用压制权限进行暴力反射，则会抛出以下异常

```
java.lang.IllegalAccessException: Class com.oaec.classtype.ClassType can not access a member of
class com.oaec.entity.User with modifiers "private"
        at sun.reflect.Reflection.ensureMemberAccess(Unknown Source)
        at java.lang.reflect.AccessibleObject.slowCheckMemberAccess(Unknown Source)
        at java.lang.reflect.AccessibleObject.checkAccess(Unknown Source)
        at java.lang.reflect.Field.set(Unknown Source)
        at com.oaec.classtype.ClassType.main(ClassType.java:17)
```

例子：复制对象

方法

```
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class ObjectUtil {

    public static Object copy(Object obj) {
        try {
            //1.获取Class对象
            Class<? extends Object> cls = obj.getClass();
            //2.实例化一个对象
            Object instance = cls.newInstance();
            //3.获取属性(私有)
            Field[] declaredFields = cls.getDeclaredFields();
            //4.遍历属性数组
            for (Field field : declaredFields) {
                //5.根据属性名拼接为get方法名
                String getMethodName = "get"+field.getName().substring(0,
1).toUpperCase()+field.getName().substring(1);
//                System.out.println(getMethodName);
                //6.通过get方法名获取指定get方法
                Method getMethod = cls.getMethod(getMethodName);
                //7.执行get方法，获取属性值
                Object value = getMethod.invoke(obj);
//                System.out.println(value);
                //8.根据属性名拼接为set方法名
                String setMethodName = "set"+field.getName().substring(0,
1).toUpperCase()+field.getName().substring(1);
                //9.通过set方法名获取指定set方法
                Method setMethod = cls.getMethod(setMethodName, field.getType());
                //10.执行set方法，为属性赋值
                setMethod.invoke(instance, value);
            }
            return instance;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

调用方法

```
Object copy = ObjectUtil.copy(new User(1, "张三", "12345"));
System.out.println("复制成功");
System.out.println(copy);
```

执行结果

```
复制成功
User [id=1, userName=张三, password=12345]
```