关于反射机制（摘录）：

笔记本：	java基础		
创建时间：	2018/10/2/周二 13:56	**更新时间：**	2018/10/2/周二 13:56
作者：	1634896520@qq.com		
URL：	file:///F:/GIT_resposity/Learning-Notes/java%20EE/%E5%85%B3%E4%BA%8E%E5%8F%8D%E5%B0%84%E...		

关于反射机制（摘录）：

反射机制是什么

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。

反射机制能做什么

反射机制主要提供了以下功能：

- 在运行时判断任意一个对象所属的类；
- 在运行时构造任意一个类的对象；
- 在运行时判断任意一个类所具有的成员变量和方法；
- 在运行时调用任意一个对象的方法；
- 生成动态代理。

反射机制的相关API

InterFace 接口

```
package com.app;

public interface InterFace {

void read() ;
}
```

Person 类

```
package com.app;

public class Person implements InterFace {

private String id ;

private String name ;

public String age ;
```

```
//构造函数1
public Person( ){

}

//构造函数2
public Person( String id ){
this.id = id ;
}

//构造函数3
public Person( String id , String name ){
this.id = id ;
this.name = name ;
}


public String getId() {
return id;
}

public void setId(String id) {
this.id = id;
}

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}


public String getAge() {
return age;
}

public void setAge(String age) {
this.age = age;
}

/**
* 静态方法
*/
public static void update(){

}


@Override
```

```
public void read() {

}

}
```

- 获取类：3种方法

```
package com.app;

public class T1 {

public static void main(String[] args) {

//第一种方法：forName
try {
Class<?> class1 = Class.forName("com.app.Person");

System.out.println( class1 );
} catch (ClassNotFoundException e) {
e.printStackTrace();
}


//第二张方法：class
Class<?> class2 = Person.class;

//第三种方法：getClass
Person person = new Person();
Class<?> class3 = person.getClass();

System.out.println( class2 );
System.out.println( class3 );
}

}
```

运行结果：

```
class com.app.Person class com.app.Person class com.app.Person
```

- 获取所有的方法：getMethods( )

```
package com.app;

import java.lang.reflect.Method;

public class T1 {

public static void main(String[] args) {

try {
```

```
//创建类
Class<?> class1 = Class.forName("com.app.Person");

//获取所有的公共的方法
Method[] methods = class1.getMethods() ;
for (Method method : methods) {
System.out.println( method );
}

} catch (ClassNotFoundException e) {
e.printStackTrace();
}
}
}
```

运行结果：

```
//自定义方法public static void com.app.Person.update()
public java.lang.String com.app.Person.getName()
public void com.app.Person.read()
public java.lang.String com.app.Person.getId()
public void com.app.Person.setName(java.lang.String)
public void com.app.Person.setId(java.lang.String)

//父类Object类方法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 java.lang.String java.lang.Object.toString()
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()
```

- 获取所有实现的接口：getInterfaces()

```
package com.app;

public class T1 {

public static void main(String[] args) {

try {
//创建类
Class<?> class1 = Class.forName("com.app.Person");

//获取所有的接口
Class<?>[] interS = class1.getInterfaces() ;

for (Class<?> class2 : interS ) {
```

```
System.out.println( class2 );
}

} catch (ClassNotFoundException e) {
e.printStackTrace();
}
}
}
```

运行结果：

```
interface com.app.InterFace
```

- 获取父类：getSuperclass()

```
package com.app;

public class T1 {

public static void main(String[] args) {

try {
//创建类
Class<?> class1 = Class.forName("com.app.Person");

//获取父类
Class<?> superclass = class1.getSuperclass() ;

System.out.println( superclass );

} catch (ClassNotFoundException e) {
e.printStackTrace();
}
}
}
```

运行结果：

```
//父类是Object类class java.lang.Object
```

- 获取所有的构造函数：getConstructors()

```
package com.app;

import java.lang.reflect.Constructor;

public class T1 {

public static void main(String[] args) {

try {
//创建类
```

```
Class<?> class1 = Class.forName("com.app.Person");

//获取所有的构造函数
Constructor<?>[] constructors = class1.getConstructors() ;

for (Constructor<?> constructor : constructors) {
System.out.println( constructor );
}


} catch (ClassNotFoundException e) {
e.printStackTrace();
}
}
}
```

运行结果：

```
public com.app.Person(java.lang.String,java.lang.String)
public com.app.Person(java.lang.String)
public com.app.Person()
```

- 获取所有的属性：getDeclaredFields();

```
package com.app;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;

public class T1 {

public static void main(String[] args) {

try {
//创建类
Class<?> class1 = Class.forName("com.app.Person");

//取得本类的全部属性
Field[] field = class1.getDeclaredFields();

for (Field field2 : field) {
System.out.println( field2 );
}

} catch (ClassNotFoundException e) {
e.printStackTrace();
}
}
}
```

运行结果：

```
private java.lang.String com.app.Person.idprivate java.lang.String com.app.Person.name
```

可以看出属性的修饰符是：private，数据类型：String ，名字：id/name

- 创建实例：newInstance()

```
package com.app;


public class T1 {

public static void main(String[] args) {

try {
//创建类
Class<?> class1 = Class.forName("com.app.Person");;

//创建实例化：相当于 new 了一个对象
Object object = class1.newInstance() ;

//向下转型
Person person = (Person) object ;

} catch (ClassNotFoundException e) {
e.printStackTrace();
} catch (InstantiationException e) {
e.printStackTrace();
} catch (IllegalAccessException e) {
e.printStackTrace();
}

}
}
```

getDeclaredFields 和 getFields 的区别

getDeclaredFields()获得某个类的所有申明的字段，即包括public、private和proteced，但是不包括父类的申明字段。

getFields()获得某个类的所有的公共（public）的字段，包括父类。

小例子

```
package com.app;

import java.lang.reflect.Field;

public class T1 {

public static void main(String[] args) {

try {
```

```
//创建类
Class<?> class1 =  Class.forName("com.app.Person");;

//获得所有的字段属性：包括
Field[] declaredFields = class1.getDeclaredFields() ;

Field[] fields = class1.getFields() ;

for(  Field  field : declaredFields ){
System.out.println(  "de-- "  + field );
}

for(  Field  field : fields ){
System.out.println(  "fields-- "  + field );
}

}  catch  (ClassNotFoundException  e) {
e.printStackTrace();
}

}
}
```

运行结果：

```
de--  private  java.lang.String  com.app.Person.id de--  private  java.lang.String
 com.app.Person.name de--  public  java.lang.String  com.app.Person.age fields--  public
 java.lang.String  com.app.Person.age
```

实战1：通过反射，获取对象实例，并且操作对象的方法

```
package com.app;


public  class  T1  {

public  static  void  main(String[] args)  {

try  {
//创建类
Class<?> class1 = Class.forName("com.app.Person");;

//创建实例化：相当于 new 了一个对象
Object  object  = class1.newInstance() ;

//向下转型
Person person = (Person)  object  ;
person.setId(  "100");
person.setName(  "jack") ;
System.out.println(  "id: "  + person.getId() +  " name: "  + person.getName() );
}  catch  (ClassNotFoundException e) {
```

```
e.printStackTrace();
} catch (InstantiationException e) {
e.printStackTrace();
} catch (IllegalAccessException e) {
e.printStackTrace();
}

}
}
```

运行结果：

```
id: 100 name: jack
```

实战2：通过反射获取对象字段属性，并且赋值

```
package com.app;

import java.lang.reflect.Field;

public class T1 {

public static void main(String[] args) {

try {
//创建类
Class<?> class1 = Class.forName("com.app.Person");

//创建实例
Object person = class1.newInstance();

//获得id 属性
Field idField = class1.getDeclaredField( "id" ) ;
//给id 属性赋值
idField.set( person , "100") ;

//打印 person 的属性值
System.out.println( idField.get( person ));

} catch (InstantiationException e) {
e.printStackTrace();
} catch (IllegalAccessException e) {
e.printStackTrace();
} catch (NoSuchFieldException e) {
e.printStackTrace();
} catch (SecurityException e) {
e.printStackTrace() ;
} catch (ClassNotFoundException e) {
e.printStackTrace();
}
```

```
}
}
```

运行结果：

```
java.lang.IllegalAccessException: Class com.app.T1 can not access a member of class com.app.Person 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.app.T1.main(T1.java:20)
```

程序崩溃，原因是：id 这个属性的是 private 私有的，不能修改它的值。

改进：

添加 idField.setAccessible(true);

完整的代码为：

```
package com.app;

import java.lang.reflect.Field;

public class T1 {

public static void main(String[] args) {

try {
//创建类
Class<?> class1 = Class.forName("com.app.Person");

//创建实例
Object person = class1.newInstance();

//获得id 属性
Field idField = class1.getDeclaredField( "id" ) ;

//打破封装 实际上setAccessible是启用和禁用访问安全检查的开关,并不是为true就能访问为false就不能访问
//由于JDK的安全检查耗时较多.所以通过setAccessible(true)的方式关闭安全检查就可以达到提升反射速度的目的
idField.setAccessible( true );
//给id 属性赋值
idField.set( person , "100") ;

//打印 person 的属性值
System.out.println( idField.get( person ));

} catch (InstantiationException e) {
e.printStackTrace();
```

```
} catch (IllegalAccessException e) {
e.printStackTrace();
} catch (NoSuchFieldException e) {
e.printStackTrace();
} catch (SecurityException e) {
e.printStackTrace() ;
} catch (ClassNotFoundException e) {
e.printStackTrace();
}

}
}
```

运行结果：

```
100
```

实战3：综合训练，反射操作属性和方法

```
package com.app;

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

public class T1 {

public static void main(String[] args) {

try {
//创建类
Class<?> class1 = Class.forName("com.app.Person");

//创建实例
Object person = class1.newInstance();

//获得id 属性
Field idField = class1.getDeclaredField( "id" ) ;

//打破封装 实际上setAccessible是启用和禁用访问安全检查的开关,并不是为true就能访问为false就不能访问
//由于JDK的安全检查耗时较多.所以通过setAccessible(true)的方式关闭安全检查就可以达到提升反射速度的目的
idField.setAccessible( true );

//给id 属性赋值
idField.set( person , "100") ;

//获取 setName() 方法
Method setName = class1.getDeclaredMethod( "setName", String.class ) ;
//打破封装
setName.setAccessible( true );
```

```
//调用setName 方法。
setName.invoke( person , "jack" ) ;

//获取name 字段
Field nameField = class1.getDeclaredField( "name" ) ;
//打破封装
nameField.setAccessible( true );

//打印 person 的 id 属性值
String id_ = (String) idField.get( person ) ;
System.out.println( "id: " + id_ );

//打印 person 的 name 属性值
String name_ = ( String)nameField.get( person ) ;
System.out.println( "name: " + name_ );
//获取 getName 方法
Method getName = class1.getDeclaredMethod( "getName" ) ;
//打破封装
getName.setAccessible( true );
//执行getName方法，并且接收返回值
String name_2 = (String) getName.invoke( person ) ;
System.out.println( "name2: " + name_2 );

} catch (IllegalArgumentException e) {
e.printStackTrace();
} catch (InvocationTargetException e) {
e.printStackTrace();
} catch (NoSuchMethodException e) {
e.printStackTrace();
} catch (InstantiationException e) {
e.printStackTrace();
} catch (IllegalAccessException e) {
e.printStackTrace();
} catch (NoSuchFieldException e) {
e.printStackTrace();
} catch (SecurityException e) {
e.printStackTrace() ;
} catch (ClassNotFoundException e) {
e.printStackTrace();
}

}
}
```

运行结果：

```
id: 100name: jack
name2: jack
```

实战4：静态属性、静态方法调用

定义 Util 类

```
package com.app;

public class Util {

public static String name = "json" ;

/**
* 没有返回值，没有参数
*/
public static void getTips(){
System.out.println( "执行了---------1111");
}

/**
* 有返回值，没有参数
*/
public static String getTip(){
System.out.println( "执行了---------2222");
return "tip2" ;
}

/**
* 没有返回值，有参数
* @param name
*/
public static void getTip( String name ){
System.out.println( "执行了---------3333 参数： " + name );
}

/**
* 有返回值，有参数
* @param id
* @return
*/
public static String getTip( int id ){
System.out.println( "执行了---------4444 参数： " + id );
if ( id == 0 ){
return "tip1 444 --1 " ;
}else{
return "tip1 444 --2" ;
}
}

}
```

完整小例子：

```
package com.app;

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

public class T1 {

public static void main(String[] args) {

try {
//创建类
Class<?> class1 = Class.forName("com.app.Util");

//获取 nameField 属性
Field nameField = class1.getDeclaredField( "name" ) ;
//获取 nameField 的值
String name_ = (String) nameField.get( nameField ) ;
//输出值
System.out.println( name_ );


//没有返回值，没有参数
Method getTipMethod1 = class1.getDeclaredMethod( "getTips" ) ;
getTipMethod1.invoke( null ) ;
//有返回值，没有参数
Method getTipMethod2 = class1.getDeclaredMethod( "getTip" ) ;
String result_2 = (String) getTipMethod2.invoke( null ) ;
System.out.println( "返回值："+ result_2 );
//没有返回值，有参数
Method getTipMethod3 = class1.getDeclaredMethod( "getTip" , String.class ) ;
String result_3 = (String) getTipMethod3.invoke( null , "第三个方法" ) ;
System.out.println( "返回值："+ result_3 );
//有返回值，有参数
Method getTipMethod4 = class1.getDeclaredMethod( "getTip" , int.class ) ;
String result_4 = (String) getTipMethod4.invoke( null , 1 ) ;
System.out.println( "返回值："+ result_4 );
} catch (InvocationTargetException e) {
e.printStackTrace();
} catch (IllegalArgumentException e) {
e.printStackTrace();
} catch (NoSuchMethodException e) {
e.printStackTrace();
} catch (IllegalAccessException e) {
e.printStackTrace();
} catch (NoSuchFieldException e) {
e.printStackTrace();
} catch (SecurityException e) {
e.printStackTrace() ;
```

```
} catch (ClassNotFoundException e) {
e.printStackTrace();
}

}
}
```

运行结果：

```
json
执行了---------1111
执行了---------2222
返回值： tip2
执行了---------3333 参数： 第三个方法
返回值： null
执行了---------4444 参数： 1
返回值： tip1 444 --2
```

当参数是 int 类型 和 Integer 类型，反射获取方法不一样

- 当参数是 int 类型时

```
/**
* 没有返回值，有参数
* @param id
*/
public static void getTip( int id ){
}
```

获取方法的时候需要用：`int.class`。不能使用 `Integer.class`. 会报错。

```
Method getTipMethod4 = class.getDeclaredMethod( "getTip" , int.class ) ;
String result_4 = (String) getTipMethod4.invoke( null , 1 ) ;
System.out.println( "返回值： "+ result_4 );
```

- 当参数是 Integer类型时

```
/**
* 没有返回值，有参数
* @param id
*/
public static void getTip( Integer id ){
}
```

获取方法的时候需要用：`Integer .class`。不能使用 `int.class`. 会报错。

```
Method getTipMethod4 = class.getDeclaredMethod( "getTip" , Integer .class ) ;
String result_4 = (String) getTipMethod4.invoke( null , 1 ) ;
System.out.println( "返回值： "+ result_4 );
```

创建对象实例

Person 类

```
package com.app;

public  class  Person{

private  String id ;

private  String name ;


//构造函数1
public  Person(  ){
System.out.println(  "构造函数 无参"  );
}

//构造函数2
public  Person(  String id  ){
this.id = id ;
System.out.println(  "构造函数 id : "  + id );
}

//构造函数3
public  Person(  String id , String name  ){
this.id = id ;
this.name = name ;
System.out.println(  "构造函数 id : "  + id +  " name: "  + name );
}


public  String  getId()  {
return  id;
}

public  void  setId(String id)  {
this.id = id;
}

public  String  getName()  {
return  name;
}

public  void  setName(String name)  {
this.name = name;
}

}
```

创建实例实战

```
package com.app;
```

```
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;

public class T1 {

public static void main(String[] args) {

try {
//创建类
Class<?> class1 = Class.forName("com.app.Person");

//无参构造函数
Object object = class1.newInstance() ;
//有参构造函数：一个参数
Constructor<?> constructor = class1.getDeclaredConstructor( String.class ) ;
constructor.newInstance( "1000" ) ;
//有参构造函数：二个参数
Constructor<?> constructor2 = class1.getDeclaredConstructor( String.class , String.class ) ;
constructor2.newInstance( "1001" , "jack" ) ;

} catch (InstantiationException e) {
e.printStackTrace();
} catch (IllegalAccessException e) {
e.printStackTrace();
} catch (InvocationTargetException e) {
e.printStackTrace();
} catch (IllegalArgumentException e) {
e.printStackTrace();
} catch (NoSuchMethodException e) {
e.printStackTrace();
} catch (SecurityException e) {
e.printStackTrace() ;
} catch (ClassNotFoundException e) {
e.printStackTrace();
}

}
}
```

运行结果

```
构造函数 无参
构造函数 id : 1000
构造函数 id : 1001 name: jack
```

总结

- Class类提供了四个public方法，用于获取某个类的构造方法。

Constructor getConstructor(Class[] params) 根据构造函数的参数，返回一个具体的具有public属性的构造函数
Constructor getConstructors() 返回所有具有public属性的构造函数数组
Constructor getDeclaredConstructor(Class[] params) 根据构造函数的参数，返回一个具体的构造函数（不分public和非public属性）
Constructor getDeclaredConstructors() 返回该类中所有的构造函数数组（不分public和非public属性）

- 四种获取成员方法的方法

Method getMethod(String name, Class[] params) 根据方法名和参数，返回一个具体的具有public属性的方法
Method[] getMethods() 返回所有具有public属性的方法数组
Method getDeclaredMethod(String name, Class[] params) 根据方法名和参数，返回一个具体的方法（不分public和非public属性）
Method[] getDeclaredMethods() 返回该类中的所有的方法数组（不分public和非public属性）

- 四种获取成员属性的方法

Field getField(String name) 根据变量名，返回一个具体的具有public属性的成员变量
Field[] getFields() 返回具有public属性的成员变量的数组
Field getDeclaredField(String name) 根据变量名，返回一个成员变量（不分public和非public属性）
Field[] getDelcaredField() 返回所有成员变量组成的数组（不分public和非public属性）