反射机制介绍

什么是反射

Java 反射机制是Java语言一个很重要的特性，它使得Java具有了“动态性”。在Java程序运行时，对于任意的一个类，我们能不能知道这个类有哪些属性和方法呢？对于任意的一个对象，我们又能不能调用它任意的方法？答案是肯定的！这种动态获取类的信息以及动态调用对象方法的功能就来自于Java 语言的反射（Reflection）机制。

反射的作用

简单来说两个作用，RTTI（运行时类型识别）和DC（动态创建）。

我们知道反射机制允许程序在运行时取得任何一个已知名称的class的内部信息，包括其modifiers(修饰符)，fields(属性)，methods(方法)等，并可于运行时改变fields内容或调用methods。那么我们便可以更灵活的编写代码，代码可以在运行时装配，无需在组件之间进行源代码链接，降低代码的耦合度；还有动态代理的实现等等；但是需要注意的是反射使用不当会造成很高的资源消耗！

实时效果反馈

1.如下对Java反射描述错误的是?

A 反射可以使代码在运行时装配;

B 反射可以降低代码的耦合度;

C 通过反射可以实现动态代理;

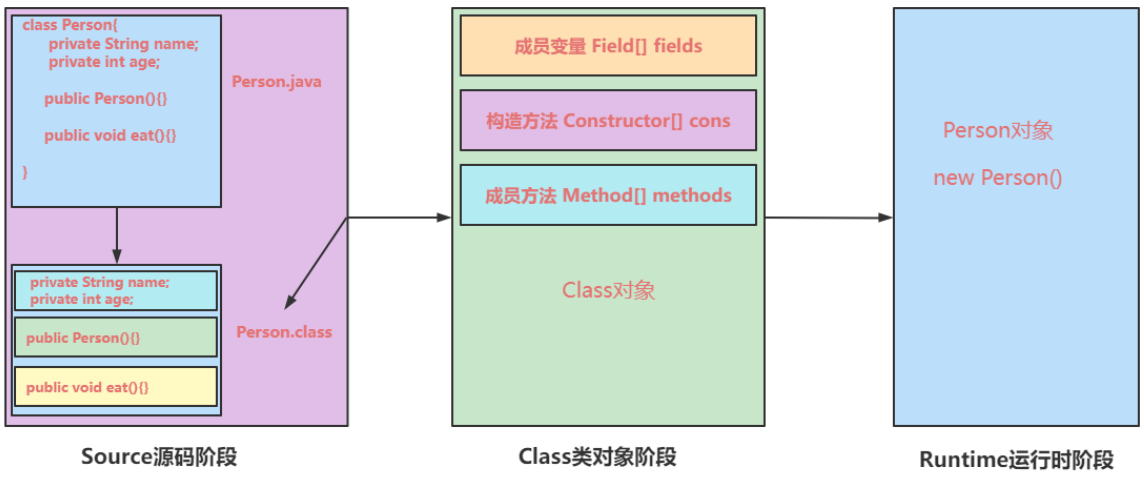
D 反射不会造成很高的资源消耗;

答案

1=>D

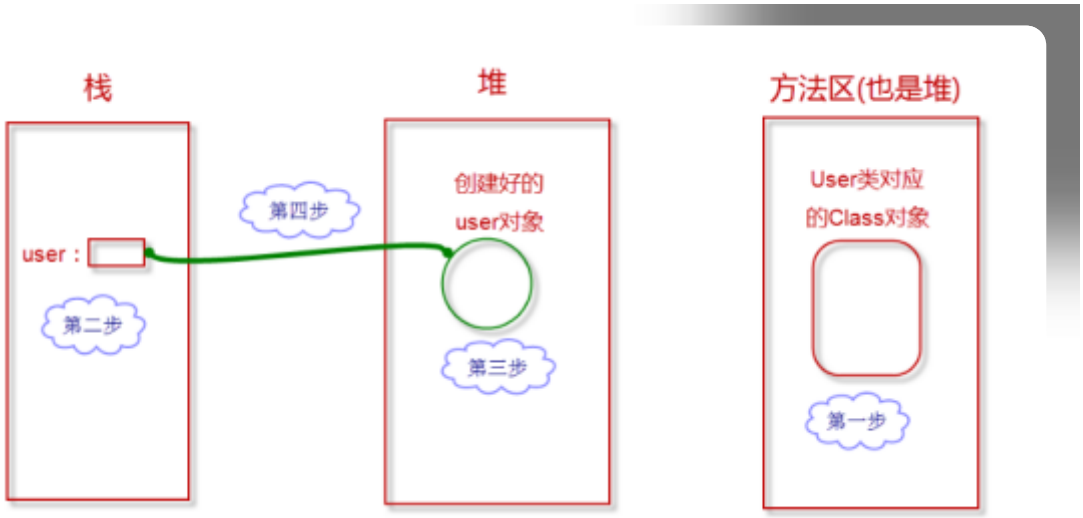
创建对象过程

Java创建对象的三个阶段



创建对象时内存结构

```
Users user = new Users();
```



实际上，我们在加载任何一个类时都会在方法区中建立“这个类对应的Class对象”，由于“Class对象”包含了这个类的整个结构信息，所以我们可以通过这个“Class对象”来操作这个类。

我们要使用一个类，首先要加载类；加载完类之后，在堆内存中，就产生了一个 Class 类型的对象（一个类只有一个 Class 对象），这个对象就包含了完整的类的结构信息。我们可以通过这个对象知道类的结构。这个对象就像一面镜子，透过这个镜子可以看到类的结构，所以，我们形象的称之为：反射。 因此，“Class对象”是反射机制的核心。

实时效果反馈

1.如下对Class对象描述错误的是?

A Class对象包含了这个类的整个结构信息；

B 通过Class对象可以获取类的相关信息；

C 一个类可以有多个 Class 对象；

D Class对象是反射机制的核心；

答案

1=>C

反射的具体实现

获取Class对象的三种方式

- 通过getClass()方法;
- 通过.class 静态属性;
- 通过Class类中的静态方法forName();

创建Users类

```
1  public class Users {
2      private String username;
3      private int userage;
4
5      public String getUsername() {
6          return username;
7      }
8
```

```
    public void setUsername(String username) {
        this.username = username;
    }

    public int getUserage() {
        return userage;
    }

    public void setUserage(int userage) {
        this.userage = userage;
    }
}

```

通过getClass()方法获取Class对象

```
 * 通过getClass()方法获取该类的Class对象
 */
public class GetClass1 {
    public static void main(String[] args) {
        Users users = new Users();
        Users users1 = new Users();
        Class clazz = users.getClass();
        System.out.println(clazz);
        System.out.println(clazz.getName());
        System.out.println(users.getClass() == users1.getClass());
    }
}
```

通过.class 静态属性获取Class对象

```
/**
 * .class静态属性获取Class对象
 */
public class GetClass2 {
    public static void main(String[] args) {
        Class clazz = Users.class;
        Class clazz2 = Users.class;
        System.out.println(clazz);
        System.out.println(clazz.getName());
        System.out.println(clazz == clazz2);
    }
}
```

通过forName()获取Class对象

```java
/**
 * 通过Class.forName("class Name")获取Class对象
 */
public class GetClass3 {
    public static void main(String[] args)throws Exception {
        Class clazz = Class.forName("com.bjsxt.Users");
        Class clazz2 = Class.forName("com.bjsxt.Users");
        System.out.println(clazz);
        System.out.println(clazz.getName());
        System.out.println(clazz == clazz2);
    }
}
```

获取类的构造方法

方法介绍

方法名	描述
getDeclaredConstructors()	返回 Constructor 对象的一个数组，这些对象反映此 Class 对象表示的类声明的所有构造方法。
getConstructors()	返回一个包含某些 Constructor 对象的数组，这些对象反映此 Class 对象所表示的类的所有公共（public）构造方法。
getConstructor(Class<?>... parameterTypes)	返回一个 Constructor 对象，它反映此 Class 对象所表示的类的指定公共（public）构造方法。
getDeclaredConstructor(Class<?>... parameterTypes)	返回一个 Constructor 对象，该对象反映此 Class 对象所表示的类或接口的指定构造方法。

方法使用

修改Users类

```
public class Users {
    private String username;
    private int userage;
    public Users(){
    }
    public Users(String username,int userage){
        this.username= username;
        this.userage=userage;
    }
    public  Users(String username){
        this.username= username;
    }
    private Users(int userage){
        this.userage = userage;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public int getUserage() {
        return userage;
    }
```

```
28
29      public void setUserage(int userage) {
30          this.userage = userage;
31      }
32  }
```

获取构造方法

```
1  public class GetConstructor {
2      public static void main(String[]
   args)throws Exception {
3          Class clazz = Users.class;
4          Constructor[]  arr =
   clazz.getDeclaredConstructors();
5          for(Constructor c:arr){
6              System.out.println(c);
7          }
8          System.out.println("----------------
   -----");
9          Constructor[]  arr1 =
   clazz.getConstructors();
10         for(Constructor c:arr1){
11             System.out.println(c);
12         }
13         System.out.println("----------------
   --------");
14         Constructor c =
   clazz.getDeclaredConstructor(int.class);
15         System.out.println(c);
16
```

```
17          System.out.println("----------------
    --------");
18          Constructor c1 =
    clazz.getConstructor(null);
19          System.out.println(c1);
20      }
21  }
```

通过构造方法创建对象

```
1  public class GetConstructor2 {
2      public static void main(String[]
    args)throws Exception {
3          Class clazz = Users.class;
4          Constructor constructor =
    clazz.getConstructor(String.class,int.class);
5          Object o =
    constructor.newInstance("OldLu",18);
6          Users users = (Users)o;
7
     System.out.println(users.getUsername()+"\t"+
    users.getUserage());
8      }
9  }
```

获取类的成员变量

方法介绍

方法名	描述
getFields()	返回Field类型的一个数组,其中包含 Field对象的所有公共(public)字段。
getDeclaredFields()	返回Field类型的一个数组,其中包含 Field对象的所有字段。
getField(String fieldName)	返回一个公共成员的Field指定对象。
getDeclaredField(String fieldName)	返回一个 Field指定对象。

方法使用

修改Users类

```
public class Users {
    private String username;
    public int userage;
    public Users(){
    }
    public Users(String username,int userage){
        this.username= username;
        this.userage=userage;
    }
    public  Users(String username){
        this.username= username;
    }
    private Users(int userage){
        this.userage = userage;
    }
```

```
16
17      public String getUsername() {
18          return username;
19      }
20
21      public void setUsername(String username)
{
22          this.username = username;
23      }
24
25      public int getUserage() {
26          return userage;
27      }
28
29      public void setUserage(int userage) {
30          this.userage = userage;
31      }
32  }
33
```

获取成员变量

```
1   public class GetField {
2       public static void main(String[]
args)throws Exception {
3           Class clazz = Users.class;
4           Field[] fields = clazz.getFields();
5           for(Field f:fields){
6               System.out.println(f);
7               System.out.println(f.getName());
```

```
        }
        System.out.println("----------------
--------");
        Field[] fields2 =
clazz.getDeclaredFields();
        for(Field f:fields2){
            System.out.println(f);
            System.out.println(f.getName());
        }
        System.out.println("----------------
--------");
        Field field =
clazz.getField("userage");
        System.out.println(field);
        System.out.println("----------------
-----");
        Field field1 =
clazz.getDeclaredField("username");
        System.out.println(field1);
    }
}
```

操作成员变量

```
public class GetField2 {
    public static void main(String[]
args)throws Exception  {
        Class clazz = Users.class;
        Field field =
clazz.getField("userage");
```

```
            //对象实例化
            Object obj = clazz.newInstance();
            //为成员变量赋予新的值
            field.set(obj,18);
            //获取成员变量的值
            Object o = field.get(obj);
            System.out.println(o);

        }
}
```

获取类的方法

方法介绍

方法名	描述
getMethods()	返回一个Method类型的数组,其中包含 所有公共(public)方法。
getDeclaredMethods()	返回一个Method类型的数组,其中包含 所有方法。
getMethod(String name, Class<?>... parameterTypes)	返回一个公共的Method方法对象。
getDeclaredMethod(String name, Class<?>... parameterTypes)	返回一个方法Method对象

方法使用

修改Users类

```
public class Users {
    private String username;
```

```
3       public int userage;
4       public Users(){
5       }
6       public Users(String username,int
    userage){
7           this.username= username;
8           this.userage=userage;
9       }
10      public  Users(String username){
11          this.username= username;
12      }
13      private Users(int userage){
14          this.userage = userage;
15      }
16
17      public String getUsername() {
18          return username;
19      }
20
21      public void setUsername(String username)
    {
22          this.username = username;
23      }
24
25      public int getUserage() {
26          return userage;
27      }
28
29      public void setUserage(int userage) {
30          this.userage = userage;
31      }
32      private void suibian(){
```

```
33          System.out.println("Hello Oldlu");
34      }
35  }
36
```

获取方法

```
1  public class GetMethod {
2      public static void main(String[]
   args)throws Exception{
3          Class clazz = Users.class;
4          Method[] methods =
   clazz.getMethods();
5          for(Method m: methods){
6              System.out.println(m);
7              System.out.println(m.getName());
8          }
9          System.out.println("----------------
   -----------");
10         Method[] methods2 =
   clazz.getDeclaredMethods();
11         for(Method m: methods2){
12             System.out.println(m);
13             System.out.println(m.getName());
14         }
15         System.out.println("----------------
   ----------");
16         Method method =
   clazz.getMethod("setUserage", int.class);
17
    System.out.println(method.getName());
```

```
        System.out.println("----------------------");
        Method method1 = clazz.getDeclaredMethod("suibian");

         System.out.println(method1.getName());
    }
}
```

调用方法

```
public class GetMethod2 {
    public static void main(String[] args)throws  Exception {
        Class clazz = Users.class;
        Method method = clazz.getMethod("setUsername",String.class);
         //实例化对象
        Object obj = clazz.getConstructor(null).newInstance();
        //通过setUserName赋值
        method.invoke(obj,"oldlu");

        //通过getUserName获取值
        Method method1 = clazz.getMethod("getUsername");
        Object value = method1.invoke(obj);
        System.out.println(value);
    }
}
```

获取类的其他信息

```
public class GetClassInfo {
    public static void main(String[] args) {
        Class clazz = Users.class;
        //获取类名
        String className = clazz.getName();
        System.out.println(className);
        //获取包名
        Package p = clazz.getPackage();
        System.out.println(p.getName());
        //获取超类
        Class superClass =
clazz.getSuperclass();

 System.out.println(superClass.getName());
        //获取该类实现的所有接口
        Class[] interfaces =
clazz.getInterfaces();
        for(Class inter:interfaces){

 System.out.println(inter.getName());
        }
    }
}
```

反射应用案例

需求：根据给定的方法名顺序来决定方法的执行顺序。

```
class Reflect {
    public void method1(){

 System.out.println("Method1.......");
    }
    public void method2(){

 System.out.println("Method2.......");
    }
    public void method3(){

 System.out.println("Method3.......");
    }
}
public class ReflectDemo {
    public static void main(String[]
args)throws Exception {
        Reflect rd = new Reflect();
        if(args != null && args.length > 0){
            //获取ReflectDemo的Class对象
            Class clazz = rd.getClass();
            //通过反射获取ReflectDemo下的所有方
法
            Method[] methods =
clazz.getMethods();
            for(String str :args){
                for(int
i=0;i<methods.length;i++){
```

```
22
 if(str.equalsIgnoreCase(methods[i].getName(
))){
23
 methods[i].invoke(rd);
24                          break;
25                      }
26                  }
27              }
28          }else{
29              rd.method1();
30              rd.method2();
31              rd.method3();
32          }
33      }
34  }
```

反射机制的效率

由于Java反射是要解析字节码，将内存中的对象进行解析，包括了一些动态类型，而JVM无法对这些代码进行优化。因此，反射操作的效率要比那些非反射操作低得多！

接下来我们做个简单的测试来直接感受一下反射的效率。

反射机制的效率测试

```
public class Test{
    public static void main(String[] args) {
        try {
            //反射耗时
            Class clazz = Class.forName("com.bjsxt.Users");
            Users users = (Users) clazz.getConstructor(null).newInstance();
            long reflectStart = System.currentTimeMillis();
            Method method = clazz.getMethod("setUsername", String.class);
            for(int i=0;i<100000000;i++){

 method.invoke(users,"oldlu");
            }
            long reflectEnd = System.currentTimeMillis();
            //非反射方式的耗时
            long start = System.currentTimeMillis();
            Users u = new Users();
            for(int i=0;i<100000000;i++){
                u.setUsername("oldlu");
            }
            long end = System.currentTimeMillis();
            System.out.println("反射执行时间: "+(reflectEnd - reflectStart));
```

```
            System.out.println("普通方式执行时间："+(end - start));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

setAccessible方法

setAccessible()方法：

setAccessible是启用和禁用访问安全检查的开关。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。值为 false 则指示反射的对象应该实施 Java 语言访问检查;默认值为false。

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。

```
public class Test2 {
    public static void main(String[] args)throws Exception {
        Users users = new Users();
        Class clazz = users.getClass();
        Field field = clazz.getDeclaredField("username");
        //忽略安全检查
        field.setAccessible(true);
        field.set(users,"oldlu");
        Object object = field.get(users);
```

```
10          System.out.println(object);
11          System.out.println("----------------
            -------------");
12          Method method =
            clazz.getDeclaredMethod("suibian");
13          method.setAccessible(true);
14          method.invoke(users);
15      }
16  }
```

本章总结

- Java 反射机制是Java语言一个很重要的特性，它使得Java具有了“动态性”。
- 反射机制的优点：
  - 更灵活。
  - 更开放。
- 反射机制的缺点：
  - 降低程序执行的效率。
  - 增加代码维护的困难。
- 获取Class类的对象的三种方式：
  - 运用getClass()。
  - 运用.class 语法。
  - 运用Class.forName()（最常被使用）。
- 反射机制的常见操作
  - 动态加载类、动态获取类的信息（属性、方法、构造器）。
  - 动态构造对象。
  - 动态调用类和对象的任意方法。
  - 动态调用和处理属性。
  - 获取泛型信息。

- 处理注解。