[Spring中应用反射机制浅析](http://blog.csdn.net/woshixuye/article/details/7700455)

标签： [spring](http://www.csdn.net/tag/spring)[getter](http://www.csdn.net/tag/getter)[class](http://www.csdn.net/tag/class)[setter](http://www.csdn.net/tag/setter)[string](http://www.csdn.net/tag/string)[exception](http://www.csdn.net/tag/exception)

2012-06-28 19:30 6547人阅读 [评论](http://blog.csdn.net/woshixuye/article/details/7700455#comments)(4) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/woshixuye/article/details/7700455#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

spring&springMVC（17） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

我们知道,Spring中大量使用了反射机制，那么究竟是什么地方使用了呢？就从源头说起吧。

**一  反射源头Class类**

对类的概念我们已经非常熟悉了。比如可以有Student这个类，Person这个类。但是我们要知道，有一个叫**Class的类**，它是反射的源头。

**正常方式：通过完整的类名—>通过new实例化—>取得实例化对象**

**反射方式：实例化对象—>getClass()方法—>通过完整的类名**

一个简单的例子：

package cn.classes;

public class OneClass {

}

package cn.test;

import cn.classes.OneClass;

public class Test {  
    public static void main(String[] args) {  
        OneClass c = new OneClass();  
        **System.out.println(c.getClass().getName());**   }  
}

输出结果：cn.classes.OneClass

我们需要使用反射，就要获得**Class这个类**，有三种方法：

package cn.classes;

public class OneClass {

}

import cn.classes.OneClass;

public class Test {  
public static void main(String[] args) {  
     Class<?> c1 = null;  
     Class<?> c2 = null;  
     Class<?> c3 = null;

     try   
    {  
         // 方法一：forName(重要)  
         **c1 = Class.forName("cn.classes.OneClass");**    }   
    catch (ClassNotFoundException e)   
    {  
         e.printStackTrace();  
    }  
     // 方法二  
     **c2 = new OneClass().getClass();**    
     // 方法三  
     **c3 = OneClass.class;**

     System.out.println(c1.getName());  
     System.out.println(c2.getName());  
     System.out.println(c3.getName());  
   }  
}

输出结果：cn.classes.OneClass

**二 利用Class这个类实例化类**

**①无参构造**

package cn.classes;

public class Person {  
    private String name;  
    private int age;

     .............省略getter,setter..............

    @Override  
    public String toString()  
    {  
          return "Person [name=" + name + ", age=" + age + "]";  
    }

}

package cn.test;

import cn.classes.Person;

public class Test  
{  
    // 这样做必须在类中有一个空构造方法  
    public static void main(String[] args)  
    {  
              Class<?> c = null;  
               try  
               {  
                      **c = Class.forName("cn.classes.Person");  
                      Person p = (Person)c.newInstance();**                      p.setName("xy");  
                      p.setAge(20);  
                      System.out.println(p);  
               }   
              catch (Exception e)  
              {  
                      e.printStackTrace();  
               }    
     }  
}

**②有参构造**

package cn.classes;

public class Person  
{  
    private String name;  
    private int age;

**public Person(Stringname,int age)**{

                 this.name = name;

                 this.age = age;

    }

    .............省略getter,setter..............

    @Override  
    public String toString()  
    {  
          return "Person [name=" + name + ", age=" + age + "]";  
    }  
}

package cn.test;

import java.lang.reflect.Constructor;

import cn.classes.Person;

public class Test  
{  
    // 如果没有一个空构造方法  
    public static void main(String[] args)  
    {  
          Class<?> c = null;  
          try  
          {  
                  **c = Class.forName("cn.classes.Person");  
                  Constructor<?> con = c.getConstructors(String.class,int.class);**                  **Person p = (Person)con.newInstance("xy",20);**                  System.out.println(p);  
          }   
         catch (Exception e)  
         {  
                e.printStackTrace();  
         }    
    }  
}

**三 Spring中使用Class实例化**

bean.xml  
<bean id="id" class="com.xy.Student" />

Spring将采用的代码创建代码Java实例  
**Class c = Class.forName("com.xy.Student");  
Object bean = c.newInstance();**

**四 Class类调用方法**

package cn.classes;

public class Person  
{  
    public void add()  
    {  
           System.out.println("add");  
    }

    public void addWithParameters(String name, int age)  
    {  
            System.out.println("add带参数方法" + name + age);  
    }  
}

package cn.test;

import java.lang.reflect.Method;

public class Test  
{  
    public static void main(String[] args)  
    {  
             Class<?> c1 = null;  
             try  
             {

                   c1 = Class.forName("cn.classes.Person");

                   // 不带参数的方法调用  
                   **Method m = c1.getMethod("add");**                   **m.invoke(c1.newInstance());**

                   // 带参数方法调用  
                   **Method m1 = c1.getMethod("addWithParameters", String.class, int.class);  
                   m1.invoke(c1.newInstance(), "xy", 22);**            }  
            catch (Exception e)  
            {  
                   e.printStackTrace();  
            }  
    }  
}

**五 Class获得getter，setter方法**

Class这个类可以获得类的很多信息，比如获得该类的接口，构造函数，属性，方法等。我们来看如何获得getter，setter方法。

package cn.classes;

public class Person  
{  
    private String name;  
    private int age;

    省略getter,setter

 }

package cn.test;

import java.lang.reflect.Method;

public class Test  
{  
    public static void main(String[] args)  
    {  
           Class<?> c1 = null;  
           Object obj = null;  
           try  
           {  
                     c1 = Class.forName("cn.classes.Person");  
                     obj = c1.newInstance();  
                     setter(obj, "name", "xy", String.class);  
                     setter(obj, "age", 20, int.class);  
                     getter(obj, "name");  
                     getter(obj, "age");  
           }  
           catch (Exception e)  
            {  
                    e.printStackTrace();  
            }  
   }

   /\*\*  
    \* @param obj：要操作的对象  
    \* @param att：要操作的属性  
    \* @param value：要设置的属性内容  
    \* @param type：要设置的属性类型  
    \*/  
    public static void setter(Object obj, String att, Object value, Class<?> type)  
    {  
         try  
         {  
            // 得到setter方法  
          Method m = obj.getClass().getMethod("set" + initStr(att), type);  
          m.invoke(obj, value);  
         }  
         catch (Exception e)  
         {  
          e.printStackTrace();  
         }  
   }

   /\*\*  
    \* @param obj：要操作的对象  
    \* @param att：要操作的属性  
    \*/  
   public static void getter(Object obj, String att)  
   {  
        try  
        {  
               // 得到getter方法  
               Method m = obj.getClass().getMethod("get" + initStr(att));  
               System.out.println(m.invoke(obj));  
        }  
       catch (Exception e)  
       {  
               e.printStackTrace();  
       }  
   }

   public static String initStr(String oldStr)  
   {  
         String newStr = oldStr.substring(0, 1).toUpperCase() + oldStr.substring(1);  
         return newStr;  
   }  
}

**六 Spring调用getter,setter方法**

我们以setter注入例子

bean.xml  
<bean id="id" class="com.xy.Student">  
    **<property name="stuName" value="xy" />**</bean>

Spring将采用的代码创建代码Java实例，并注入值：  
**Class c = Class.forName("com.xy.Student");  
Object bean = c.newInstance();**

通过一些操作获取对stuName对应的setter方法名  
**String setname = "set" + "StuName";  
Method method = c.getMehod(setname,String.Class);  
method.invoke(bean,"xy");**

这样就完成了最基本的注入操作。当然，Spring还可以通过构造函数进行注入。这样就参考第二点有参构造的Class的使用。

Class还可以访问Annotation，这样就Spring使用注解的时候，可以完成注入的功能。