转自：http://www.cnblogs.com/rollenholt/archive/2011/09/02/2163758.html

【案例3】通过Class实例化其他类的对象

通过无参构造实例化对象

```
package Reflect;
class Person{

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public String toString(){
        return "["+this.name+"  "+this.age+"]";
    }
    private String name;
    private int age;
}

class hello{
    public static void main(String[] args) {
        Class<?> demo=null;
        try{
            demo=Class.forName("Reflect.Person");
        }catch (Exception e) {
            e.printStackTrace();
        }
        Person per=null;
        try {
            //demo.newInstance()是Object类型，强转成Person类型
            per=(Person)demo.newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
```

```
        per.setName("Rollen");
        per.setAge(20);
        System.out.println(per);
        //~out: [Rollen  20]
    }
}
```

但是注意一下，当我们把Person中的默认的无参构造函数取消的时候，比如自己定义只定义一个有参数的构造函数之后，会出现错误：

比如我定义了一个构造函数：

```
public Person(String name, int age) {
    this.age=age;
    this.name=name;
}
```

然后继续运行上面的程序，会出现：

```
java.lang.InstantiationException: Reflect.Person
    at java.lang.Class.newInstance0(Class.java:340)
    at java.lang.Class.newInstance(Class.java:308)
    at Reflect.hello.main(hello.java:39)
Exception in thread "main" java.lang.NullPointerException
    at Reflect.hello.main(hello.java:47)
```

所以大家以后在编写使用Class实例化其他类的对象的时候，一定要自己定义无参的构造函数

【案例】通过Class调用其他类中的构造函数（也可以通过这种方式通过Class创建其他类的对象）

```
package Reflect;
import java.lang.reflect.Constructor;
class Person{

    public Person() { }
    public Person(String name){
        this.name=name;
    }
    public Person(int age){
        this.age=age;
    }
    public Person(String name, int age) {
        this.age=age;
        this.name=name;
    }
    public String getName() {
        return name;
```

```
    }
    public int getAge() {
        return age;
    }
    @Override
    public String toString(){
        return "["+this.name+"  "+this.age+"]";
    }
    private String name;
    private int age;
}

class hello{
    public static void main(String[] args) {
        Class<?> demo = null;
        try{
            demo = Class.forName("Reflect.Person");
        }catch (Exception e) {
            e.printStackTrace();
        }
        Person per1=null;
        Person per2=null;
        Person per3=null;
        Person per4=null;
        //取得全部的构造函数
        Constructor<?> cons[] = demo.getConstructors();
        try{
         //cons是无序的，但可以根据构造方法的参数
         //判断出cons[0],cons[1],cons[2],cons[3]分别对应哪个构造方法
         for(int i=0;i<cons.length;i++){
                if(cons[i].getParameterTypes().length==2){
                    per4=(Person)cons[2].newInstance("Rollen",20);
                } else if(cons[i].getParameterTypes().length==1
                    && cons[i].getParameterTypes()
[0].equals(Integer.TYPE)){
                    per3=(Person)cons[1].newInstance(20);
                } else if(cons[i].getParameterTypes().length==0){
                    per1=(Person)cons[3].newInstance();
                } else {
                    per2=(Person)cons[0].newInstance("Rollen");
                }
         }
        }catch(Exception e){
            e.printStackTrace();
        }
```

```
        System.out.println(per1);
        System.out.println(per2);
        System.out.println(per3);
        System.out.println(per4);
        //~out : [null  0]
        //~out : [Rollen  0]
        //~out : [null  20]
        //~out : [Rollen  20]
    }
}
```