三，获取所有的构造器

1，getConstructors（）获取运行时类及其父类声明为public的构造器

2，getDeclaredConstructors（）获取运行时类本省声明的全部的构造器

其余的获取构造器的其他东西跟方法类似

四，其他

1，getSuperclass（）获取运行时类的父类

```
//1.获取运行时类的父类
@Test
public void test1(){
    Class clazz = Person.class;
    Class superClass = clazz.getSuperclass();
    System.out.println(superClass);
}
```

2，getGenericSuperclass（）获取带泛型的运行时的父类

```
//2.获取带泛型的父类
@Test
public void test2(){
    Class clazz = Person.class;
    Type type1 = clazz.getGenericSuperclass();
    System.out.println(type1);
}
```

3，获取父类的泛型

```
//3.获取父类的泛型
@Test
public void test3(){
    Class clazz = Person.class;
    Type type1 = clazz.getGenericSuperclass();
    ParameterizedType param = (ParameterizedType)type1;
    Type[] ars = param.getActualTypeArguments();
    System.out.println(((Class)ars[0]).getName());
}
```

4，getInterfaces（）获取实现的接口

```
//4.获取实现的接口
@Test
public void test4(){
    Class clazz = Person.class;
    Class[] interfaces = clazz.getInterfaces();
    for(Class i : interfaces){
        System.out.println(i);
    }
}
```

5，getPackage（）获取所在的包

```java
//5.获取所在的包
@Test
public void test5(){
    Class clazz = Person.class;
    Package pack = clazz.getPackage();
    System.out.println(pack);
}
```

6，getAnnotations（）获取注解（声明为runtime类型的）

```java
//6.获取注解
@Test
public void test6(){
    Class clazz = Person.class;
    Annotation[] anns = clazz.getAnnotations();
    for(Annotation a : anns){
        System.out.println(a);
    }
}
```

5.全部的Field

- public Field[] getFields()

返回此Class对象所表示的类或接口的public的Field。

- public Field[] getDeclaredFields()

返回此Class对象所表示的类或接口的全部Field。

- Field方法中：
- public int getModifiers() 以整数形式返回此Field的修饰符
- public Class<?> getType() 得到Field的属性类型
- public String getName() 返回Field的名称。

三、通过反射调用类中的指定方法、指定属性

1.调用指定方法

通过反射，调用类中的方法，通过Method类完成。步骤：

1.通过Class类的**getMethod(String name,Class...parameterTypes)**方法取得一个Method对象，并设置此方法操作时所需要的参数类型。

2.之后使用**Object invoke(Object obj, Object[] args)**进行调用，并向方法中传递要设置的obj对象的参数信息。