为了利用java反射获取属性和方法，需要先了解三个反射包中的类。

1.Constructor：代表类的单个构造方法，通过Constructor我们可以执行一个类的某个构造方法，有参或者无参　来创建对象。

2.Method：代表类中的单个方法，可以用于执行类的某个普通方法，有参或者无参，并可以接受返回值。

3.Field：代表类中的单个属性，用于set和get属性。

使用Class类中的方法可以获得该类中的所有Constructor对象，Method对象和Field对象。

但是仍然无法访问私有化的构造方法，普通方法和私有属性，此时我们可以使用他们继承父类的setAccessiable()方法，来设置或取消访问检查，以达到访问私有对象的目的。

```java
public class Beans1 {
      /**
     * 设置私有属性，并没有设置setters和getters
     */
    private String adminName = "adminName";
    private String userName = "username";
      public Beans1(){
            System.out.println("无参构造调用");
        }
        public Beans1(String name){
            System.out.println("带参数构造函数"+name);
        }
        public String getName(){
            return "getName返回值";
        }
    public static void main(String [] args) {
        try {
        Class<?> clazz = Class.forName("com.JavaBasic.reflect.Beans1");
        //
        Beans1 bean = (Beans1) clazz.newInstance();
        Field[] fs = clazz.getDeclaredFields();

        for (Field field : fs) {
            // 要设置属性可达，不然会抛出IllegalAccessException异常
            field.setAccessible(true);
            // 打印初始值
            System.out.println("设置属性之前: " + field.getName() + "===" + field.get(bean));
            // 设置属性值，set(Object obj, Object value)
            // obj - 应该修改其字段的对象
            // value - 正被修改的 obj 的字段的新值（参考api）
            field.set(bean, "Liang");
            // 打印设置属性之后的值
            System.out.println("设置属性之后: " + field.getName() + "=" + field.get(bean));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    }
}
```

Markers Properties Servers Data Source Explorer Snippets Console LogCat

<terminated> Beans1 (4) [Java Application] C:\Program Files\Java\jdk1.8.0_112\bin\javaw.exe (2018年9月6日 下午2:12:10)

```
无参构造调用
设置属性之前: adminName===adminName
设置属性之后: adminName=Liang
设置属性之前: userName===username
设置属性之后: userName=Liang
```

```java
import java.lang.reflect.Field;

/**
 * @author Administrator
 *
 */
public class Test {


    private String name;
    private int age;

    private Test(int age){
        this.age = age;
    }

    private void speak(String name){
        System.out.println("我的名字是:"+name);
    }

    public Test(String name) {
        this.name = name;
    }

}
```

```java
public class Main {
    public static void main(String [] args) {
        Test test = new Test("张三");

        Method [] methods = Test.class.getMethods();
        Field [] fields = Test.class.getDeclaredFields();

        for(int i = 0; i < fields.length; i ++) {
            fields[i].setAccessible(true);
            try {
                System.out.println(fields[i].get(test));
            } catch (Exception e) {
            }
            System.out.println(fields[i].getName());
        }
        Method [] methods2 = Test.class.getDeclaredMethods();
        for(int i = 0; i < methods2.length; i ++) {
            methods2[i].setAccessible(true);
              try {
                  methods2[i].invoke(test,"成功调用");
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (IllegalArgumentException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }
            System.out.println(methods2[i].getName());
        }

    }
```

```
Markers  Properties  Servers  Data Source Explorer  Snippets  Console  LogCat
<terminated> Main (1) [Java Application] C:\Program Files\Java\jdk1.8.0_112\bin\javaw.exe (2018年9月6日 下午2:22:09)
张三
name
0
age
java.lang.IllegalArgumentException: argument type mismatch
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at com.JavaBasic.reflect.Main.main(Main.java:28)
main
我的名字是:成功调用
```

这样，我们就获得了私有属性的值，~~当然，java反射也存在优缺点：

优点：

1. 能够运行时动态获取类的实例，大大提高系统的灵活性和扩展性。

2. 与java动态编译结合，可以实现无比强大的功能。

缺点：

1. 使用反射的性能较低。

2. 使用反射来说相对不安全。

3. 破坏了类的封装性，可以通过反射来获取这个类的属性，和私有方法。