单元测试简单说明文档

张雄彪

Table of Contents

1. 单元测试说明 1
2. 一般单元测试任务 2
3. 好处 3
4. 坏处 4
5. 基本规范 5
    5.1. 基本原则 5
    5.2. 编写规范 5
6. 单元测试工具 7
7. 普通 *Java* 类测试 8
8. Spring 与 Junit 集成测试 11
    8.1. 编写 BaseTestCase 11
    8.2. 编写普通Case 12
9. SpringMVC 测试 15

Chapter 1. 单元测试说明

单元测试（模块测试）是开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。单元测试并不一定保证程序功能是正确的，更不保证整体业务是准备的。

单元测试是由程序员自己来完成，最终受益的也是程序员自己。可以这么说，程序员有责任编写功能代码，同时也就有责任为自己的代码编写单元测试。执行单元测试，就是为了证明这段代码的行为和我们期望的一致。

Chapter 2. 一般单元测试任务

1. 接口功能测试：用来保证接口功能的正确性。
2. 局部数据结构测试（不常用）：用来保证接口中的数据结构是正确的
    a. 比如变量有无初始值
    b. 变量是否溢出
3. 边界条件测试
    a. 变量没有赋值（即为NULL）
    b. 变量是数值（或字符)
        i. 主要边界：最小值，最大值，无穷大（对于DOUBLE等）
        ii. 溢出边界（期望异常或拒绝服务）：最小值-1，最大值+1
        iii. 临近边界：最小值+1，最大值-1
    c. 变量是字符串
        i. 引用“字符变量”的边界
        ii. 空字符串
        iii. 对字符串长度应用“数值变量”的边界
    d. 变量是集合
        i. 空集合
        ii. 对集合的大小应用“数值变量”的边界
        iii. 调整次序：升序、降序
    e. 变量有规律
        i. 比如对于Math.sqrt，给出n^{2-1}，和n^2+1的边界
4. 所有独立执行通路测试：保证每一条代码，每个分支都经过测试
    a. 代码覆盖率
        i. 语句覆盖：保证每一个语句都执行到了
        ii. 判定覆盖（分支覆盖）：保证每一个分支都执行到
        iii. 条件覆盖：保证每一个条件都覆盖到true和false（即if、while中的条件语句）
        iv. 路径覆盖：保证每一个路径都覆盖到
5. 各条错误处理通路测试：保证每一个异常都经过测试

Chapter 3. 好处

编写单元测试用例的好处

1. 提高代码质量
2. 可快速测试，不要重复启tomcat
3. 测试用例可复用
4. 不仅仅用来保证当前代码的正确性，更重要的是用来保证代码修复、改进或重构之后的正确性

Chapter 4. 坏处

短时间内会增加开发人员负担

Chapter 5. 基本规范

5.1. 基本原则

1. 单元测试必须保持每一个测试方法是独立的，不依赖于其它测试用例或方法，也不影响其它测试用例或方法。
2. 单元测试不应该对数据库造成脏数据。即应使用事务回滚数据。
3. 单元测试不应该依赖于数据库已有数据。即测试数据应自己写代码准备。需要保证你的测试用例在别人的环境下也可以正常运行。
4. 单元测试必须覆盖每一条语句，每一个异常，每一个if/else分支。

5.2. 编写规范

1. 测试类命令

```
public class CatalogControllerTest extends BaseTest { ①

}
```

① 被测试类名 *CatalogController* 加上 *Test* 为测试类的命名

2. 测试方法命令

以下几种写法都可以

```
@Test
public void tree() throws Exception { ①

}

@Test
public void testTree() throws Exception { ②

}

@Test
public void testTree_emptyParam() throws Exception { ③

}

@Test
public void testTreeEmptyParam() throws Exception { ④

}
```

① 直接与被测试方法同名

② 使用 *test* 加 被测试方法名 *Tree*

③ 同一方法，多个不同测试条件时，使用下划线 _ 连接后面的测试说明

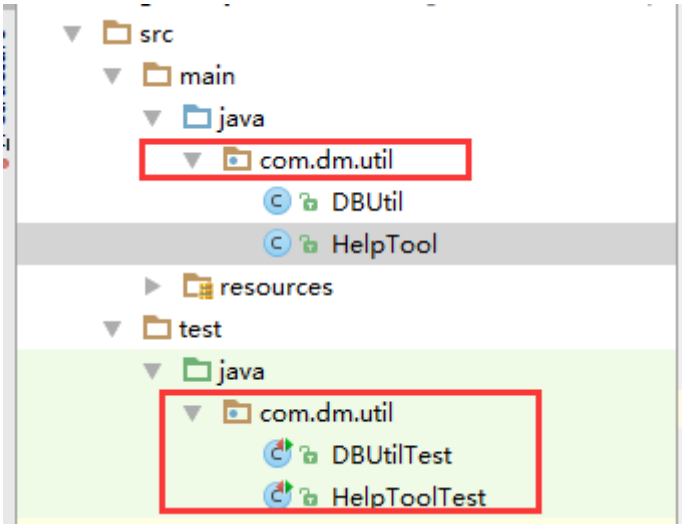
④ 也可以不用下划线

Chapter 6. 单元测试工具

本例中直接使用的是 *Junit4* 。基于注解配置，较方便。

Chapter 7. 普通 *Java* 类测试

1. 测试类所在包与被测试类所在包保持一致



2. 示例

HelpTool

```
package com.dm.util;

/**
 * @author zxb
 * @version 1.0.0
 * @date 2016年04月25日 11:08
 * @since Jdk1.6
 */
public class HelpTool {

    public String doHelp(Boolean flag){
        if(flag == null){
            throw new IllegalArgumentException("flag can't be null!");
        }
        if(flag){
            return "do it yourself!";
        }else{
            return "hello, " + flag;
        }
    }
}
```

HelpToolTest

```
package com.dm.util;

import org.junit.Test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;

/**
 * @author zxb
 * @version 1.0.0
 * @date 2016年04月25日 11:22
 * @since Jdk1.6
 */
public class HelpToolTest {
    @Test
    public void doHelp() throws Exception {
        HelpTool helpTool = new HelpTool();

        String returnStr = helpTool.doHelp(true);
        assertNotNull(returnStr); ①
        assertEquals("do it yourself!", returnStr); ②

        returnStr = helpTool.doHelp(false); ③
        assertNotNull(returnStr);
        assertEquals("hello, false", returnStr);
    }

    @Test
    public void doHelp_nullParam() throws Exception {
        HelpTool helpTool = new HelpTool();

        Throwable tx = null;
        try {
            String returnStr = helpTool.doHelp(null);
            fail(); ④
        } catch (Exception e) {
            tx = e;
        }
        assertNotNull(tx); ⑤
        assertTrue(IllegalArgumentException.class.isAssignableFrom(tx.getClass()));
        assertEquals("flag can't be null!", tx.getMessage());
    }
}
```

① 断言，returnStr不为空。断言失败时，测试用例将不会通过。

② 断言 *returnStr* 一定为 "do it yourself!"

③ 测试 *else* 分支

④ 测试 异常处理情况，此处未抛异常则应该 *fail*，使测试用例不通过。

⑤ 断言异常类型及异常 *message*

Chapter 8. Spring 与 Junit 集成测试

详见 *Spring 官方文档*

8.1. 编写 BaseTestCase

为了复用测试用例的一些通用配置，建议编写一个 *BaseTestCase* 类，子类直接继承该类即可。

BaseTestCase

```
package com.dm.catalog;

import com.dm.cas.client.filter.AuthFilter;
import com.dm.wmf.core.context.ContextUtils;
import com.dm.wmf.core.util.SetCharacterEncodingFilter;
import com.dm.wmf.core.util.SetCurrentRequestFilter;
import org.junit.Before;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.mock.web.MockHttpServletRequest;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.transaction.TransactionConfiguration;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.RequestPostProcessor;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.context.WebApplicationContext;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

/**
 * 单元测试类基类，提供公共的测试环境配置信息。
 * <br/>
 * 建议所有的子测试类直接继承该类。
 *
 * @author zxb
 * @version 1.0
 *          Created by zxb on 2016/3/14.
 */
@RunWith(SpringJUnit4ClassRunner.class)   ①
@ContextConfiguration(locations = {       ②
```

```
        "classpath*:com/dm/**/beans-*.xml",
        "classpath*:resource/**/beans-*.xml",
        //"classpath*:com/dm/**/servlet-*.xml",
        "classpath*:resource/**/servlet-*.xml"
})
@WebAppConfiguration // 定义webapp目录    ③
@Transactional
@TransactionConfiguration(transactionManager = "transactionManager")        ④
public class BaseTest {

    @Autowired
    protected WebApplicationContext wac;  ⑤

    @Before
    public void setUp() { ⑥
        ContextUtils.setContext(this.wac.getServletContext());    ⑦
         //设置wmf框架中的ContextUtils的servlet context
    }
}
```

① 与 *Spring* 集成时，需要用这个 *SpringJUnit4ClassRunner.class* 来跑

② 指定要加载的 *Spring* 配置文件。建议与 *web.xml* 中配置的保持一致。

③ 指定 *webapp* 目录，默认为 *src/main/webapp*

④ 指定事务 *transactionManger*

⑤ 注入 *WebApplicationContext* ，如果非 *web* 工程，此处可以注入 *ApplicationContext*

⑥ *setUp* 配置 @Before注解。将在每一个测试方法执行前执行该方法。

⑦ 设置框架使用的 *ContextUtils* 中的 *servletContext*

8.2. 编写普通Case

UserMapperServiceDefaultImplTest

```
package com.dm.cas.validation.service.impl;

import com.dm.cas.server.domain.User;
import com.dm.cas.server.service.UserMapperService;
import com.dm.cas.validation.BaseTest;
import com.dm.cas.validation.service.UserService;
import com.dm.cas.validation.util.EncryptUtil;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.UUID;

import static org.junit.Assert.*;
```

```java
/**
 * @author zxb
 * @version 1.0.0
 * @date 2016年04月17日 18:05
 * @since Jdk1.6
 */
public class UserMapperServiceDefaultImplTest extends BaseTest {

    @Autowired
    private UserMapperService userMapperService; ①

    @Autowired
    private UserService userService;

    private User user = null;

    @Before
    public void setUp() throws Exception { ②
        user = new User();
        user.setUser_id(UUID.randomUUID().toString().replaceAll("-", ""));
        user.setUser_name("张雄彪");
        user.setUser_code("zxb_test001");
        user.setUser_password(EncryptUtil.md5Degest("11223344"));
        user.setUser_sfzh("420921199111105711");
        user.setValid_flag(1);
        user = userService.create(user);
    }

    @After
    public void tearDown() throws Exception { ③
        user = null;
    }

    @Test
    public void testGetUser() throws Exception { ④
        try {
            User user = this.userMapperService.getUser("zxb_test001");
            assertNotNull(user);
            assertEquals("张雄彪", user.getUser_name());
            assertEquals("zxb_test001", user.getUser_code());
            assertEquals(EncryptUtil.md5Degest("11223344"), user.getUser_password());
            assertEquals("420921199111105711",user.getUser_sfzh());
            assertTrue(1 == user.getValid_flag());
        } catch (Exception e) {
            e.printStackTrace();
            fail();
        }
    }
```

```
    @Test
    public void testGetMappingUser_emptyParam() throws Exception{ ⑤
        User mappingUser = null;
        Throwable tx = null;
        try {
            user.setUser_code("  ");
            mappingUser = this.userMapperService.getUser(user);
            fail();
        } catch (Exception e) {
            tx = e;
        }
        assertNotNull(tx);
        assertTrue(IllegalArgumentException.class.isAssignableFrom(tx.getClass()));
        assertEquals("user is null or user_code is empty!", tx.getMessage());

        try {
            user = null;
            mappingUser = this.userMapperService.getUser(user);
            fail();
        } catch (Exception e) {
            tx = e;
        }
        assertNotNull(tx);
        assertTrue(IllegalArgumentException.class.isAssignableFrom(tx.getClass()));
        assertEquals("user is null or user_code is empty!", tx.getMessage());
    }

}
```

① 可以直接注入要测试的 *service* 类

② @Before *setUp* 中准备创建测试数据。不应该直接使用库中现有的数据，此处应准备数据，确保其它开发人员机器环境下可以正常跑过。

③ @After *tearDown* 每个测试方法执行完成后会执行该方法。要保证每个测试方法互不影响。

④ 测试

⑤ 异常测试

Chapter 9. SpringMVC 测试

除了可以测试 *Spring* 的 *service bean* 外，还可以使用测试 *springMvc* 中的 *Controller*

参考示例： .BaseTest

```
package com.dm.catalog;

import com.dm.cas.client.filter.AuthFilter;
import com.dm.wmf.core.context.ContextUtils;
import com.dm.wmf.core.util.SetCharacterEncodingFilter;
import com.dm.wmf.core.util.SetCurrentRequestFilter;
import org.junit.Before;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.mock.web.MockHttpServletRequest;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.transaction.TransactionConfiguration;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.RequestPostProcessor;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.context.WebApplicationContext;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

/**
 * 单元测试类基类，提供公共的测试环境配置信息。
 * <br/>
 * 建议所有的子测试类直接继承该类。
 *
 * @author zxb
 * @version 1.0
 *          Created by zxb on 2016/3/14.
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {
        "classpath*:com/dm/**/beans-*.xml",
        "classpath*:resource/**/beans-*.xml",
        //"classpath*:com/dm/**/servlet-*.xml",
        "classpath*:resource/**/servlet-*.xml"
```

```
})
@WebAppConfiguration // 定义webapp目录
@Transactional
@TransactionConfiguration(transactionManager = "transactionManager")
public class BaseTest {

    @Autowired
    protected WebApplicationContext wac;

    protected MockMvc mvc; ①

    @Before
    public void setUp() {
        mvc = MockMvcBuilders.webAppContextSetup(wac)
                .defaultRequest( ②
                        get("/")
                                .accept(MediaType.parseMediaType(
"text/html;charset=UTF-8"))
                                .contextPath("/catalog")
                                .with(new RequestPostProcessor() {
                                    @Override
                                    public MockHttpServletRequest postProcessRequest
(MockHttpServletRequest request) {
                                        request.setRemoteUser("zxb");
//设置登录的用户账号 ③
                                        return request;
                                    }
                                })
                )
                .alwaysDo(print()) ④
                .alwaysExpect(status().isOk()) ⑤
                // .alwaysExpect(content().contentType("text/html;charset=UTF-8"))
                .addFilter(new SetCharacterEncodingFilter(), "/*")
                // .addFilter(new AuthFilter(), "*.do") //避免登录验证
                .addFilter(new SetCurrentRequestFilter(), "*.do", "*.html") ⑥
                .build();
        ContextUtils.setContext(this.wac.getServletContext());    //设置wmf
框架中的ContextUtils的servlet context
    }
}
```

① *MockMvc* 使用该对象模拟请求

② 设置默认请求设置，*contextPath* 指定被测试 *web* 工程的上下文路径

③ 设置用户账号，模拟登陆。

④ 打印请求响应信息

⑤ 断言请求返回状态为 200

⑥ 添加过滤器

CatalogControllerTest

```
public class CatalogControllerTest extends BaseTest {
  @Test
  public void listCatalog() throws Exception {
      try {
          super.mvc.perform(get("/catalog/catalog/Catalog_testUser.do")) ①
                  .andDo(print()

                  ) //打印信息
                  .andExpect(status().isOk())
                  //.andExpect(model().attribute("error", "查询方案不存在！")) ②
                  //.andExpect(view().name("/catalog/catalog_viewColumns.vm"))
                  //.andExpect(content().contentType("text/html;charset=UTF-8"))
                  //.andExpect(content().contentType("text/html;charset=GBK"))
                  .andExpect(jsonPath("$.user_code").value("zxb")) ③
                  .andReturn();
      } catch (Exception e) {
          fail();
      }
  }
}
```

① 指定请求地址

② 断言 *ModelAndView* 对象中的信息

③ 断言返回的 *JSON* 对象信息