单元测试

#### 定义

**单元测试**： 一个单元测试是一段自动化的代码，这段代码调用被测试的工作单元，之后对这个单元的单个最终结果的某些假设进行检验。单元测试几乎都是用单元测试框架编写的。单元测试容易编写，能快速运行。单元测试可靠、可读，并且可维护。只要产品代码不发生变化，单元测试的结果是稳定的。

#### 单元测试要点

1. 接口功能测试：保证接口功能的正确性。
2. 边界条件测试
   * 变量没有赋值（即为NULL）
   * 变量是数值（或字符)
     1. **主要边界**：最小值，最大值，无穷大（对于DOUBLE等）
     2. **溢出边界**（期望异常或拒绝服务）：最小值-1，最大值+1
     3. **临近边界**：最小值+1，最大值-1

如果文本输入域允许输入1－255个字符。

尝试：输入1个字符和255个字符（合法区间），也可以加入254个字符作为合法测试。

输入0个字符和256个字符作为非法区间。

* + 变量是字符串
    1. 引用“字符变量”的边界
    2. 空字符串
    3. 对字符串长度应用“数值变量”的边界
  + 变量是集合
    1. 空集合
    2. 对集合的大小应用“数值变量”的边界
    3. 调整次序：升序、降序
  + 变量有规律
    1. 比如对于Math.sqrt，给出n^2-1，和n^2+1的边界

1. 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
   * 代码覆盖率
     1. 语句覆盖：保证每一个语句都执行到了
     2. 判定覆盖（分支覆盖）：保证每一个分支都执行到
     3. 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）
     4. 路径覆盖：保证每一个路径都覆盖到
2. 各条错误处理通路测试：保证每一个异常都经过测试

#### JUNIT

* JUnit是Java单元测试框架，已经在Eclipse中默认安装。目前主流的是JUnit4。JUnit4中，测试用例无需继承TestCase类，只需要使用@Test等注解。JAVA采用JUnit进行单元测试是比较通用的做法。

JUnit4通过注解的方式来识别测试方法。目前支持的主要注解有：

@BeforeClass 全局只会执行一次，而且是第一个运行

@Before 在测试方法运行之前运行

@Test 测试方法

@After 在测试方法运行之后允许

@AfterClass 全局只会执行一次，而且是最后一个运行

@Ignore 忽略此方法

* 运行测试方法

可以在Eclipse中运行，工程通过maven进行管理的话，也可通过mvn test命令运行。

#### Maven

###### 利用 Maven 初始化一个简单的 Java 应用：

mvn archetype:generate -DartifactId=test-demo -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

Maven 会自动创建好类文件和测试类，路径如下：

test-demo

├── pom.xml ---- pom 依赖配置文件

└── src ---- 源码路径

   ├── main ---- 类文件

   │   └── java

   │   └── com

   │   └── App.java

   └── test ---- 测试类

   └── java

   └── com

   └── AppTest.java

在 pom.xml 中引入 JUnit4：

<dependencies>

<!-- junit4 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

</dependencies>

引入 JUnit 依赖后，就能在测试类中，通过 JUnit 提供的注解和静态方法，进行测试了。

一般在执行mvn package 或者 mvn install时，maven会默认先执行test命令。

###### maven-surefire-plugin的使用

maven-surefire-plugin是maven里执行测试用例的插件，不显示配置就会用默认配置。这个插件的surefire:test命令会默认绑定maven执行的test阶段

JUnit3.8以下的版本surefire不支持。建议用最新的JUnit版本，目前是4.12。

跳过测试

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.19</version>

<configuration>

<skipTests>true</skipTests>

</configuration>

</plugin>

若有测试执行失败则跳过其他测试

在打包时,默认情况会执行全部测试用例,然后给出一个执行的统计结果,如下所示:

Results :

Tests run: 3, Failures: 1, Errors: 0, Skipped: 0

很多情况下我们希望测试用例没有失败的才能打包,如果出现打包失败,需要立刻停止执行其他测试用例.为满足这个要求,我们需要增加一些配置设定.

<configuration>

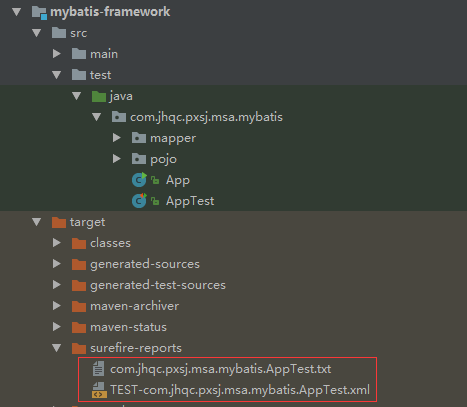
<skipAfterFailureCount>1</skipAfterFailureCount>

</configuration>

里面的数字1表示当有一个测试用例执行失败或发生异常时,跳过后续的其他测试用例.这个数字其实只要是一个大于零的数就可以.表达的意思就是当有N个测试用例执行失败或异常时,跳过后续的其他测试用例.

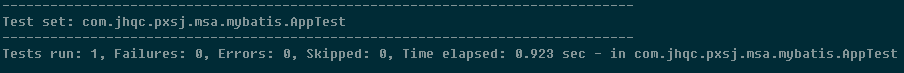
查看测试报告

在执行完mvn test后,会在target目录下生成测试报告

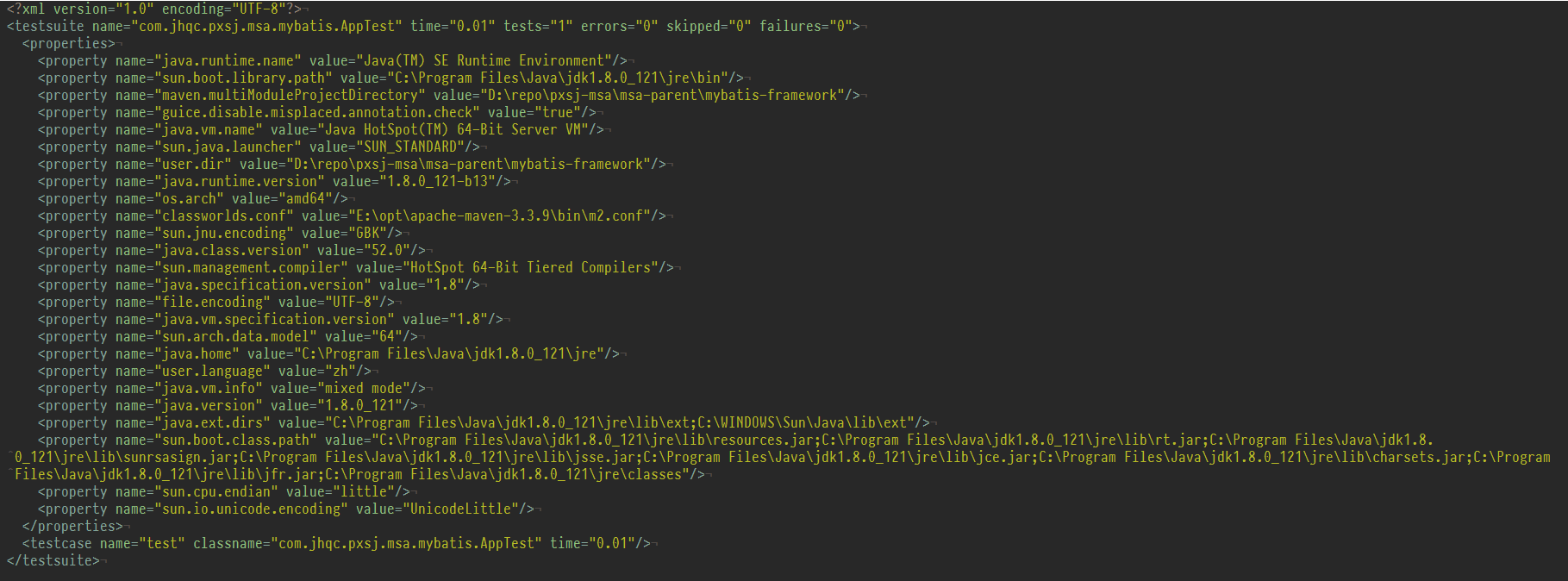


一共两个文件,txt文件记录了汇总信息,xml文件里记录了测试用例执行的环境和执行情况,而且方便程序解析展现。

txt文件内容：



xml文件内容：



#### Spring构建RESTful API与单元测试

API网关和微服务对外提供的全是RESTful 接口，确保对外的API进行完全测试。下面一个例子说明怎样测试。

User实体定义：

public class User {

private Long id;

private String name;

private Integer age;

// 省略setter和getter

}

实现对User对象的操作的RestController：

@RestController

@RequestMapping(value="/users") // 通过这里配置使下面的映射都在/users下

public class UserController {

// 创建线程安全的Map

static Map<Long, User> users = Collections.synchronizedMap(new HashMap<Long, User>());

@RequestMapping(value="/", method=RequestMethod.GET)

public List<User> getUserList() {

// 处理"/users/"的GET请求，用来获取用户列表

// 还可以通过@RequestParam从页面中传递参数来进行查询条件或者翻页信息的传递

List<User> r = new ArrayList<User>(users.values());

return r;

}

@RequestMapping(value="/", method=RequestMethod.POST)

public String postUser(@ModelAttribute User user) {

// 处理"/users/"的POST请求，用来创建User

// 除了@ModelAttribute绑定参数之外，还可以通过@RequestParam从页面中传递参数

users.put(user.getId(), user);

return "success";

}

@RequestMapping(value="/{id}", method=RequestMethod.GET)

public User getUser(@PathVariable Long id) {

// 处理"/users/{id}"的GET请求，用来获取url中id值的User信息

// url中的id可通过@PathVariable绑定到函数的参数中

return users.get(id);

}

@RequestMapping(value="/{id}", method=RequestMethod.PUT)

public String putUser(@PathVariable Long id, @ModelAttribute User user) {

// 处理"/users/{id}"的PUT请求，用来更新User信息

User u = users.get(id);

u.setName(user.getName());

u.setAge(user.getAge());

users.put(id, u);

return "success";

}

@RequestMapping(value="/{id}", method=RequestMethod.DELETE)

public String deleteUser(@PathVariable Long id) {

// 处理"/users/{id}"的DELETE请求，用来删除User

users.remove(id);

return "success";

}

}

针对该Controller编写测试用例验证正确性，具体如下：

// SpringJUnit支持，由此引入Spring-Test框架支持

@RunWith(SpringJUnit4ClassRunner.class)

// 指定我们工程的Application启动类

@SpringApplicationConfiguration(classes = MockServletContext.class)

// 由于是Web项目，Junit需要模拟ServletContext，因此我们需要给我们的测试类加上@WebAppConfiguration

@WebAppConfiguration

public class ApplicationTests {

private MockMvc mvc;

@Before

public void setUp() throws Exception {

mvc = MockMvcBuilders.standaloneSetup(new UserController()).build();

}

@Test

public void testUserController() throws Exception {

// 测试UserController

RequestBuilder request = null;

// 1、get查一下user列表，应该为空

request = get("/users/");

mvc.perform(request)

.andExpect(status().isOk())

.andExpect(content().string(equalTo("[]")));

// 2、post提交一个user

request = post("/users/")

.param("id", "1")

.param("name", "测试大师")

.param("age", "20");

mvc.perform(request)

.andExpect(content().string(equalTo("success")));

// 3、get获取user列表，应该有刚才插入的数据

request = get("/users/");

mvc.perform(request)

.andExpect(status().isOk())

.andExpect(content().string(equalTo("[{\"id\":1,\"name\":\"测试大师\",\"age\":20}]")));

// 4、put修改id为1的user

request = put("/users/1")

.param("name", "测试终极大师")

.param("age", "30");

mvc.perform(request)

.andExpect(content().string(equalTo("success")));

// 5、get一个id为1的user

request = get("/users/1");

mvc.perform(request)

.andExpect(content().string(equalTo("{\"id\":1,\"name\":\"测试终极大师\",\"age\":30}")));

// 6、del删除id为1的user

request = delete("/users/1");

mvc.perform(request)

.andExpect(content().string(equalTo("success")));

// 7、get查一下user列表，应该为空

request = get("/users/");

mvc.perform(request)

.andExpect(status().isOk())

.andExpect(content().string(equalTo("[]")));

}

}

执行mvn test就可以对上面的测试用例进行测试，在上述例子中，一个测试用例把UserController的接口全部覆盖。测试报告可参看[[查看测试报告](#_查看测试报告)]节