单元测试用例编写

前言：在实际项目中写单元测试的过程中我们会发现需要测试的类有很多依赖，这些依赖项又会有依赖，导致在单元测试代码里几乎无法完成构建，尤其是当依赖项尚未构建完成时会导致单元测试无法进行。为了解决这类问题我们引入了Mock的概念，简单的说就是模拟这些需要构建的类或者资源，提供给需要测试的对象使用。业内的Mock工具有很多，也已经很成熟了，这里我们将直接使用流行的Mockito进行。

1.1 Mockito准备工作

通过Maven管理的，需要在项目的Pom.xml中增加如下的依赖：

```
<properties>
    <java.version>1.8</java.version>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.
outputEncoding>
    <maven.compiler.encoding>UTF-8</maven.compiler.encoding>
    <jacoco.version>0.8.10</jacoco.version>
    <jacoco.report-path>${project.basedir}/target/jacoco.exec</jacoco.
report-path>
</properties>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

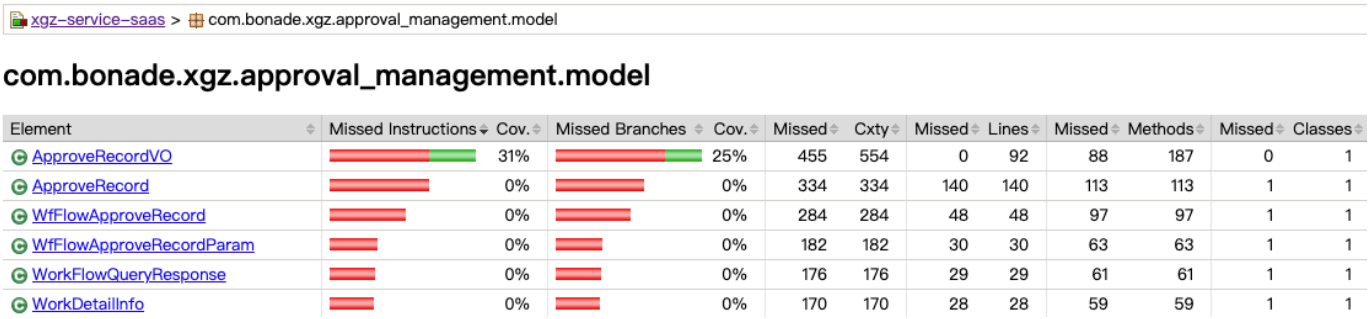
引入jacoco插件

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.22.2</version>
    <dependencies>
        <dependency>
            <groupId>org.apache.maven.surefire</groupId>
            <artifactId>surefire-junit47</artifactId>
            <version>2.22.2</version>
        </dependency>
    </dependencies>
    <configuration>
        <includes>
            <include>**/*Test.java</include>
            <include>**/*Application.java</include>
        </includes>
        <parallel>all</parallel>
```

```
        <threadCount>16</threadCount>
        <useUnlimitedThreads>true</useUnlimitedThreads>
        <perCoreThreadCount>true</perCoreThreadCount>
        <useSystemClassLoader>false</useSystemClassLoader>
        <forkCount>6</forkCount>
        <reuseForks>true</reuseForks>
        <testFailureIgnore>true</testFailureIgnore>
        <argLine>-Dfile.encoding=UTF-8 ${surefireArgLine}</argLine>
        <systemPropertyVariables>
            <jacoco-agent.destfile>${jacoco.report-path}</jacoco-agent.
destfile>
        </systemPropertyVariables>
    </configuration>
</plugin>

<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>${jacoco.version}</version>
<configuration>
    <excludes>
        <exclude>**/chain/**</exclude>
        <exclude>**/feign/**</exclude>
        <exclude>**/dto/**</exclude>
        <exclude>**/model/**</exclude>
        <exclude>**/entity/**</exclude>
        <exclude>**/mapper/**</exclude>
        <exclude>**/exception/**</exclude>
        <exclude>**/*Application.class</exclude>
        <exclude>**/*Builder.class</exclude>
        <exclude>**/*BO.class</exclude>
        <exclude>**/*DO.class</exclude>
        <exclude>**/*DTO.class</exclude>
        <exclude>**/*Dto.class</exclude>
        <exclude>**/*VO.class</exclude>
        <exclude>**/*Vo.class</exclude>
        <exclude>**/*Param.class</exclude>
        <exclude>**/*Params.class</exclude>
        <exclude>**/*Factory.class</exclude>
        <exclude>**/*Handler.class</exclude>
        <exclude>**/*dao.class</exclude>
        <exclude>**/*Dao.class</exclude>
        <exclude>**/*Util.class</exclude>
        <exclude>**/vo/**</exclude>
        <exclude>**/util/**</exclude>
        <exclude>**/config/**</exclude>
        <exclude>**/common/**</exclude>
    </excludes>
</configuration>
    <executions>
        <execution>
            <id>default-prepare-agent</id>
            <goals>
```

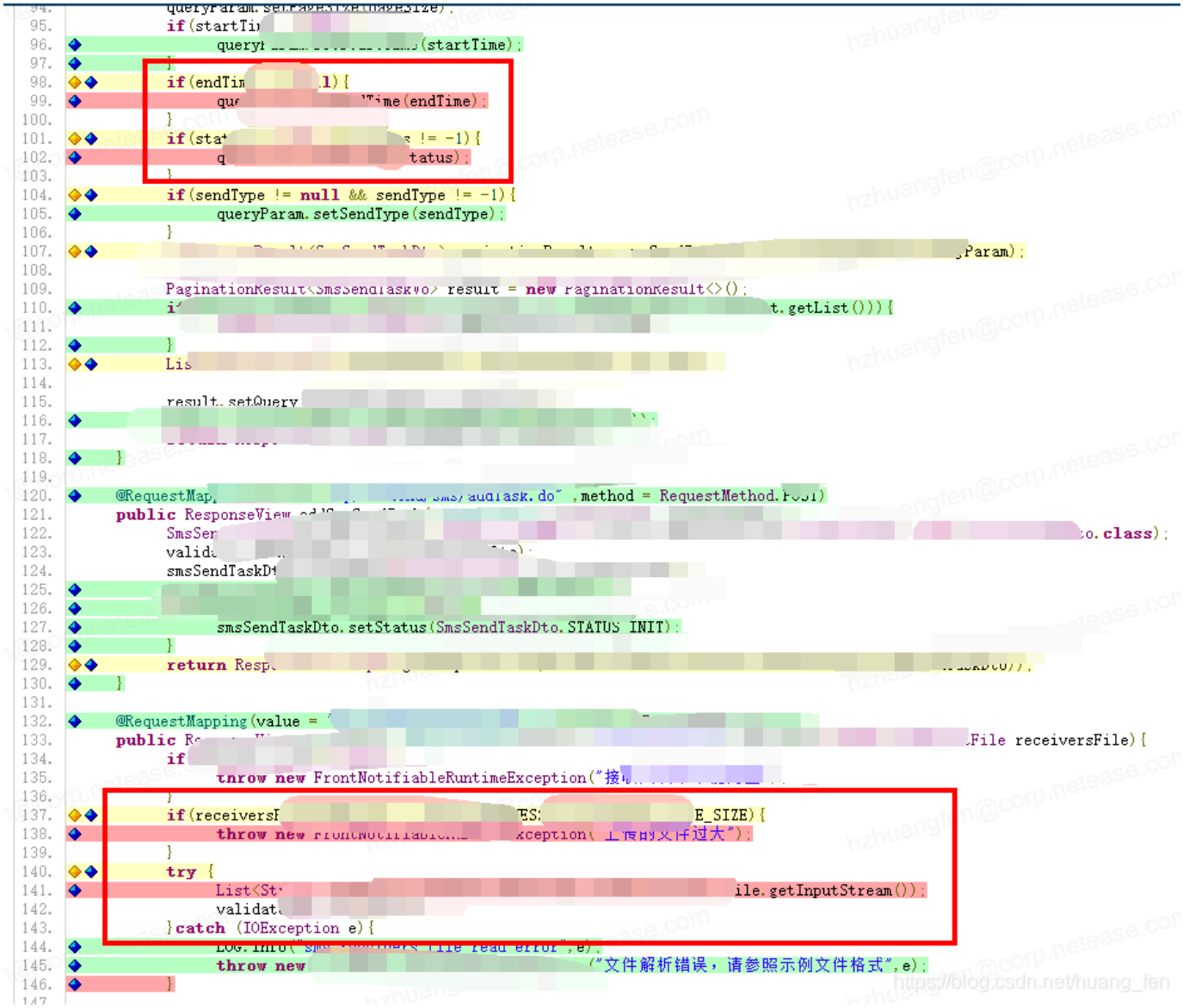
```
                <goal>prepare-agent</goal>
            </goals>
            <configuration>
                <propertyName>surefireArgLine</propertyName>
            </configuration>
        </execution>
        <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
            <configuration>
                <dataFile>${jacoco.report-path}</dataFile>
                <outputDirectory>target/jacoco-report</outputDirectory>
            </configuration>
        </execution>
    </executions>
</plugin>
```

执行打包命令后，会生成对应的报告：



显示用例的覆盖情况（红色表示未覆盖，绿色表示已覆盖，黄色表示部分覆盖）：

```
public class ApproveRecordController {

    @Autowired
    private ApproveRecordService approveRecordService;

    @Resource
    private UserInfoServiceImpl userInfoService;

    @ApiOperation(value = "分页获取审批记录接口", httpMethod = "POST", tags = SwggerCommonTags.XGZ_APPROVE)
    @RequestMapping(value = "/queryApproveRecordListPage", method = RequestMethod.POST)
    @ApiImplicitParams({
            @ApiImplicitParam(paramType = "body", dataType = "ApproveRecordVO", name = "approveRecordVo", value = "", required = true)
    })
    public Result<?> queryUseWaterListPage(@RequestBody @Valid ApproveRecordVO approveRecordVo, @EnableUser LoginUser loginUser) {
        List<ApproveRecordVO> list = approveRecordService.queryApproveRecordListPage(approveRecordVo);
        IPage<ApproveRecordVO> page = new Page<>(approveRecordVo.getOffset(), approveRecordVo.getLimit());
        page.setRecords(list);
        return Result.data(page);
    }

    @ApiOperation(value = "审批记录详情", httpMethod = "POST", tags = SwggerCommonTags.XGZ_APPROVE)
    @RequestMapping(value = "/getApproveRecordDetail", method = RequestMethod.POST)
    public Result<?> getApproveRecordDetail(
            @ApiParam(required = true, name = "approveRecordId", value = "审批记录ID") @RequestParam(defaultValue = "0") Integer approveRecordId) {
        ApproveRecordVO vo = approveRecordService.getApproveRecordDetail(approveRecordId);
        return Result.data(vo);
    }

```



1.2 模拟对象

```
mock(Class classToMock);
mock(Class classToMock, String name)
mock(Class classToMock, Answer defaultAnswer)
mock(Class classToMock, MockSettings mockSettings)
mock(Class classToMock, ReturnValues returnValues)
```

可以对类和接口进行mock对象的创建，创建时可以为mock对象命名。对mock对象命名的好处是调试的时候容易辨认mock对象。

Mock对象的期望行为和返回值设定

假设我们创建了LinkedList类的mock对象：

LinkedList mockedList = mock(LinkedList.class);

1.3 设置对象调用的预期返回值

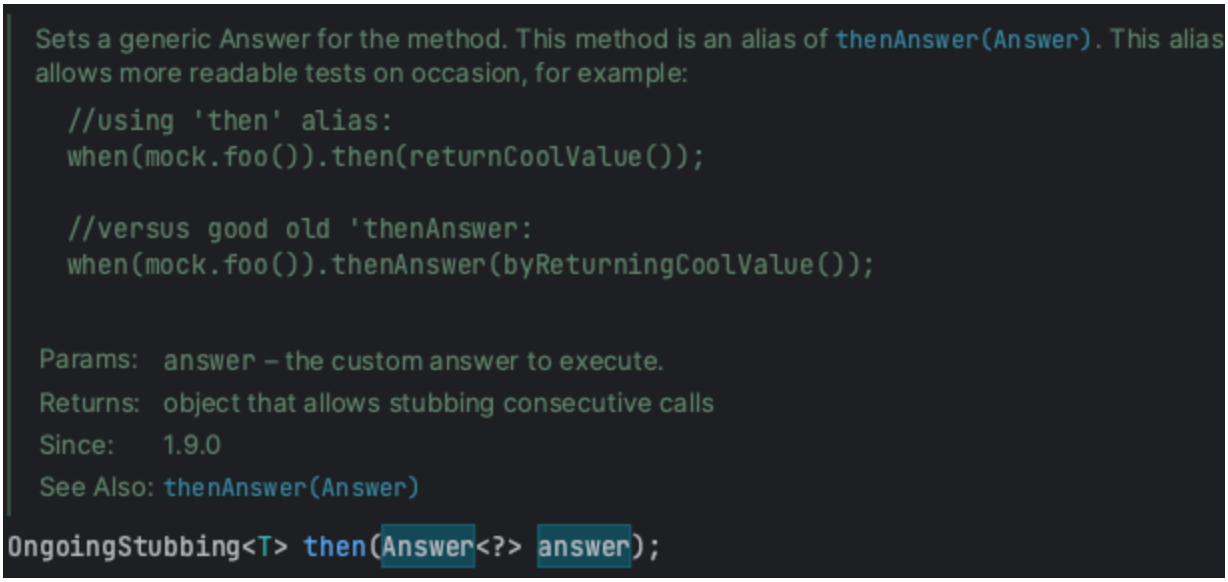
通过 when(mock.someMethod()).thenReturn(value) 来设定 Mock 对象某个方法调用时的返回值。我们可以看看源码中关于thenReturn方法的注释：

```
Sets a return value to be returned when the method is called. E.g:
  when(mock.someMethod()).thenReturn(10);

See examples in javadoc for Mockito.when
Params: value – return value
Returns: object that allows stubbing consecutive calls
OngoingStubbing<T> thenReturn(T value);

Sets consecutive return values to be returned when the method is called. E.g:
  when(mock.someMethod()).thenReturn(1, 2, 3);

Last return value in the sequence (in example: 3) determines the behavior of further consecutive calls.
See examples in javadoc for Mockito.when
Params: value – first return value
        values – next return values
Returns: object that allows stubbing consecutive calls
// Additional method helps users of JDK7+ to hide heap pollution / unchecked g
/unchecked, varargs/
OngoingStubbing<T> thenReturn(T value, T... values);
```

使用when(mock.someMethod()).thenThrow(new RuntimeException) 的方式来设定当调用某个方法时抛出的异常：

```
Sets Throwable objects to be thrown when the method is called. E.g:
  when(mock.someMethod()).thenThrow(new RuntimeException());

If throwables contain a checked exception then it has to match one of the checked exceptions of method signature.
You can specify throwables to be thrown for consecutive calls. In that case the last throwable determines the behavior of further consecutive calls.
If throwable is null then exception will be thrown.
See examples in javadoc for Mockito.when
Params: throwables – to be thrown on method invocation
Returns: object that allows stubbing consecutive calls
OngoingStubbing<T> thenThrow(Throwable... throwables);
```

Answer 是个泛型接口。到调用发生时将执行这个回调，通过 Object[] args = invocation.getArguments();可以拿到调用时传入的参数，通过 Object mock = invocation.getMock();可以拿到mock对象。



有些方法可能接口的参数为一个Listener参数，如果我们使用Answer打桩，我们就可以获取这个Listener,并且在Answer函数中执行对应的回调函数。

1.4 验证被测试类方法

Mock 对象一旦建立便会自动记录自己的交互行为，所以我们可以有选择的对它的 交互行为进行验证。在 Mockito 中验证 Mock 对象交互行为的方法是 verify(mock).someMethod(…)。最后 Assert() 验证返回值是否和预期一样。

1.5 编写Demo

下面以具体代码演示如何使用Mockito，代码有三个类，分别如下：

Person类：

```
public class Person {
    private int id;
    private String name;
    public Person(int id, String name) {
        this.id = id;
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

PersonDao类：

```
public interface PersonDao {
    Person getPerson(int id);
    boolean update(Person person);
    boolean add(Person person);
}
```

PersonService类：

```
@Service
public class PersonService {
    private final PersonDao personDao;
    public PersonService(PersonDao personDao) {
        this.personDao = personDao;
    }
    public boolean update(int id, String name) {
        Person person = personDao.getPerson(id);
        if (person == null) {
            return false;
        }
        Person personUpdate = new Person(person.getId(), name);
        return personDao.update(personUpdate);
    }
        public Result<Boolean> add(Person person) {
            Person p = personDao.getPerson(person.getId());
            if (p != null) {
                return Result.fail("");
            }
            Boolean flag = personDao.add(person);
            return Result.condition(flag);
        }
}
```

Mock测试类：

```
public class PersonServiceTest {

    private PersonDao     mockDao;
    private PersonService personService;

    @Before
    public void setUp() throws Exception {
        //PersonDao
        mockDao = mock(PersonDao.class);
        when(mockDao.getPerson(1)).thenReturn(new Person(1, "Person1"));
        when(mockDao.update(isA(Person.class))).thenReturn(true);

        personService = new PersonService(mockDao);
    }
        @Test
        public void testAdd() {
                //
                Person person = new Person(666, "");
                //
                Result<Boolean> result = personService.add(person);
                //
                Assert.assertTrue(result.isSuccess());
                Assert.assertNotNull(result.getData());
    }
    @Test
    public void testUpdate() throws Exception {
        //
        boolean result = personService.update(1, "new name");
        assertTrue("must true", result);
        //getPerson(1)
        verify(mockDao, times(1)).getPerson(eq(1));
        //update
        verify(mockDao, times(1)).update(isA(Person.class));
    }

    @Test
    public void testUpdateNotFind() throws Exception {
        boolean result = personService.update(2, "new name");
        assertFalse("must true", result);
        //getPerson(1)
        verify(mockDao, times(1)).getPerson(eq(1));
        //update
        verify(mockDao, never()).update(isA(Person.class));
    }
}
```

Mockito

```
@RunWith(SpringRunner.class)
```

```
@SpringBootTest
@AutoConfigureMockMvc
public class SalaryIncomeStatementControllerTest2 {
    @Autowired
    private MockMvc mockMvc;
    @MockBean
    private SalaryIncomeStatementRecordService mockRecordService;
    @MockBean
    private RedisService mockRedisService;
    @Autowired
    private WebApplicationContext webApplicationContext;
    private static final String  TENANT_ID =
"00ptbnd1594895868165716557492316";
    private static final String  AUTHORIZATION = "Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzeXN0ZW1GbGFnIjoic2Fhcy1jbGllbnQiLCJ1c2VySW5mbyI6eyJ1c2VyRnJvbSI6ImJvbmF
kZV9zYWFzX2hybV9mb3VuZGF0aW9uIiwicGhvbmVOdW1iZXIiOiIxODgyNjQ3NDUyNiIsImdlbm
RlciI6bnVsbCwib3BlbklkIjoiMDBwdGJuZDIyNDQxMTMxMTE2ODQ1NzY0MzM2MTExNjgiLCJhd
mF0YXIiOiIiLCJ1c2VyTmFtZSI6IuW8oOmUoeaWjCJ9LCJjbGllbnRJZCI6IjExOWFlNTY2ZGY4
NjRhNTRhMGMwMzkyNWU2N2VlNmM0IiwiZXhwaXJlIjo2MDQ4MDAsImNoYW5uZWwiOiJwYyIsInZ
lcnNpb24iOiIyLjAiLCJqdGkiOiIwM2JjMjQxMy1jNjlhLTQ4MTMtYTAwMC04MjQ5ZDAyMWY0OD
IiLCJ1c2VyS2V5IjoiMTg4MjY0NzQ1MjYjcGMjc2Fhcy1jbGllbnQifQ.
o9zCoQtk7v3sdU3vVzLgUTbUUwP20D1uIIvGx3Sme80";
    @Before
    public void setUp(){
        mockMvc =  MockMvcBuilders.webAppContextSetup
(webApplicationContext).addFilter((request, response, chain) -> {
                    MockHttpServletRequest mq = (MockHttpServletRequest)
request;
                    mq.addHeader(TENANT_ID, TENANT_ID);
                    mq.addHeader("authorization", AUTHORIZATION);
                    response.setCharacterEncoding("UTF-8");
                    chain.doFilter(request, response);
                },"/*").build();
        // mysqlId
        TenantContextHolder.setTenantId(TENANT_ID);
        //
        TenantDataSourceProvider tenantDataSourceProvider =
webApplicationContext.getBean(TenantDataSourceProvider.class);
        tenantDataSourceProvider.pushDs(TENANT_ID);
        // mongodb
        TenantMongoContextHolder.push(TENANT_ID);
    }

    @After
    public void tearDown(){
        SaasTenantCut.clear();
    }

    /**
     * // 1
     * // 2
     * // 3
```

```
     * @throws Exception
     */
    @Test
    public void testCreateIncomeStatement() throws Exception {
        // 1
        // Setup
        // Configure SalaryIncomeStatementRecordService.
createIncomeStatementController(...).
        final SalaryIncomeStatementRecord vo = new
SalaryIncomeStatementRecord();
        vo.setTenantName("001");
        vo.setTemplateId(1702980416418349056L);
        vo.setResultTableName("resultTableName");
        vo.setDataFilter(new ArrayList<>());
        vo.setCycleDateType(0);
        vo.setCycleDate("2023-09");
        vo.setPaySubjectId("");
        vo.setPaySubjectName("");
        vo.setSalaryType(1);
        when(mockRecordService.createIncomeStatementController
(AUTHORIZATION, vo).isSuccess()).thenReturn(Result.data(null).isSuccess());

        //
        final MockHttpServletResponse createIncomeStatementResponse =
mockMvc.perform(post("/hrm-salary/incomeStatementRecord/v1.0
/createIncomeStatement").content(JSONUtil.toJsonStr(vo))
                                .contentType(MediaType.APPLICATION_JSON)
                                .accept(MediaType.APPLICATION_JSON))
                .andReturn().getResponse();

        // Verify the results
        assertThat(createIncomeStatementResponse.getStatus()).isEqualTo
(HttpStatus.OK.value());
        assertThat(JSONUtil.toBean(createIncomeStatementResponse.
getContentAsString(), Result.class).isSuccess()).isEqualTo(Result.data
(null).isSuccess());
        //
        final SalaryComplexDetailedListDTO dto  = new
SalaryComplexDetailedListDTO();
        dto.setConfId(1702208089312612352L);
        dto.setPageNum(1);
        dto.setPageSize(10);
        // 2
        final MockHttpServletResponse detailedListResponse = mockMvc.
perform(
                        post("/hrm-salary/salaryComplexConf/v1.0
/detailedList").content(JSONUtil.toJsonStr(dto))
                                .contentType(MediaType.APPLICATION_JSON)
                                .accept(MediaType.APPLICATION_JSON))
                .andReturn().getResponse();

        // Verify the results
        assertThat(detailedListResponse.getStatus()).isEqualTo(HttpStatus.
```

```
OK.value());
        Result<SalaryComplexDetailedListVO> result = JSONUtil.toBean
(detailedListResponse.getContentAsString(), Result.class);
        //
        assertThat(result.isSuccess()).isEqualTo(Result.data(null).
isSuccess());
        //
        assertThat(result.getData().getCalculateFlag()).isEqualTo(Boolean.
TRUE);
        //
    }
}
```

处理mvm test运行久的问题，优化编写内容:

每个业务模块指定BaseTest来作为基类，由编号A0(第一个)的Controller或者Serivce测试类继承BaseTest类。其他Controller或者Serivce测试类都通过加A+序号前缀来传递继承来实现，最终由AxxxStartApplication启动。

顺序：BaseTest------>A1------>A2------>A3------>A4------>A5------>AxStartApplication

注意：BaseTest和StartApplication是一比一的关系，不同的人写的不同业务测试类，可以定义自己的BaseTest和StartApplication

1.其他Controller或者Serivce测试类必须声明为abstract(抽象)类，可参照下列的事例代码。

2.AxxxStartApplication------这个是启动类，需要继承最后一个Controller或者Serivce测试类，可参照下列的事例代码。

3.A0BaseTest------这个是用来做基类的参数定义，可参照下列的事例代码。

4.项目路径不能带有中文，否则会导致无法输出本地的jacoco报告。

步骤1:

在BaseTest类上声明一些通用的变量，以便后面的测试类使用。

```
import org.mockito.Mock;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.web.context.WebApplicationContext;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.get;


public abstract class A0BaseTest {

    @Autowired
    public MockMvc mockMvc;

    @Autowired
    public WebApplicationContext webApplicationContext;

    @Mock
    public AutoCloseable mockitoCloseable;
    public static final String TENANT_ID = "";
    public static final String AUTHORIZATION = "Bearer ";

}
```

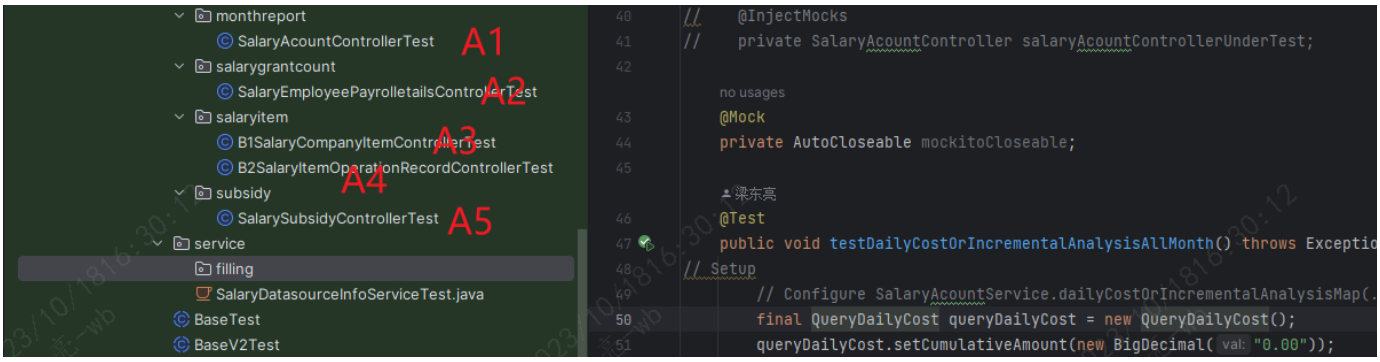
步骤2:

每个模块的每个业务controller类加前缀A+序号的类

其他类按照序号加前缀比如A2，A3，并且序号往后的类继承序号前的类。例如：A2xx extends A1xx, A3extends A2xx ,

A2继承A1

这几个小模块,大模块方式继承!



步骤3：进行测试，启动AxxxStartApplication类，会执行所有继承的全部类。

如果只启动单个文件，则单个文件按原来的写法，AxxxStartApplication直接继承此类即可。

```
import com.bonade.boot.common.context.TenantContextHolder;
import com.bonade.ious.client.controller.trade.
C9WhiteBarTradeControllerTest;
import com.bonade.saas.tenant.dynamic.datasource.TenantDataSourceProvider;
import com.bonade.saas.tenant.dynamic.mongodb.TenantMongoContextHolder;
import org.junit.After;
import org.junit.Before;
import org.junit.runner.RunWith;
import org.springframework.boot.test.autoconfigure.web.servlet.
AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.mock.web.MockHttpServletRequest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;

/**
 *
 */
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class A1StartApplication extends C9WhiteBarTradeControllerTest {

    @Before
    public void setUp() {
        mockMvc = MockMvcBuilders.webAppContextSetup
(webApplicationContext).addFilter((request, response, chain) -> {
            MockHttpServletRequest mq = (MockHttpServletRequest) request;
            mq.addHeader("tenant-id", TENANT_ID);
            mq.addHeader("authorization", AUTHORIZATION);
            response.setCharacterEncoding("UTF-8");
            chain.doFilter(request, response);
        }, "/*").build();
        // mysqlId
        TenantContextHolder.setTenantId(TENANT_ID);
        //
        TenantDataSourceProvider tenantDataSourceProvider =
```

```
webApplicationContext.getBean(TenantDataSourceProvider.class);
        tenantDataSourceProvider.pushDs(TENANT_ID);
        // mongodb
        TenantMongoContextHolder.push(TENANT_ID);
    }

    @After
    public void tearDown() throws Exception {
        TenantDataSourceProvider tenantDataSourceProvider =
webApplicationContext.getBean(
                TenantDataSourceProvider.class);
        // mysqlId
        TenantContextHolder.clear();
        //
        tenantDataSourceProvider.clearAllDs();
        mockitoCloseable.close();
    }


}
```

参考代码：JUnit-demo.zip

当前验证效果如下

只13个启动类，整个过程只编译一次，开5个线程，从原来的20分钟缩减到8分钟以内。