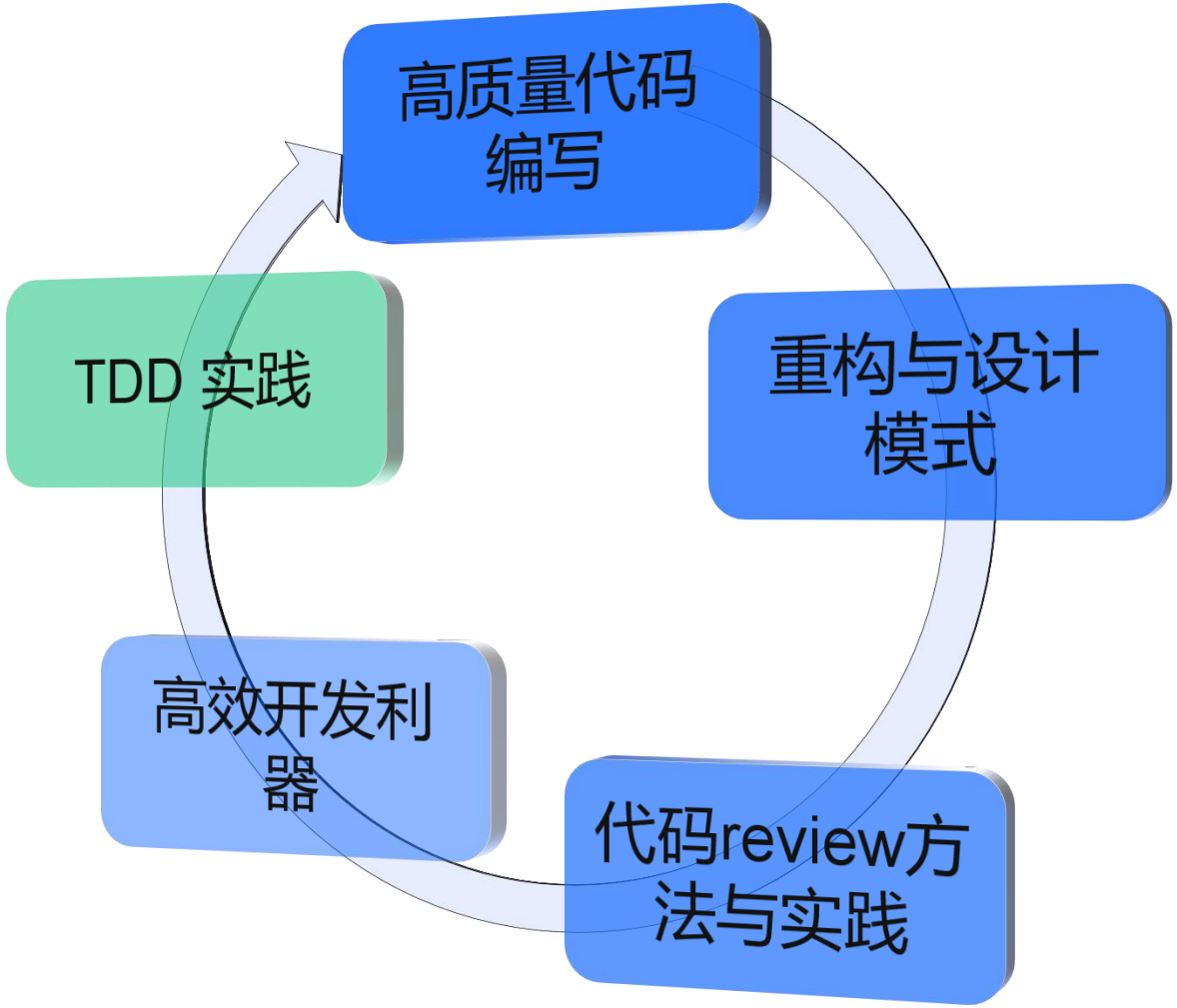
高效编码
之
TDD 实践

钟明星

2021 年 4月 28 日

高效编码

- 从理论到实践，从编程工具到代码编写、测试、review，全方面提升一线编码人员实际代码编写能力，提升编写的代码质量



课程目录

基本概念	• TDD 的特点 • TDD 的流程
使用框架	• Junit、Hamcrest、PowerMock • Spring Test
覆盖率与报表	• Idea Coverage • SonarQube、Jacoco
DDD 概念	• 领域驱动设计基本概念 • 领域驱动实践示例
项目落地	• 项目落地流程 • 实践案例

互动

你有做过单元测试吗?

举例

```
/**
 * 计算学生平均分
 **/
public double calculateScoresAvg(double totalScores, int totalCourses){
        return totalScores / totalCourses;
}
```

```
/**
 * 返回总分高于200的学生id
 **/
public List<Long> getStudentsCoursesPass(List<Student> students){
    return students.stream()
    .filter(st->st.getTotalPoints() >= 200)
    .map(Student::getId)
    .collect(Collectors.toList());
}
```

```
/**
 * 修改学生的姓名
 **/
public int updateStudentName(Long studentId, String name) {
    Student student = studentMapper.selectById(studentId);
    student.setName(name);
    return studentMapper.updateById(student);
}
```

TDD 简介

Test Driven Develop

1. TDD 的特点
2. TDD 的流程

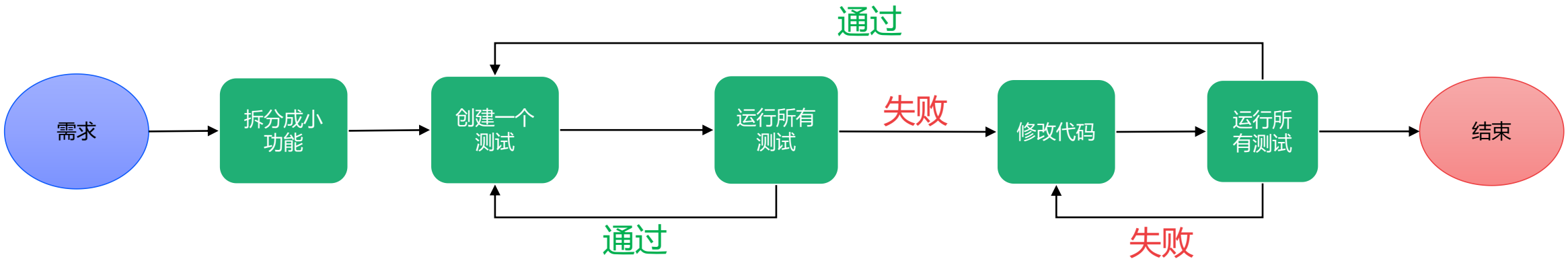
遇到的问题

- 测试人员漏测
- 新人业务不熟悉产生 Bug
- 老人疏忽产生 Bug
- 重构困难且风险很大

TDD——定义

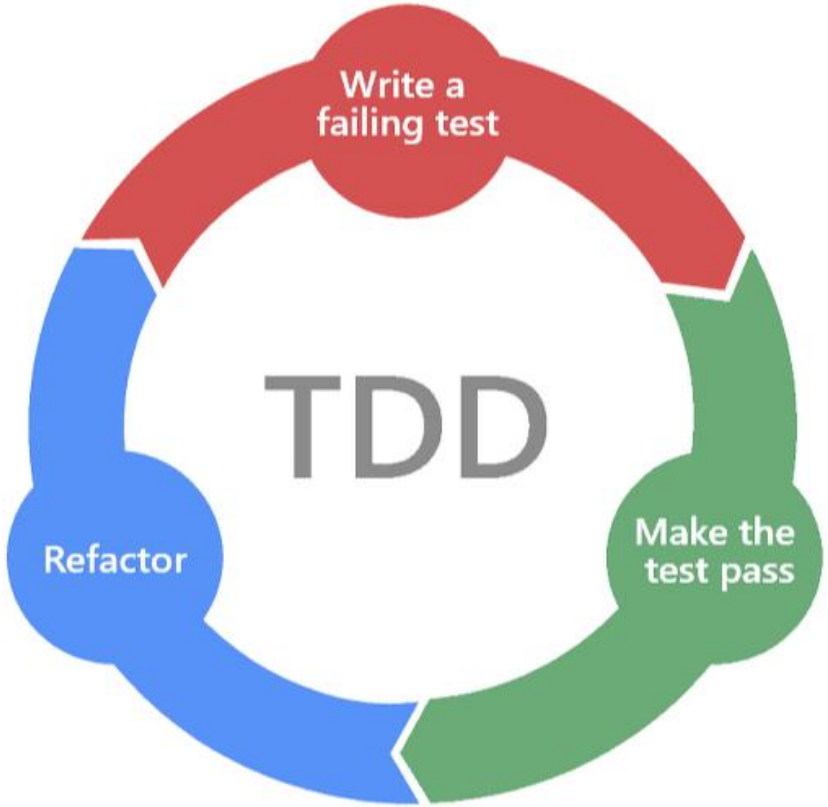
- TDD 是一个软件开发过程
- 将需求转换为具体的测试用例
- 非常短的开发周期
- 对代码进行改进，以使测试通过
- 测试用例的不断演进，完善出符合要求的代码

TDD——流程



TDD——好处

1. 降低了 Bug 的发生率和严重性
2. 测试倒逼设计
3. 测试用例是一个健全的 API
4. 当修改老功能时，测试会确保代码正确性



了解 TDD 理念，我们还需了解常用的框架

框架

Junit 概述与特点

Junit 是一个开放源代码的 Java 单元测试框架用于编写和运行可重复的测试

特点:

- 接口方法的单元测试
- 自动判断执行结果，无需人为干预
- 每个单元测试独立，不影响其它测试方法

Junit 举例

```java
@RunWith(BlockJUnit4ClassRunner.class);
public class TestCase {

  @BeforeClass
  public static void setUpBeforeClass() throws Exception {
    //只执行只一次
  }

  @Before
  public void setUp() throws Exception {
    //每个测试方法前都执行一次
  }

  @Test
  public void test1() {
    assertEquals(1, 1);
  }

  @Test(timeout = 1000)
  public void test2() {
    assertEquals("a","b");
  }

  @Test(expected = xxxException.class)
  public void test3() {
      new ArrayList().get(0)
  }

  @After
  public void tearDown() throws Exception {
    //每个测试方法结束都执行一次
  }

  @AfterClass
  public static void tearDownAfterClass() throws Exception {
    //所有方法结束执行一次
  }
}
```

Junit 注解与生命周期

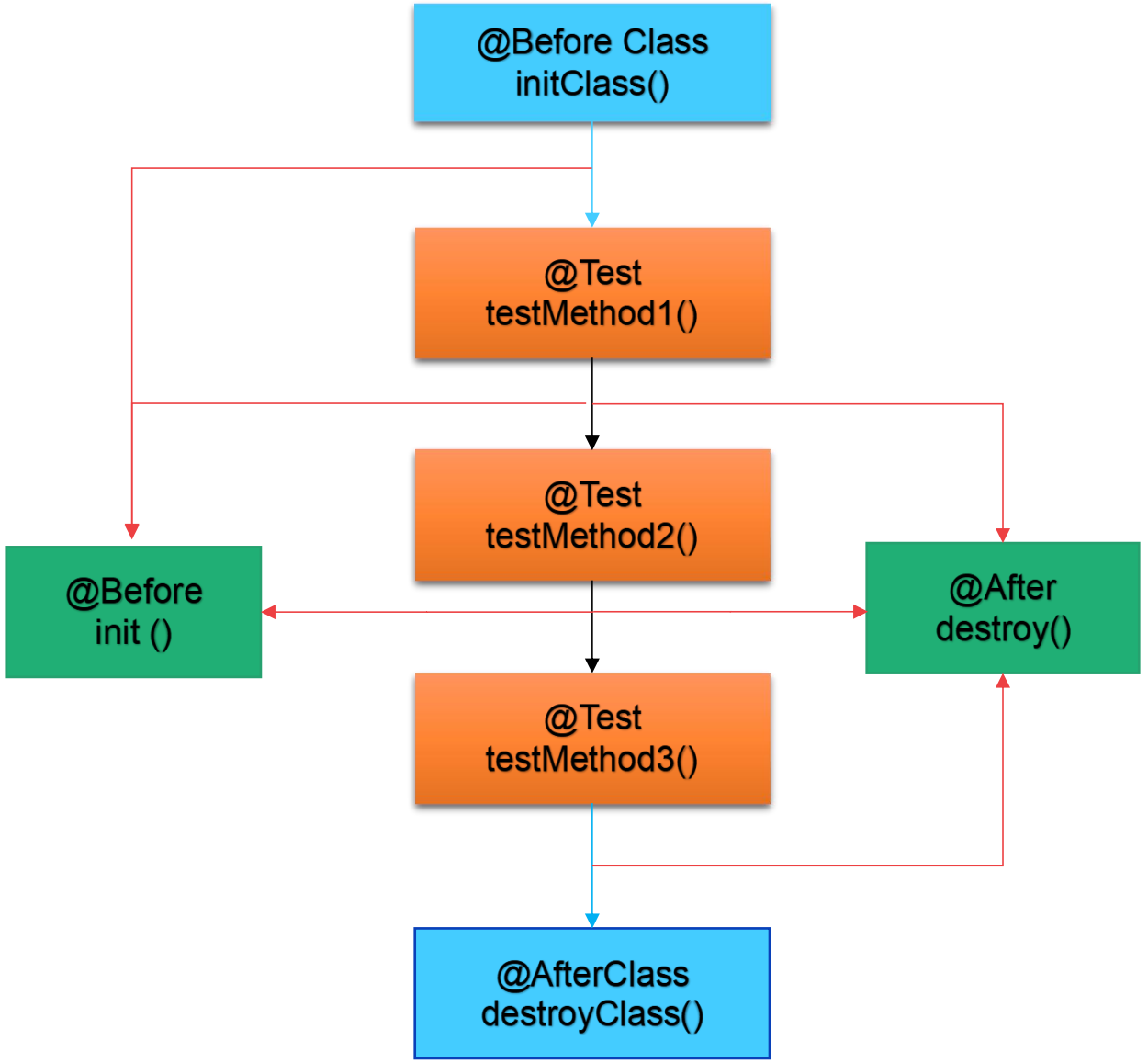
- Assertions 断言
  第一个参数为期望值
  第二个参数为实际值
  assertEquals("text", "text");

- @RunWith 单元测试运行器
  @RunWith(SpringRunner.class)
  @RunWith(PowerMockRunner.class)

- @Test
  表示可执行的测试方法

- @Test(timeout=1000)
  测试方法执行超过 1 秒会失败

- @Test(expected = xxxException.class)
  异常测试



Junit 结论概述

通过输入得到输出，并断言输出是我们期望的结果

了解 Junit，前边示例的测试现在是不是能写出来了呢?

Junit 举例1

计算平均分方法

```
2   /**
3    * 计算学生平均分
4    **/
5   public double calculateScoresAvg(double totalScores, int totalCourses){
6           return totalScores / totalCourses;
7   }
```

Junit 测试代码

```
public class StudentServiceTest {
    @Test
    public void testCalculateScoresAvg() {
      assertEquals(20, calculateScoresAvg(40, 2));
    }

    @Test(expected = ArithmeticException.class)
    public void testCalculatePointsAvgWithZero() {
      assertEquals(3, new Calculate().divide(9, 0));
    }
  }
```

异常情况测试

输入：40，2
输出：calculatePointsAvg(40, 2) = 20
期望的结果：20
判断结果相等断言： assertEquals

Junit 举例2

返回学生集合

```java
/***
 * 返回总分高于200的学生id
 */
public List<Long> getStudentsCoursesPass(List<Student> students){
    return students.stream()
    .filter(st->st.getTotalPoints()>=200)
    .map(Student::getId)
    .collect(Collectors.toList());
}
```

Junit 测试代码

```java
@Test
public void testCalculatePointsAvg(){
  Student s1=new Student(1,200);
  Student s2=new Student(2,150);
  Student s3=new Student(3,210);
  students=new ArrayList<Student>();
  students.add(s1);
  students.add(s2);
  students.add(s3);
  //测试getStudentsCoursesPass方法
  List<Long> studentIds = studentServiceImpl.getStudentsCoursesPass(students);
  //期望的结果是studentIds值为1和3
  //此处如何assert？Junit很难断言出结果，有没有方便的API？
}
```

如何断言复杂的结构
如 List, Map

Hamcrest 介绍

- Hamcrest 是一个以测试为目的，且能组合成灵活表达式的匹配器类库
- 它解决了 Junit 应对复杂断言的困扰

Hamcrest Junit 书写对比

运用 Hamcrest 使断言更简洁丰富

Hamcrest 断言	Junit 断言

`List<Integer> list = Arrays.asList(5, 2, 4);`
`assertThat(list, everyItem(greaterThan(1)));`
	`List<Integer> list=Arrays.asList(5,2,4);`
	`for (Integer result:list) {`
	`  assertTrue(result>1);`
	`}`

`String stringToTest = "";`
`assertThat(stringToTest, blankOrNullString());`
	`String stringToTest = "";`
	`assertNull(stringToTest);`
	`assertEquals("",  stringToTest)`

`Todo todo1=new Todo(1,"Learn Hamcrest","Important");`
`Todo todo2=new Todo(1,"Learn Hamcrest","Important");`
`assertThat(todo1,samePropertyValuesAs(todo2));`
	`assertEquals(todo1.getId(),todo2.getId())`
	`assertEquals(todo1.getName(),todo2.getName())`
	`assertEquals(todo1.getImp(), todo2.getImp())`

`assertThat(list, containsInAnyOrder(2, 4, 5));`	`assertTrue(list.contains(4));`
	`assertTrue(list.contains(2));`
	`assertTrue(list.contains(5));`

互动

```java
/**
 * 修改学生的姓名
 **/
public int updateStudentName(Long studentId, String name) {
    Student student = studentMapper.selectById(studentId);
    student.setName(name);
    return studentMapper.updateById(student);
}
```

PowerMock 解决的问题

PowerMock 是 Java 开发中的一种 Mock 框架，用于单元模块测试

解决的问题：

- 测试一个需要认证的 Service 接口
- 分模块开发时且进度不同，对方只需提供接口，就能模拟获取数据
- 减少外部类、系统和依赖给单元测试带来的耦合
- 可实现完成对 private/static/final 方法的 Mock

PowerMock 常见用法

- 通过注解 @Mock 模拟出一个实例
- Mock 模拟调用第三方接口
- Spy局部 Mock，返回当前类对象调用方法结果

```
31  @RunWith(PowerMockRunner.class)
32  public class TeacherServiceImplTest{
33      @Mock
34      private StudentServiceImpl studentServiceImpl;
35
36      @Test
37      public void testTeacherMock(){
38          when(studentServiceImpl.getSomeMethod()).thenReturn(result);
39          ...
40          //assert做断言
41      }
42  }
```

```
46  public class TeacherServiceImpl{
47      public boolean getResult(){
48          boolean resultTrue = isLongMethod();
49          // resultTrue =  needInitMethod();
50          // resultTrue =  justWantGetResultMethod();
51          if(resultTrue){
52              resultTrue = isBadScoreStudent();
53          }
54          return resultTrue;
55      }
56  o
57      protected boolean isLongMethod(){
58          //1000行代码
59      }
60
61      protected boolean needInitMethod(){
62          //初始化 client
63      }
64
65      protected boolean justWantGetResultMethod(){
66          //只想这个方法发挥true
67      }
68
69      private isBadScoreStudent(){
70          ...
71      }
72
73  }
```

```
48  @RunWith(PowerMock.class)
49  public class TeacherServiceImplTest{
50      @Mock
51      TeacherServiceImpl teacherServiceImpl;
52
53      @Test
54      public void testIsBadScoreStudent(){
55          TeacherServiceImpl teacherServiceImplSpy = PowerMockito.spy(teacherServiceImpl);
56          when(teacherServiceImplSpy.isLongMethod()).thenReturn(true);
57          boolean result = teacherServiceImplSpy.getResult();
58          assertTrue(result);
59      }
60  }
```

PowerMock 举例

修改学生的姓名

```
3   public int updateStudentName(Long studentId,String name){
4       Student student=studentMapper.selectById(studentId);
5       student.setName(name);
6       return studentMapper.updateById(student);
7   }
```

输入： id = 100L name = 测试
期望结果：学生name属性更新为测试，并返回成功1
问题：由于selectById与updateById需要访问数据库，具体结果是不可控的，所以此处需要使用mock

PowerMock 测试代码

```
3   @RunWith(PowerMock.class)
4   public class StudentServiceImplTest {
5     @Mock
6     StudentMapper studentMapper;
7     @Test
8     public void testUpdateStudentNameSuccessful() {
9       Student student = new Student();
10      student.setName("同学");
11      when(studentMapperMock.selectById(100L)).thenReturn(student);
12      when(studentMapperMock.updateById(student)).thenReturn(1);
13      int result = studentServiceImpl.updateStudentName(100L, "测试");
14      assertEquals(1, result);
15      assertEquals("测试", student.getName());
16    }
17  }
```

```
3   when(studentMapperMock.selectById(anyLong())).thenReturn(student);
4   when(studentMapperMock.updateById(anyObject())).thenReturn(1);
```

异常情况测试

是否存在其它情况需要测试?

```
2   @RunWith(PowerMock.class)
3   public class StudentServiceImplTest {
4     @Mock
5     StudentMapper studentMapper;
6     @Test
7     public void testUpdateStudentNameSuccessful() {
8       Student student = new Student();
9       student.setName("同学");
10      when(studentMapperMock.selectById(100L)).thenReturn(null);
11      when(studentMapperMock.updateById(student)).thenReturn(1);
12      int result = studentServiceImpl.updateStudentName(100L, "测试");
13      assertEquals(1, result);
14      assertEquals("测试", student.getName());
15    }
16  }
```

通过 Mock 返回空情况，会出现 NullPointException，测试失败

PowerMock 代码修改后测试

代码修改后

```
public int updateStudentName(Long studentId,String name){
    Student student=studentMapper.selectById(studentId);
     if(student==null){
       return 2;
      }
     student.setName(name);
     return studentMapper.updateById(student);
}
```

修改后的测试代码

```
@Test
public void test updateStudentNameResultNull(){
    Student student=new Student();student.setName(“同学”);
    when(studentMapperMock.selectById(100L)).thenReturn(null);
    when(studentMapperMock.updateById(student)).thenReturn(1);
    int result = studentService.updateStudentName(100L, “测试”);
    assertEquals(2,result);
    assertEquals(“同学”,student.getName());
}
```

Redis 模拟调用

场景：Redis 查询结果，并返回和 test 拼装的结果

```
15    public class Filter {
16      private String convertDataFormRedis(@NonNull String key) {
17        ...
18        String value = getValue(key);
19        ...
20        return "test" + value;
21      }
22
23      protected String getValue(String key) {
24        return AdxSystem.sysMgrModule().getGnomeByteCacheRedisClient().get(key);
25      }
26    }
```

输入：key = **lowuv_0_IDFA_C7FD1C8264FC43A11F4F3685C2DB047F**
输出：test1

```
8   public void filter()throws Exception{
9       Filter spyFilter=PowerMockito.spy(filter);
10      sessInfo.key="lowuv_0_IDFA_C7FD1C8264FC43A11F4F3685C2DB047F";
11      doReturn("1").when(spyFilter,"getValue","lowuv_0_IDFA_C7FD1C8264FC43A11F4F3685C2DB047F");
12      String result=convertDataFormRedis(sessInfo);
13      assertEquals("test1",result)
14  }
```

Spring 中 Mock RedisTemplate

```
6   RedisTemplate redisTemplate = Mockito.mock(RedisTemplate.class);
7   HashOperations hashOperations = redisTemplate.opsForHash();
8   ……
```

静态方法单例调用

外部静态方法的调用 Mock：

外部静态方法
Mock

```
3   @Test
4   public void testUpPlatOtherPlatToken(){
5       PowerMockito.when(CacheUtils.getOtherPlatToken(269)).thenReturn(null);
6       boolean result=upPlatPddDsp.handlerReq(sessInfo,upPlatInfo);
7       assertFalse(result);assertThat(upPlatInfo.getErrInfos(),
8       hasItem(PlatErrInfo.REQUEST_ERR_TOKEN_NULL));
9   }
```

单例对象的 Mock：

单例模式方法
Mock

```
3   PlatMipMapHandler instanceMock=PowerMockito.mock(PlatMipMapHandler.class);
4   Whitebox.setInternalState(PlatMipMapHandler.class,"instance",instanceMock);
5   when(instanceMock.useMapStrategy(sessInfo,182)).thenReturn(true);
6   Integer winnerId=platHandler.optimize.WinId(sessInfo,winnerBidder);
7   assertNull(winnerId);
```

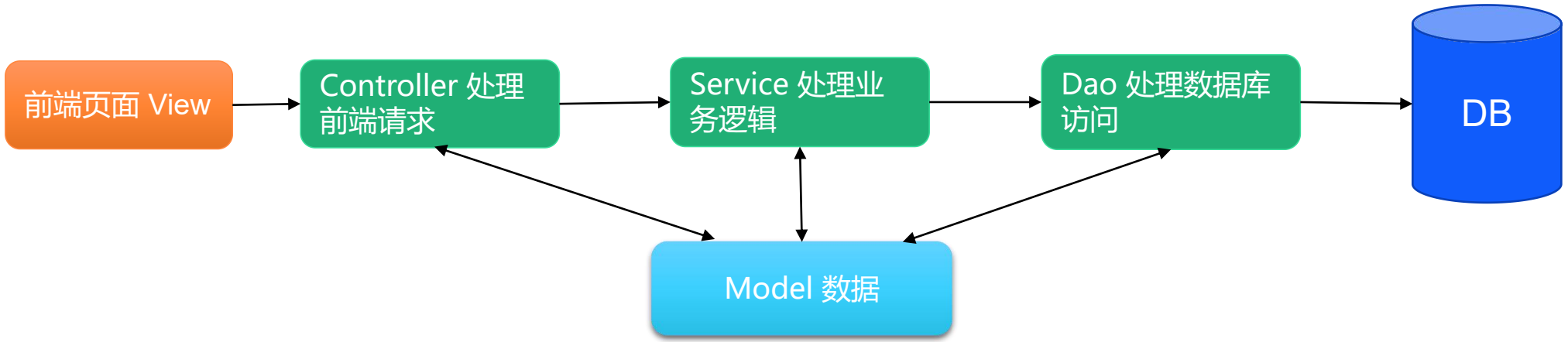
private final static模拟

```
123  private class Student{
124      private static final Gson GSON = new GsonBuilder().disableHtmlEscaping().create();
125  }
126
```

```
127  private class StudentTest{
128      @Mock
129      Gson gson;
130
131      @Before
132      public void setUp(){
133          PreTestInfoUtils.INSTANCE.setFinalStatic(Student.class.getDeclaredField("GSON"),gson);
134      }
135
136      //由于GSON是private static final 普通mock不起作用需要运用反射，得到mock对象gson
137      public void setFinalStatic(Field field,Object newValue)throws Exception{
138          field.setAccessible(true);
139          Field modifiersField=Field.class.getDeclaredField("modifiers");
140          modifiersField.setAccessible(true);
141          modifiersField.setInt(field,field.getModifiers()&~Modifier.FINAL);
142          field.set(null,newValue);
143      }
144  }
```

Spring 项目如何测试?

Spring Test 解决了什么



Spring Test 是针对 Spring 项目测试提出的解决方案

- 单元测试，面向方法的测试
- 集成测试，面向整体业务的测试
- 切面测试

主要功能：

- 测试运行环境，通过 @RunWith 和 @SpringBootTest 启动 Spring 容器
- Mock 能力，Mockito 提供 Mock 能力
- 断言能力，AssertJ、Harmcrest、JsonPath 提供断言能力

Restful 接口测试

```java
@ApiOperation(value = "查询学生信息", notes = "查询学生信息")
@PostMapping(value = "/commom/getStudent")
public ResponseDto<Student> getStudent(Integer id) {
    return ResponseDto.success(studentServiceImpl.getStudentById(id));
}
```

```java
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class StudentControllerTest {
    @Autowired
    private MockMvc mockMvc;
    private MockHttpSession session;

    @Before
    public void setup() {
        session = new MockHttpSession();
    }

    @Test
    public void testGetUsablePlat() throws Exception {
        // 登陆
        MockHttpServletRequestBuilder loginRequestBuilder =
            MockMvcRequestBuilders.post("/common/getStudent")
            .param(id, 100L)
            .accept(MediaType.parseMediaType("application/json;charset=UTF-8"))
            .session(session);

        mockMvc.perform(loginRequestBuilder)
                .andExpect(MockMvcResultMatchers.status().isOk())
                .andExpect(jsonPath("$.name", is("测试")));
    }
}
```

当希望能够通过输入 URL 对 Controller 进行接口测试，如果通过启动服务器，建立 Http client进行测试，这样会使得测试变得很麻烦，比如，启动速度慢，测试验证不方便，依赖网络环境等，所以为了可以对 Controller 进行测试，Spring 引入了 MockMvc

Service 层测试

Spring 提供的
Runner（运行环境）

通过 MockBean 注解
Mock 对象

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class StudentServiceImplTest {
  @Autowired
  StudentServiceImpl studentServiceImpl;
  @MockBean
  private StudentMapper studentMapperMock;

  @Test
  public void testUpdateStudentNameResultNotNull() {
    Student student = new Student();
    student.setName("同学");
    Mockito.when(studentMapperMock.selectById(100L)).thenReturn(student);
    Mockito.when(studentMapperMock.updateById(student)).thenReturn(1);
    int result = studentServiceImpl.updateFatherAge(100L,"测试"); assertEquals(1, result);
    assertEquals("测试", student.getName());
  }
}
```

Dao 层测试

```
2   @RunWith(SpringRunner.class)
3   @SpringBootTest
4   @Transactional
5   public class StudentMapperTest {
6     @Autowired
7     private StudentMapper studentMapper;
8
9     @Test
10    @Rollback
11    public void testInsertStudent() {
12      Student student = new Student();
13      student.setName("测试");
14      studentMapper.insert(student);
15      LambdaQueryWrapper<Student> lambdaQuery = new LambdaQueryWrapper<>();
16      lambdaQuery.eq(Student::getName, "测试");
17      Student studentResult = studentMapper.selectOne(lambdaQuery);
18      assertEquals("测试", student.getName());
19    }
20  }
```

Dao 层通过事务
防止测试数据污染
测试环境

Dao 层测试数据的
CIUD 操作，并通过
Rollback 将数据及时
回滚

测试书写建议

- TDD 流程任何操作之前建议先创建 Test case
- 避免编写的 Test 对公用测试库和环境有影响
- 有意义的 Test case 名称 举例遵循包名和类名方法名规则
- 最大化的断言 (Assert) 如引入 Hamcrest
- Mock 数据是很有必要的，如测试 Redis，测外部依赖接口
- Test case 尽可能的小，单一变量原则
- 测试覆盖率不低于 90%，关键模块 100%

引言

如果关注质量，那么长期来看质量会提升，成本会降低

如果关注成本，那么长期来看成本会提升，质量会降低

世界著名的质量管理专家 Edwards.Deming

如何监督与检测?

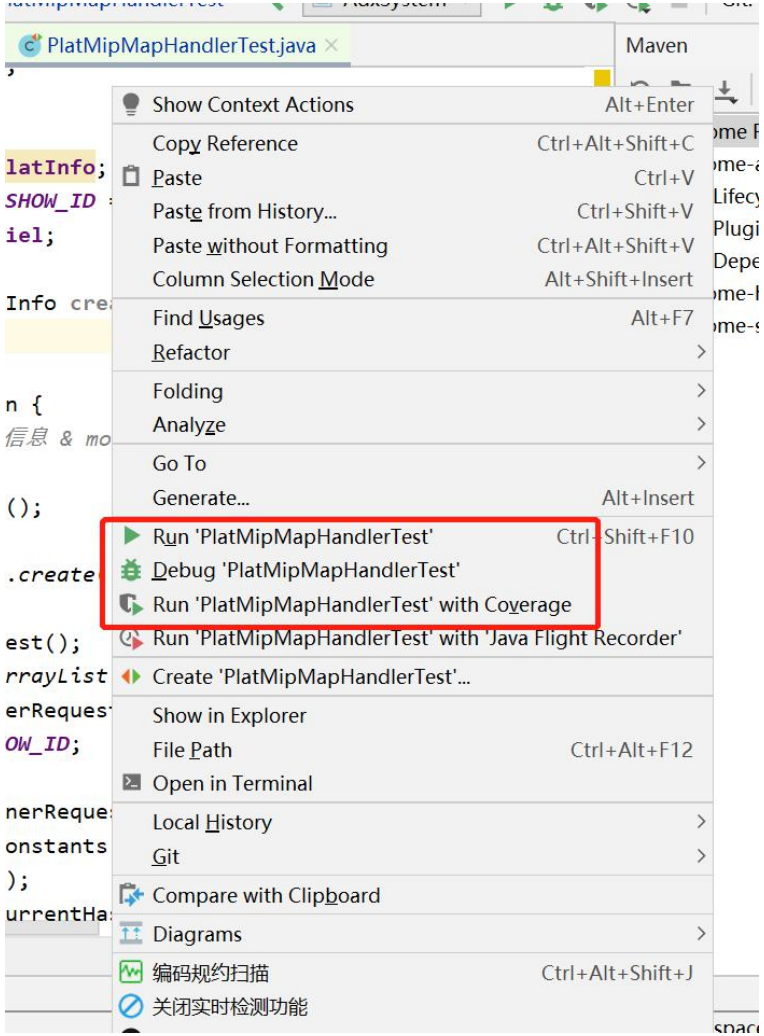
PART 03

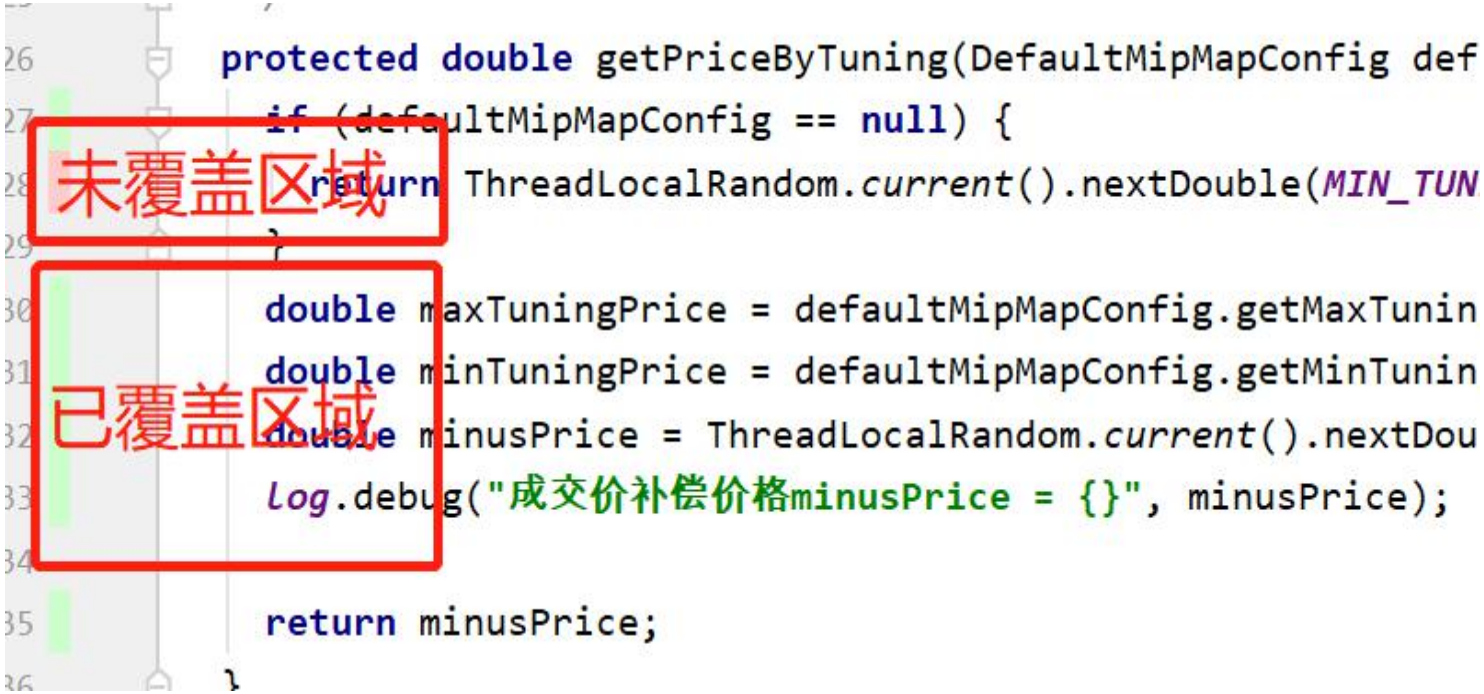
检测工具与报表

1. Idea Coverage
2. SonarQube、 Jacoco

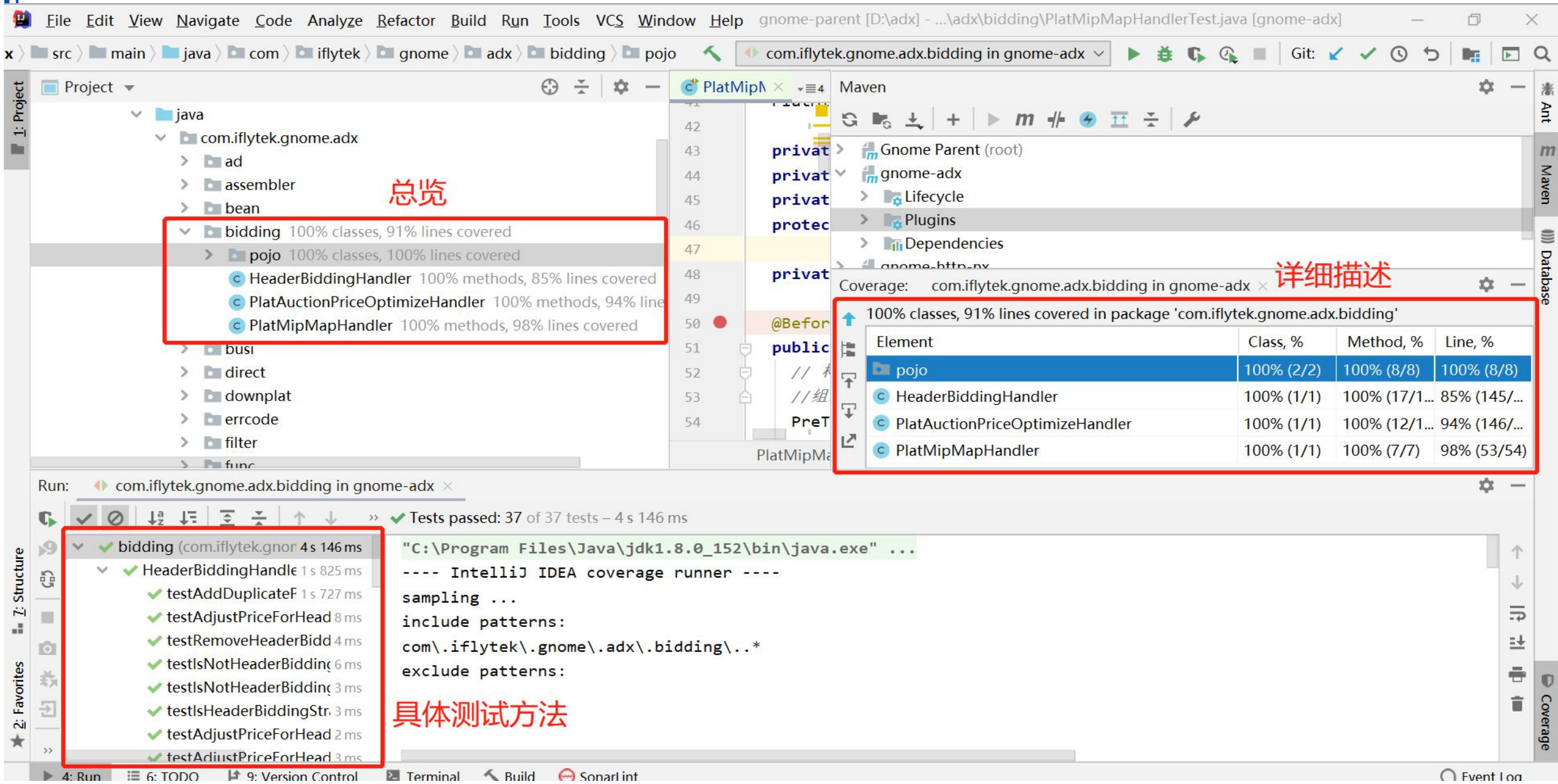
Idea coverage检测工具

- Idea 自带测试覆盖率工具可帮助研发快速定位覆盖率问题





Idea Coverage覆盖率检测

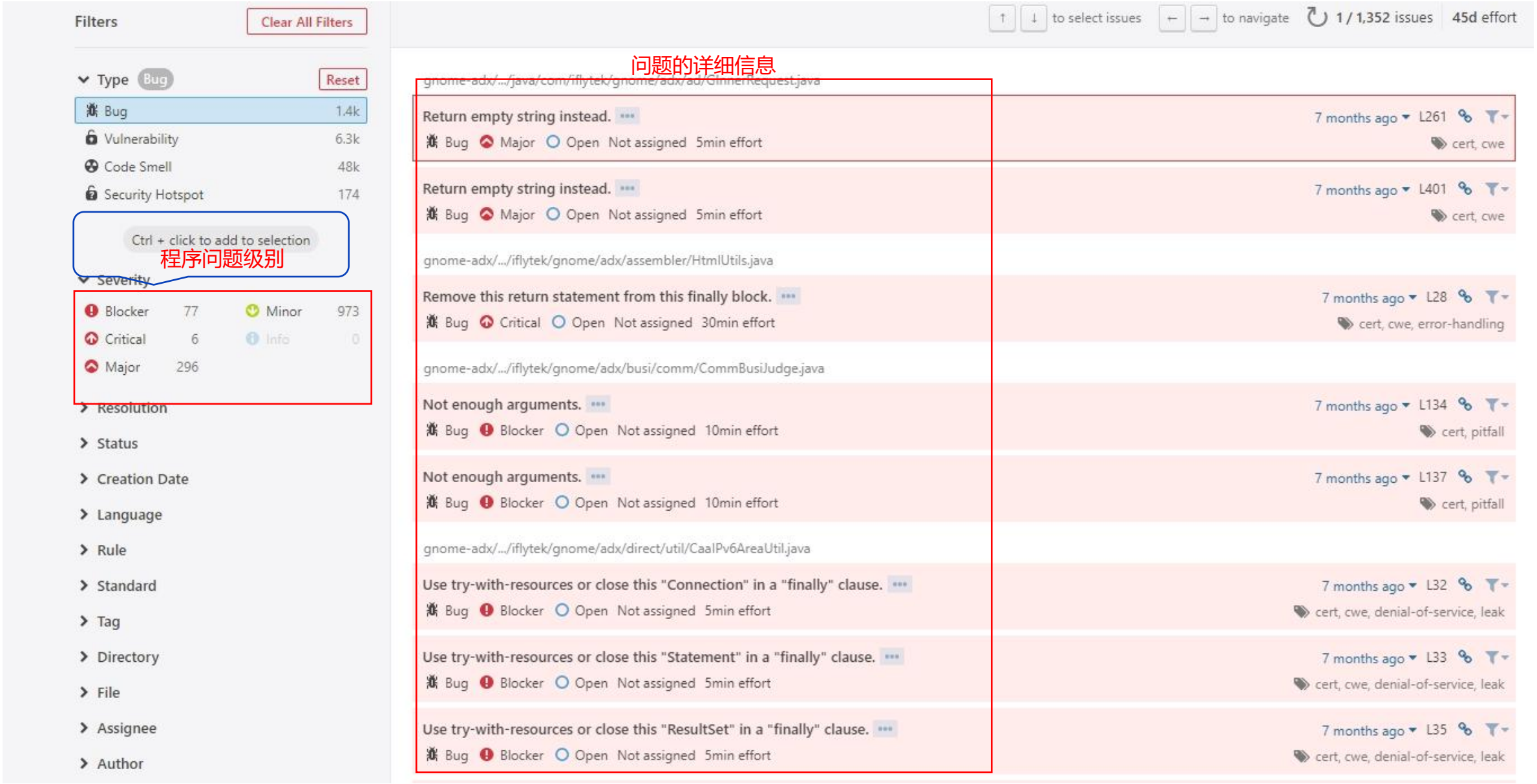


SonarQube 集成与检测

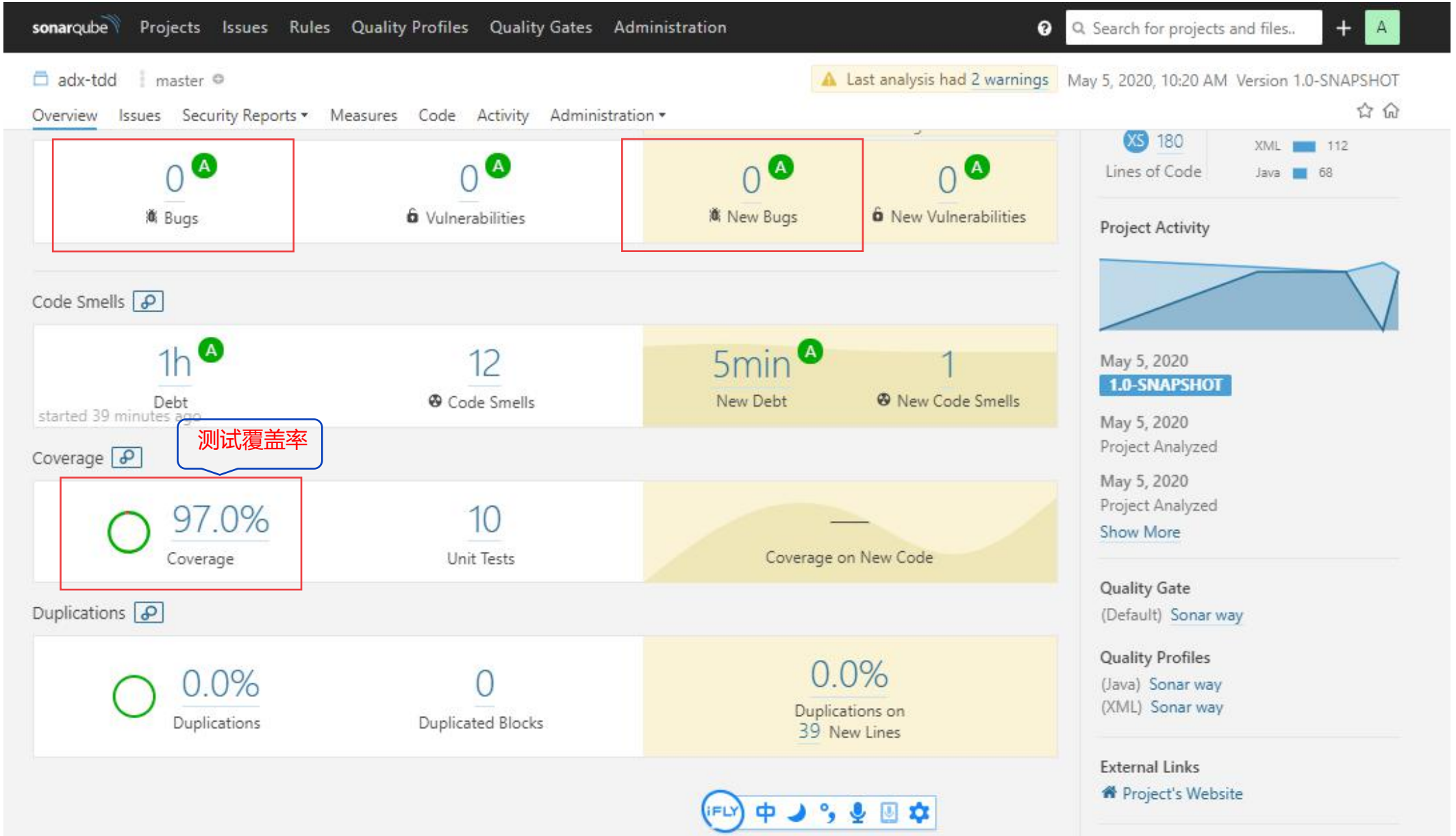
- SonarQube 是一个用于代码质量管理的开源平台,用于管理源代码的质量

- 评判维度
  - 可靠性：码中的语法是否合规，是否有语法逻辑错误或是一些不推荐使用的方法，主要用 Bugs 来衡量
  - 安全性：项目代码是否有潜在的“可被攻击”的漏洞
  - 代码重复率：代码是否精简，将一些通用的类或方法进行封装
  - 单元测试覆盖率 需要集成 Jacoco：完善的单元测试可以使代码更加可靠，也可以减少很多后续的风险呢，SonarQube 用 Coverage 来考量单元测试覆盖率

SonarQube 集成检测



SonarQube 测试报表



Jacoco 覆盖率工具

- Jacoco 是一个开源的覆盖率工具，针对 Java 语言
  - Jacoco 包含了多种尺度的覆盖率计数器,包含指令级 分支、行、方法、类
  - SonarQube 测试覆盖率报表需使用 Jacoco 插件

report

report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.hundsun.model		10%		9%	2,897	3,493	4,564	5,402	2,068	2,637	18	45
com.hundsun.service		22%		11%	287	338	469	633	51	88	4	18
com.hundsun.util		45%		43%	298	426	509	1,027	58	129	1	12
com.hundsun.service.http		56%		30%	85	126	134	295	20	61	10	31
com.hundsun.service.check		51%		41%	48	80	65	171	0	16	0	1
com.hundsun.model.dto		29%		50%	123	178	193	268	118	172	8	24
com.hundsun.service.combine		77%		67%	77	167	96	421	4	42	0	8
com.hundsun.service.listener		63%		35%	15	30	29	76	8	23	1	7
com.hundsun.model.enums		0%		n/a	8	8	15	15	8	8	1	1
com.hundsun.exception		10%		n/a	8	9	15	17	8	9	1	2
com.hundsun.configuration		97%		87%	5	53	9	155	4	49	0	7
com.hundsun		100%		n/a	0	2	0	3	0	2	0	1
Total	29,863 of 40,221	25%	2,548 of 3,283	22%	3,851	4,910	6,098	8,483	2,347	3,236	44	157

测试倒逼设计?

DDD 领域驱动模型

DDD 领域驱动设计—引入

领域驱动设计：为解决场景下的问题而形成的一套模型，然后使用这套模型来解决业务问题，根据重复劳动经验我们会形成一套模式

当前模式遇到的困境是什么？当前架构在测试方面有什么困难？

DDD设计

主要困境：

在 Service 层通常我们非常喜欢用 Service去管理大部分的逻辑，Model 作为数据在 Service 层不停地变换和组合，Service 层在这里是一个巨大的加工工厂

DDD 设计：

每个领域都是完成属于自己应有的行为，领域模型并不完成业务，Service 层就是基于这些模型做的业务操作（代码不再那么臃肿，很多动作交给了领域对象去处理）

假如有学生和老师这两个表，生成Model

```
3   public class Teather {
4       ……
5       private String name;
6       private int age;
7       List<Long> studentIds;
8       ……
9   }
10
11  public class Student {  ……
12      private String name;
13      private int score;
14      List<Long> teacherIds;  ……
15  }
```

传统架构例子

学校准备冲刺，希望有经验的老师教导成绩差的学生，Service 通常这么做

```
3   public class TeacherServiceImpl {
4       public Teacher techerStudent(Teacher teacher, Student student) {
5         //首先要判断是有经验的老师
6         if (!isExperienceTeacher(teacher)) {
7           return null;
8         }
9         if (!isBadScoreStudent(student)) {
10          return null;
11        }
12        teacher.getStudentIds().add(student.getId());
13        ……
14      }
15
16      private boolean isExperienceTeacher(Teacher teacher) {
17        return teacher.getAge() > 40; //老师的年龄必须大于40岁
18      }
19
20      private boolean isBadScoreStudent(Student student) {
21        return student.getScore () <60; //学生的分数必须小于60分
22      }
23    }
```

传统架构例子

有另外一个 Service 成绩差学生要感谢额外培训的老师

```
3   public class StudentServiceImpl {
4       public Student thanksTeacher(Student student) {
5         if (!isBadScoreStudent(student)) {
6           return null;
7         }
8         for (Long id : student.getTeacherIds()) {
9           if (!isExperienceTeacher(teacher)) {
10            continue;
11          }
12          list.add(id)
13        }
14        sendFlower(list);
15        ……
16      }
17
18      private boolean isExperienceTeacher(Teacher teacher) {
19        return teacher.getAge() > 40; //老师的年龄必须大于40岁
20      }
21
22      private boolean isBadScoreStudent(Student student) {
23        return student.getScore()<60; //学生的分数必须小于60分
24      }
25    }
```

对于复杂的业务，存在别的 Service 同样存在验证与老师的查询操作

传统架构测试

这里，Service 充当了管理的角色，什么事都得他去做
当我们对这些 Service 写测试时，每个 Service 都需写验证老师与学生的操作的分支

```
3   public class TeacheServiceImplTest {
4       @Test
5       private void testIsNotExperienceTeacher() {
6         Teacher teacher = new Teacher();
7         teacher.setAge(21);
8         Student student = new Student();
9         student.setid(1);
10        student.setScore(39);
11        Teacher teacher = teacherService.techerStudent(Teacher teacher, Student student)
12        assertFalse(teacher.getStudentids,  hasItem(1));
13      }
14    }
```

测试是否是有经验的老师

当teacher的age>40时，该断言仍然成功

问题：

1. 通过断言 teacher.getStudentIds 不能确保该方法是老师验证被过滤还是学生验证被过滤
2. 当 techerStudent 方法前又加入了其它验证过滤，使断言的准确性更差
3. 这段验证测试代码会出现在不同 Service 的测试类里

DDD编写案例

如果用 DDD 编写

```
2   public class Teacher {
3       ……
4       public boolean isExperienceTeacher() {
5           return teacher.getAge() > 40; //老师的年龄必须大于40岁
6       }
7   }
8
9   public class Student {
10      ……
11      public boolean isBadScoreStudent() {
12          return socre < 60; //学生的分数必须小于60分
13      }
14      public List<String> getSendFlowerTeacher() {
15          for (Long id : this.getTeacherIds()) {
16              teacher = mapper.getteacher(id);
17              if (teacher.isExperienceTeacher()) {
18                  continue;
19              }
20              list.add(id);
21          }
22          return list;
23      }
```

DDD例子 1

原 Service 测试

```
3  public class TeacherServiceImplTest {
4      @Test
5      private void testIsNotExperienceTeacher() {
6          Teacher teacher = new Teacher();
7          teacher.setAge(21);
8          Student student = new Student();
9          student.setid(1);
10         student.setScore(39);
11         Teacher teacher = teacherService.techerStudent(Teacher teacher, Student student);
12         assertFalse(teacher.getStudentids,  hasItem(1));
13     }
```

- 只能通过 techerStudent 调用测试是否是有经验的老师

使用 DDD 后测试

```
3  public class TeacherTest {
4      @Test
5      private void testIsNotExperienceTeacher() {
6          Teacher teacher = new Teacher();
7          teacher.setAge(21);
8          boolean result = teacher.isExperienceTeacher();
9          //直接测试
10         assertFalse(result);
11     }
12 }
```

- 书写简单
- 只测试一次
- 目的很明确，不受其它条件分支干扰
- 后期修改重构方便

DDD 例子 2

原代码逻辑

```java
public class TeacherServiceImpl{

    public List<Teacher> getFlowerTeacher(Teacher teacher, Student student){
      boolean isExperienceTeacher = isExperienceTeacher(teacher);
      boolean isBadScoreStudent = isBadScoreStudent(student);
      List<String> teachers= getTop2Teachers(student);
      if(isExperienceTeacher
         && isBadScoreStudent
         && Collections.isNotEmpty(flowerTeachers)){
         //测试该sendFlower方法
         List<String> flowerTeacher = sendFlower(flowerTeachers);
         return flowerTeacher;
      }
    }

    private boolean isExperienceTeacher(Teacher teacher) {
        return teacher.getAge() > 40;
    }

    private boolean isBadScoreStudent(Student student) {
        return socre < 60;
    }

    private List<String> getTop2Teachers(Student student) {
        Student students = studentMapper.selectById(student.getId());
        if(student == null){
            return null;
        }
        //其它逻辑代码
        List top2Teachers = getTop2Teachers(student);
        return top2Teachers;
    }

    private List<Teacher> sendFlower(List list){
        return list.stram()
        .map(t->t.setFlower(1))
        .collect(Collectors.toList());
    }
}
```

原测试代码

```java
@RunWith(PowerMock.class)
public class TeacherServiceImplTest{
    @Mock
    StudentMapper studentMapper;

    @Test
    public void testSendFlowerTeacher(){
        Teacher teacher = new Teacher();
        teacher.setAge(50);
        Student student = new Student();
        student.setId(10L);
        student.setScore(60);
        List Lists = new ArrayList<>();
        list.addAll(1L,3L);
        student.setTeacherIds(Lists);
        when(studentMapper.selectById(student.getId(10))).then(student);
        List<String> flowerTeachers = getFlowerTeacher(teacher,  student);
        assertThat(flowerTeachers, hasSize(2));
        assertThat(flowerTeachers, containsAnyOrder(1L, 3L));
    }
}
```

问题：

1：需要模拟正确的值，让 isExperienceTeacher，isBadScoreStudent = true 且 teachers 不为空时才能测到 sendFlower 方法分支。

2：任意一个方法修改后，都会影响当前测试结果，如有新增条件经验的老师必须是要有证书

3：实际项目中遇到的同样困境

DDD 重构后代码与测试

DDD 修改后代码

```
public class Teacher(){
    private Long id;
    private int age;
    private boolean isExperienceTeacher() {
        return age > 40;
    }
}
public class Student(){
    private Long id;
    private boolean isBadScoreStudent() {
        return socre < 60;
    }
    private List<String> getTop2Teachers() {
        Student students = studentMapper.selectById(id);
        if(student == null){
            return null;
        }
        //其它逻辑代码
        List top2Teachers = getTop2Teachers(student);
        return top2Teachers;
    }
}

public class TeacherServiceImpl{
    public List<String> getFlowerTeacher(Teacher teacher, Student student){
      boolean isExperienceTeacher = teacher.isExperienceTeacher();
      boolean isBadScoreStudent = student.isBadScoreStudent();
      List<String> teachers= student.getTop2Teachers();
      if(isExperienceTeacher
         && isBadScoreStudent
         && Collections.isNotEmpty(flowerTeachers)){
         //测试该sendFlower方法
         List<String> flowerTeacher = sendFlower(flowerTeachers);
         return flowerTeacher;
      }
    }
    private List<String> sendFlower(List list){
        return list.stram()
        .map(t->t.setFlower(1))
        .collect(Collectors.toList());
    }
  }
```

DDD 修改后测试代码

```
@RunWith(PowerMock.class)
public class TeacherServiceImplTest{
    @Mock
    StudentMapper studentMapper;

    @Test
    public void testSendFlowerTeacher(){
        Teacher teacher = new Teacher();
        teacher.setAge(50);
        Student student = new Student();
        student.setId(10L);
        student.setScore(60);
        List Lists = new ArrayList<>();
        list.addAll(1L,3L);
        student.setTeacherIds(Lists);
        when(teacher.isExperienceTeacher()).thenReturn(true)
        when(student.isBadScoreStudent()).thenReturn(true);
        when(student.getTop2Teachers()).thenReturn(list);
        List<String> flowerTeachers = getFlowerTeacher(teacher,  student);
        assertThat(flowerTeachers, hasSize(2));
        assertThat(flowerTeachers, containsAnyOrder(1L, 3L));
    }
}
```

优点：

- 当方法需要修改时直接改 Model 里的方法即可
- 无论 isExperienceTeacher isBadScoreStudent getTop2Teachers 如何修改都不影响测试结果

DDD 领域驱动设计—案例

```java
@Data
public class Device {
  /**
   * 设备 user-agent
   */
  private String ua;
  private String carrier;
  private String imei;
  private String imei_md5;
  private Geo geo;


  @Data
  public class Geo {
    private float latitude;
    private float longitude;
  }

```

```java
  /**
   * 构建设备信息
   */
  public boolean buildDevice(xxx) { }

  /**
   * 设备Id校验
   */
  private boolean checkDid(xxx) { }
  /**
   * geo 转换
   */
  private void convertGeo(xxx) { }
  /**
   * 运营商字符串
   */
  private String convertCarrier(xxx) {}

  /**
   *链接类型
   */
  private Integer convertConnectionType(Integer connectionType) {}

  /**
   * 设备类型
   */
  private Integer convertDeviceType(Integer deviceType) {}
```

理论如何落地？新项目如何引入？执行情况如何？


PART 05
项目实践与落地

实施背景

实施背景

- 混合云建设过程中涉及大量模块重构，同时还有很多新功能模块编写
- 当前程序化广告服务代码有 37 万行，下个 6 年会有更多的版本功能叠加，如何保障工程质量
- 程序员专业性提升，很多人不写，是不想或者不会还是所谓时间不够

业务现状

- 超过四百亿广告竞价，对代码质量要求很高
- 系统内单路会话平均耗时 30ms
- 年毛利亿级

问题与成果

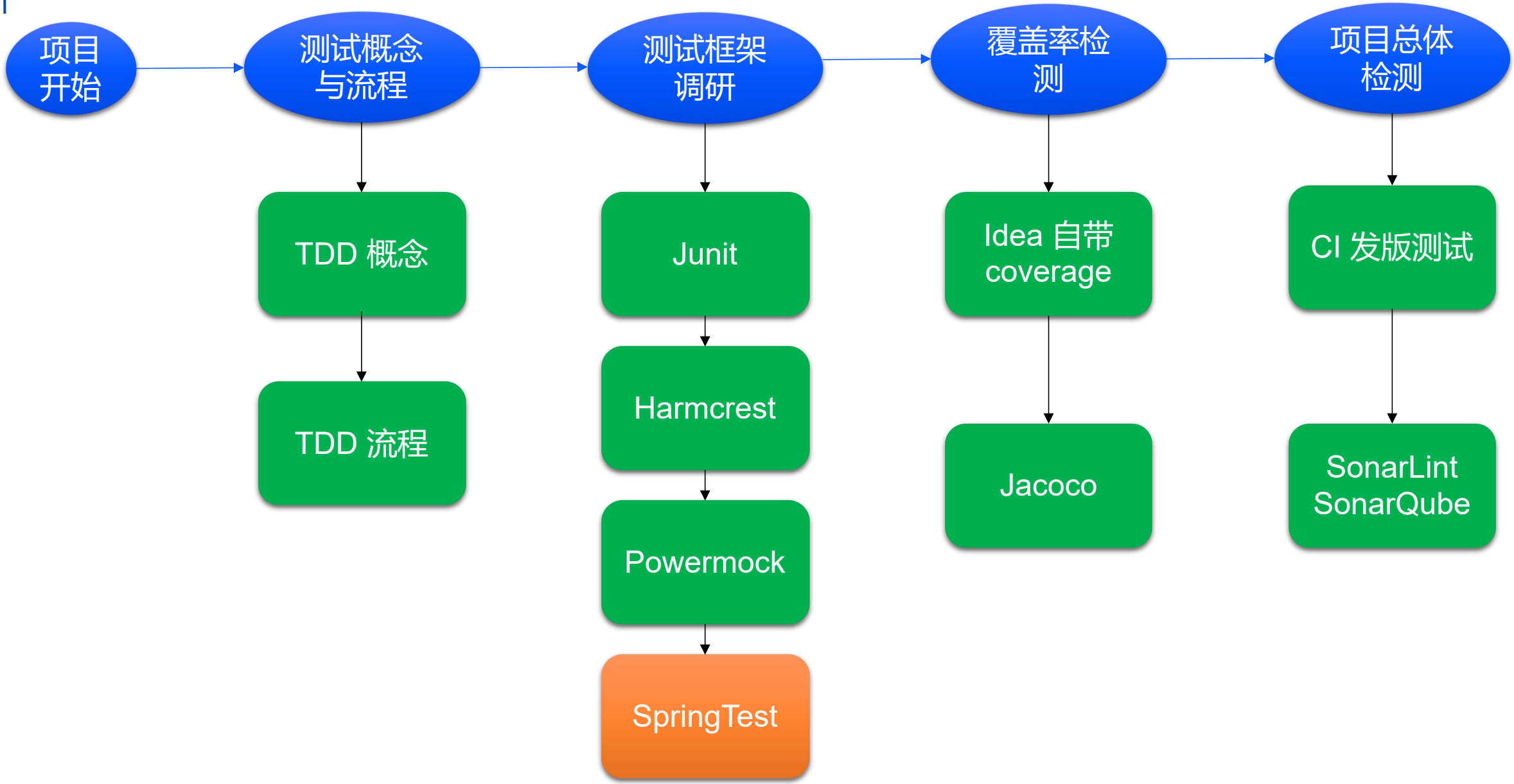
遇到的问题：

- 研发对测试的不熟悉，耗费一些时间写测试
- 部分逻辑层过于臃肿，重构困难
- 相同验证分布在各个类中，测试代码重复
- private 方法 mock 后，当前类所有测试覆盖率统计不上
- private static final 静态变量无法 mock
- 单元测试数据污染公共测试数据库

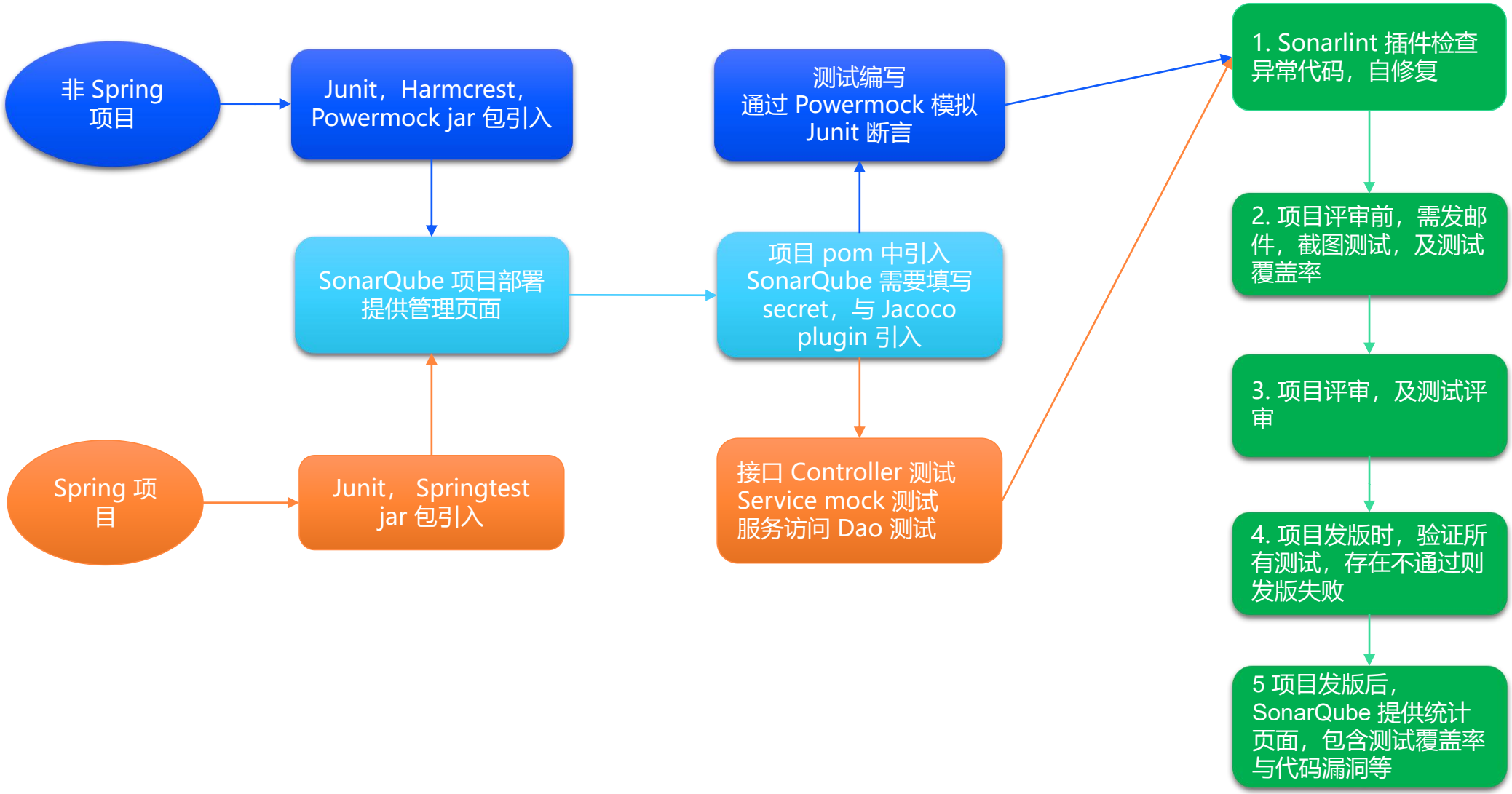
目前实施成果：

- 单元测试覆盖率从不到 5% 提升到逼近 100%
- 新需求都已被测试覆盖
- 测试没有拖延进度，且提高了发版的质量
- 大胆重构老功能

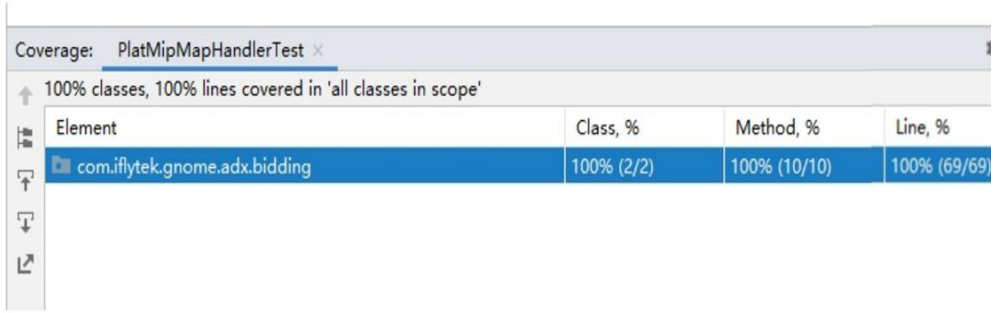
项目落地流程

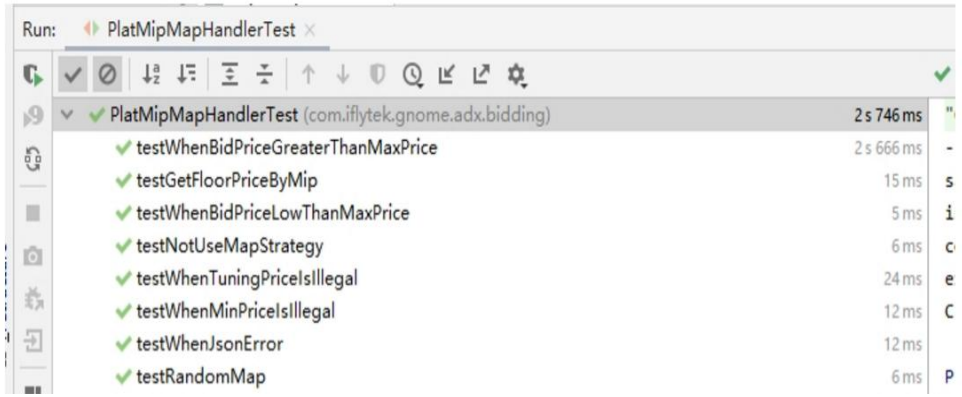

项目开始
测试概念与流程
TDD 概念
TDD 流程
测试框架调研
Junit
Harmcrest
Powermock
SpringTest
覆盖率检测
Idea 自带 coverage
Jacoco
项目总体检测
CI 发版测试
SonarLint
SonarQube

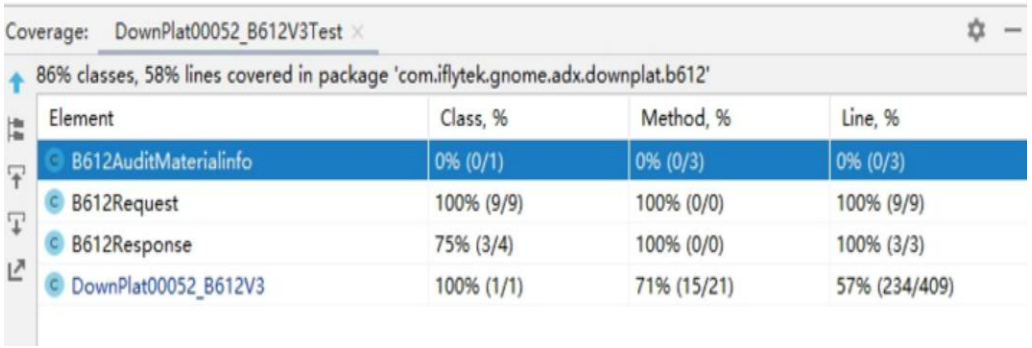
项目落地流程

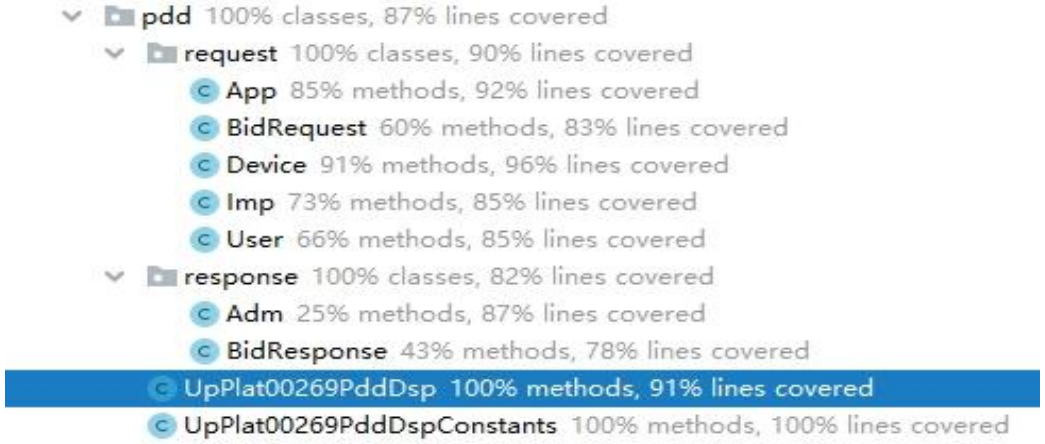

非 Spring 项目
Junit，Harmcrest，Powermock jar 包引入
SonarQube 项目部署提供管理页面
Spring 项目
Junit， Springtest jar 包引入
测试编写
通过 Powermock 模拟 Junit 断言
项目 pom 中引入 SonarQube 需要填写 secret，与 Jacoco plugin 引入
接口 Controller 测试
Service mock 测试
服务访问 Dao 测试
1. Sonarlint 插件检查异常代码，自修复
2. 项目评审前，需发邮件，截图测试，及测试覆盖率
3. 项目评审，及测试评审
4. 项目发版时，验证所有测试，存在不通过则发版失败
5 项目发版后，SonarQube 提供统计页面，包含测试覆盖率与代码漏洞等

执行情况

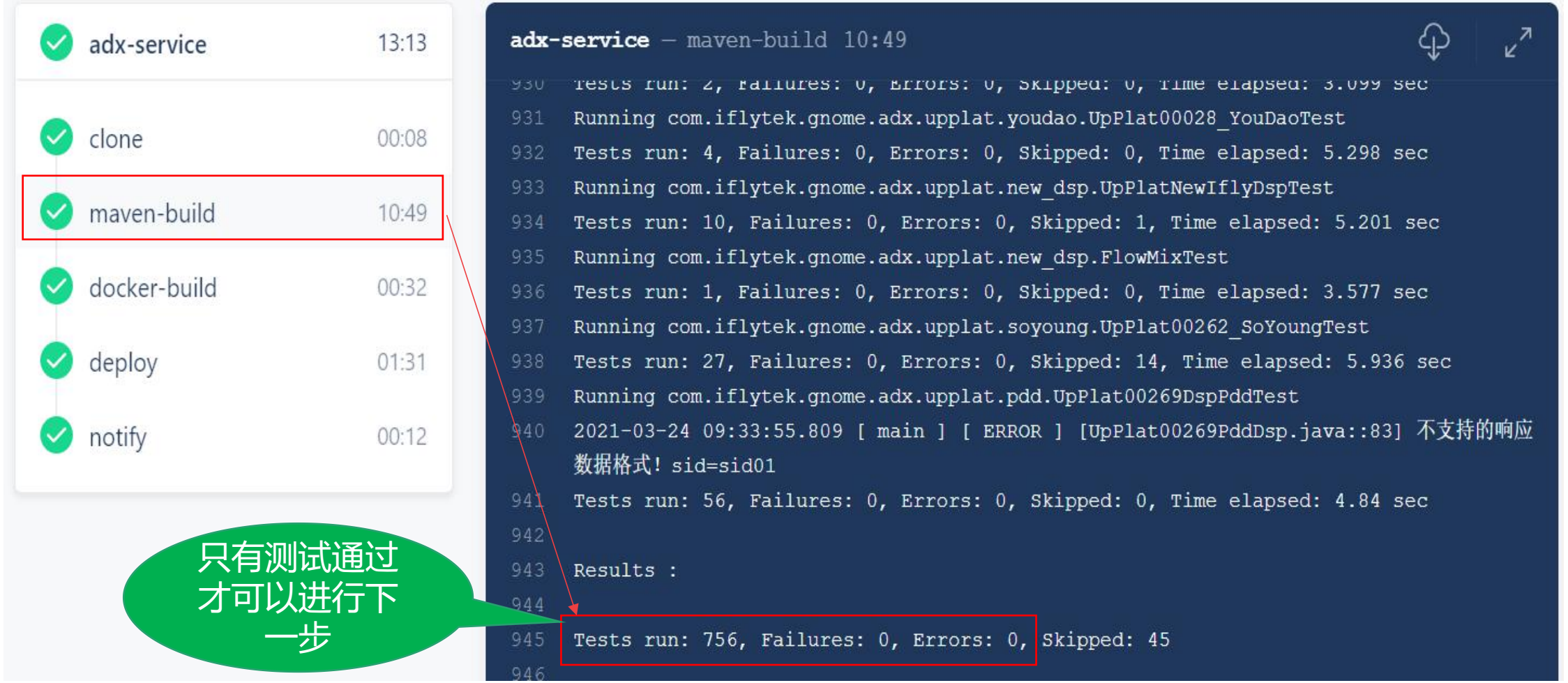

Coverage: PlatMipMapHandlerTest
100% classes, 100% lines covered in 'all classes in scope'
Element
Class, %
Method, %
Line, %
com.iflytek.gnome.adx.bidding
100% (2/2)
100% (10/10)
100% (69/69)


Run: PlatMipMapHandlerTest
PlatMipMapHandlerTest (com.iflytek.gnome.adx.bidding) 2 s 746 ms
testWhenBidPriceGreaterThanMaxPrice 2 s 666 ms
testGetFloorPriceByMip 15 ms
testWhenBidPriceLowThanMaxPrice 5 ms
testNotUseMapStrategy 6 ms
testWhenTuningPriceIsIllegal 24 ms
testWhenMinPriceIsIllegal 12 ms
testWhenJsonError 12 ms
testRandomMap 6 ms


Coverage: DownPlat00052_B612V3Test
86% classes, 58% lines covered in package 'com.iflytek.gnome.adx.downplat.b612'
Element
Class, %
Method, %
Line, %
B612AuditMaterialinfo
0% (0/1)
0% (0/3)
0% (0/3)
B612Request
100% (9/9)
100% (0/0)
100% (9/9)
B612Response
75% (3/4)
100% (0/0)
100% (3/3)
DownPlat00052_B612V3
100% (1/1)
71% (15/21)
57% (234/409)


pdd 100% classes, 87% lines covered
request 100% classes, 90% lines covered
App 85% methods, 92% lines covered
BidRequest 60% methods, 83% lines covered
Device 91% methods, 96% lines covered
Imp 73% methods, 85% lines covered
User 66% methods, 85% lines covered
response 100% classes, 82% lines covered
Adm 25% methods, 87% lines covered
BidResponse 43% methods, 78% lines covered
UpPlat00269PddDsp 100% methods, 91% lines covered
UpPlat00269PddDspConstants 100% methods, 100% lines covered

构建执行测试


adx-service 13:13
clone 00:08
maven-build 10:49
docker-build 00:32
deploy 01:31
notify 00:12
只有测试通过才可以进行下一步
adx-service – maven-build 10:49
931 Running com.iflytek.gnome.adx.upplat.youdao.UpPlat00028_YouDaoTest
932 Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 5.298 sec
933 Running com.iflytek.gnome.adx.upplat.new_dsp.UpPlatNewIflyDspTest
934 Tests run: 10, Failures: 0, Errors: 0, Skipped: 1, Time elapsed: 5.201 sec
935 Running com.iflytek.gnome.adx.upplat.new_dsp.FlowMixTest
936 Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 3.577 sec
937 Running com.iflytek.gnome.adx.upplat.soyoung.UpPlat00262_SoYoungTest
938 Tests run: 27, Failures: 0, Errors: 0, Skipped: 14, Time elapsed: 5.936 sec
939 Running com.iflytek.gnome.adx.upplat.pdd.UpPlat00269DspPddTest
940 2021-03-24 09:33:55.809 [main] [ERROR] [UpPlat00269PddDsp.java::83] 不支持的响应数据格式! sid=sid01
941 Tests run: 56, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.84 sec
942
943 Results :
944
945 Tests run: 756, Failures: 0, Errors: 0, Skipped: 45
946

执行效果

指标名称	使用TTD前	开始使用半年	最近一个月
单需求耗费的时长	无耗时	6 小时	3.5 小时
开发人员接受的程度	了解，很少写	上手写，遇到部分测试场景书写比较困难	按要求正常写测试
每月 Bug 数量	2020.03 63 2020.04 27 2020.05 21 2020.06 36 2020.07 71 2020.08 46 2020.09 36	2020.10 49 2020.11 58 2020.12 17 2021.01 18 2021.02 9 2021.03 8	2021.04 10
发版的成功率	78%	85%	90%
是否考虑过设计与重构	考虑并会使用，但不迫切	尝试使用 DDD 与其它设计	重构老代码

后期展望：自动化测试

交流 & 讨论