CODE A BETTER LIFE
一 行 代 码　　亿 万 生 活

美团外卖客户端 UI 自动化测试实践

美团点评 · 外卖测试组 · 刘健 · 2018.09

美团点评

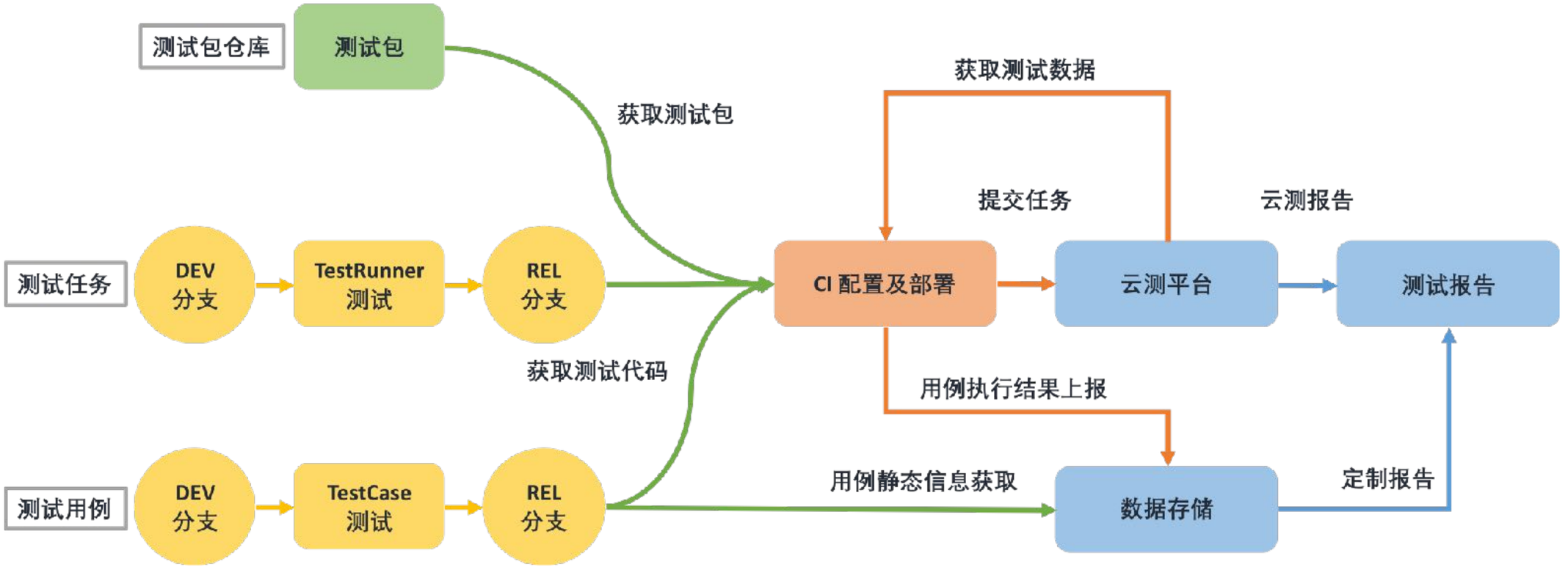
刘健　2015年9月加入原美团

先后在外卖用户端、外卖商家端从事客户端测试工作

主要负责客户端 UI 自动化测试及客户端质量体系的建设

内容概要

1. 方案总览
2. 测试框架设计
3. 测试任务执行
4. 持续集成
5. 应用场景
6. 工程体系



1. 方案总览

美团外卖自动化测试的对象

方案总览

外卖 Android

外卖 频道 Android

外卖 iOS

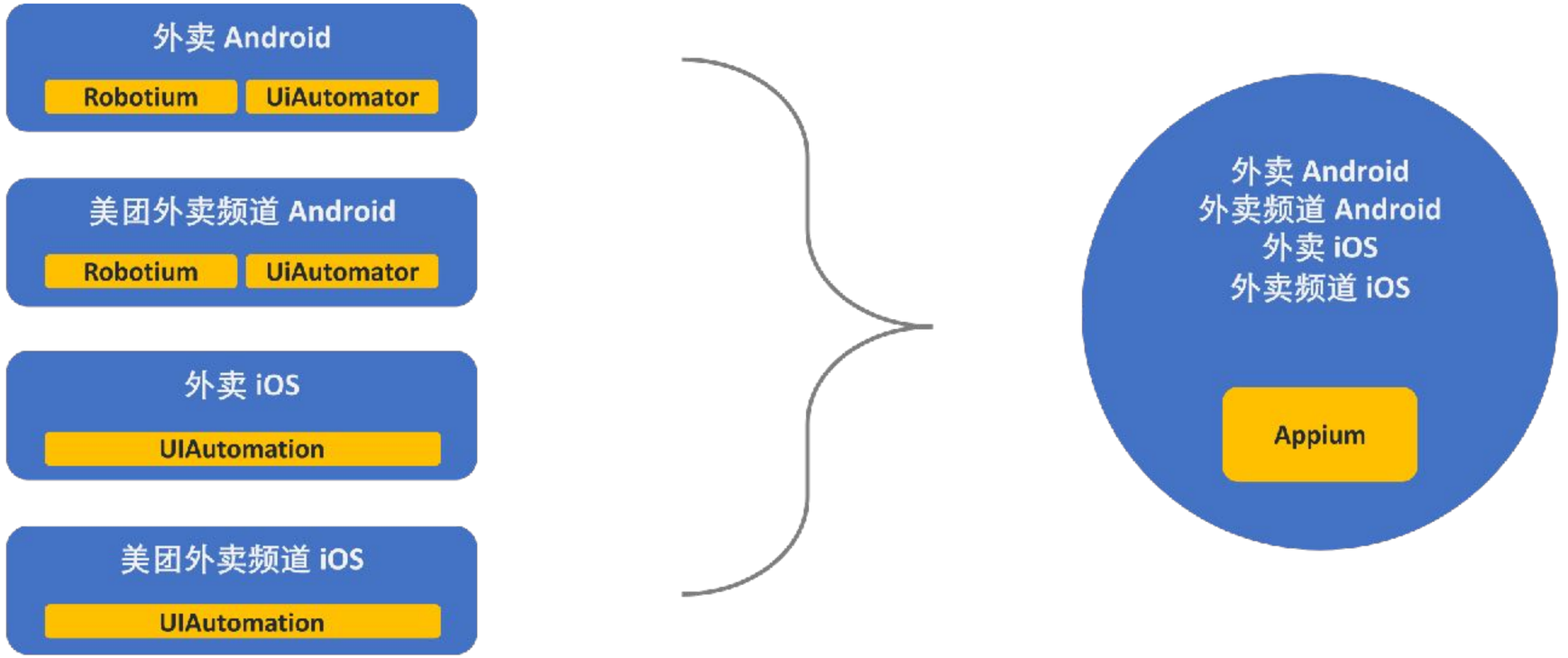
外卖 频道 iOS

{美团外卖 App} 和 {美团 App 内部外卖 频道} 共计 4 端

自动化测试用例如何实现 4 端通用的目标？

自动化测试方案的演进

方案总览

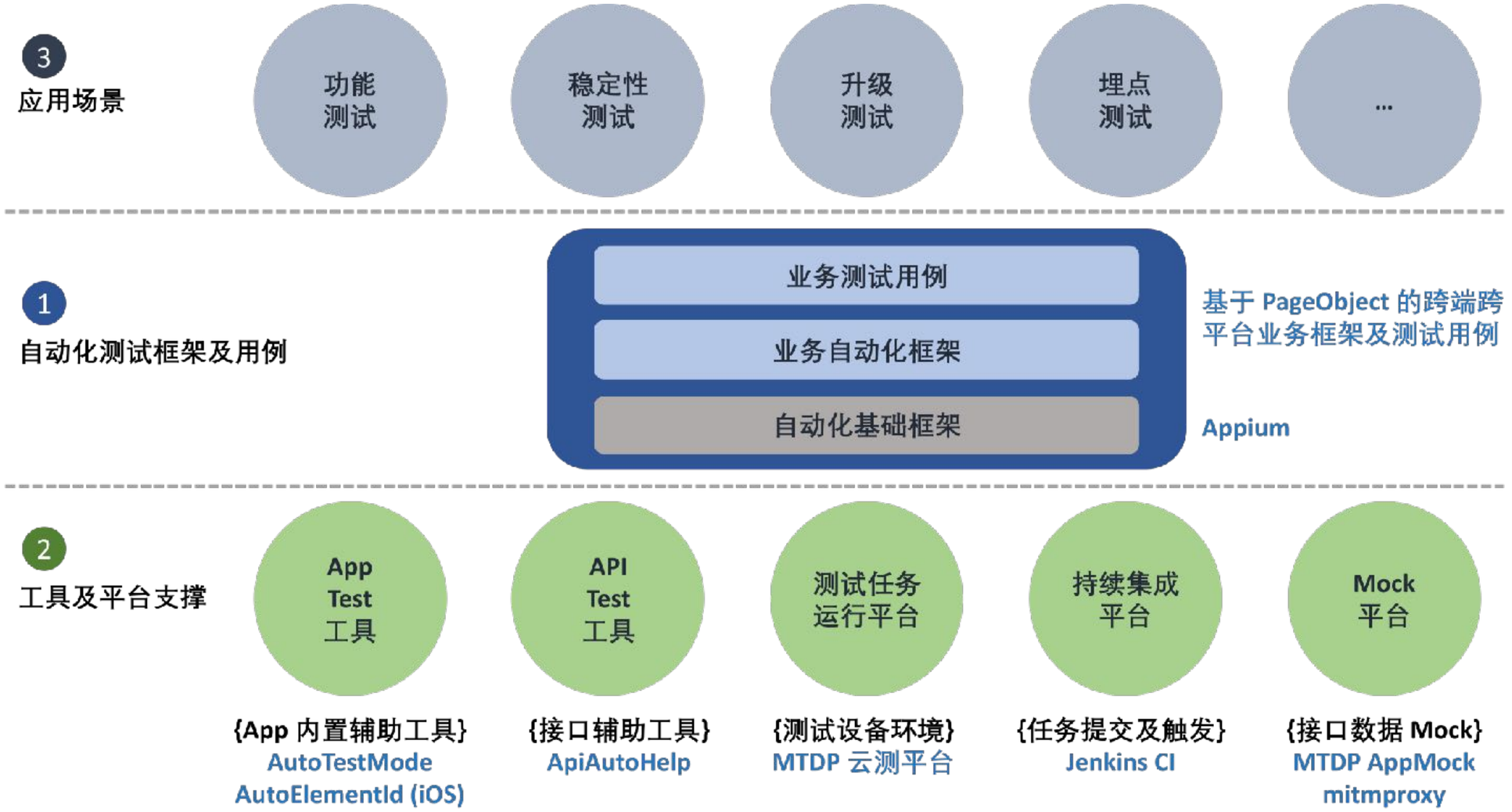


框架和工具不统一，重复建设，复用困难

四端统一及高度复用的工具和框架

自动化测试系统的构成

方案总览

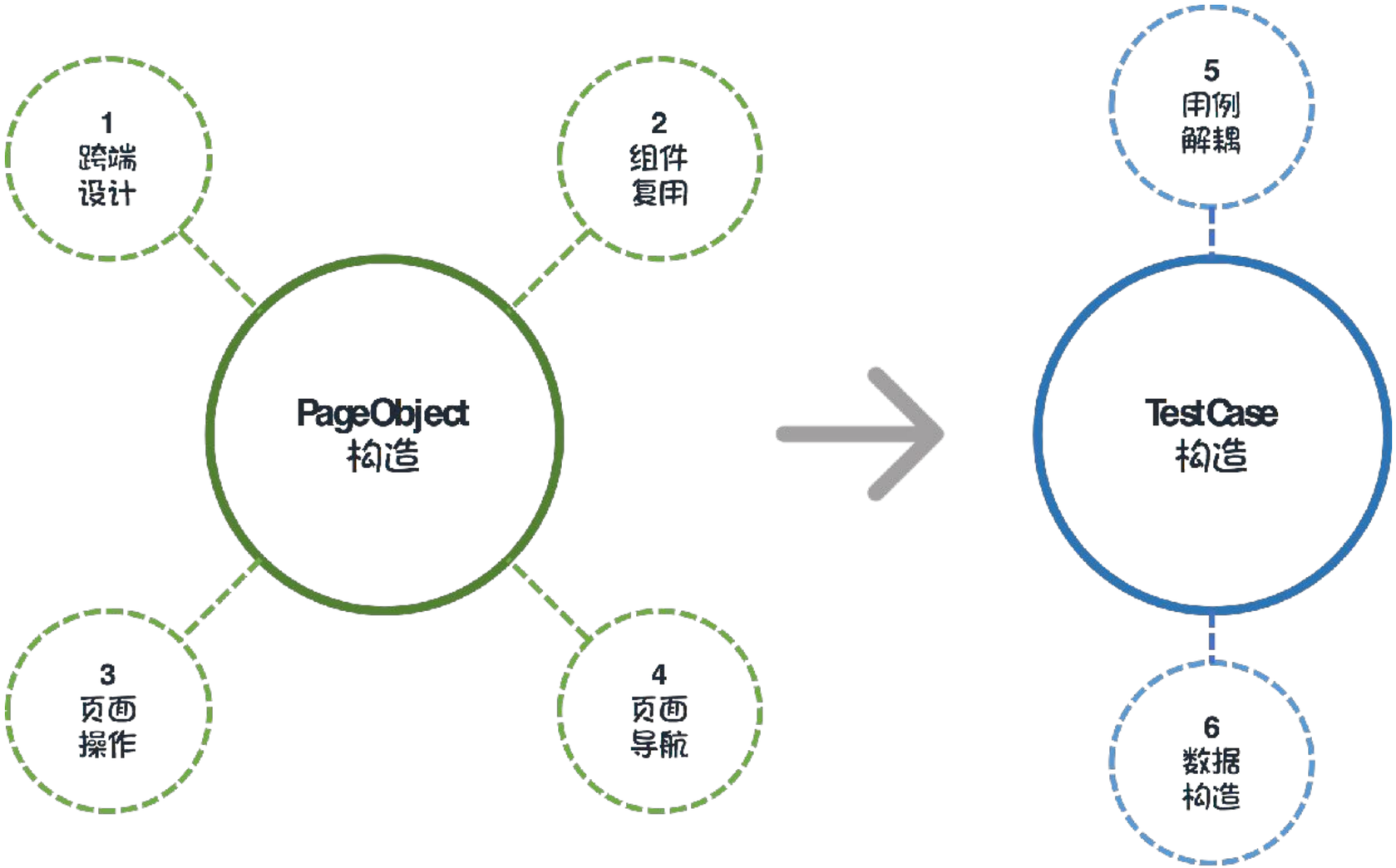


2. 测试框架设计

- 基于 PageObject 模式设计跨端跨平台测试用例

内容概要

测试框架设计

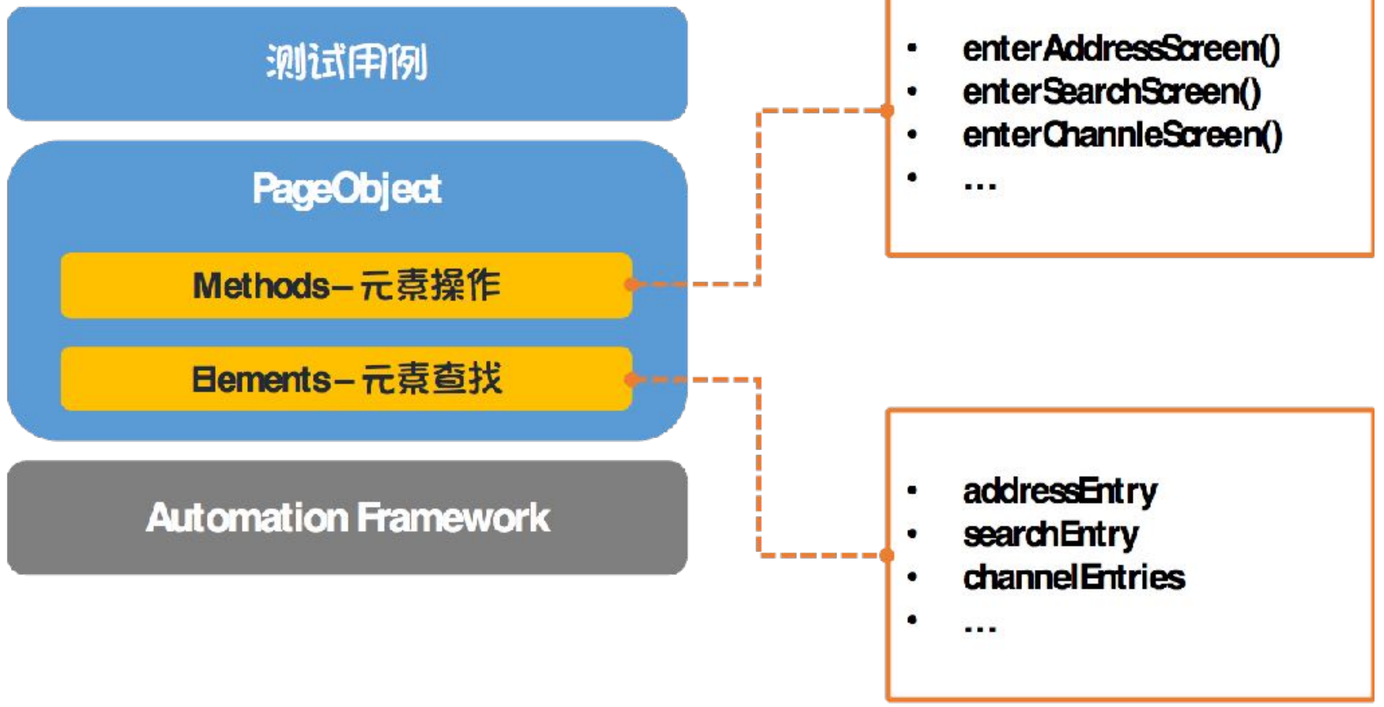

1
跨端
设计
2
组件
复用
PageObject
构造
3
页面
操作
4
页面
导航
5
用例
解耦
TestCase
构造
6
数据
构造

PageObject 基本概念

测试框架设计 (1/6)

美团外卖 首页

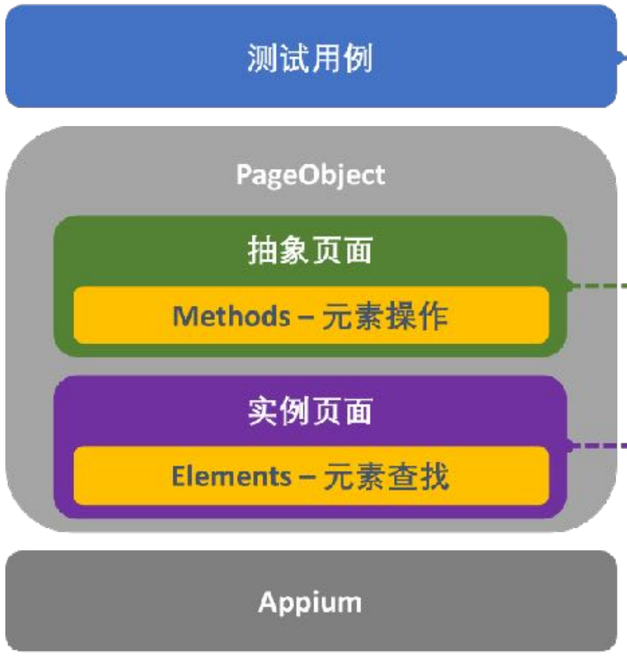
以美团外卖 首页为例, 其 PageObject 封装的元素和操作方法



跨端跨平台的 PageObject 设计

测试框架设计 (1/6)

以从首页进入地址选择页为例的 PageObject 构建




```
public class HomeScreenTestCase extends TestCaseBase {
   @Test
   public void testEnterAddress() {
      HomeScreenBase homeScreen = getHomeScreen();
      homeScreen.enterAddressScreen().assertIsDisplayed()
            .assertIsNotBlank();
   }
}
```

3

测试用例中引用抽象页面类型
(四端通用)

```
public abstract class HomeScreenBase extends UiScreenBase {
   public abstract MobileElement getAddressEntry();

   public AddressScreenBase enterAddressScreen() { tap(getAddressEntry()); }
}
```

1

抽象页面定义元素操作方法
(四端通用)

```
public class WmChannelHomeScreen extends HomeScreenBase {
   @AndroidFindBy(id = "...")   @iOSXCUITFindBy(id = "...")
   private MobileElement mAddressEntry;

   @Override
   public MobileElement getAddressEntry() { return mAddressEntry; }
}
```

```
public class WmHomeScreen extends HomeScreenBase {
   @AndroidFindBy(id = "...")   @iOSXCUITFindBy(id = "...")
   private MobileElement mAddressEntry;

   @Override
   public MobileElement getAddressEntry() { return mAddressEntry; }
}
```

2

实例页面继承抽象页面

实例页面提供元素查找方法
(四端存在差异)

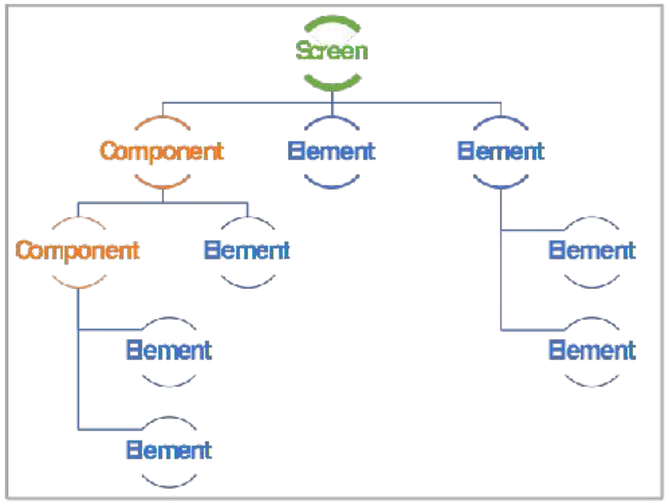
元素操作方法四端存在差异时
在实例页面中复写基类方法

基于页面继承: 通过 <抽象 & 实例> 页面分层及 Appium 页面元素注解实现跨端跨平台

页面组件的分类和抽象

测试框架设计 (2/6)

页面组件是元素的集合



通过组件可以实现 PageObject 内部和 PageObject 间的复用

页面组件归类

列表形式

特点：多个组件构成一个列表
例如：商家列表卡片

跨页面复用模块

特点：组件在多个页面出现
例如：购物车

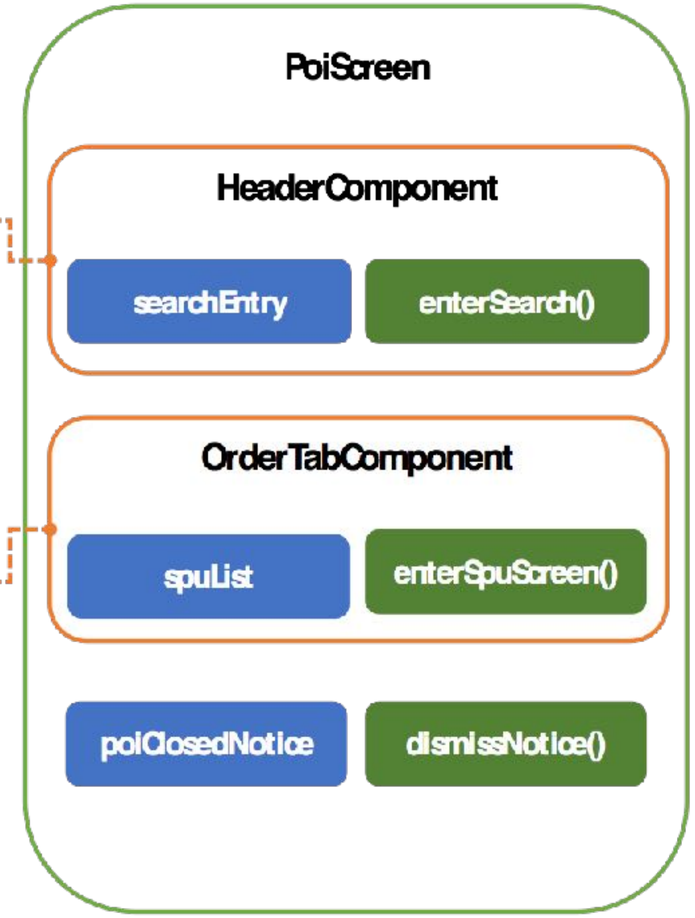
页面模块拆分

特点：一页面逻辑拆分
例如：商家 TAB 页

商家页面的组件实现

测试框架设计 (2/6)

例:商家页面头部信息及点餐部分 TAB部分



组件　页面元素　操作方法

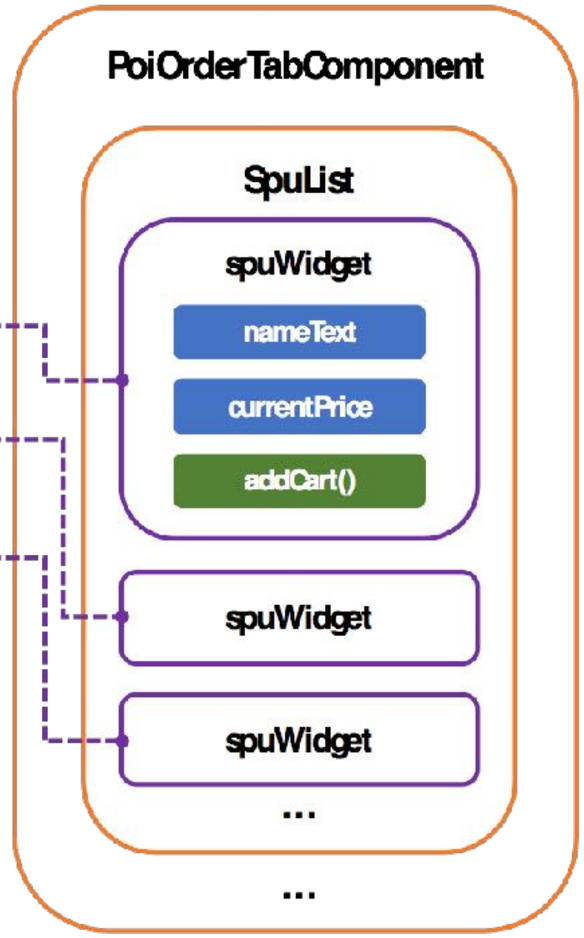
```
public abstract class PoiScreenBase extends UiScreenBase {
  /* 获取头部信息组件 */
  public PoiHeaderComponentBase getPoiHeaderComponent() {
    return mHeaderComponent;
  }

  /* 获取点餐部分 TAB 组件 */
  public PoiTabOrderComponentBase getOrderComponent() {
    return mOrderComponent;
  }

  /* 检查商家是否处在休息状态 */
  public boolean isPoiClosed() {
    return getPoiClosedNotice().isDisplayed();
  }

  /* 添加任意商品到购物车并提交订单 */
  public OrderPreviewScreenBase makeOrder() {
    return mOrderComponent.makeOrder();
  }
}
```

注:商家页抽象页面类实现页面操作方法

商家页面的组件实现

测试框架设计 (2/6)

例:酸品市片列表部分



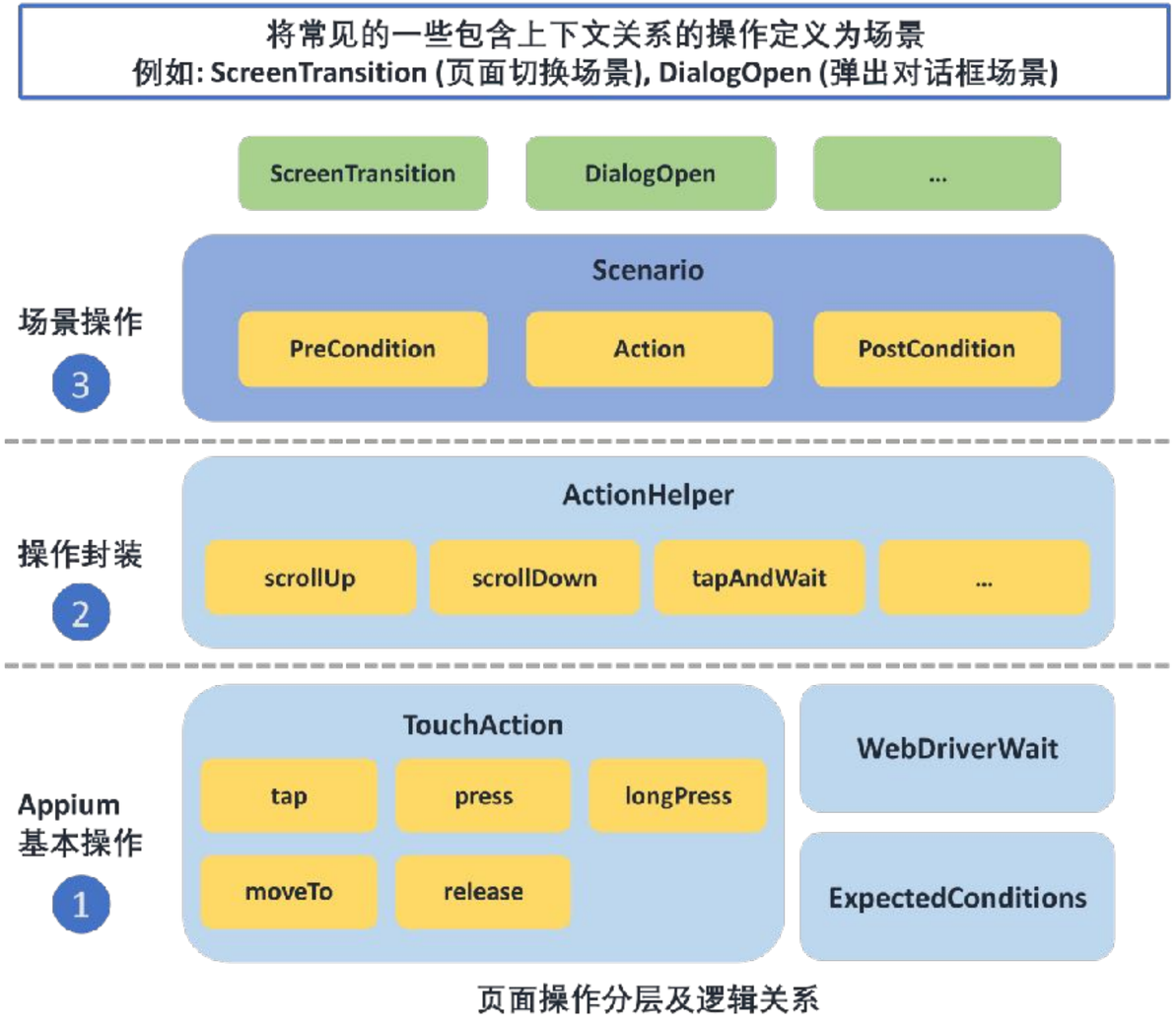
组件 | Widget | 页面元素 | 操作方法

```
public abstract class PoiTabOrderComponentBase extends TabComponentBase {
  /* 获取酸品市片列表 */
  public abstract List<? extends BaseSpuInfoWidget> getSpuWidgets();
  /* 获取指定位置的酸品市片 */
  public abstract BaseSpuInfoWidget getSpuWidget(int index);

  /* 酸品市片组件定义：Widget 作为容器包含子页面元素 */
  public abstract class BaseSpuInfoWidget extends Widget {
    public BaseSpuInfoWidget(MobileElement element) {
      super(element);
      PageFactory.initElements(fieldDecorator, this);
    }

    public abstract MobileElement getAddButton();// 添加酸品按钮
    public abstract MobileElement getRemoveButton();// 移除酸品按钮

    public boolean addCart() { /* ... */ }
    public boolean removeCart() { /* ... */ }
  }
}
```

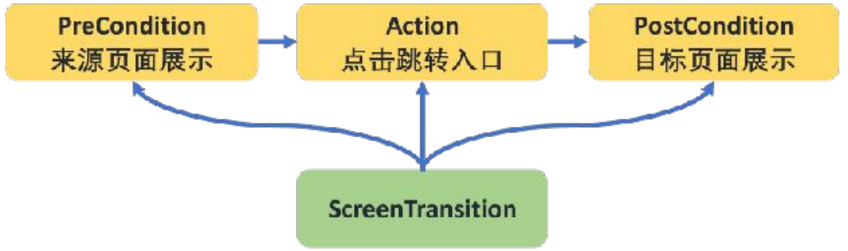
注:哪考 Appium Widget 的使用方式

页面操作的实现方式

测试框架设计 (3/6)

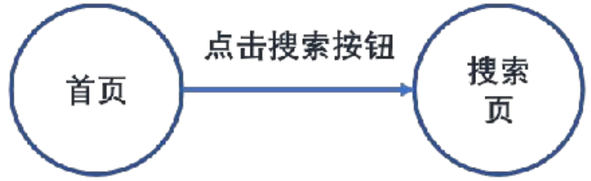


页面操作分层及逻辑关系



ScreenTransition (页面跳转场景) 示例: 进入全局搜索

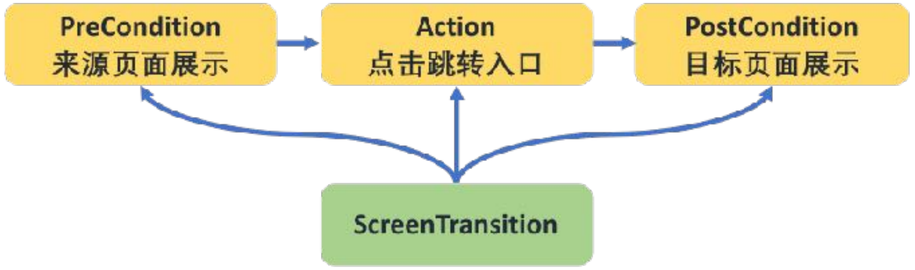
- PreCondition: 当前页面为首页
- Action: 点击搜索入口元素
- PostCondition: 跳转至搜索页面



同一类型的场景的内部处理逻辑和流程相似
可以减少重复的代码逻辑, 并方便进行重试等控制

页面操作的实现方式

测试框架设计 (3/6)



1

```
public class ScreenNavigatorBase extends UiObjectBase {
  /* 页面跳转操作的定义 */
  public <T extends UiScreenBase> T enterScreen(
                                UiScreenBase fromScreen,
                                T toScreen,
                                MobileElement entry) {
    fromScreen.assertIsDisplayed();
    mAutoUtil.tap(entry);
    fromScreen.onExit();
    return (T) toScreen.onEnter();
  }
}
```

方式1: 通过定义方法来实现, 提供多个重载版本支持不同的参数

2

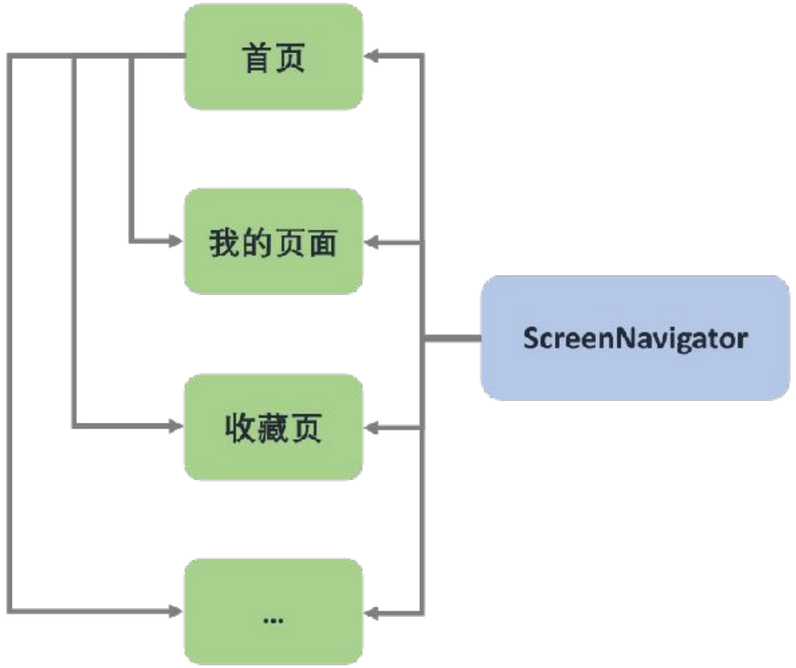
```
public class ScreenTransition<R extends UiScreenBase> implements Scenario<R> {
  private ScreenTransition(Builder<R> builder) { /* ... */ }
  public static class Builder<R> { /* ... */ }

  @Override
  public R perform() {
    // 检查前置条件
    for (Check c : getPreconditions()) {
      Assert.assertTrue(c.performCheck(), "Precondition check failed");
    }
    // 执行操作
    mAction.perform();
    // 检查结果
    for (Check c : getConsequences()) {
      Assert.assertTrue(c.performCheck(), "Consequence check failed");
    }
    return mToScreen;
  }
}
```

方式2: 把场景抽象成类, 基于 Builder 模式构造参数

页面跳转关系的实现方式

测试框架设计 (4/6)

通过 ScreenNavigator 进行页面跳转的管理



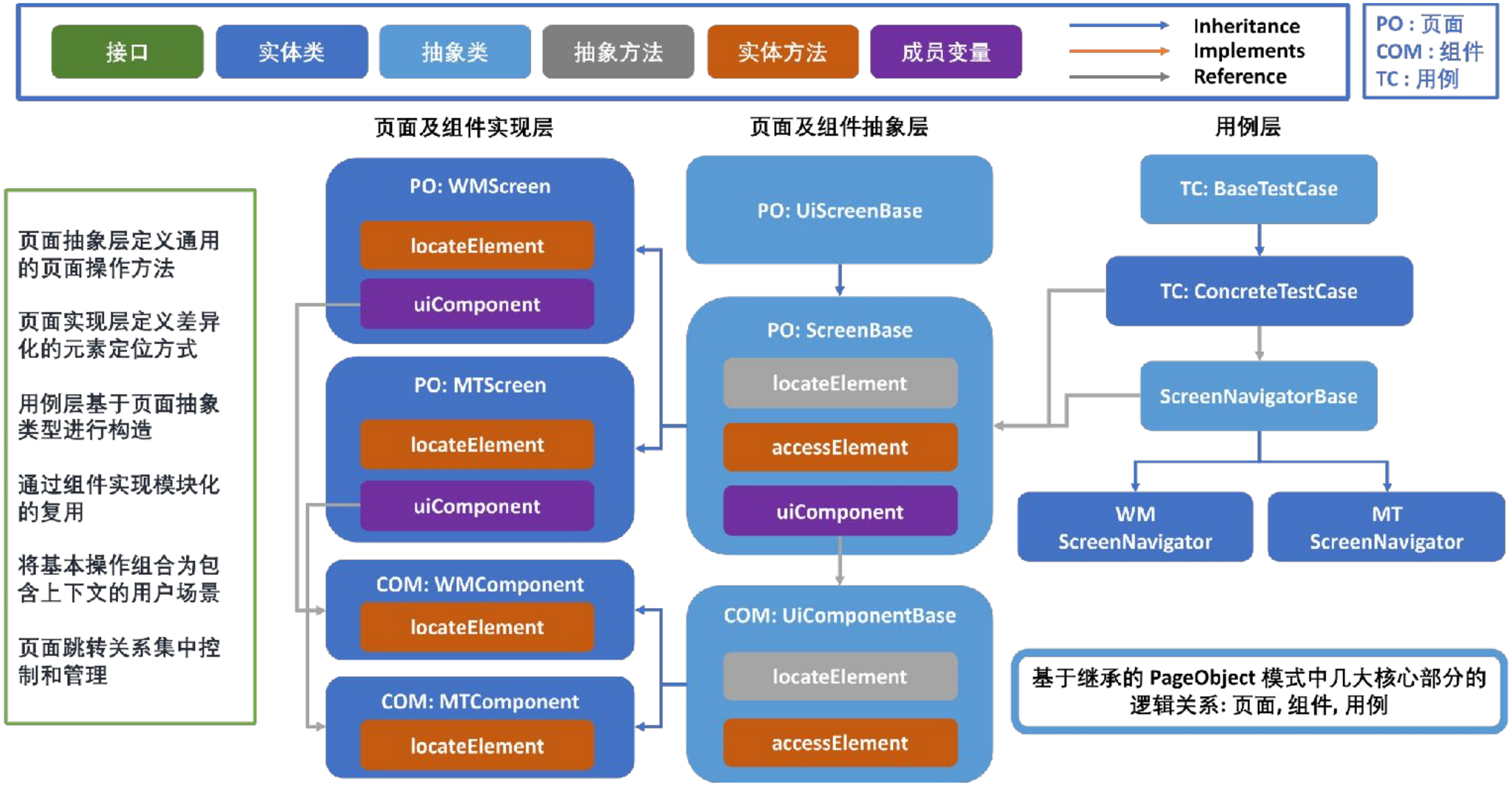
1 ScreenNavigatorBase 基类实现或定义通用方法

2 3 ScreenNavigator 实体类覆写通用方法，实现不同端的差异部分

```
public abstract class ScreenNavigatorBase extends UiObjectBase {                    1
  private Map<Class<? extends UiScreenBase>, UiScreenBase> mScreens =
      UiScreenFactoryBase.getFactory().getScreens();

  /* 抽象方法: 由子类提供具体实现 */
  public abstract MyFavoritesScreenBase enterMyFavoriteScreen();

  /* 通用页面跳转方法 */
  public <T extends UiScreenBase> T transferToScreen(T screen, Map<String, String> params) {
    /* ... 省略 ... */
  }
  public <T extends UiScreenBase> T enterScreen(UiScreenBase from, MobileElement elem, T to) {
    /* ... 省略 ... */
  }
}

public class WmScreenNavigatorBase extends ScreenNavigatorBase {                    2
  /* 外卖 APP 进入收藏列表页的路径 */
  @Override public MyFavoritesScreenBase enterMyFavoriteScreen() {
    return this.enterHomeScreen().enterMyProfileScreen().enterMyFavoriteScreen();
  }
}

public class MtScreenNavigatorBase extends ScreenNavigatorBase {                    3
  /* 外卖频道进入收藏列表页的路径 */
  @Override public MyFavoritesScreenBase enterMyFavoriteScreen() {
    return this.enterHomeScreen().openMoreMenu().enterMyFavoriteScreen();
  }
}
```

PageObject 设计小结

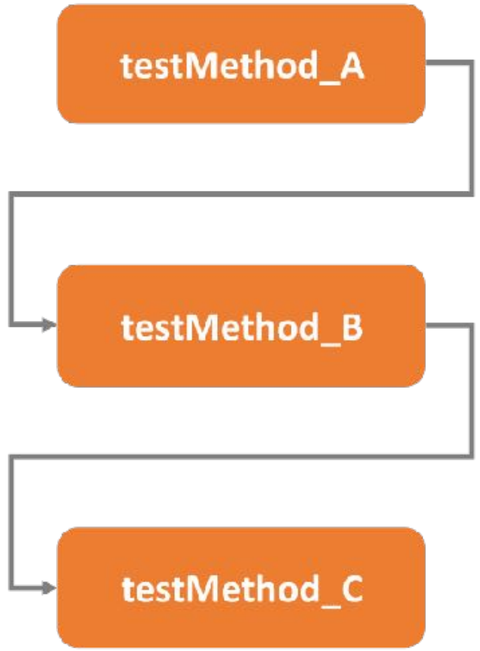
测试框架设计



测试用例解耦及异常恢复

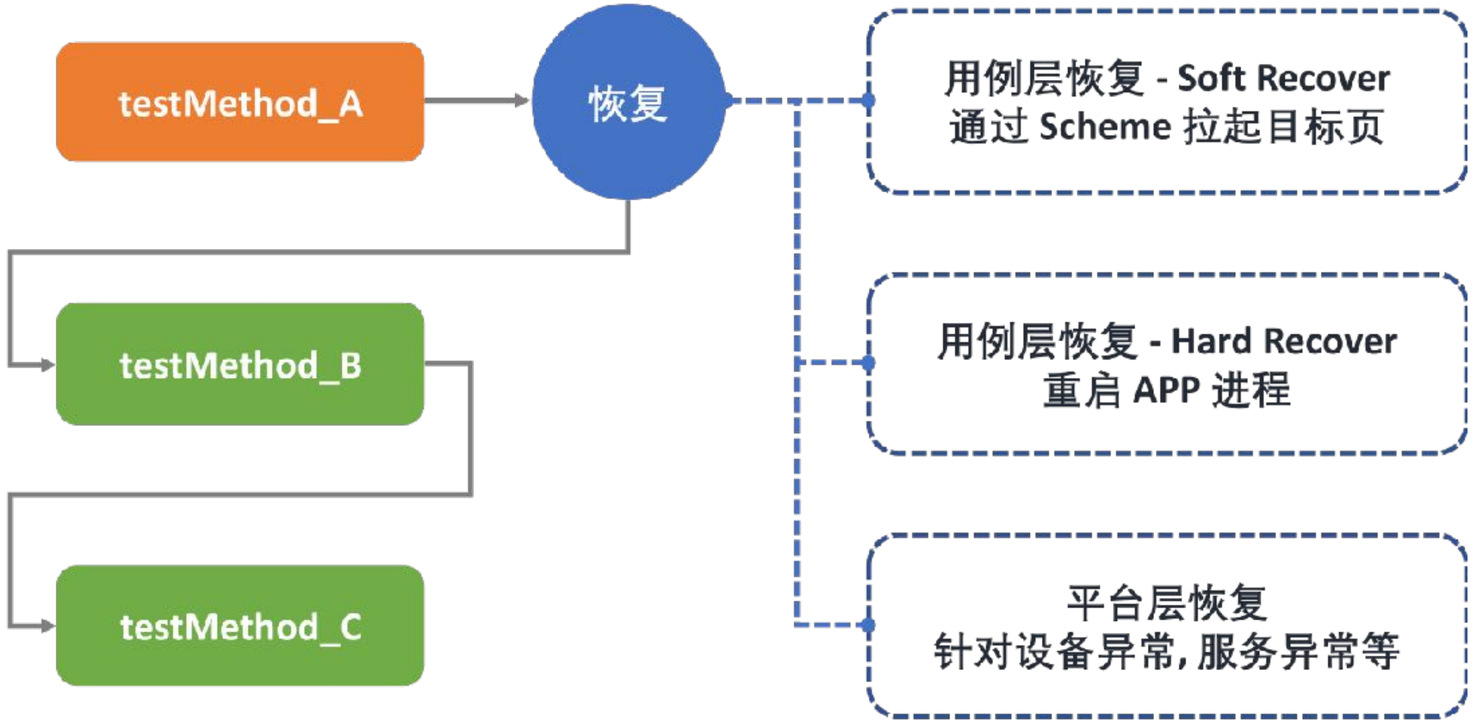
测试框架设计 (5/6)

解耦前



用例失败后导致后继用例全部失败

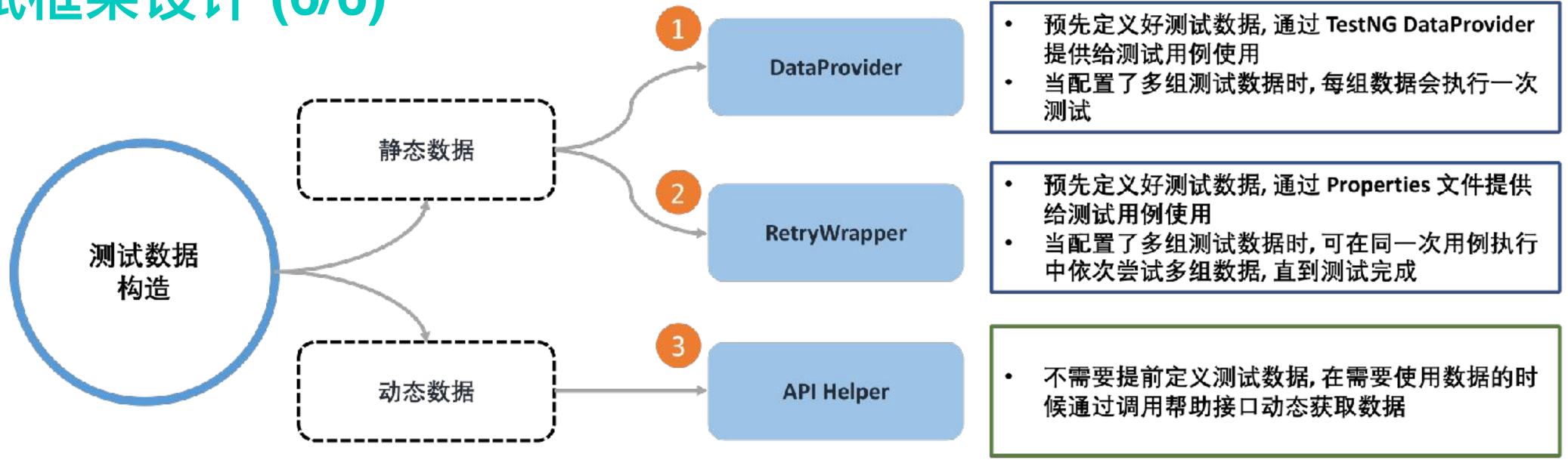
解耦后

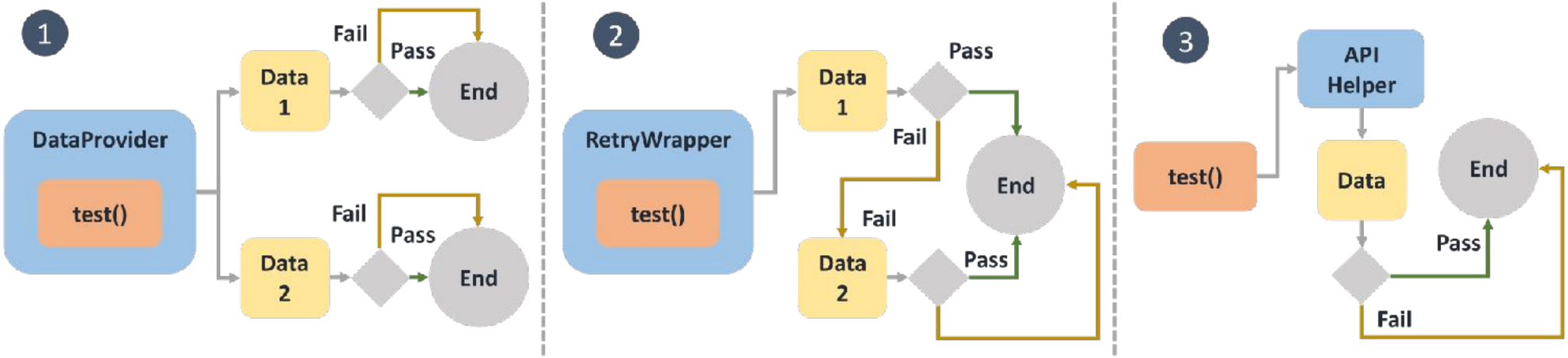


用例失败后后继用例不受影响

测试数据构造

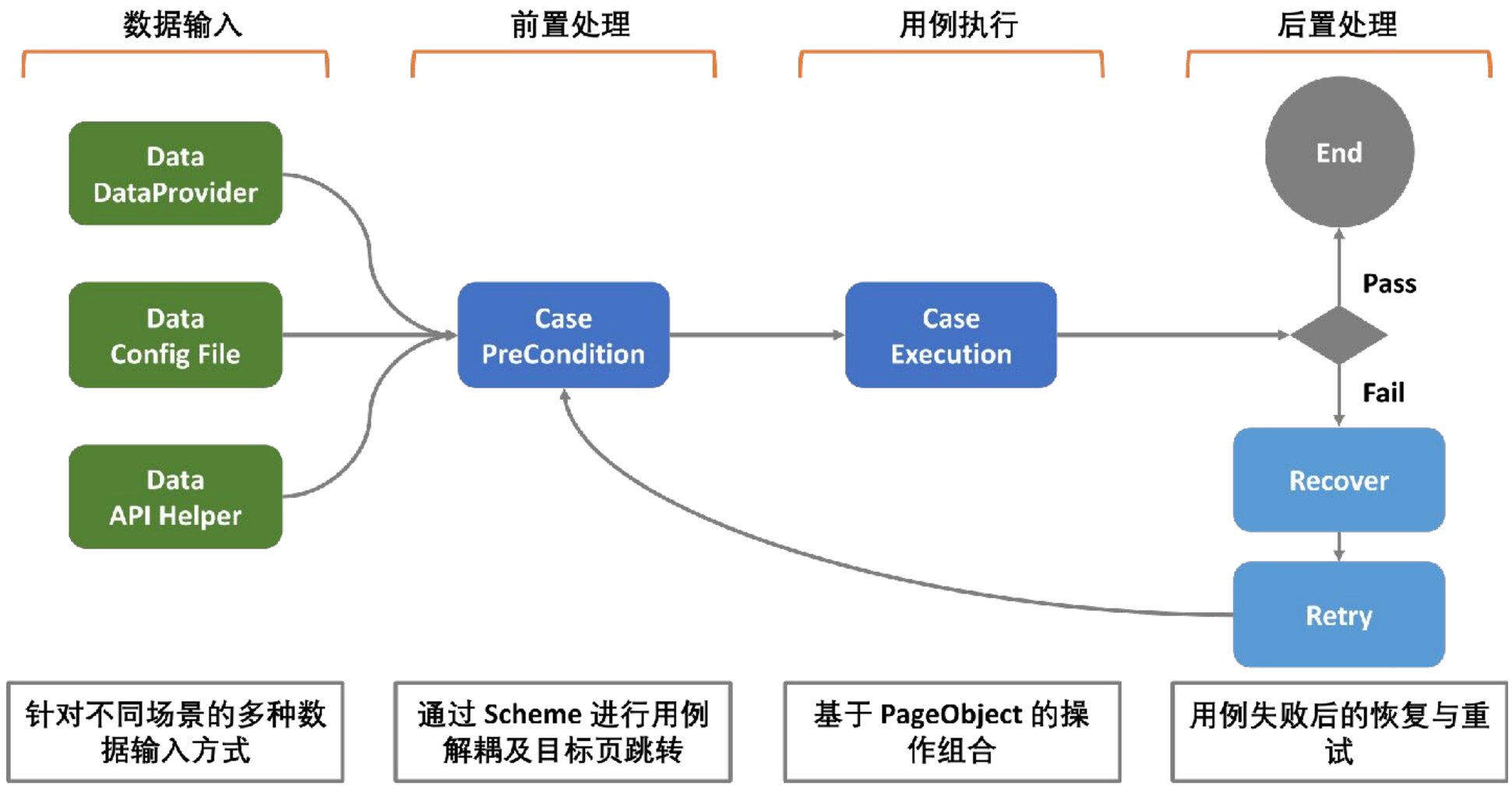
测试框架设计 (6/6)





测试用例设计小结

测试框架设计



应用效果

140+

单端测试用例数量

实际的有效用例数
大约为单端用例数
量的 4 倍

90+

用例稳定性

任意设备的条件
下, 用例整体达到
90% 以上稳定性

30

用例运行时间

单端全部用例运行耗时
Android < 30 mins
iOS < 60 mins

6+

应用场景

API 发版巡检
兼容性测试
埋点测试
…




3. 测试任务执行

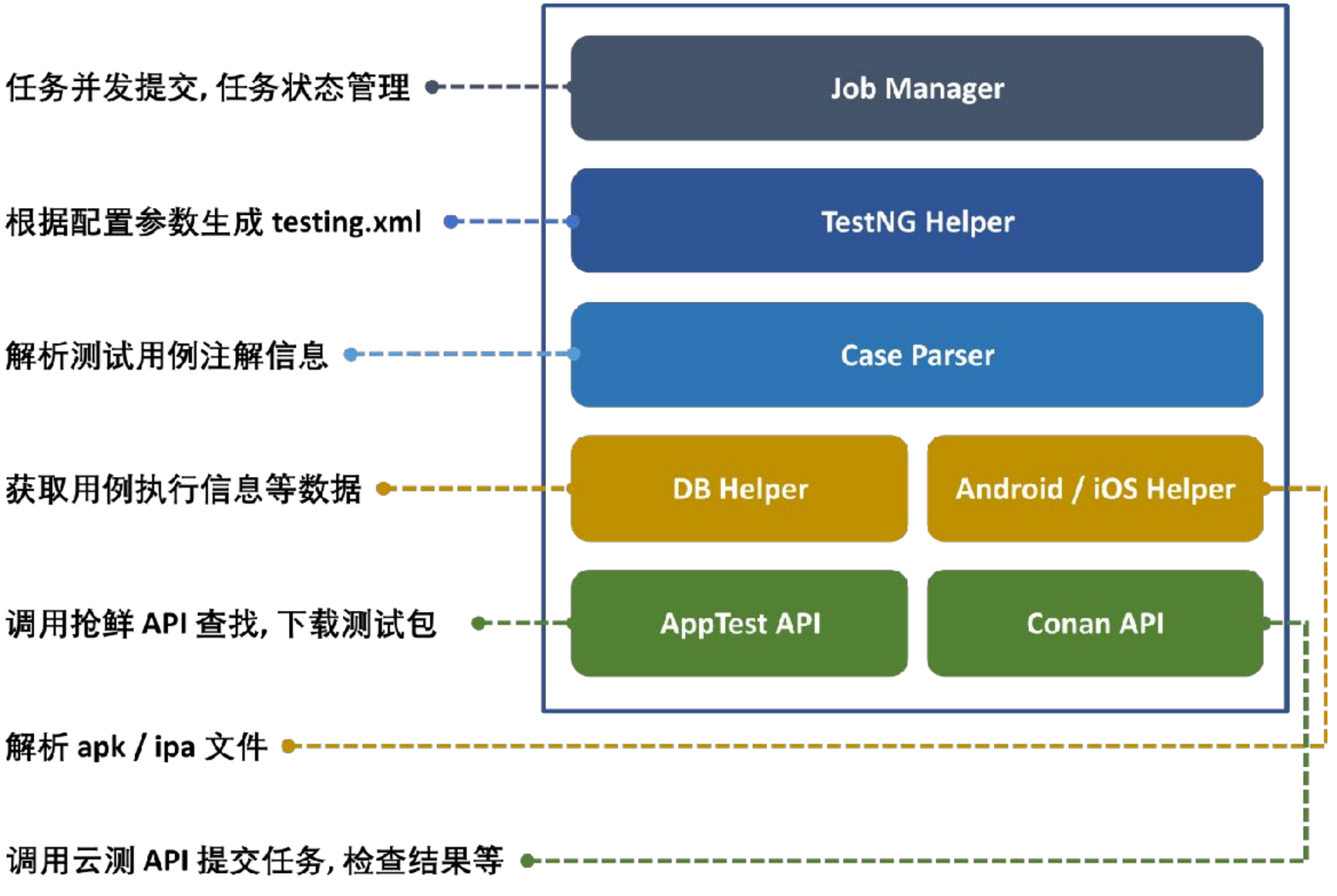
- 通过 TestRunner 规划测试用例的执行

TestRunner 的设计

测试任务执行

主要功能

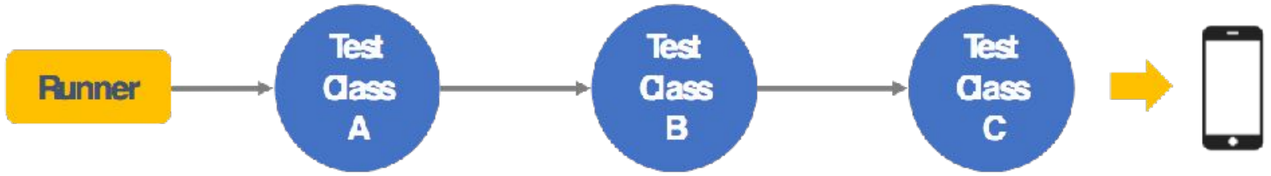
- 云测接口封装
- 云测任务流程控制
- 用例动态组合
- 任务并发提交及执行



任务执行效率提升

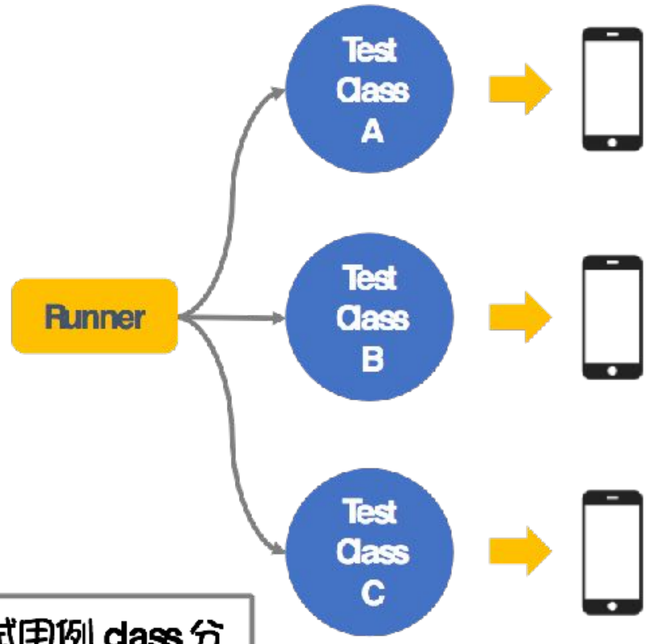
测试任务执行

将所用测试用例在一个设备上串行执行

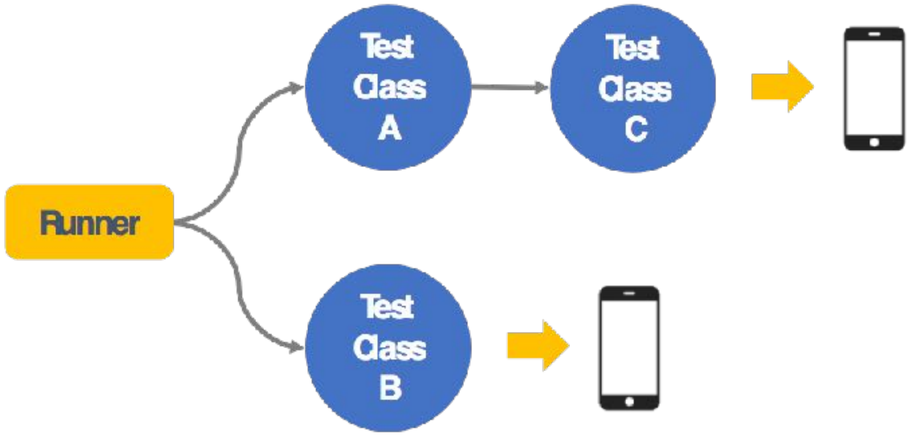


串行执行

并行执行



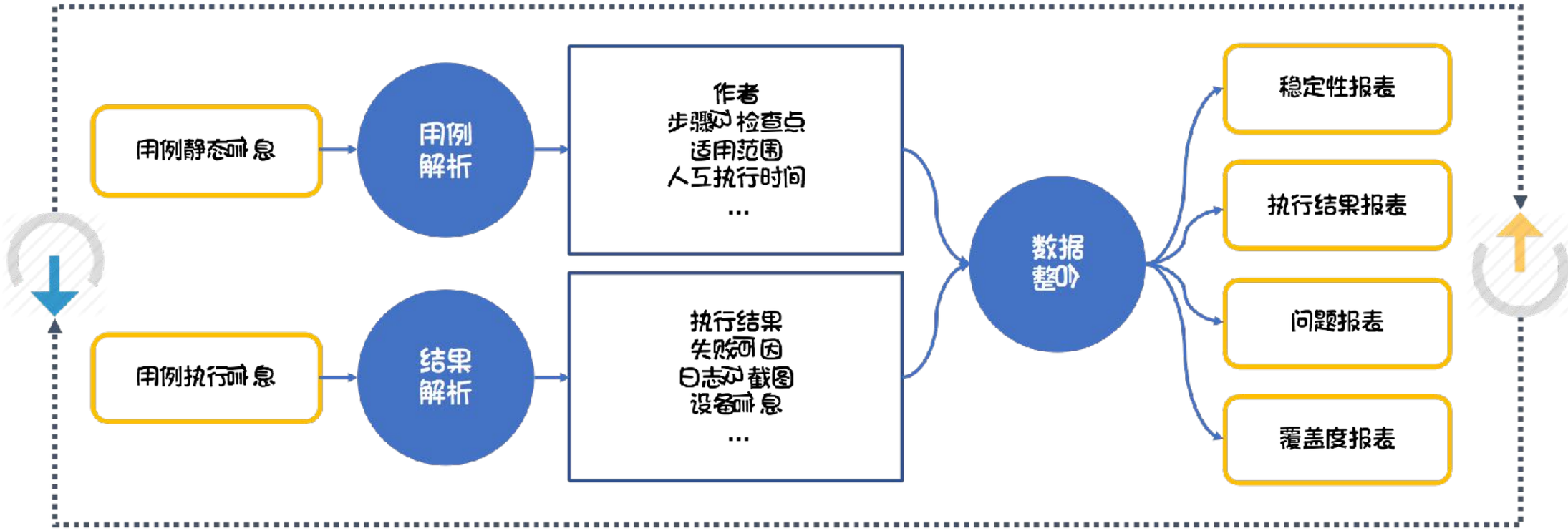
将不同的测试用例 class 分别到不同设备上并行执行



根据历史运行时间，当前可用设备数等信息将用例重新组合并分别到多个设备上并行执行

数据收集与测试报告

测试任务执行



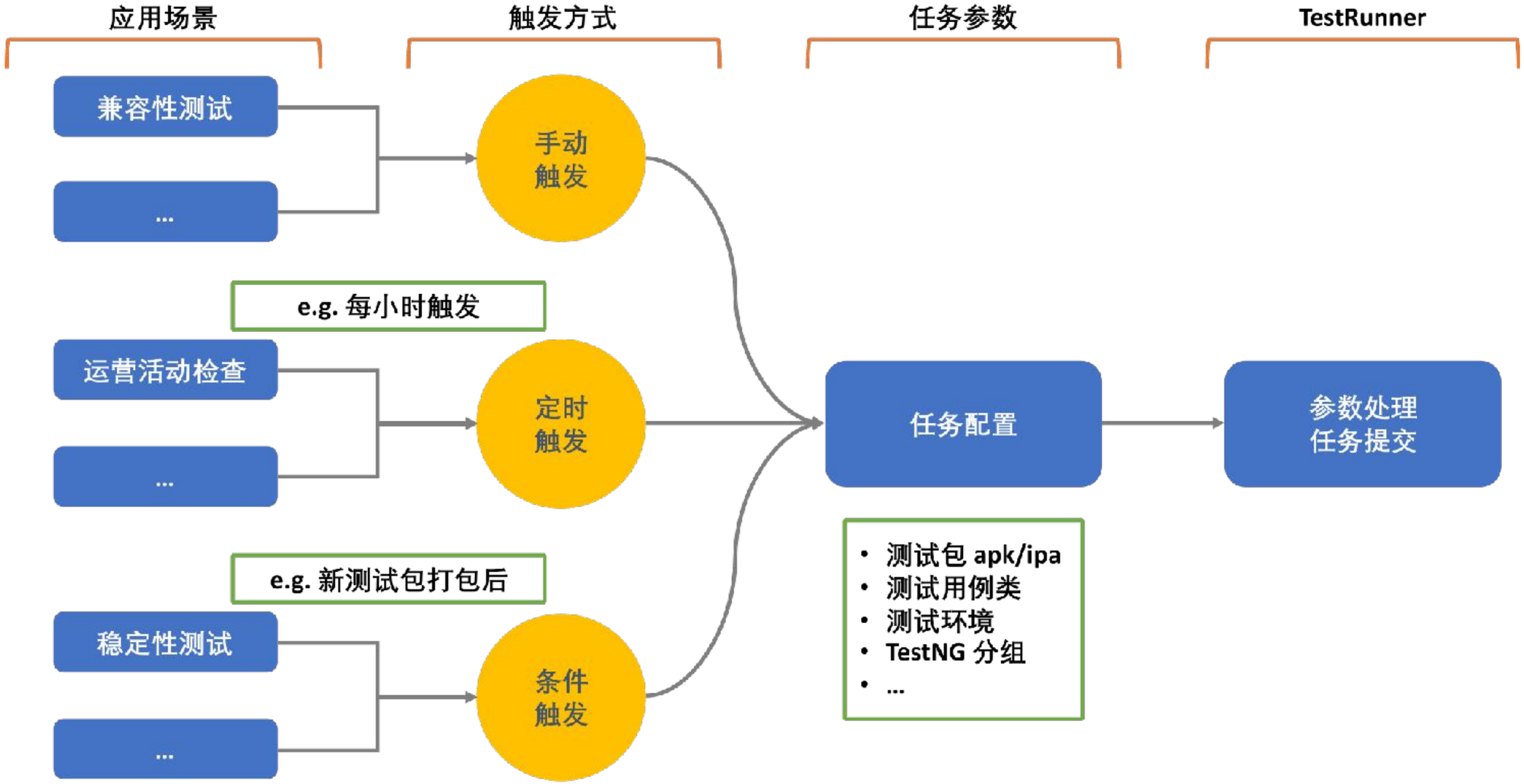
- 用例静态信息指通过注解在用例代码中添加的信息，如：测试步骤等，通过用例解析获取
- 用例执行信息指与用例执行结果相关的信息，如：成功/失败，执行日志等，通过云测接口获取

4. 持续集成

- 持续集成任务的部署及触发方式

任务部署及触发方式

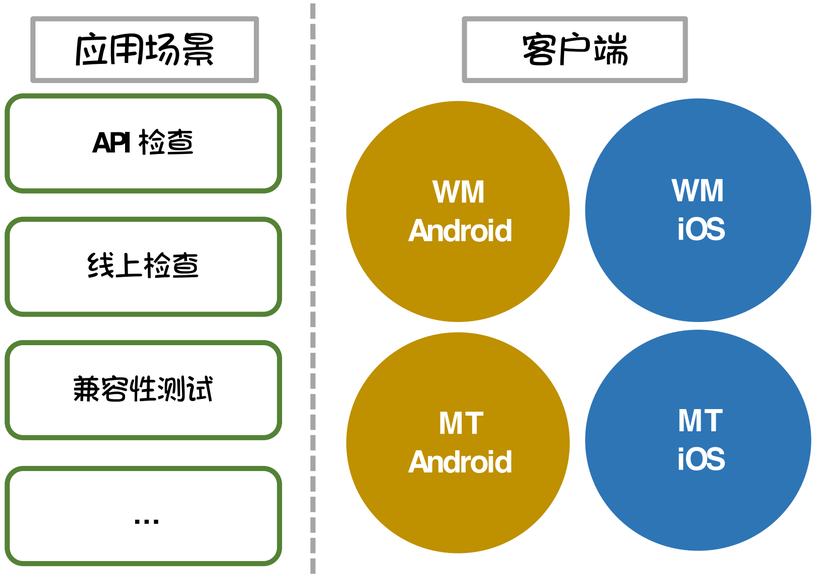
持续集成



集中化的配置方案

持续集成

Gradle 任务配置面临的问题



多应用场景: M

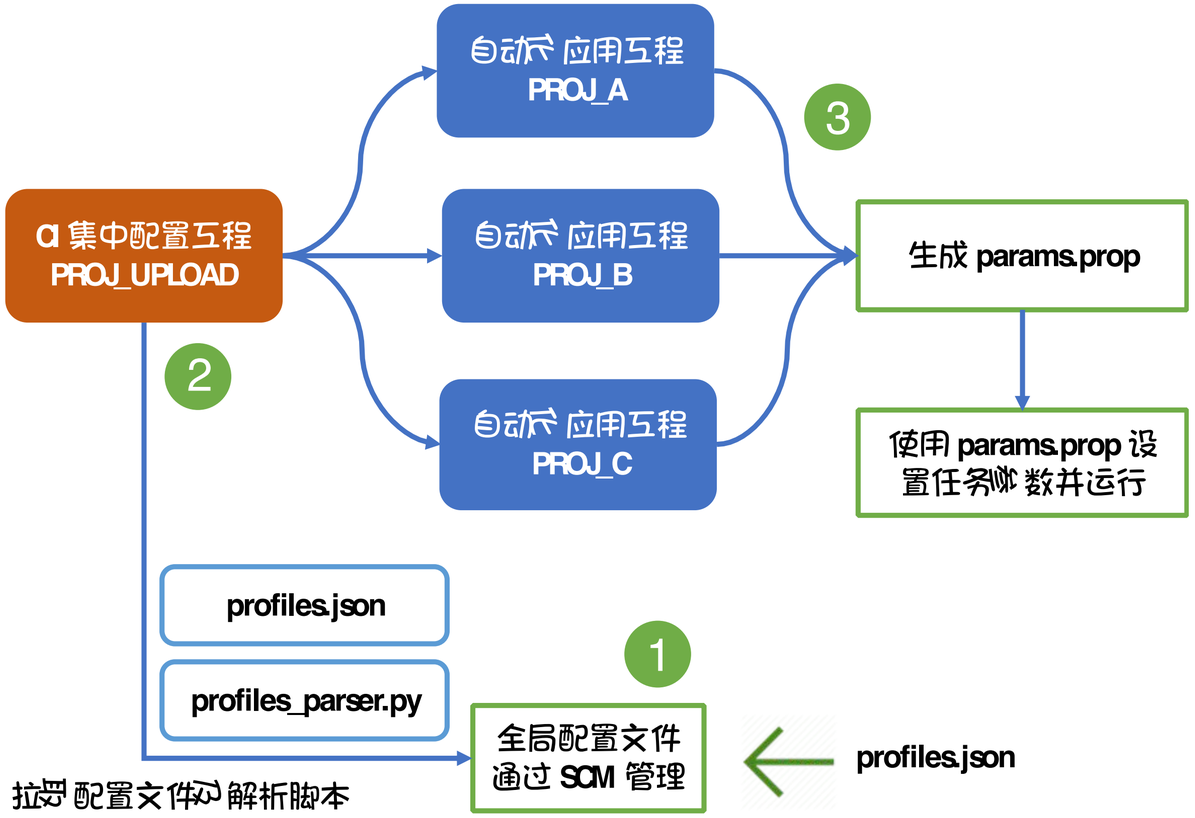
多客户端端: N

需要 Gradle 任务数量: M * N

每个部署需要不同的参数且需持续更新

M × N

Gradle 任务配置的解决方案



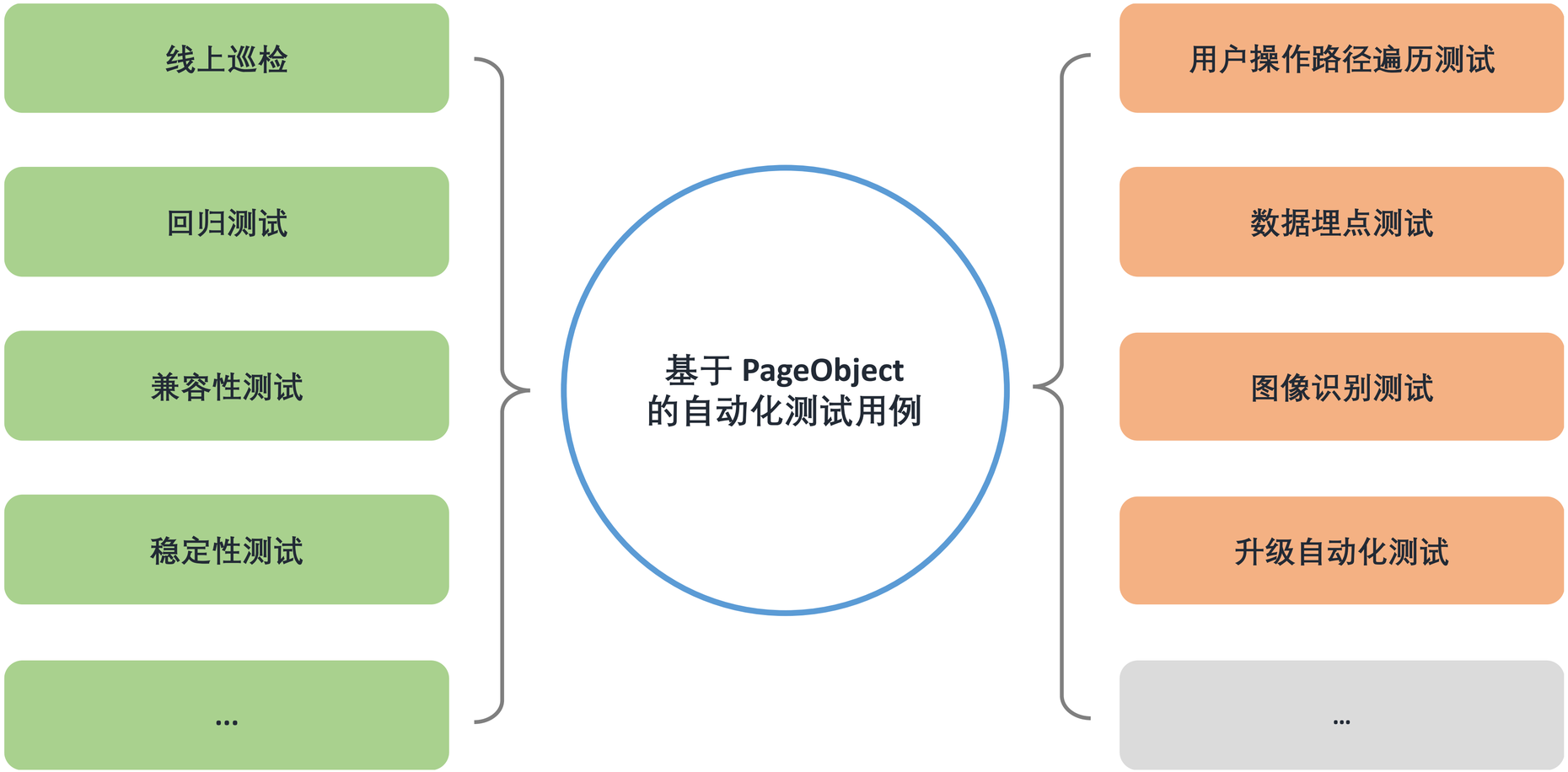
例: Gradle 集中式 部署方案

5. 应用场景

- 常规应用场景及扩展应用场景

常规应用场景及扩展应用场景

应用场景

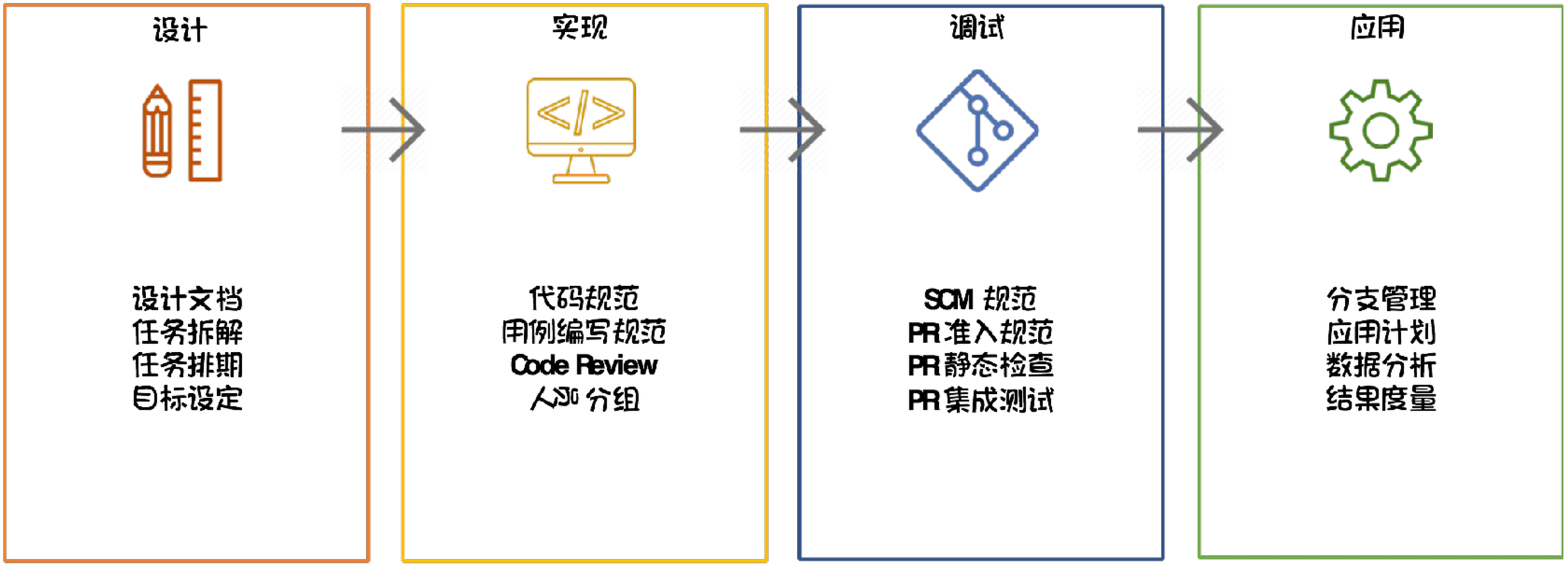


6. 工程体系

- 通过工程管理保证测试用例质量

自动化测试的工程管理

工程体系



自动化测试工程，同时也是完整的严格的软件研发工程

PR 质量保证

工程体系

人工代码 Review

用例编译检查

通过 Jenkins Stash Integration

云测环境下试运行用例

通过 Jenkins Stash Integration

PR Review

PR-Validation
静态检查

PR-Check
PR合入前用例运行检查

例: PR-Validation 与 PR-Check 执行结果

静态代码检查账号
[*BuildFinished* **waimai-c-app-ui-case__pr-check**] 9bb4bccb0f3ed21
✓ **BUILD SUCCESS** - Build *#462* which took *12 min*
Reply · Delete · Create task · Like · 17 May 2018

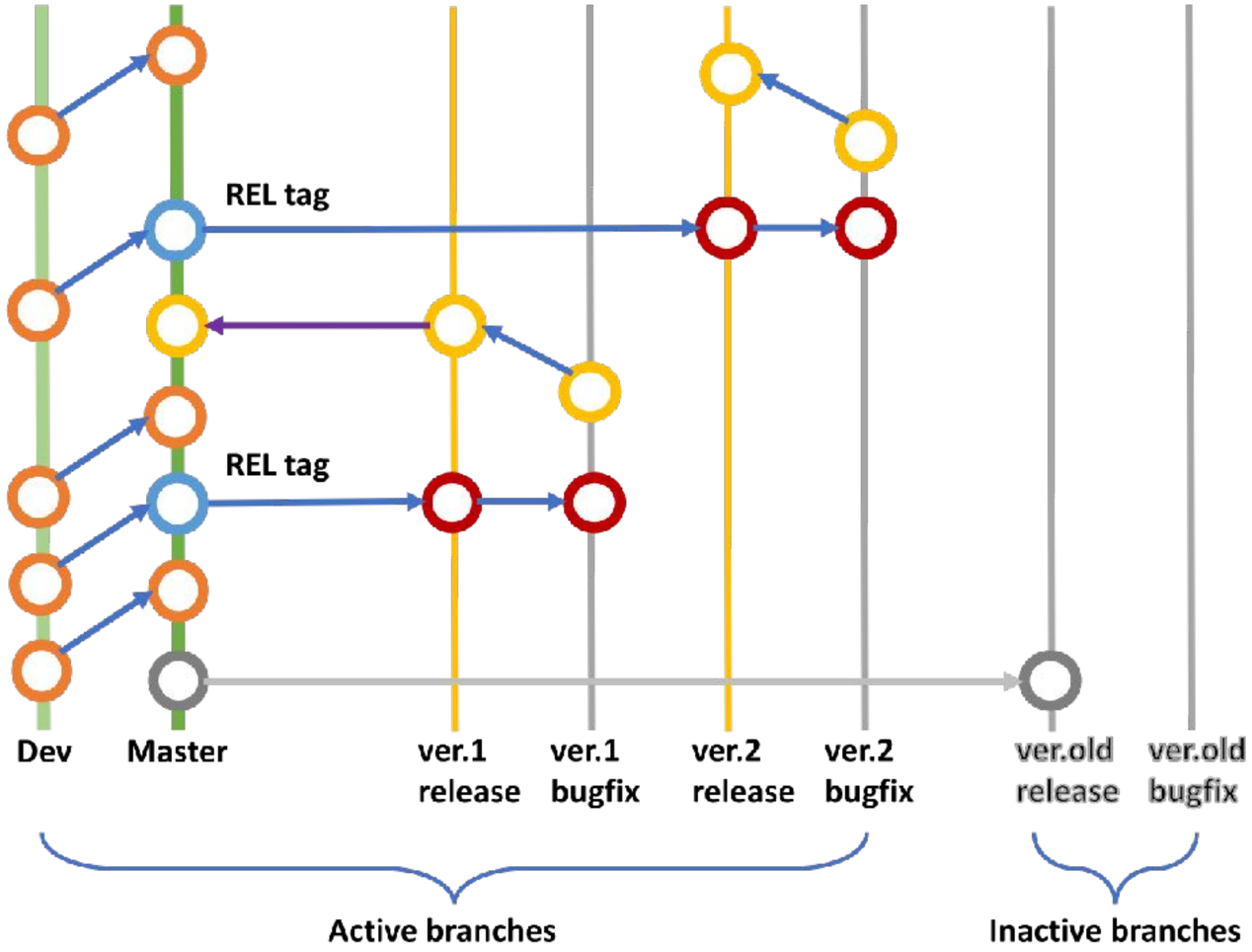
静态代码检查账号
[*BuildFinished* **waimai-c-app-ui-case__validation**] 9bb4bccb0f3ed2
✓ **BUILD SUCCESS** - Build *#576* which took *29 sec*
Reply · Delete · Create task · Like · 17 May 2018

静态代码检查账号
[*BuildStarted* **waimai-c-app-ui-case__validation**] 9bb4bccb0f3ed21
Reply · Delete · Create task · Like · 17 May 2018

静态代码检查账号
[*BuildStarted* **waimai-c-app-ui-case__pr-check**] 9bb4bccb0f3ed21f
Reply · Delete · Create task · Like · 17 May 2018

Git Workflow 及分支管理

工程体系



发布分支管理

- 测试用例分 <开发> / <发布> 分支进行管理
- 用例达到稳定性标准后创建发布分支
- 每个发布分支对应一个历史版本的稳定用例

The Goal of Testing

The goal of testing is not so much to find errors as it is to increase confidence that the code works as expected.

Since it is unrealistic to assume you can find all the bugs, the goal of a quality assurance (QA) plan should be to achieve the greatest possible confidence given the testing resources available.

Java Concurrency in Practice

by Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea

Publisher: Addison-Wesley Professional

Q&A

CODE A BETTER LIFE

一 行 代 码　　亿 万 生 活

更多技术资料

欢迎关注“美团点评技术团队”

美团点评