构建通用的自动化测试平台

腾讯公司 – 吴凯华

2011-04-09

自动化测试面临不同需求

UI界面

中间件

```
libruby-static.a
libruby.dll.a
libfreetype.la
libfreetype.dll.a
libfreetype.a
libmagic.la
libmagic.dll.a
libmagic.a
libssh2.dll.a
libssh2.la
libssh2.a
libcurl.dll.a
libcurl.la
```

Mobile应用

单元测试

```
// Tests the Increment() method.
TEST(Counter, Increment) {
    Counter c;

    // EXPECT_EQ() evaluates its arguments
    // can have side effects.

    EXPECT_EQ(0, c.Increment());
    EXPECT_EQ(1, c.Increment());
    EXPECT_EQ(2, c.Increment());
}
```

。。。。。。

一些自动化测试现状

- **敏捷研发冲击**
  - 自动化测试：先有量的投入才能有回报
  - 产品快速迭代和变更
  - 开发测试人力比
- **开展中的困难**
  - 缺乏优秀测试开发能力的工程师
    - 单纯开发背景的工程师对测试理解认知不够
  - 频繁的临时救火让长期规划开展举步维艰
  - 自动化测试ROI贡献度不够
  - **缺乏有效的测试工具和平台支撑**

而它们都有很多不足和弊端！

- 学习上手难度大
- 测试脚本/代码编写规则不严谨
- 测试数据管理乱或不好
- 不同平台需求支持度不够
- 仍需更多二次封装和开发
- 后期优化时维护代价大
- 天价购买

测试平台的选择和架设

直接决定自动化的效率和成败！

通用测试平台要具备

1. 跨平台支持
2. 各种需求都可接入、开放性好
3. 测试输入输出数据丰富、易管理
4. 测试资源控制和分配管理
5. 测试用例编写规则要清晰和有体系
6. 定位、调试、自我外接性要友好容易

特别提醒（*尤其重要*）：
- 工具接入的规则和机制设计
- 脚本格式和编写规则制定和设计

测试平台概要架构图 –方案1

再加细化规则：

- API开发规则
- 工具接入规则
- 脚本管理规则
- 测试数据/结果管理规则
- 调度执行规则
- 版本变更管理规则

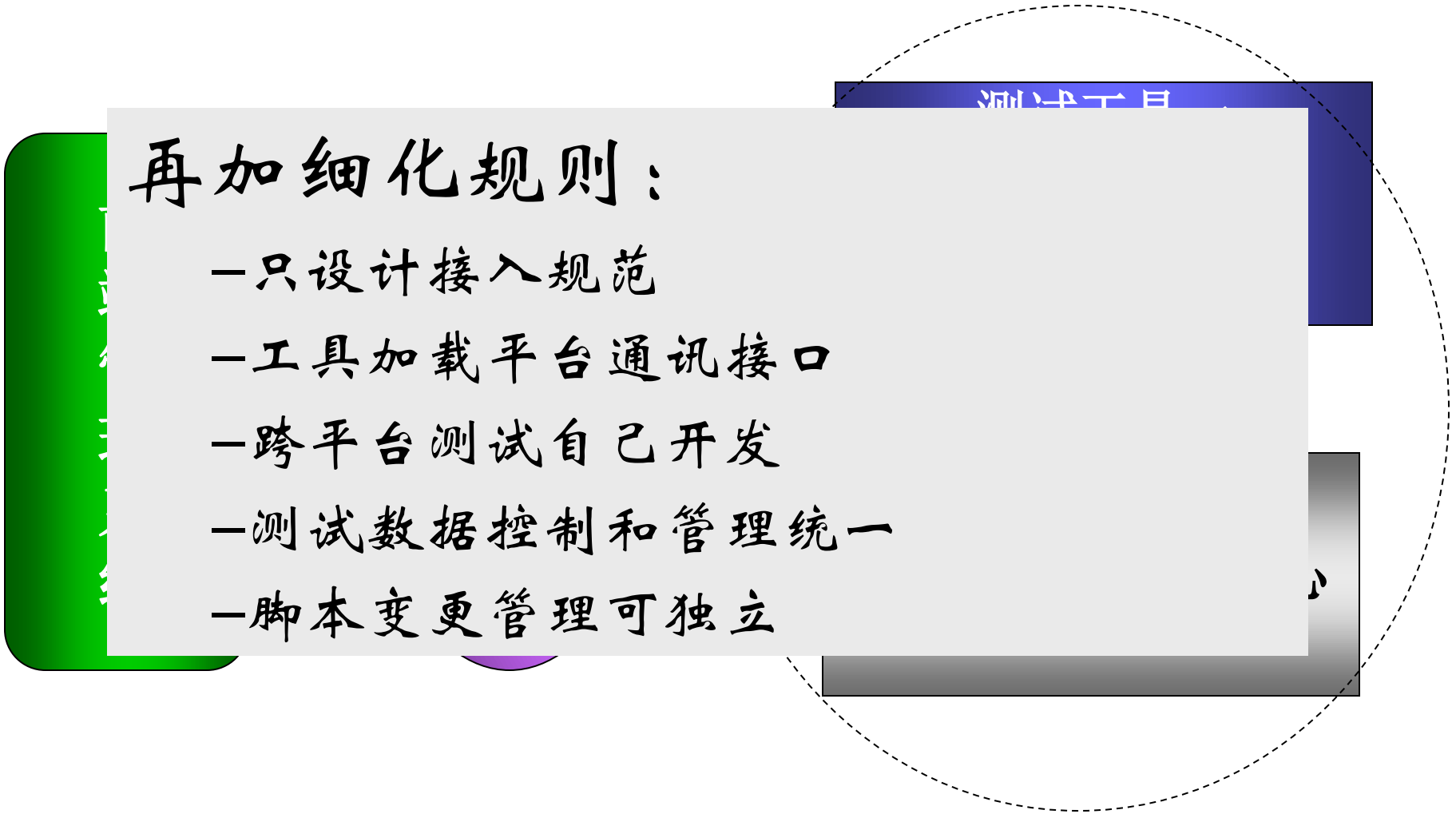
(Windows)

-2(Linux)

r-3(Mac)

N(Android)

测试平台概要架构图 –方案2



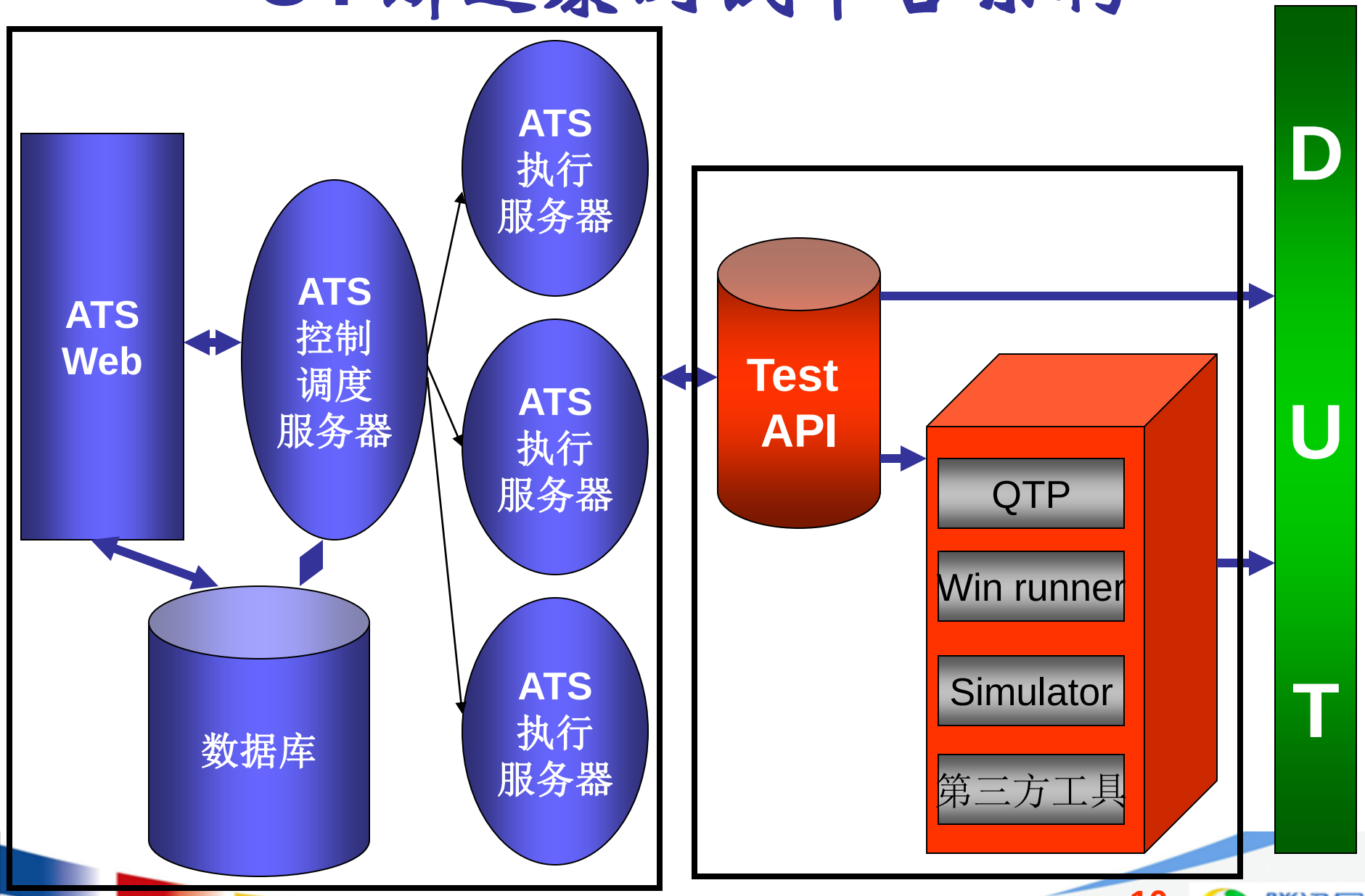
再加细化规则：

- –只设计接入规范
- –工具加载平台通讯接口
- –跨平台测试自己开发
- –测试数据控制和管理统一
- –脚本变更管理可独立

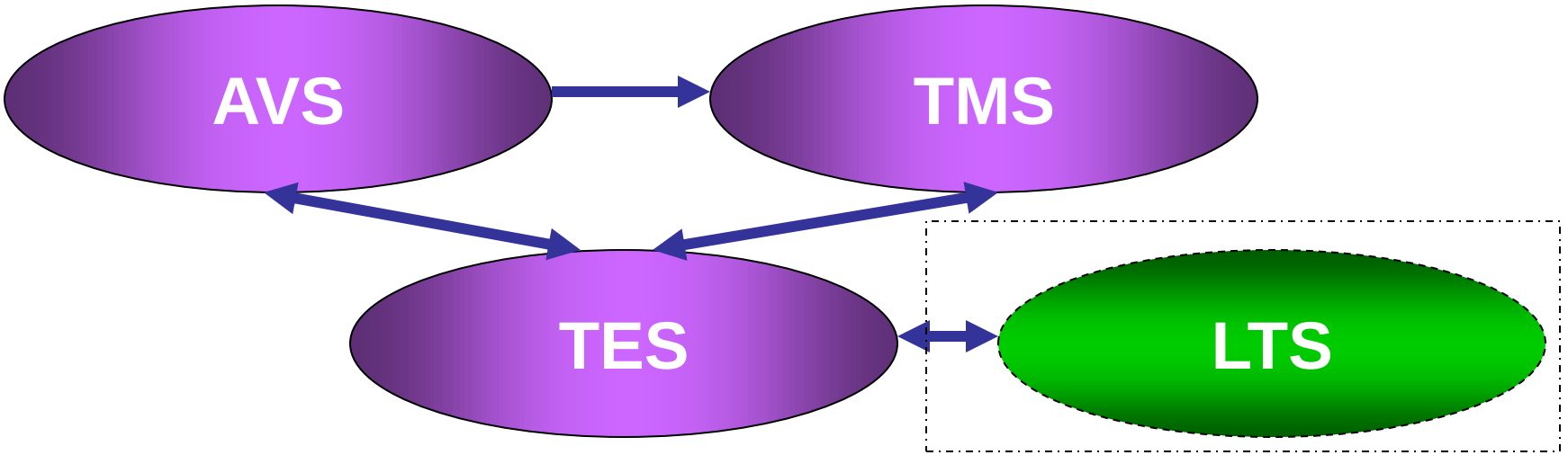
两种平台的差异

- 方案一 适合企业自建使用
  - 平台本身要完成所有跨平台的测试驱动和调度
    - 如LINUX、Windows、iOS、Android下自动化执行调度
  - 平台开发工作量更大
    - 可分期支持（主要是分布式执行agent的开发）
  - 后续的使用接入成本小
    - 不同平台上的工具不需额外在平台内注册
    - 工具以平台内API方式驱动（API为测试脚本的步骤）
    - API要在前端提供管理机制
- 方案二 适合商业推广（如QC）
  - 平台专注前端管理和接入规则的开发
  - 工具同时要负责内置平台接口和自己支持跨平台
  - 测试资源控制和管理调度比方案一更加方便组织
    - 当然方案一仍然可独立对资源做管控

UT斯达康测试平台架构



Lucent 测试平台系统框架



- **AVS: Automated Verification System**
  - 负责任务创建、管理和执行调度
- **TMS: Test Management System**
  - 负责脚本编写、(版本)管理，也可自动执行和计划管理
  - 功能和自动化统一集成
- **TES: Test Execution System**
  - 资源控制和申请/管理
  - Lab Test Real Environment， TMS脚本执行环境
- **LTS: Laboratory Test System**
  - 各种测试工具和调试工具集

前端管理系统功能介绍

前台功能（B/S or C/S）

- 自动化测试脚本/用例管理
- 自动化测试任务管理
- API管理
- 测试数据和结果管理及度量

自动化脚本管理

- 高端功能：IDE方式支持
- 可导入导出模式支持
  - 便于离线开发，方便用户使用自己喜欢的IDE环境进行开发
- 设置脚本的格式和规则要求
  - 推荐xUnit框架模式思路，但要更简洁和容易理解
- 可动态和单步调试
- Set/Suite和Case概念要体现
  - 利用公共测试数据管理和提取

几种自动化脚本风格

```
TESTSUITE SuiteID Suite_Name {
    测试套功能介绍
}{
    测试套参数定义，Case里的公共全局参数
}{
TEST 【SuiteName SuiteTestFunction】 {
    ……
}

TEST 【SuiteName SuiteTestFunction】 {
    ……
}
}{
    用例参数/数据定义和声明
}{
    用例初始化部分
}{
    用例执行主体和具体测试步骤
}{
    用例清除部分
}
```

API管理和开发

- Javadoc风格管理和提供API帮助指引

自动化测试任务管理

- 任务管理
  - 增删改查
- 任务指定时间执行
- 任务并发执行
- 任务控制（暂停、Kill、调试模式等）
- 测试数据在任务里的可动态调整和抽取

自动化测试任务管理（demo）

Pass	Fail	Error	Killed	Total
1	10	0	0	11

CaseNum	CaseId	CaseName	Result
1	85551	Zero Usage Service Stat	FAIL
2	85553	New User Usage Stat	FAIL
3	99999999	Charge Detail Stat(Not in TPMS)	FAIL
4	85564	PostPaid Balance Stat	FAIL
5	85565	PostPaid Owe Grade Stat	PASS
6	85568	Charge Deduction Stat	FAIL
7	85569	Penalty Stat	FAIL
8	85570	Penalty Deduction Stat	FAIL
9	85567	Payment Detail Stat	FAIL
10	85572	Overpay Monthly Stat	FAIL
11	85573	PrePaid User Deposit Stat	FAIL

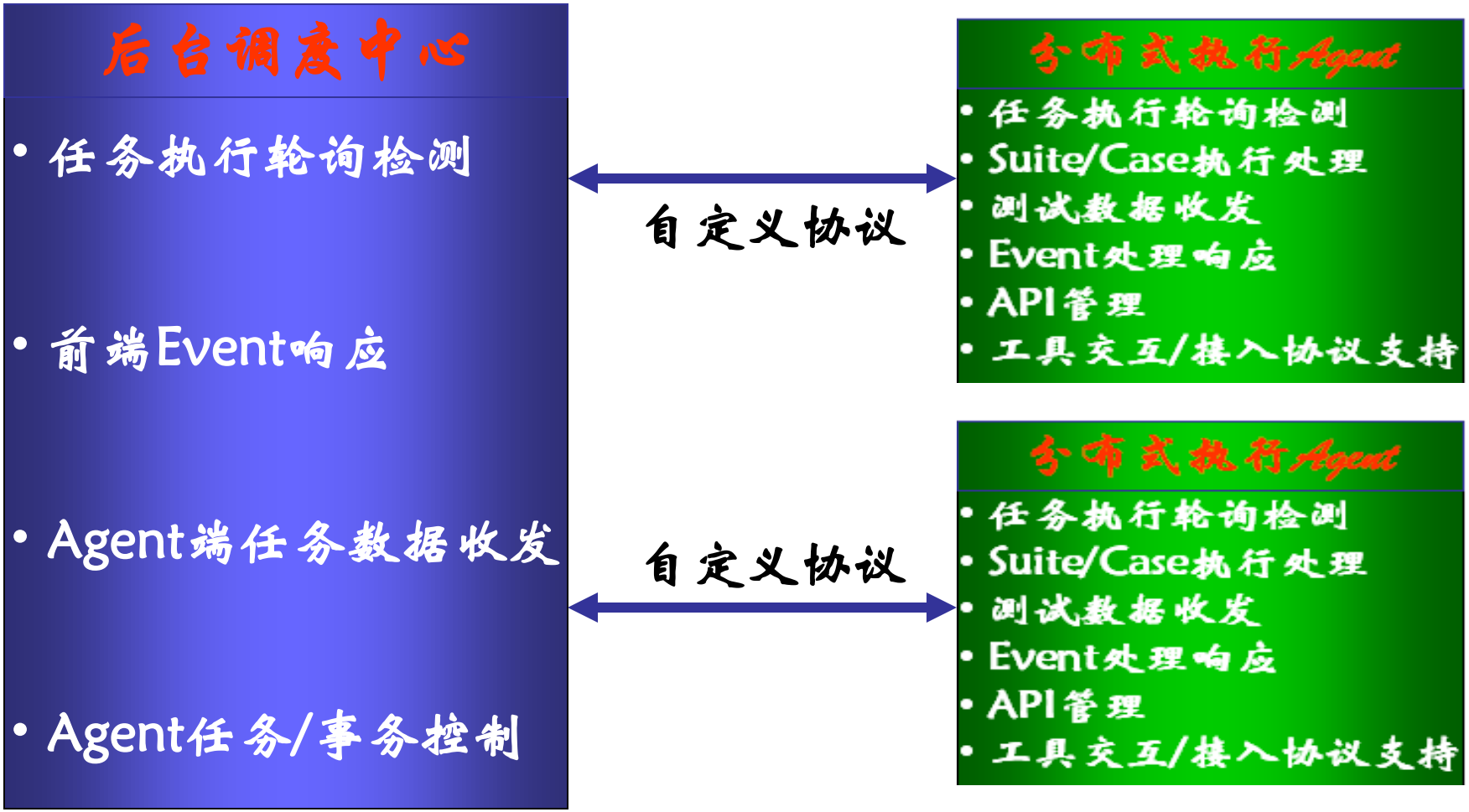
log info, caller single_update (update job_suite suite set state = 'R', start_time = sysdate, processid = 22556, state_comments = '' where jobid = 'he2004

log debug, caller commit, time 18:42:15, ATSOracle commit OK

Test Suite 1044, /home/ATS/2004-11-26/he20041126184210/2000021.suite.

后台调度执行中心

后端架构概要功能


后台调度中心
• 任务执行轮询检测
• 前端Event响应
• Agent端任务数据收发
• Agent任务/事务控制
自定义协议
分布式执行Agent
• 任务执行轮询检测
• Suite/Case执行处理
• 测试数据收发
• Event处理响应
• API管理
• 工具交互/接入协议支持
自定义协议
分布式执行Agent
• 任务执行轮询检测
• Suite/Case执行处理
• 测试数据收发
• Event处理响应
• API管理
• 工具交互/接入协议支持

统一调度Server设计和功能

- 提供前端时时执行任务结果展示和控制接口
- 多任务分发和并行执行/调度能力
  - FSM或多线程支持的架构都OK
- 分布式设计模式
- Client/Server间通讯的数据结构/协议要设计好
- 性能支持要到位

后端执行Agent设计难点

- 兼容不同操作系统、同时做到功能接口统一
- Suite/Case执行Controller/Framework
- API安装/管理和调用规则

第三方工具接入方案介绍

平台支持接入设计要求

- 可以允许使用或定义不同脚本格式和脚本语言
  - 如QTP/VBScript、Win Runner/TSL等
- 平台可以把测试数据和对应Case映射关系保存在平台DB，细化的不同脚本以SVN/CVS等做保存和管理
- 平台以传入对应case信息和测试数据及结果封装调度API
- 平台可以不同脚本或风格实施平台侧脚本管理

QTP/Win Runner接入方案

- 基于QTP/Win Runner开发一个内置框架
  - 可脱离开测试平台单独运行的自动化框架
  - 数据管理规范/格式统一
  - 脚本编写自定义固定格式
  - OR库和GUI Map文件方式管理和动态加载
  - 执行日志和结果按照规定方法上报和收集
- 封装一个长/短连接API完成用例脚本和Test Data Input的发送，驱动QTP/Win Runner执行用例
  - TSL或VBScript脚本借助SVN管理，执行时由框架动态按需拉取
- 封装一个回传Test Result和入库的API
- Win Runner或QTP脚本可轻松在测试平台里实施管理甚至编写
- 分布式执行Agent侧的接入支持

测试平台API

- parseTestInputData
- checkoutScriptFromSVN
- checkoutORFileFromSVN
- createQTPTestConfigFile
- openQTP
- runTest
- getCaseExecResult
- getCaseExecDetailLog
- exitQTP

- 两种使用方式：
  - 供执行Agent调度
  - 或作为提供不同测试数据，但格式一致的用例编写

QTP框架内API

- logReport (logType, logMsg)
  - logReport “ERROR”, “Failed to execute this step”
  - logReport “FAIL”, “The case execution failed”
  - logReport “STEP”, “The current executing step is xxx”
  - 上述三种日志类型是在Case里频繁使用的
- paramName=GetParam(“QQNumber”)
  - GetParam会从配置文件里获得此Case的参数值，如果此参数在Case_Param里没有指定，GetParam还会遍历Suite_Param表来试图获得此参数值
  - 获得的值赋给paramName变量
- reportResult (caseResult, caseResultMsg)
  - reportResult “PASS”, “The case execution succeeded”
  - reportResult “FAIL”, “The execution failed, message is xxx”

一个简单的测试用例

```
Function openQzoneWeb()
        url=getParam("QZone_URL")
        logReport "STEP", "Opening the URL: " + url
        ret=openWeb(url)
        If ret <>0  Then
                reportResult "FAIL","Failed to open the URL: " + url
        Else
                reportResult "PASS","The url has been successfully
        opened"
        End If
        Exit Function
End Function
```

Q&A

QCon

杭州站 · 2011年10月20日~22日

www.qconhangzhou.com（6月启动）

QCon北京站官方网站和资料下载

www.qconbeijing.com

全球企业开发大会

THE ANNUAL INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE