Deep in iOS Testing

mqc.aliyun.com

阿里云测移动质量中心
2016/08/12

大家怎么测试iOS应用？

手工测试

零门槛、无部署维护成本

周期长、难回归、测试难以标准化

VS

自动化测试

持续集成、随时回归、避免风险累积

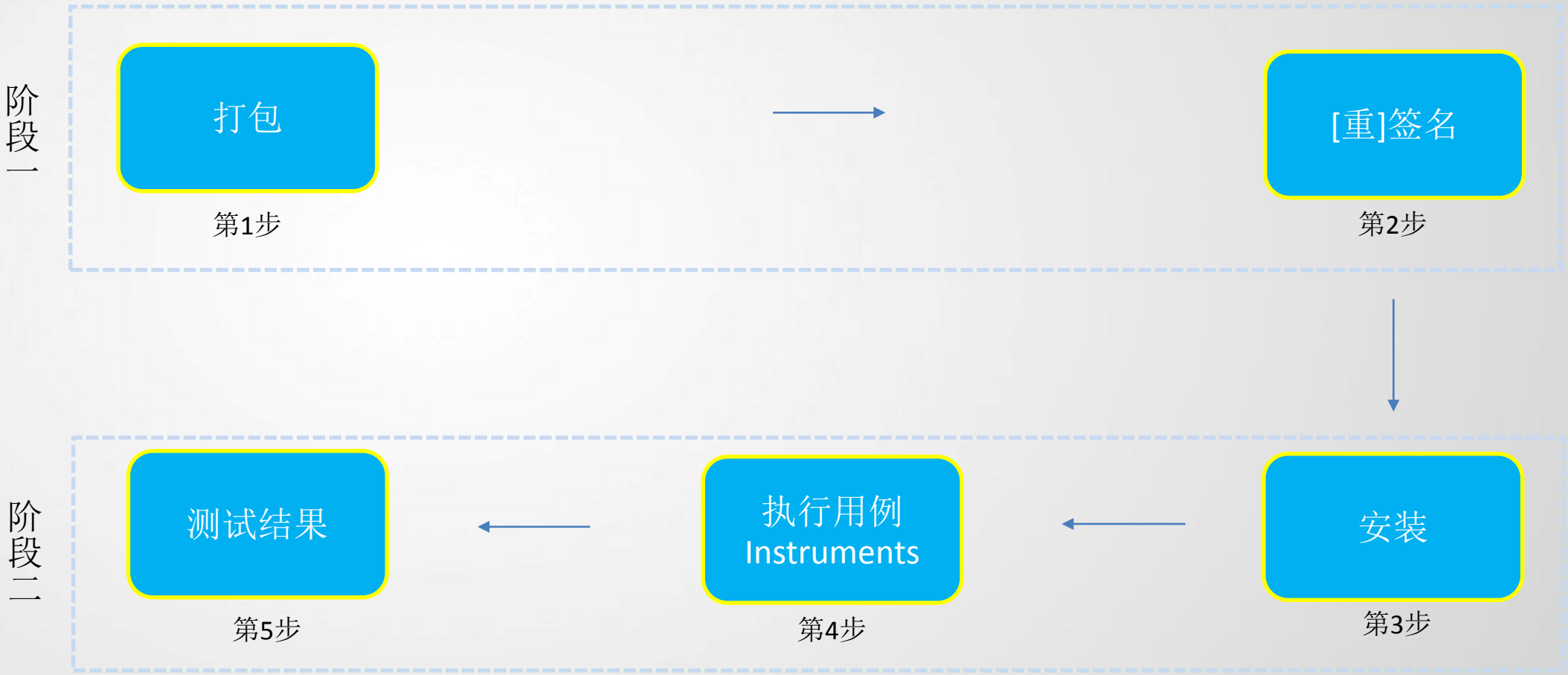
技术门槛高、有维护成本

iOS 自动化测试

目录

让测试流程自动化

自动化测试测试流程



阶段一

1. 自动编译打包

环境配置

Provision Profile
Developer Certificate

自动打包

shenzhen

2. 提示

Shenzhen 可以直接部署包到包括fir.im在内的各个渠道

重签名： **ota-tools ipa-sign** [unzip ipa -> remove signature -> copy mobileprovision -> codesign -> zip to new ipa]

编码错误：打包前设置本地环境的编码为UTF-8

阶段二 - 自动安装

1. 自动安装iOS App

Start Session
Start AFC service
Push App

Start Session
Start installation_proxy Service
Send Installation Cmd

Start Session
Start debugServer Service
Send startup Cmd

2. 替代工具

3. 可能遇到的坑

无法安装： Provisioning Profile没添加目标设备

Monkey测试

Monkey测试

1. 为什么使用Monkey?

通用性强：可以适用于所有应用

简单有效：对于发现应用崩溃等稳定性问题有奇效

维护成本低：相比功能用例维护，成本极低

2. 如何在iOS测试中使用Monkey?

通过UIAutomation 脚本，并用Instruments驱动。

开源工具：ui-auto-monkey。

3. 定制Monkey

Monkey进入特定界面退不出来。

ANR (App Not Responding) 出现。

系统弹窗，比如“是否接受应用通知弹窗”。

iOS 崩溃分析

iOS崩溃分析

1. 报告头(Header)

报告头包含了应用程序以其运行环境的一些基本信息。

```
Incident Identifier: E6EBC860-0222-4B82-BF7A-2B1C26BE1E85
CrashReporter Key: 6196484647b3431a9bc2833c19422539549f3dbe
Hardware Model: iPhone6,1
Process: TheElements [4637]
Path: /private/var/mobile/Containers/Bundle/Application/5A9E4FC2-D03B-4E19-9A91-
104A0D0C1D44/TheElements.app/TheElements
Identifier: com.example.apple-samplecode.TheElements
Version: 1.12
Code Type: ARM (Native)
Parent Process: launchd [1]

Date/Time: 2015-04-06 09:14:08.775 -0700
Launch Time: 2015-04-06 09:14:08.597 -0700
OS Version: iOS 8.1.3 (12B466)
Report Version: 105
```

iOS崩溃分析

2. 异常代码(Exception Codes)

```
Exception Type: EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000
Triggered by Thread: 0
```

2.1 组成结构

异常类型(Exception Type)、异常子类型(Exception Subtype)、处理器的异常代码(processor-specific Exception Codes)、 其他Crash信息的字段、触发Crash的线程索引

2.2 常见异常类型

Bad Memory Access： SIGSEGV、SIGBUS、SEGV、EXC_BAD_ACCESS

Abnormal Exit： SIGABRT、EXC_CRASH

其它异常类型： 0x8badf00d、0xdead10cc、0xdeadfa11等等

程序启动或者恢复时间过长被watch dog终止

iOS崩溃分析

3. 应用详情(Application Specific Information)

有些Crash出现时，会产生额外的信息，帮助更好地了解应用程序终止时的运行环境。

```
Application Specific Information:
MyApp[134] was suspended with locked system files:
/private/var/mobile/Library/AddressBook/AddressBook.sqlitedb
```

4. 回溯(Backtrace)

调用栈

```
Thread 0 name: Dispatch queue: com.apple.main-thread
Thread 0 Crashed:
0   TheElements               0x0000000100063fdc -[AtomicElementViewController
myTransitionDidStop:finished:context:] (AtomicElementViewController.m:201)
1   UIKit                     0x000000018ca5c2ec -[UIViewAnimationState sendDelegateAnimationDidStop:finished:] +
184
2   UIKit                     0x000000018ca5c1f4 -[UIViewAnimationState animationDidStop:finished:] + 100
3   QuartzCore                0x000000018c380f60 CA::Layer::run_animation_callbacks(void*) + 292
4   libdispatch.dylib         0x0000000198fb9368 _dispatch_client_callout + 12
5   libdispatch.dylib         0x0000000198fbd97c _dispatch_main_queue_callback_4CF + 928
6   CoreFoundation            0x000000018822dfa0 __CFRUNLOOP_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__ + 8
7   CoreFoundation            0x000000018822c048 __CFRunLoopRun + 1488
8   CoreFoundation            0x00000001881590a0 CFRunLoopRunSpecific + 392
9   GraphicsServices          0x00000001912fb5a0 GSEventRunModal + 164
10  UIKit                     0x000000018ca8aaa0 UIApplicationMain + 1484
11  TheElements               0x0000000100005d800 main (main.m:55)
12  libdyld.dylib             0x0000000198fe2a04 start + 0

Thread 1 name: Dispatch queue: com.apple.libdispatch-manager
Thread 1:
0   libsystem_kernel.dylib    0x00000001990e0c94 kevent64 + 8
1   libdispatch.dylib         0x0000000198fc897c _dispatch_mgr_invoke + 272
2   libdispatch.dylib         0x0000000198fbb3b0 _dispatch_mgr_thread + 48

...
```

iOS崩溃分析

5.线程状态(Thread State)

发生Crash的线程的状态，即寄存器和寄存器的值

```
Thread 0 crashed with ARM Thread State (64-bit):
    x0: 0x0000000000000000   x1: 0x0000000000000000   x2: 0x0000000000000000   x3: 0x00000001995f8020
    x4: 0x0000000000000000   x5: 0x0000000000000001   x6: 0x0000000000000000   x7: 0x0000000000000000
    x8: 0x0000000000000000   x9: 0x0000000000000015  x10: 0x0000000199601df0  x11: 0x0000000b0000000f
   x12: 0x00000001741e8700  x13: 0x000001a5995f5779  x14: 0x0000000000000000  x15: 0x0000000044000000
   x16: 0x00000001989724d8  x17: 0x0000000188176370  x18: 0x0000000000000000  x19: 0x00000001701dda60
   x20: 0x0000000000000001  x21: 0x0000000136606e20  x22: 0x00000001000f6238  x23: 0x0000000000000000
   x24: 0x000000019cc640a8  x25: 0x0000000000000020  x26: 0x0000000000000000  x27: 0x0000000000000000
   x28: 0x000000019cc577c0  fp: 0x000000016fd1a8d0   lr: 0x00000001000effcc
    sp: 0x000000016fd1a860   pc: 0x00000001000effdc cpsr: 0x60000000
```

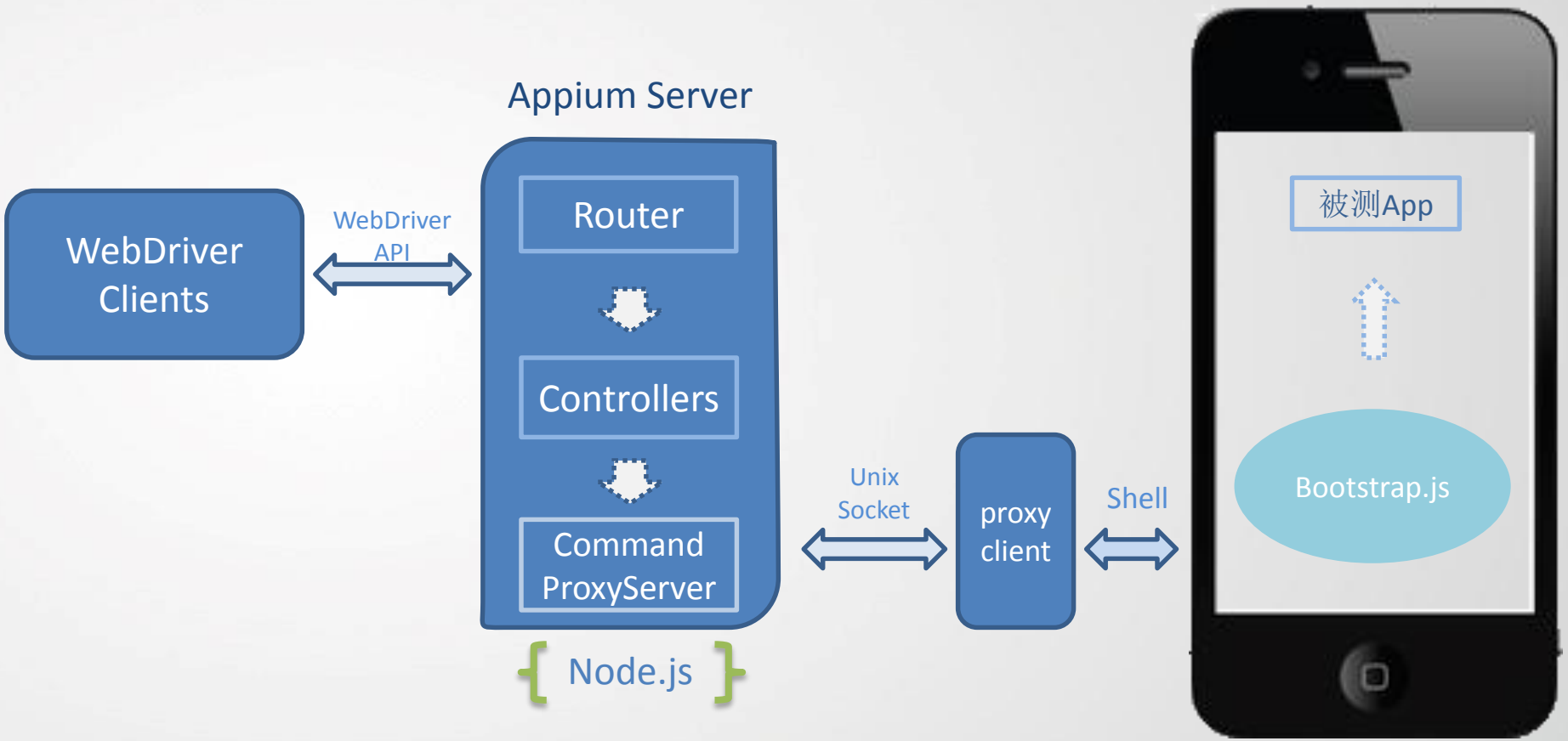
6. 二进制映像(Binary Images)

发生Crash时，被装载进进程内存空间的依赖库或者模块

```
Binary Images:
0x100058000 - 0x10006bfff TheElements arm64 <77b672e2b9f53b0f95adbc4f68cb80d6>
/var/mobile/Containers/Bundle/Application/CB86658C-F349-4C7A-B73B-CE3B4502D5A4/TheElements.app/TheElements
```

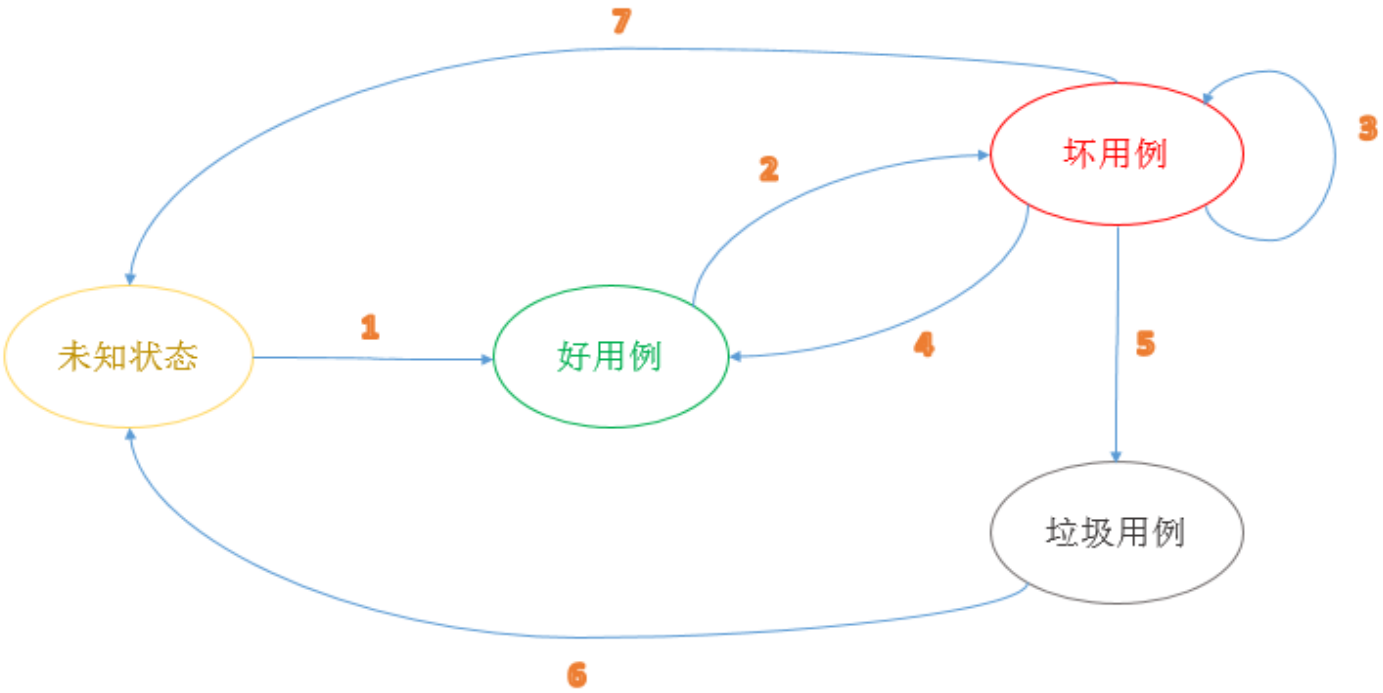
功能测试最佳实践

Appium的iOS架构图



performTaskWithPathArgumentsTimeout

如何管理自动化测试用例



测试用例生命周期

自动化测试最佳实践

持续集成

通过“CI”（持续集成）发现“坏用例”

防微杜渐

守护自动化用例，避免“破窗理论”发生

解决并行执行问题

避免并行执行时，系统出现冲突，导致错误

避免测试脚本过长

脚本太长导致用例不稳定，尽量控制在30步以内

避免用例相互依赖

用例保持独立，相互依赖会导致整个用例集不稳定

使用异步等待

异步等待UI操作，避免过早或过晚执行下一步

避免执行环境差异

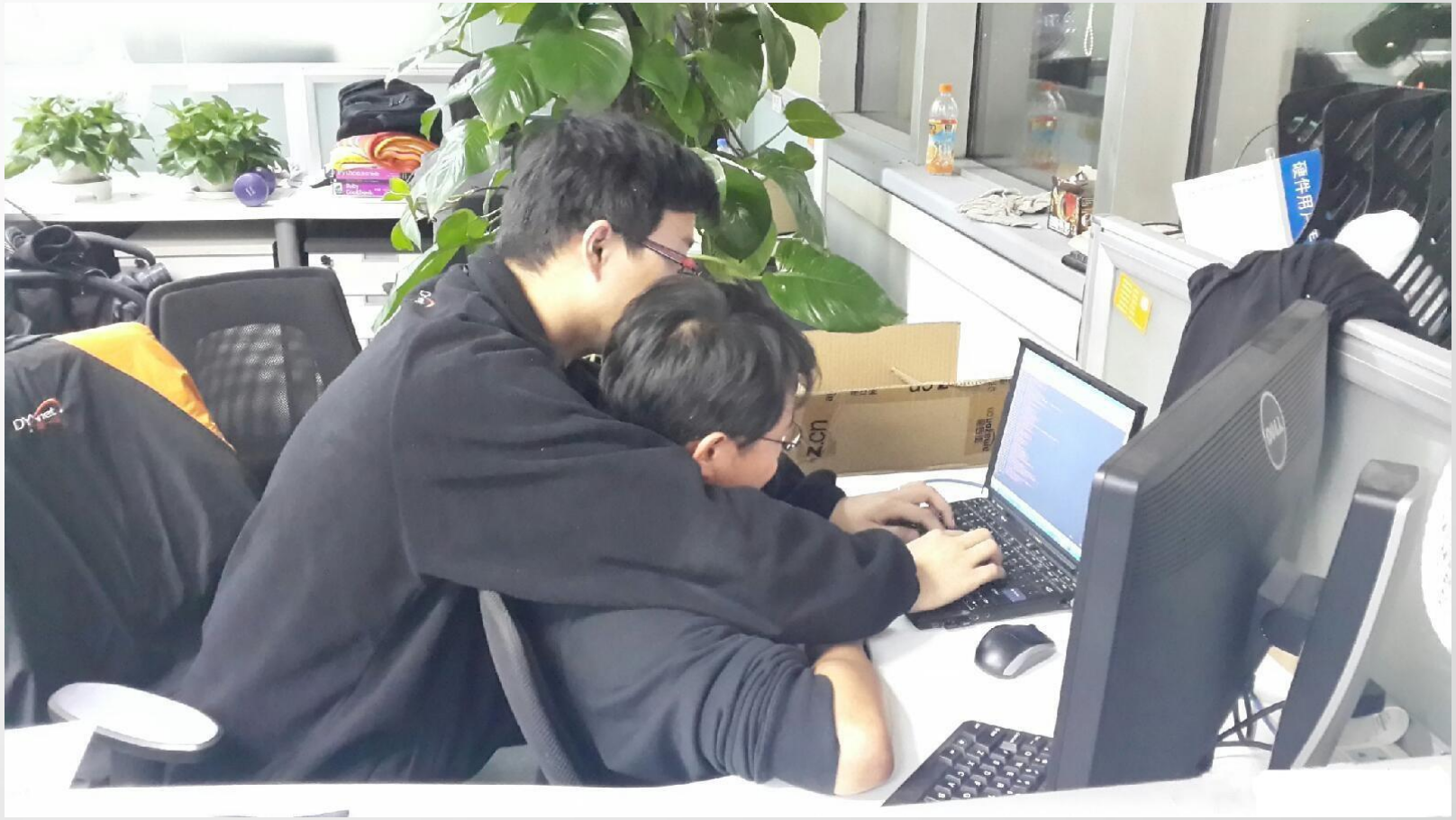
确保本地执行环境和CI环境一致，避免异常发生

提高用例编写水平

用例不是简单脚本，也需要封装和维护

自动化测试最佳实践

把开发同学变成好朋友



谢谢大家