29 | 自动化测试：如何把Bug杀死在摇篮里？

2019-05-04 宝玉 来自北京

《软件工程之美》

你好，我是宝玉。前不久我所在项目组完成了一个大项目，把一个网站前端的 jQuery 代码全部换成 React 代码，涉及改动的项目源代码文件有一百多个，变动的代码有几千行，最终上线后出乎意料的稳定，只有几个不算太严重的 Bug。

能做到重构还这么稳定，是因为我们技术水平特别高吗？当然不是。还是同样一组人，一年前做一个比这还简单的项目，上线后却跟噩梦一样，频繁出各种问题，导致上线后不停打补丁，一段时间才逐步稳定下来。

这其中的差别，只是因为在那次失败的上线后，我们总结经验，逐步增加了自动化测试代码的覆盖率。等我们再做大的重构时，这些自动化测试代码就能帮助我们发现很多问题。

当我们确保每一个以前写好的测试用例能正常通过后，就相当于把 Bug 杀死在摇篮里，再配合少量的人工手动测试，就可以保证上线后的系统是稳定的。

其实对于自动化测试，我们专栏已经多次提及，它是敏捷开发能快速迭代很重要的质量保障，是持续交付的基础前提。

所以今天我将带你一起了解什么是自动化测试，以及如何在项目中应用自动化测试。

为什么自动化测试能保障质量?

自动化测试并不难理解，你可以想想人是怎么做测试的：首先根据需求写成测试用例，设计好输入值和期望的输出，然后按照测试用例一个个操作，输入一些内容，做一些操作，观察是不是和期望的结果一致，一致就通过，不一致就不通过。

自动化测试，就是把这些操作，用程序脚本来完成的，本质上还是要输入和操作，要检查输出是不是和期望值一致。只要能按照测试用例操作和检查，其实是人来做还是程序来做，结果都是一样的。

不过，自动化测试有一个手工测试没有的优势，那就是可以直接绕过界面，对程序内部的类、函数进行直接测试，如果有一定量的自动化测试代码覆盖，相对来说软件质量是更有保障的。

而且，一旦实现了自动化，每测试一次的成本其实大幅降低了的，几百个测试用例可能几分钟就跑完了。尤其是每次修改完代码，合并到主干之前，把这几百个测试用例跑一遍，可以有效地预防“修复一个 Bug 而产生新 Bug”的情况发生。

但现阶段，自动化测试还是不能完全代替手工测试的，有些测试，自动化测试成本比手工测试成本要高，比如说测试界面布局、颜色等，还是需要一定量的手工测试配合。

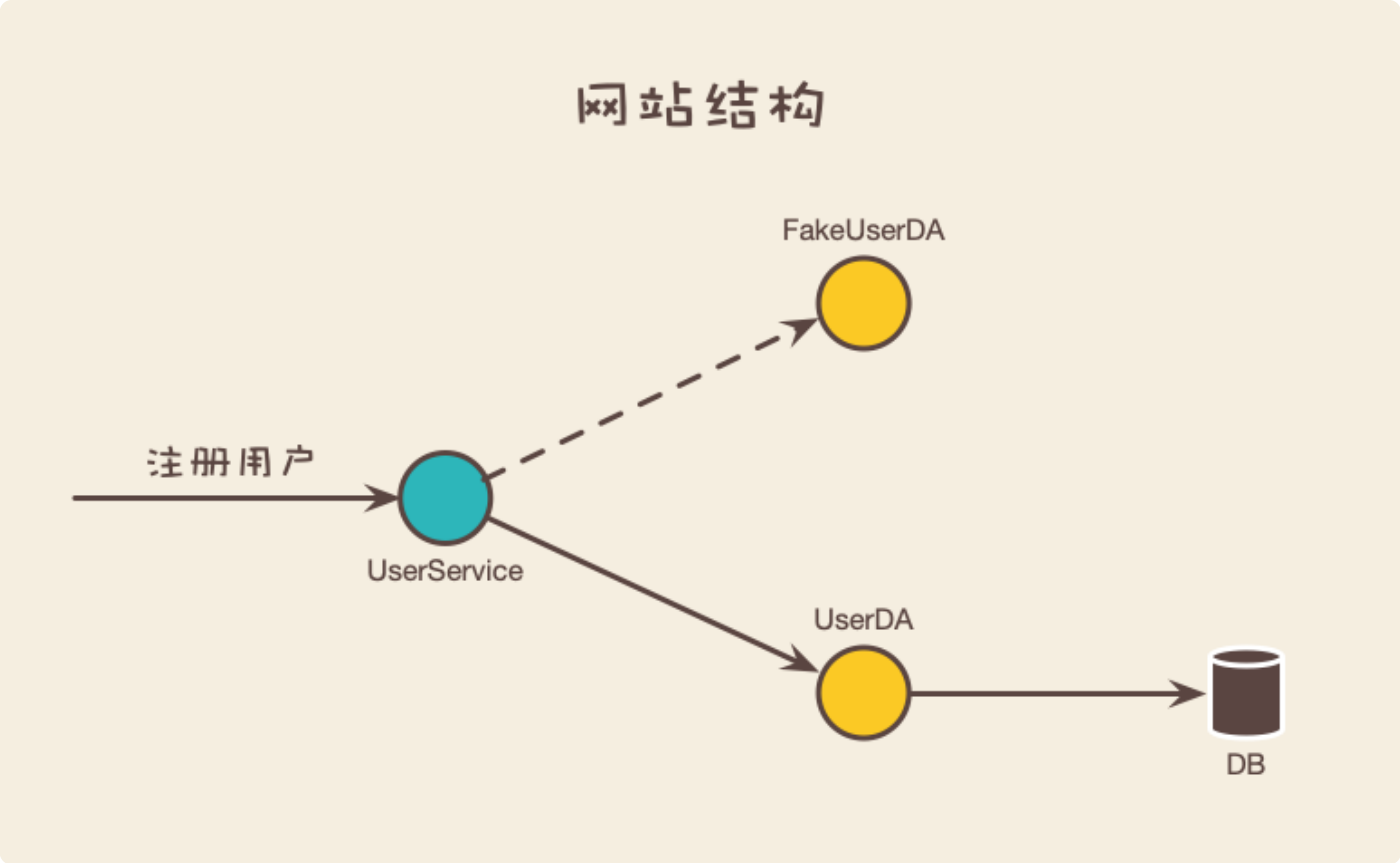
有哪些类型的自动化测试?

现在说到自动化测试，已经有很多的概念，除了大家熟悉的单元测试，还有像集成测试、UI 测试、端到端测试、契约测试、组件测试等。而很多时候同一个名字还有不同的解读，很容易混淆。

在对自动化测试类型的定义方面，Google 的分类方法我觉得比较科学：根据数据做出决策，而不仅仅是依靠直觉或无法衡量和评估的内容。Google 将自动化测试分成了三大类：小型测

试、中型测试和大型测试。

假设我们有一个网站，是基于三层架构（如下图所示），业务逻辑层的类叫 UserService 类，数据访问层的类叫 UserDA，我们将以用户注册的功能来说明几种测试的区别 。



小型测试

小型测试是为了验证一个代码单元的功能，例如针对一个函数或者一个类的测试。我们平时说的单元测试就是一个典型的小型测试。

比如说 UserService 这个类，有一个注册用户的函数，现在要对它写一个单元测试代码，那么看起来就像下面这样：

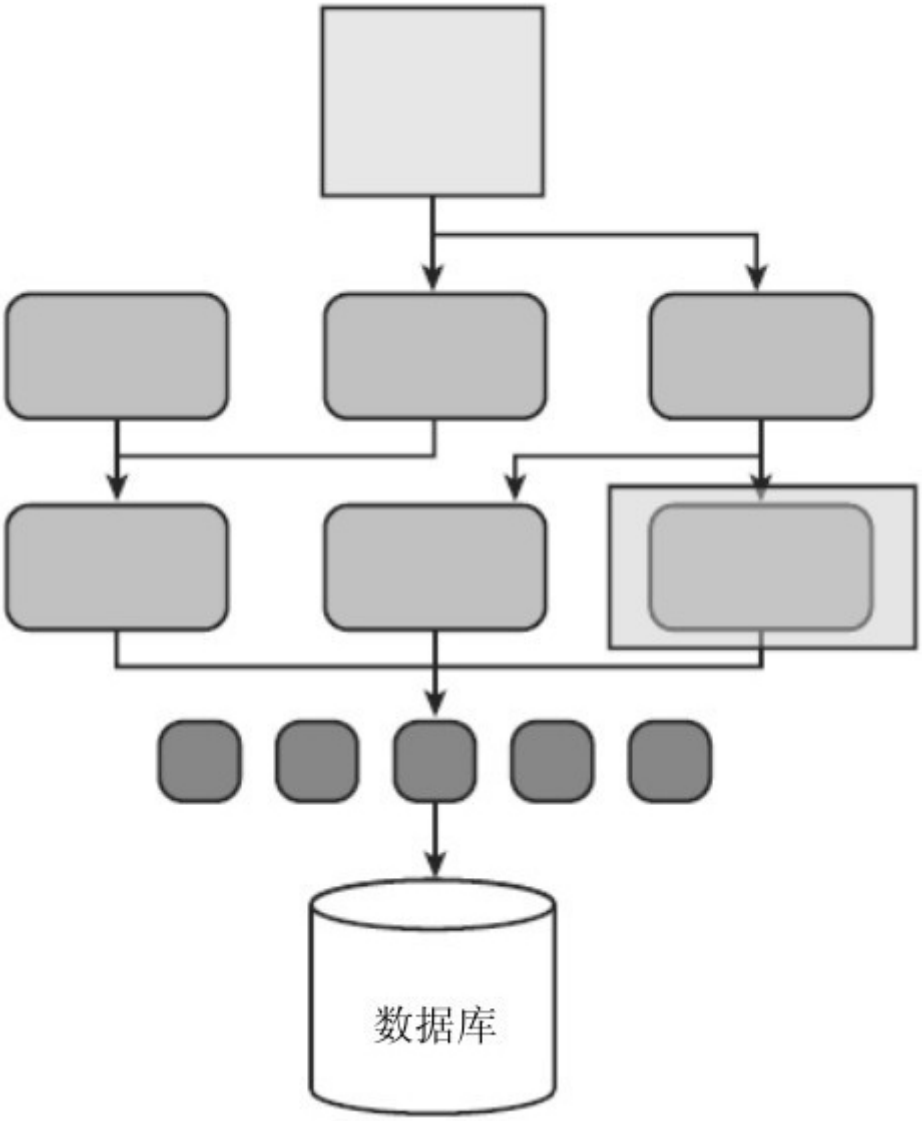
```
// Unit test
describe('User Service Unit Tests', () => {
  describe('Create new user', () => {
    it('should create the new use with valid username and password', () => {
      // 1. 准备
      const fakeUserDA = new FakeUserDA(); // 使用模拟的数据库访问类
      const userService = new UserService(fakeUserDA);
      // 2. 执行
      const newUser = userService.create({
        username: 'test',
        password: 'password',
      }); // 创建测试用户

      // 3. 断言
      expect(newUser.status).to.equal('approval'); // 用户注册成功
      expect(newUser.username).to.equal('test'); // 用户名和注册前匹配

      // 4. 清理
      userService.remove(newUser); // 清理用户
    });
  });
});
```

通过这样的测试代码，就可以清楚的知道 UserService 类的 create 这个函数是不是能正常工作。

小型测试的运行，不需要依赖外部。如果有外部服务（比如文件操作、网络服务、数据库等），必须使用一个模拟的外部服务。比如上面例子中我们就使用了 FakeUserDA 这个模拟的数据库访问类，实际上它不会访问真实的数据库。这样可以保证小型测试在很短时间内就可以完成。



小型测试，图片来源：《Google软件测试之道》

中型测试

中型测试是验证两个或多个模块应用之间的交互，通常也叫集成测试。

如果说要对用户注册的功能写集成测试，那么就会同时测试业务逻辑层的 UserService 类和数据访问层的 UserDA 类。如下所示：

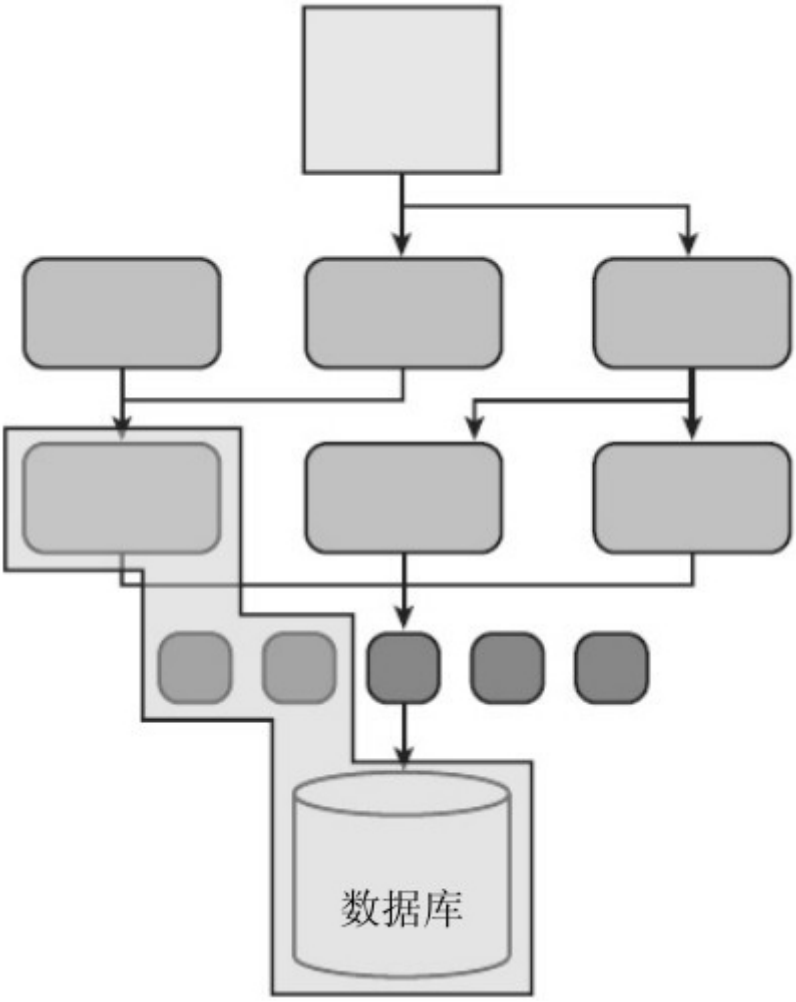
```
// Integration test
describe('User Service Integration Tests', () => {
  describe('Create new user', () => {
    it('should create the new use with valid username and password', () => {
      // 1. 准备
      const userDA = new UserDA(); // 使用真实的数据库访问类
      userDA.mode = 'memory'; // 使用内存数据库
      const userService = new UserService(userDA);
      // 2. 执行
      const newUser = userService.create({
        username: 'test',
        password: 'password',
      }); // 创建测试用户

      // 3. 断言
      expect(newUser.status).to.equal('approval'); // 用户注册成功
      expect(newUser.username).to.equal('test'); // 用户名和注册前匹配

      // 4. 清理
      userService.remove(newUser); // 清理用户
    });
  });
});
```

对于中型测试，可以使用外部服务（比如文件操作、网络服务、数据库等），可以模拟也可以使用真实的服务。比如上面这个例子，就是真实的数据库访问类，但是用的内存数据库，这样可以提高性能，也可以减少依赖。

至于中型测试要不要使用模拟的服务，有个简单的标准，就是看能不能在单机情况下完成集成测试，如果可以就不需要模拟，如果不能，则要模拟避免外部依赖。



中型测试，图片来源：《Google软件测试之道》

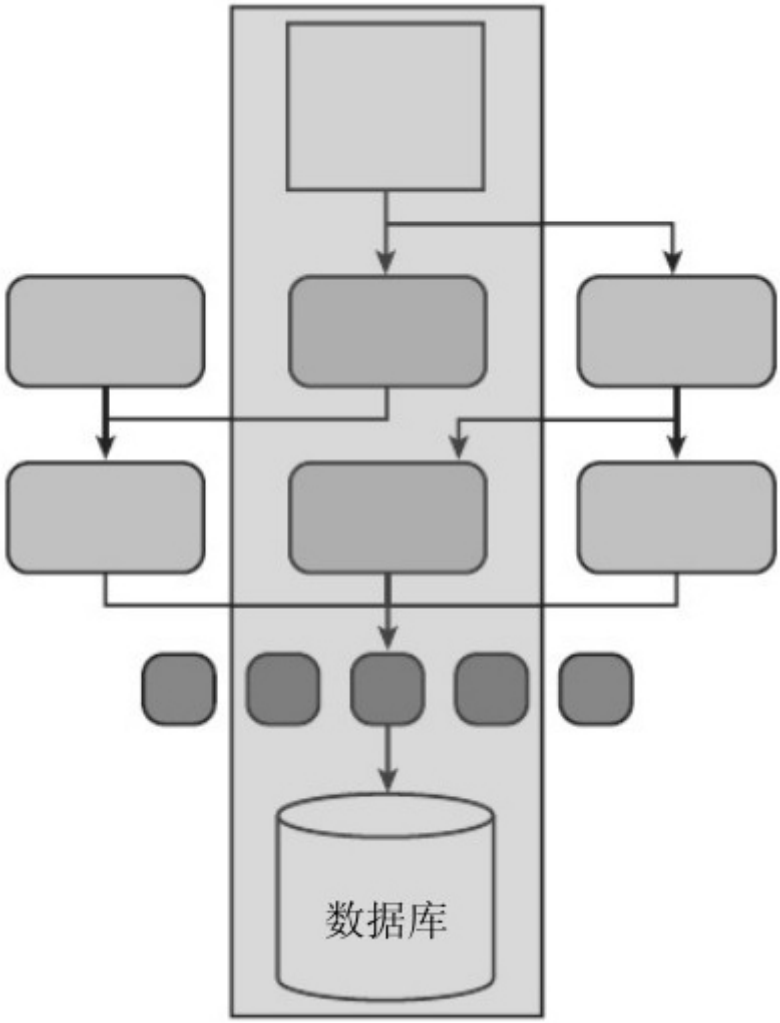
大型测试

大型测试则是从较高的层次运行，把系统作为一个整体验证。会验证系统的一个或者所有子系统，从前端一直到后端数据存储。大型测试也叫系统测试或者端对端测试。

如果说要对用户注册写一个端对端测试的例子，那么看起来会像这样：

```
// End2End test
// 基于 http://nightwatchjs.org
describe('User Service End2End Tests', () => {
  describe('Create new user', () => {
    it('should create the new use with valid username and password', () => {
      return (
        // 1. 准备
        // 调用Chrome Headle浏览器对象
        browser
          // 加载页面
          .url('http://yourdomain.com/signup')
          // 确保注册按钮正常显示
          .waitForElementVisible('input[name=btnCreate]')
          // 2. 执行
          // 在用户名的输入框中输入一个带随机数的用户名
          .setValue('input[name=username]', `test-${new Date().getTime()}`)
          // 在密码的输入框中输入密码
          .setValue('input[name=password]', `password`)
          // 点击注册按钮
          .click('input[name=btnCreate]')
          // 等待3秒
          .pause(3000)
          // 3. 断言
          // 检查是否显示注册成功的消息
          .assert.containsText('#message', 'Success')
          // 4. 清理
          .end()
      );
    });
  });
});
```

对于大型测试，通常会直接使用外部服务（比如文件操作、网络服务、数据库等），而不会去模拟。比如上面这个例子，就是直接访问测试环境的地址，通过测试库提供的 API 操作浏览器界面，输入测试的用户名密码，点击注册按钮，最后检查输出的结果是不是符合预期。



大型测试，图片来源：《Google软件测试之道》

区分测试类型的依据是什么?

以上就是主要的自动测试类型了。捎带着补充一个测试类型，那就是契约测试，这个测试最近出现的频率比较高，主要是针对微服务的。其实就是让微服务在测试时，不需要依赖于引用的外部的微服务，在本地就可以模拟运行，同时又可以保证外部微服务的接口更新时，本地模拟的接口（契约）也能同步更新。对契约服务更多的说明可以参考这篇文章：《 聊一聊契约测试 》

那么契约测试，属于大型测试还是中型测试呢?

Google 针对这几种测试类型列了一张表，根据数据给出了明确区分：

Feature	Small	Medium	Large
Network access	No	localhost only	Yes
Database	No	Yes	Yes
File system access	No	Yes	Yes
Use external systems	No	Discouraged	Yes
Multiple threads	No	Yes	Yes
Sleep statements	No	Yes	Yes
System properties	No	Yes	Yes
Time limit (seconds)	60	300	900+

图片来源：Google Testing Blog

结合上面的表格其实就很好区分了：

小型测试，没有外部服务的依赖，都是要模拟的；

中型测试，所有的测试几乎都不需要依赖其他服务器的资源，如果有涉及其他机器的服务，则本地模拟，这样本机就可以完成测试；

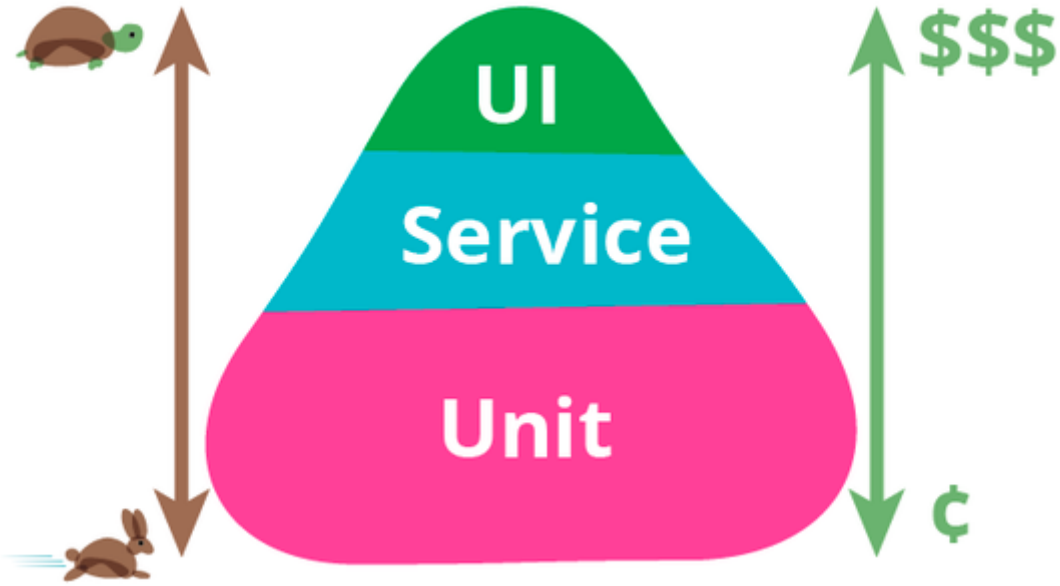
大型测试，几乎不模拟，直接访问相关的外部服务。

所以现在你应该就知道契约测试，也是中型测试的一种了，因为它不需要依赖外部服务，本机就可以完成测试。

为什么中型测试这么看重“能单机运行”这一点呢？因为这样才方便在持续集成上跑中型测试，不用担心外部服务不稳定而导致测试失败的问题。

上面的表中还反映出一个事实：**越是小型测试，执行速度越快，越是大型测试，执行速度越慢。**通常一个项目的小型测试，不超过一分钟就能全部跑完，一个中型测试，包括一些环境准备的时间，可能要几分钟甚至更久，而大型测试就更久了。

另外越是大型测试，写起来的成本也相应的会更高，所以一般项目中，小型测试最多，中型测试次之，大型测试最少。就像下面这张金字塔图一样。所以我们也常用测试金字塔来区分不同类型的测试粒度。



测试金字塔，图片来源： TestPyramid

如果你对测试类型很感兴趣，可以参考《测试金字塔实战》这篇文章作为补充。

怎么写好自动化测试代码？

很多人认为写自动化测试很复杂，其实测试代码其实写起来不难，包含四部分内容即可，也就是：准备、执行、断言和清理，我再把第一段代码示例贴一下：

```
// Unit test
describe('User Service Unit Tests', () => {
  describe('Create new user', () => {
    it('should create the new use with valid username and password', () => {
      // 1. 准备
      const fakeUserDA = new FakeUserDA(); // 使用模拟的数据库访问类
      const userService = new UserService(fakeUserDA);
      // 2. 执行
      const newUser = userService.create({
        username: 'test',
        password: 'password',
      }); // 创建测试用户

      // 3. 断言
      expect(newUser.status).to.equal('approval'); // 用户注册成功
      expect(newUser.username).to.equal('test'); // 用户名和注册前匹配

      // 4. 清理
      userService.remove(newUser); // 清理用户
    });
  });
});
```

第一步就是准备，例如创建实例，创建模拟对象；第二步就是执行要测试的方法，传入要测试的参数；第三步断言就是检查结果对不对，如果不对测试会失败；第四步还要对数据进行清理，这样不影响下一次测试。

上面还有几个测试代码示例，都是这样的四部分内容。

这是针对写一个自动化测试的代码结构。对于同一个功能，通常需要写几个自动化测试才完整。

一个完整的自动化测试要包括三个部分的测试：

验证功能是不是正确：例如说输入正确的用户名和密码，要能正常注册账号；

覆盖边界条件： 比如说如果用户名或密码为空，应该不允许注册成功；

异常和错误处理：比如说使用一个已经用过的用户名，应该提示用户名被使用。

```
// Unit test
describe('User Service Unit Tests', () => {
  describe('Create new user', () => {
    it('should create the new use with valid username and password', () => {
      // 1. 准备
      const fakeUserDA = new FakeUserDA(); // 使用模拟的数据库访问类
      const userService = new UserService(fakeUserDA);
      // 2. 执行
      const newUser = userService.create({
        username: 'test',
        password: 'password',
      }); // 创建测试用户

      // 3. 断言
      expect(newUser.status).to.equal('approval'); // 用户注册成功
      expect(newUser.username).to.equal('test'); // 用户名和注册前匹配

      // 4. 清理
      userService.remove(newUser); // 清理用户
    });

    it('should NOT create the new use with an exists username ', () => {
      // 1. 准备
      const fakeUserDA = new FakeUserDA(); // 使用模拟的数据库访问类
      const userService = new UserService(fakeUserDA);
      const user1 = userService.create({
        username: 'test1',
        password: 'password',
      }); // 创建测试用户
      expect(user1.status).to.equal('approval'); // 用户注册成功
      // 2. 执行
      const newUser = userService.create({
        username: 'test1',
        password: 'password',
      }); // 创建测试用户

      // 3. 断言
      expect(newUser.status).to.equal('duplicate username'); // 用户注册失败

      // 4. 清理
      userService.remove(user1); // 清理用户
    });

    it('should NOT create the new use with an empty username', () => {
      // 1. 准备
      const fakeUserDA = new FakeUserDA(); // 使用模拟的数据库访问类
      const userService = new UserService(fakeUserDA);
      // 2. 执行
      const newUser = userService.create({
        username: '',
        password: 'password',
      }); // 创建测试用户
```

```
      // 3. 断言
      expect(newUser.status).to.equal('invalid username'); // 用户注册失败

      // 4. 清理
    });
  });
});
```

所以你看，写一个测试代码并没有你想的那么复杂，那还有什么理由不去写测试呢？

如何为你的项目实施自动化测试？

现在你了解了有哪些类型的测试，如何写自动化测试代码，也许迫不及待想在项目中实施自动化测试。

选择好自动化测试框架

要写好自动化测试代码，首先要找对自动测试化框架。不同的语言，不同的平台，测试的框架都不一样。好在现在搜索引擎很方便，根据“你的语言 + 自动测试框架”的关键字，就能找到很多的结果。这里我也帮你找了一些，供参考。

Web 前端

Jest： Facebook 的前端测试框架；

Mocha：历史悠久的一个 JS 测试框架；

Nighwatch: 一个 API 很简单，但是功能很强大，可以直接操作浏览器的自动测试框架。

iOS 开发

可以参考这篇文章《iOS 自动化测试框架对比》。

安卓开发

可以参考这篇文章《Android 谈谈自动化测试》。

在持续集成环境上跑你的自动化测试

选好自动化测试框架后，你的自动化测试代码，其中的小型测试和中型测试，最好要能在持续集成环境上运行起来。

让自动化测试在持续集成上运行非常重要，只有这样才能最大化地发挥自动化测试的作用。

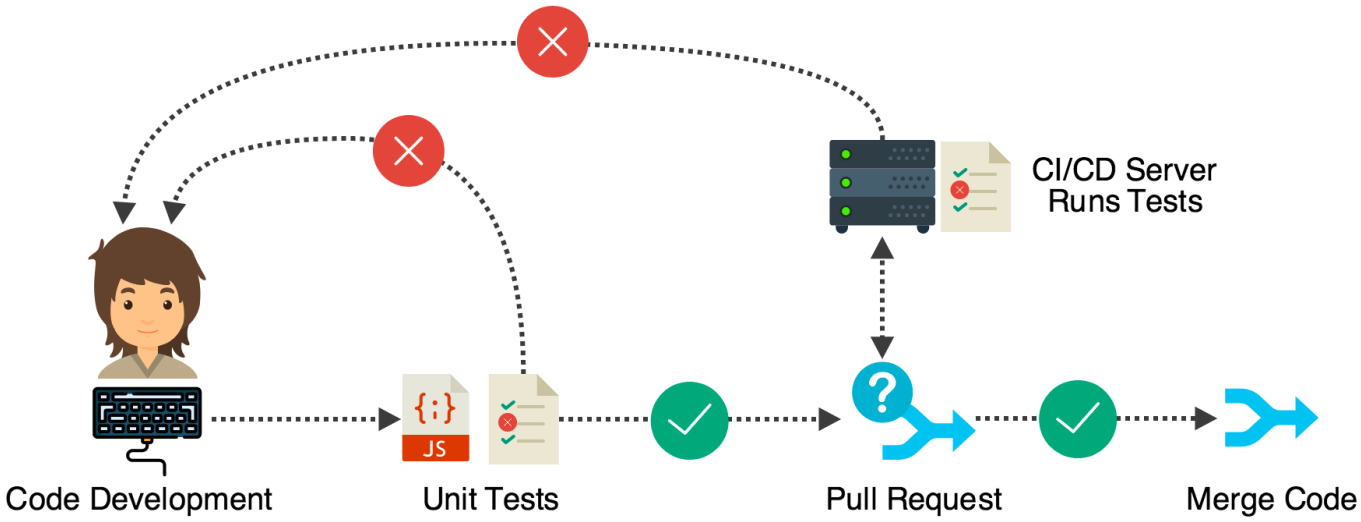
因为持续集成，会强制测试通过才能合并代码，在合并代码之前就能知道测试是不是都通过了，可以帮助程序员获得最直观的反馈，知道哪里可能存在问题，这样才能真正做到防患于未然，把 Bug 杀死在摇篮里。

下图描述的就是自动测试配合持续集成的一个标准流程：

在提交代码前，先本地跑一遍单元测试，这个过程很快的，失败了需要继续修改；

单元测试成功后就可以提交到源代码管理中心，提交后持续集成服务会自动运行完整的自动化测试，不仅包括小型测试，还有中型测试；

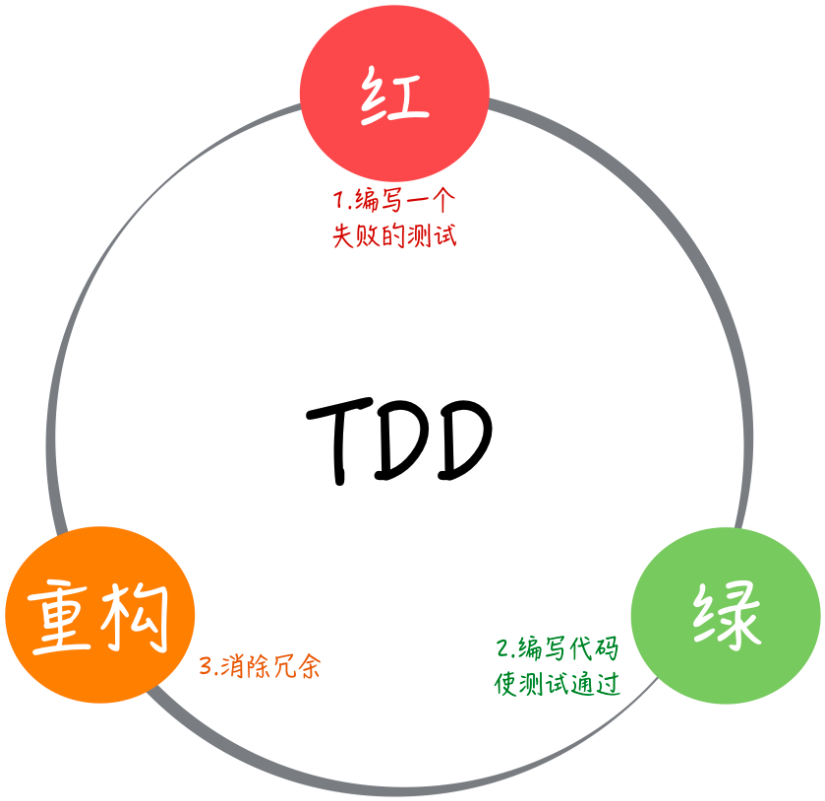
通过所有的测试后，就可以合并到主分支，如果失败，需要本地修改后再次提交，直到通过所有的测试为止。



图片来源：Microservice Testing: Unit Tests

新项目和老项目的不同策略

如果是新项目，那么可以在一开始就保持一定的自动化测试代码的覆盖率，你甚至还可以试试测试驱动（TDD）的开发模式，也就是先写测试代码，再写实现代码，保证测试通过，最后对代码进行重构。



图片来源：郑晔 《10x程序员工作法》专栏

如果是老项目，短期内要让自动化测试代码有覆盖是有难度的，可以先把主要的功能场景的中型测试写起来，这样可以保证这些主要功能不会轻易出问题。

后面在维护的过程中：

增加新功能的时候，同步对新功能增加自动化测试代码；

修复 Bug 的时候，针对 Bug 补写自动化测试代码。

这样一点一点，把自动化测试代码覆盖率加上去。

如果时间紧任务重，来不及写自动化测试怎么办？

确实遇到时间紧的情况，我建议你要优先保证中型测试代码的覆盖，因为这样至少可以保证主要的用户使用场景是正常的。然后把来不及完成的部分，创建一个 Ticket，放到任务跟踪系统里面，后面补上。

总结

今天我带你一起学习了关于自动化测试有关的知识。自动化测试，分为三类：

小型测试，主要针对函数或者类进行验证，不调用外部服务，执行速度快；

中型测试，主要验证两个或多个模块应用之间的交互，可能会调用外部服务，尽可能让所有测试能在本机即可完成，执行速度比较快；

大型测试，对服务整体进行验证，执行速度慢。

写好单元测试代码，基本结构就是：准备、执行、断言和清理；基本原则就是：

要验证正确性；

覆盖边界条件；

验证是否有异常和错误的处理。

自动化测试，一定要配合好持续集成，才能最大化发挥其效用。

对于自动化测试的实施，开头是最难的，因为需要花时间选择自动化测试框架，需要针对自动化测试框架搭建环境，甚至要去搭建持续集成环境。但搭建持续集成和搭建自动化测试环境，并且保证持续更新维护自动测试代码，这个技术投资，一定是你在项目中最有价值的投资之一。

搭建持续集成环境和集成自动化测试框架的事情，要作为一个正式的项目任务去做，当作一个很重要的任务去推进。

课后思考

你所在项目中，自动化测试代码覆盖如何？保持高覆盖率的主要阻力或者障碍是什么？打算怎么改善项目中自动化测试代码的覆盖？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言 (24)

勇闯天涯

2019-10-14

请教老师，我现在做的是嵌入式设备，要跟很多硬件外设打交道，这块的自动化测试和持续集成有什么好的建议吗？我看到文章中大多提到的都是互联网相关的方法和工具

> 作者回复: 我对嵌入式和硬件不懂，没有这方面的经验。不过软件和硬件的开发，都属于工程开发，这里面其实有很多道理是相通的。
>
> 吴军老师写过一篇很好的文章：《把事情做好的三条边》，里面举了莱特兄弟发明飞机的例子，莱特兄弟在试飞之前，打造了一个风洞，进行了上千次的风洞试验，这样可以不需要上天就可以对飞机进行测试。而同时期的很多飞行发明家，自己打造了一个飞机就上天，很多都摔死了。
>
> 自动化测试其实有些类似于风洞试验，相对于手工测试，可以“低成本”、“高效率”的对产品进行反复

的测试和验证，每次提交代码就可以自动运行测试，马上收到反馈。

从软件的自动化测试和飞机的风洞试验，可以找到一些可以借鉴的点：

1. 找到了一条低成本测试的方式
风洞试验，可以降低飞行测试的成本，不必付出生命代价就可以测试结果；
软件的自动化测试，刚开始写的时候是要付出一些成本的，但磨刀不误砍柴工，运行次数越多成本越低。

嵌入式系统是不是也可以找到一条低成本、模拟的、反复测试的方式？

2. 可以模块化的测试
软件的自动化测试，主要有单元测试、集成测试、端对端测试这三种自动化测试类型。单元测试成本是最低效率最高的，而集成测试和端对端测试成本要高很多，所以通常单元测试最多，集成测试次之，端对端测试最少。

我想莱特兄弟发明飞机飞机之前，在组装成整个飞机之前，对各个模块也是有单独的测试，而且测试的应该和自动化测试的比例也是类似。

那么嵌入式系统的自动化测试，是否也可以分模块化的进行测试？让模块的测试占更大比例？

3. 可以即时得到反馈
软件的自动化测试，尤其是结合CI（持续集成），可以即时得到反馈。每次提交代码修改，就会出发自动化测试任务，这样一旦有导致自动化测试失败的代码，能即时发现。
风洞试验也是类似的，飞机不用上天，通过风洞的测试，马上就能得到反馈。

嵌入式系统的自动化测试，是否也可以做到即时反馈？在作出修改后，能否马上得到测试结果的反馈？

以上三点是我认为可以从软件自动化测试借鉴的地方，希望有所帮助！

  16

yu
2019-05-06

.net core 的同學，我們項目使用 NUint 進行單元測試，集成測試可以使用 WebApplicationFactory，模擬工具可以使用 Moq

作者回复: 赞，感谢回复！🙏

8

易林林
2019-05-04

请教宝玉老师：团队成员的能力和素质参差不齐，如何有效的去组织和管理项目的自动化测试，自动化集成？

作者回复: 首先，你要先搭建好自动化测试环境，让自动化测试代码能跑起来，最好最好要和CI（持续集成工具）整合在一起，每次提交代码CI都会跑自动测试，然后能看到运行结果。

然后，把自动化测试作为开发流程的一部分，比如说要代码审查和自动化测试通过后才能合并代码。这部分工作如何和CI集成会容易很多。

再有就是要培训，比如遇到不会写的，开始先带着他写几个，确保他学会了自己能写，然后下次代码审查的时候，看到缺了就要求补上，还不会就继续教，来不及写的就创建个TIcket跟踪起来。

简单来说，就是代码审查+CI+培训

7

Joey
2019-06-26

请教宝玉老师：
消息类接口应该通过哪种方式高效、有效维护？

现状：
系统A属于联机类系统（高并发、低延迟），其中接口B与多个应用相关，当接口B的定义发生变化时，往往忘记通知相关方或者漏通知，从而引发生产事件。

尝试过的手段：
1、通过流程约束，需求评审阶段，强制增加是否有接口变化的评审，但是落实结果不理想，主要因为增加流程，开发人员嫌浪费精力，最后流于形式。
2、通过自动化手段约束，原则上要求接口必须在CI阶段有自动化用例守护，但是效果也不理想，自动化用例缺失或者开发人员懒的写自动化用例，最后流于形式。（我们部门研发和测试属于不同的团队，所以开发人员对于代码质量，都指望测试人员守好最后一道关卡。）

作者回复: 我觉得对于API，应该要有版本的概念，也就是一个版本的API在确定前，多论证，多确认，确认后就不要做大的改动，大改动就用新版本，新版本上线时，旧API应该持续运行一段时间。

然后对于API修改后，应该当作一个小项目来看待，也就是要确保通知所有相关方，确定API切换的时间点，帮助调用房升级迁移到新版本。

最后再是自动化测试帮助检测。

5

小老鼠

2019-09-24

1，小型、中型、大型自动化测试是不是对应单元、集成、系统测试。2、现在测试金字塔模型已经被防锤模型替代了，GUI自动化减少，Interface自动化测试增多。3、有没有必要小、中、大型自动化测试覆盖率均达到100%？4、开头你们前端改造项目自动化测试采用GUI还是Interface？若是GUI，有多少个测试用例，每个测试用例执行时间有多长，所有测试用例执行有多长。若是Interface ，如何测试前端？5、前端有没有自动化测试框架？

作者回复: 1. 一般来说是这么对应的
3. 没有必要达到100%，这样做成本太高，需要有一个平衡和取舍
4. 我们项目集成测试和单元测试都有。
我们前端项目基于React开发的，所以接口的测试基于React的单元测试接口。
各有几百个测试用例。
整个测试单元测试很快，一分钟不到，集成测试要长一点，5-10分钟。
5. 前端框架都有测试框架，比如React/Vue都有单元测试框架，可以查阅其文档

3

熊猫

2019-05-07

老师你好，请问下有没有介绍开发如何写好测试不错的书？

作者回复: 推荐：《how we test software at microsoft》中文版《微软的软件测试之道》

不过没有书其实你也可以找到很多资料的。比如我平时写前端程序，那么我会去GitHub或者Google，通过关键字、语言找跟我项目类似的开源项目，然后看其中有没有自动化测试写得好的，找到了(例如：reactstrap、electron-react-boilerplate、kitematic)就照葫芦画瓢好了，因为都是真实项目，所以

特别简单有效，建议你也可以试试。

另外耐心一点，你也可以看到很多关于测试知识分享的技术文章，多看一看也有收获。

4

hua168

2019-05-04

宝哥，我想问一下：
1.开发哪些测试需要自己写的呀，“测试驱动开发”的概念，开发应该要会写测试吧？
到底要求会写哪些测试？
2.现中小公司都没有自动化测试工程师，写好测试手工检查的多，怎搞？
开发学一点selenium3自动化测试之类会不会好点？
3.单元测试是不是数据越简单越好，最好不使用数据库，在dao层组或map
4.集成测试和大型测试用数据库则比较好，对吗？

作者回复: 1. 文章中的小型测试和中型测试都应该是开发来写的。大型测试一般是测试开发工程师来写，也可以开发写。

2. 这个必须要去学习的，

3. 单元测试不能使用真实数据库，必须要模拟数据访问的，否则速度太慢也不稳定

4. 集成测试一般用本机的数据库，或者也可以模拟数据。大型测试肯定用真实数据库的。

3

Liber

2019-05-13

宝玉老师你好，有个地方感觉有必要再展开谈谈：
以本文注册用户为例，本文分别对这个case写了小、中、大型测试用例，但实际开发过程中，如何权衡对一个场景是该小、中、大都写，还是只写部分？

作者回复: 实际开发中，理论上来说一个场景大中小测试都要写的。

通常来说，开发写小型测试和中型测试，测试写大型测试，或者开发帮助写大型测试。

小型测试：中型测试：大型测试比例大约为 7:2:1

小型测试尽可能多覆盖，不要求100%，谷歌是85%

中型测试覆盖大部分用户使用场景

小型测试覆盖主要用户场景

3

Zebin
2019-05-08

宝玉老师，请教下，我们现在LINUX环境下开发项目，主要编程需要是C/C++。
现在想搭建持续集成开发环境，有什么合适的工具可以推荐下吗

作者回复: 单元测试你可以用gtest，集成测试工具你可以参考我之前 那篇集成测试的文章，比如说试试Jenkins或者Gitlab CI。

具体怎么搭你可以参考网上的教程，应该已经有很多了

共 2 条评论>

2

起而行
2019-05-04

老师，持续集成怎么理解呢，我看知乎上说就是团队成员在一天内多次进行编译，发布或自动化测试

作者回复: 狭义的持续集成不包括发布，主要指集成，持续的（每次提交代码变更都触发，频繁的提交）对代码进行集成（合并到主干），但集成前要确保自动化测试通过。

广义的持续集成还包括部署，也就是集成后自动部署测试环境(持续交付)或者生产环境（持续部署）。

在《26 | 持续交付：如何做到随时发布新版本到生产环境？》这一篇里面有详细介绍。

2

Sam
2020-02-27

您好，请问下，我在.net framework平台下，单元测试工具如何选择（能与jenkins接合的）

作者回复: Jenkins有很多丰富的插件，你可以根据项目情况寻找适合你的源代码插件（例如Team Foundation Server Plug-in）、编译插件（例如MSBuild Plugin）、单元测试插件（例如NUnit plugin）

比如可以参考这篇教程：
https://www.swtestacademy.com/jenkins-dotnet-integration/

你也可以根据关键字".Net unit test Jenkins" Google到很多相关文章

共 2 条评论 > 1

探索无止境
2019-05-06

老师你好，各种类型的测试覆盖率你们一般采用什么指标？个人感觉在理想的情况下最好是做到百分百覆盖率

作者回复: 100%覆盖这个我觉得可以作为一种理想追求，但是没必要追求极致，还是要在进度和质量之间有个平衡比较好，毕竟进度也很重要。

另外对于前端业务，我更重视集成测试的覆盖，对于主要业务场景集成测试覆盖到位后，单元测试也就有比较多的覆盖，相对性价比更高，然后再逐步补充单元测试的覆盖率。

1

yellowcloud
2019-05-05

宝玉老师，我们现在使用的框架是.net core,语言是C#，用其进行后端开发。能否推荐一下好的自动化测试框架。我根据您的检索方法语言+自动化测试框架找到的是RedwoodHQ，不知道它在实际使用中是否可行。

作者回复: 如果是单元测试，.Net Core应该自带了，例如：《.NET Core 和 .NET Standard 单元测试最佳做法》https://docs.microsoft.com/zh-cn/dotnet/core/testing/unit-testing-best-practices

你可以换一下关键字：".Net core unit testing", ".Net core integration tests"。

这一篇《Integration tests in ASP.NET Core》https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests 写的很详细，还有实例。

另外不知道你的具体是什么类型项目，Web还是Winform?

1

hua168

2019-05-05

看到bug我又想起了网站安全，宝哥，像我们中小公司网站安全也是运维负责的
一般网站安全怎做呀？如果服务器linux(centos)被入侵一般怎么查别人是怎么入侵的？
宝哥您了解这方面的吗？小公司运维真是什么都做，打杂的~~

作者回复: 网站安全会在后面一篇有详细讲。如果你现在遇到了入侵，你可以查一下操作的日志看看，看登录的IP、账号，看是不是有什么线索。

1

丿淡忘

2019-05-04

宝玉老师，我想问一下，针对桌面开发的界面自动化测试一般是怎么进行的

作者回复: 可以试试 Appnium或者Ranorex。不过我没直接用过，不好评价是不是适合你，建议你先试试看。

1

ifelse

2022-07-02

自动化测试，一定要配合好持续集成，才能最大化发挥其效用。--记下来

。。。

2022-03-03

老师，一般，都比较常看哪些网站的文章

Ho

2021-10-28

老师讲的真好！

徐凯
2021-01-24

老师你好 之前我遇到的开发流程是本地代码修改完毕后 本地构建 构建过程中会跑单元测试 没问题后 再提交分支 然后再发起pull request 合进代码线后 jekins会触发一次与提交代码相关的服务的构建 这个过程中会构建代码并且跑单元测试 如果没过 服务会挂掉。 我想问下如果这里要加业务模块的自动化测试的话 也是在这次构建中执行的么？还有我看老师你说的好像是单元测试或者自动化测试未通过是不允许合并主干的 但是我们之前是合并主干之后才去跑测试 这里是不是存在问题？

wanghua
2020-04-02

对于单元测试，需要每个函数都写吗，这样工作量好大，有什么方法确定哪些该写，哪些不用写呢？

> 作者回复: 不需要每一个函数都写，保持一定的覆盖即可。通常你在测试一个函数的时候，其调用的函数也可以部分覆盖到。所以各种语言都有计算测试覆盖率的工具。
>
> 测试时，可以站在用户使用场景去思考，对于主要使用场景尽可能覆盖测试，在此基础上，尽可能保证高的测试覆盖率。