单元测试是怎么做的?

典型回答

使用junit、Mockito等工具在开发完成后进行自测，避免反复修复bug部署测试服耗时高。
什么是单元测试，和集成测试有什么区别?

扩展知识

面试官真的就只是想听一下什么是单元测试么？如果真的这么想是大错特错。问出这个问题的面试官考察面试者以下3点：

1. 对软件工程的生命周期是否熟悉，是否充分理解测试阶段各种手段（单测、集成测试、冒烟测试等）和重要性。
2. 面试者是否有足够的责任心，做好测试工作是对自己代码充分负责。
3. 优秀的单测用例也体现出开发者的设计和编码的基本素质。

基于以上3点，我们需要思考什么是有效的单元测试?

> 题外话，《有效的单元测试》是一本非常值得推荐看的书

整个软件工程的生命周期，大致为：

- 需求分析阶段：需求调研、设计、评审
- 设计阶段：这里指架构设计阶段
- 开发阶段：开始正式编码
- 测试阶段：完成编码，包括：
  - 自测：单元测试->集成测试
  - 提测：QA介入集成测试，多轮测试
- 发布：QA完成测试，可以进行上线，包括：
  - 预发布：部署到线上环境，QA进行回归测试，没问题放一部分流量进来，观察是否存在异常
  - 上线：若预发布没问题，则代码正式上线，当然也会根据各种灰度或ab实验策略控制新功能流量占比，稳定跑一段时间没问题再放开全部流量

我们再来仔细分析每个阶段发现bug的成本，该成本主要是指从发现到解决问题的人力时间成本。

- 需求分析阶段：若大家评审设计不合理，可以不做，只占用小时级别的会议时间
- 设计阶段：架构设计也需要进行评审，同样只占用小时级别的会议时间

- 开发阶段：若前两个阶段没有发现问题，通常小功能是小时级别，大功能是天级别（甚至是星期、月），可能导致开发出无效的功能，重新做设计，重新开发的返工局面
- 测试阶段：无论是自测还是提测的集成测试，修改一个bug就意味着需要重新部署代码，大型项目启动时间可能是分钟级别。无论是自测还是提测，意味着会阻塞当前测试进度，bug多了反复部署累计出来的时间会非常高。而单元测试一个case通常是毫秒或秒级别的，做好单元测试可大幅度提升效率。很多公司非常注重单测覆盖率，有效性，甚至把单测放入CI/CD中，所有单测都跑成功才能部署。同时QA也会非常注重阻塞测试进度的情况。
- 发布阶段：通常进入发布阶段都是经过QA严格测试过的代码，不会出现明显的bug，但不代表没有。有一些bug可能在真实用户请求或流量大的时候才能出现，bug逃过了测试和预发布环境的测试，但到线上会直接出现。灰度和ab实验的一部分目标也是为了将线上的问题产生的影响降低到最小。这也是为啥各大互联网公司那么多大佬，依然会出现事故。这时候不只是时间成本，一个致命的bug可能带来直接的经济损失和用户流失。程序员一旦出了事故，就是个故事了。所以很多公司非常注重bug逃逸率，即在测试阶段没测出来的问题。

上面说了这么多，也是为了突出单元测试的重要性。以下总结一下写好单测的方法：

1. 单测代码与正式代码一样重要，所以要层次清晰，命名符合实际用例场景，要有适当的注释，《代码整洁之道》的技巧同样可用于单测代码中。很重要的一点：case要让人能看懂
2. 单测不要盲目追求覆盖率，但要尽量测出所有可能的场景
3. 单测要保持可用，纳入CI/CD的流程，如果所有case跑不通，不允许部署
4. 确保case每次运行是确定的，不依赖外部变化和不确定因素，包括不限于：
    - 随机事件：如随机数，如果有最好Mock掉
    - IO：无论是磁盘IO、网络IO（数据库、外部接口）都需要隔离掉，否则IO抖一抖case就会失败，如果有最好Mock掉
5. 必须有断言，否则单测没有意义，不要想着打印一下结果肉眼看一下就可以了，在跑全部case的时候谁会经常想着要看一眼?
6. 验证边界和异常，这俩个是最容易被忽略的。边界可包括：

- 传入错误参数会怎么样
- 依赖（内部或外部接口、数据库环境等）返回不正确的结果会怎么样

异常包括：

- 外部异常：依赖（内部或外部接口、数据库环境等）向调用者抛出异常会怎么样
- 内部异常：代码自身抛出`RuntimeException`会怎么样

7. 正式业务代码保证单一职责，高内聚低耦合可让单测更轻松，测试粒度更细，覆盖率也更高。一个方法一个类只干一个事，单测case就可以只关注当前要测试方法的有效性就行了，不需要关注方法之间的调用，如果你的每个方法都测通了，组合起来也都会通的。当然一个case也要只干一件事！

还有一个好的方法是采用TDD（测试驱动开发），即先列出所有可能的用例，再实现逻辑代码。这样产出的代码可快速构建出单测。不过国内很少有采用TDD开发模式的公司。

> 题外话，DDD（领域驱动设计）中提倡明确的边界划分，事件风暴，防腐层的设计都给TDD和单元测试做了很好的铺垫。

上面说使用Mock屏蔽IO和随机事件，当然也可以用在各种依赖上（如spring bean之间的依赖，工具类、各种内部接口的依赖），Mock是将所依赖的资源造假，我们假装依赖都是成功或失败的，这样只要测试自身代码对其产生了什么结果。
什么是 Mock？怎么做单测的 Mock？
Java工程也可以集成 `spock framework` 进行单测，spock使用 `groovy` 语言编写单测。由于是动态语言非常灵活，很适合编写轻量的单测代码。同时spock不只局限于Mock，还提供各种高效的功能（这些都是传统Junit、Mockito无法实现的）：

1. Spy，只对部分资源进行Mock，可以很方便对同一个类内相互调用的方法进行Mock和验证
2. Mock，对依赖资源进行Mock，同时验证依赖的资源调用多少次，比如：测试一个Redis写功能，可以对Redis客户端进行Mock，验证传入的方法参数是否符合预期，验证Redis写入方法的调用次数
3. Stub，对依赖资源进行Mock一个结果，不关注调用次数、传参是否符合预期
4. 直接无视待验证方法的成员封装等级，可以直接测试 `private` 声明的方法和变量
5. 基于数据驱动的测试，可以在一个case里通过 `where:` 关键词和数据表格的方式对要测试的参数预期返回值的所有可能情况进行验证
6. 可以很方便的验证要抛出的异常
7. 可以很方便的集成spring，包括可单测spring mvc、spring boot的http接口层，不需要启动web容器

以上是如何做好单测的思路，请记住：

1. 有用户访问的项目和无用户访问的项目是不一样的，相同的代码甚至在极端的用户流量下会产生不一样的效果。对于极端的用户流量，改一行代码上线都是如履薄冰
2. 敬畏每一次上线和线上操作

所以能够写出好的单测代码，是一个优秀程序员的基本素养。