

---

## 如何做集成测试

- 1. 说明
  - 1.1 方法
  - 1.2 目标
  - 1.3 集成测试应考虑问题
- 2. 集成测试内容
  - 2.1 功能性测试
  - 2.2 异常测试
  - 2.3 规模测试
  - 2.4 并发/压力测试
- 3. 用例设计方法
  - 3.1 设计原则
  - 3.2 用例设计相关理论
  - 3.3 用例模板
  - 3.4 设计步骤
    - 3.4.1. 模块分析
    - 3.4.2. 接口分析
    - 3.4.3. 条件筛选
    - 3.4.4. 用例设计
    - 3.4.5. 场景设计
    - 3.4.6. 代码实现
    - 3.4.7. 迭代补充

## 1. 说明

集成测试是在单元测试的基础上，测试在将所有的软件单元按照概要设计规格说明的要求组装成模块、子系统或系统的过程中各部分工作是否达到或实现相应技术指标及要求的活动。

### 1.1 方法

集成测试的方法有很多种，但是归纳起来主要有两种。一种是一次性将所有单元组装起来进行测试的“大棒”模式，一种是一层一层累加递增式的测试。

### 1.2 目标

集成测试的目标是按照设计要求使用那些通过单元测试的构件来构造程序结构。单个模块具有高质量但不足以保证整个系统的质量。有许多隐蔽的失效是高质量模块间发生非预期交互而产生的。

集成测试是确保各单元组合在一起后能够按既定意图协作运行，并确保增量的行为正确。它所测试的内容包括单元间的接口以及集成后的功能。

### 1.3 集成测试应考虑问题

- 1、在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；
- 2、各个子功能组合起来，能否达到预期要求的父功能；
- 3、一个模块的功能是否会对另一个模块的功能产生不利的影响；
- 4、全局数据结构是否有问题；

- 
- 5、是采用何种系统组装方法来进行组装测试；
  - 6、组装测试过程中连接各个模块的顺序；
  - 7、模块代码编制和测试进度是否与组装测试的顺序一致；
  - 8、测试过程中是否需要专门的硬件设备；
  - 9、单个模块的误差积累起来，是否会放大，从而达到不可接受的程度。

因此，单元测试后，有必要进行集成测试，发现并排除在模块连接中可能发生的上述问题，最终构成要求的软件子系统或系统。对子系统，集成测试也叫部件测试。

## 2. 集成测试内容

这里介绍我们的模块做集成测试时主要要测试的内容：

### 2.1 功能性测试

站在使用者的角度，对模块提供的功能进行完备的测试。在做功能测试前需要设计充分的测试用例，考虑各种系统状态和参数输入下模块依然能够正常工作，并返回预期的结果。

在这一过程中，需要有一种科学的用例设计方法，依据这种方法可以充分考虑各种输入场景，保证测试功能不被遗漏，同时能够以尽可能少的用例和执行步骤完成测试过程。

常规意义上的功能测试一般是黑盒测试，对内部的实现不太关注，但是为了充分考虑各种参数取值场景并减少不必要的条件组合用例，对模块的集成测试需要对模块内部实现进行了解，采用灰盒测试。

具体的用例设计方法见用例设计方法一节。

### 2.2 异常测试

异常测试是有别于功能测试和性能测试又一种测试类型，通过异常测试，可以发现由于系统异常、依赖服务异常、应用本身异常等原因引起的性能瓶颈，提高系统的稳定性。

常见的异常如：磁盘错误、网络错误、数据出错、程序重启等等。

### 2.3 规模测试

测试模块在一定规模下是否能够正常工作，是否会出现异常或者崩溃，观察系统资源的使用情况。例如测试打开大量的文件是否会出错，内存占用是否会过大。

### 2.4 并发/压力测试

功能性的测试更多是单线程的测试，还需要在并发场景下对模块进行测试，观察在并发场景下是否能够正常工作，逻辑或者数据是否会出现错误。

考虑并发度时，使用较大的压力来测试，例如平时使用的2倍、10倍或更大的压力。

## 3. 用例设计方法

---

测试用例是为验证程序是否符合特定系统需求而开发的测试输入、执行条件和预期结果的集合，其组织性、功能覆盖性、重复性的特点能够保证测试功能不被遗漏。由于测试用例往往涉及多重选择、循环嵌套，不同的路径数目可能非常大，所以必须精心设计使之能达到最佳的测试效果。

### 3.1 设计原则

这里首先需要考虑从什么样的角度去思考用例的设计，一种是以接口的角度，根据接口划分来设计用例；另一种是以使用场景的角度来设计，例如快照过程中或者克隆过程中的操作。

#### 根据接口来设计用例

优点：

1. 可以根据输入不同的接口参数，产生不同的预期来设计用例，考虑比较完整地考虑各种调用情况；

缺点：

1. 这种方法比较抽象，不知道在什么样的场景下会出现这样的用例；
2. 根据接口提供的功能来设计用例，有时候难以考虑一些特殊场景下功能上的缺陷。

#### 根据场景来设计用例

优点：

1. 用例审核者能比较容易理解各组用例出现的场景；
2. 可以测试出功能设计之初未考虑到的一些场景。

缺点：

1. 无法很好地证明用例覆盖是否充分。

两种方式各有优缺点，可以考虑将两种方式结合起来，用接口方式来设计用例，然后用场景方式来组织用例的执行序列。

根据接口进行设计可以比较清晰地评估用例覆盖是否完全，然后以场景方式来执行这些用例，对执行过的用例就打钩记录；这样可以很清楚的知道哪些用例执行了哪些没执行，帮助发现没有想到的场景；

此外还能通过特殊的场景帮助发现接口功能是否考虑充分，两种方式可以互补帮助发现更多问题。

### 3.2 用例设计相关理论

测试用例的设计有许多的理论指导，这里列出几个常用的理论介绍。我们的用例设计会参考但不限于以下几种方法。

#### 组合测试

一个接口或功能或系统需要用到多个参数，而参数又有不同的取值，不同参数取值的组合会影响产生不同结果，但是这个组合数可能非常大，导致用例数过多。

举例说明：假设我们需要验证IE在不同硬件配置的PC上的兼容性测试，且经过数据统计，主要用户占比的PC信息如下图所示：

```
PLATFORM:  x86, ia64, amd64
CPUS:      Single, Dual, Quad
RAM:       128MB, 1GB, 4GB, 64GB
HDD:       SCSI, IDE
OS:        NT4, Win2K, WinXP, Win2K3
```

并且需要验证的IE版本如下图所示：

```
IE:        4.0, 5.0, 5.5, 6.0
```

在这种场景下，要达到对参数的所有取值组合的覆盖，共需要 $3*3*4*2*4*4=1152$ 条用例，这种情况称为组合爆炸。

而组合测试的目的，抽象的说就是为组合爆炸提供一种解决方案，简单地说就是在保证错误检出率的前提下采用较少的测试用例生成方法，它将被测系统或被测系统的模块抽象成一个受到多个因素影响的系统，并提取出每个因素的可能取值，结合组合测试方法，生成最终的测试用例。常用的组合测试方法包括：

1、两因素组合测试（也称配对测试、全对偶测试）

生成的测试集可以覆盖任意两个变量的所有取值组合。在理论上，该用例集可以暴露所有由两个变量共同作用而引发的缺陷。

2、多因素（t-way,  $t>2$ ）组合测试

生成的测试集可以覆盖任意t个变量的所有取值组合。在理论上，该测试用例集可以发现所有t个因素共同作用引发的缺陷。

3、基于选择的覆盖

要满足基于选择的覆盖，第一步是选出一个基础的组合，且基础组合中包含每个参数的基础值，建议选择最常用的有效值作为基础值。基于基础组合，每次只改变一个参数值，来生成新的组合用例。

关于组合测试的理论和方法，这里不做具体展开。

## 等价类划分

把全部输入数据合理地划分为若干等价类，在每一个等价类中取一个数据作为测试的输入条件，就可以用少量代表性的测试数据取得较好的测试结果。

有效等价类：指对于程序的规格说明来说是合理的、有意义的输入数据构成的集合。

无效等价类：与有效等价类的定义恰巧相反。

设计测试用例时，要同时考虑这两种等价类。因为软件不仅要能接收合理的数据，也要能经受意外的考验。这样的测试才能确保软件具有更高的可靠性。

## 划分等价类的方法

下面给出六条确定等价类的原则：

- 1、在输入条件规定了取值范围或值的个数的情况下，则可以确立一个有效等价类和两个无效等价类。
- 2、在输入条件规定了输入值的集合或者规定了“必须如何”的条件的情况下，可确立一个有效等价类和一个无效等价类。
- 3、在输入条件是一个布尔量的情况下，可确定一个有效等价类和一个无效等价类。
- 4、在规定了输入数据的一组值（假定n个），并且程序要对每一个输入值分别处理的情况下，可确立n个有效等价类和一个无效等价类。
- 5、在规定了输入数据必须遵守的规则的情况下，可确立一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。
- 6、在确知已划分的等价类中各元素在程序处理中的方式不同的情况下，则应再将该等价类进一步的划分为更小的等价类。

## 边界值分析

在最小值、略高于最小值、正常值、略低于最大值和最大值处取输入变量值

例如：涉及两个变量的函数x1, x2

表示方法min、min+、nom、max-、和max

X1的取值x1min, x1min+, x1nom, x1max-, x1max

X2的取值x2min, x2min+, x2nom, x2max, x2max

其他方法

状态迁移法、流程分析法、判定表分析法、因果图法等等。

总结

前面三种方法是常用的用例设计方法，并且各种方法之间可以结合使用，根据实际的场景选用合适的设计方法。

在对接口或者系统做用例设计时，需要根据参数的不同取值组合来设计用例，可以覆盖所有的情况。但是如果考虑所有的组合，在很多情况下用例集会非常庞大，这种情况下就要使用组合测试来减少用例数，同时保证充分的覆盖率。

此外有些接口的参数取值并不是离散的，这种情况下可以使用等价类划分或者边界值分析法来划分参数取值。举例来说，对于DataStore的WriteChunk接口的sn来说，sn可以为大于0的任意整数，利用等价类划分法，可以将sn与chunk的sn和correctedSn对比，划分为大于chunk的sn、等于chunk的sn和小于chunk的sn跟大于chunk的correctedSn、等于chunk的correctedSn和小于chunk的correctedSn的条件组合。

3.3 用例模板

用例设计可以遵循GWT（Given-When-Then）的模式来写，Given表示给定的前提条件，When表示要发生的操作，Then表示预期的结果。

如果要覆盖所有的用例，那么势必列出所有的前提条件，然后在特定的前提下，需要列出所有可能发生的操作。

| 编号 | Given | When | Then | 备注 | 是否执行  |
|----|-------|------|------|----|---|
| 1  |       |      |      |    | <ul style="list-style-type: none"><li>• 单元测试</li><li>• 集成测试</li></ul> |
| 2  |       |      |      |    | <ul style="list-style-type: none"><li>• 单元测试</li><li>• 集成测试</li></ul> |
| 3  |       |      |      |    | <ul style="list-style-type: none"><li>• 单元测试</li><li>• 集成测试</li></ul> |

3.4 设计步骤

整个设计可以按照以下步骤来执行：

3.4.1. 模块分析

首先需要了解要测试模块以及组成此模块的其余子模块的功能，可以通过设计文档也可以通过代码实现来了解。

---

需要知道这一模块负责的主要功能，模块维护的一些状态，模块记录了哪些信息等等。

以DataStore为例，首先DataStore下组合了LocalFileSystem和ChunkfilePool两个子模块，对外提供将对chunk的操作转化为对本地文件的操作的功能；

该模块主要管理chunk文件及其快照文件，维护chunk文件的属性和数据。

### 3.4.2. 接口分析

对接口提供的功能、输入参数、输出结果、实现原理等进行分析，根据接口内部逻辑和模块维护的状态对各参数进行等价类划分。

一般来说输入参数在内部都会存在一些判断逻辑，这些判断的逻辑就可作为等价类划分的依据；一个参数可以与不同的状态进行对比分成多个条件项，不同条件项可以根据对比结果分类分成多个具体的条件。

以DataStore的ReadChunk接口为例，主要包含了id、sn、offset、length这些参数，id这个参数对应了一个chunk的状态，即chunk是否存在、快照是否存在、是否为clone chunk，从而三个条件项，每个条件项有根据是否成立分成两个条件；

sn作用是跟chunk的sn和correctedsn进行对比分成两个条件项，对比结果可以划分为sn>chunkinfo.sn、sn<chunkinfo.sn、sn==chunkinfo.sn(边界值)三组条件，以及和correctedSn对比的三组条件；offset、length可以与chunksn对比作为一个条件项，判断读取区域chunk之前是否写过又作为一个条件项。

### 3.4.3. 条件筛选

如果要完全覆盖情况，需要将上面列出来的各组条件项进行全排列组合，这样用例数会非常大。但是这也只有在各条件项之间完全独立的情况下才可能出现，在实际设计时，各条件项之间存在着各种关系，我们可以利用这种关系来减少用例数。

常见的一些关系：

依赖：

某一条件项A存在的前提依赖于另一条件项B的某一条件m成立，这样一来就可以过滤A的各条件与m的组合。

例如DataStore的ReadChunk接口，快照是否存在这一条件项依赖于chunk存在的前提，否则这一条件项就没有意义。

互斥：

有些条件项之间是互斥的，必然不会一起出现的，这样就可以过滤掉这两个条件项的条件之间的组合。

例如DataStore的ReadChunk接口，快照存在，则chunk必然不是clone chunk；chunk如果是clone chunk就必然不存在快照。

等价处理

有些条件项不同条件在接口中的处理方式是一样的，这种情况下可以去除这一条件项，或者写用例的时候可以穿插着用不同条件来写。

例如ReadChunk接口，对于是否为clone chunk，接口处理是一样的，所以这个条件项实际上没什么意义；可以去掉，但是对这个接口来说比较好的方法是将这两个条件当成一个条件与其他条件进行组合，

但是组合时可以随机的使用clone chunk或普通chunk作为这个条件，这样可以减少一半的用例。

条件阻断

代码的实现逻辑中，条件都是顺序执行的，某一条件符合的情况下，才会判断下一个条件，如果某一条件成立情况下函数返回了，后面的判断就必然不会继续下去了，那么在这一条件成立情况下，与后面条件的组合就可以过滤掉。

但是这个需要对代码逻辑要比较了解，对于复杂的判断逻辑会较难处理。这里原则上认为对于比较简单的，且逻辑顺序基本不会发生变化的，可以应用这条方法。

---

例如DataStore的WriteChunk接口，如果offset+length>chunksize，就一定会返回OutOfRangeException，那么在这种情况下其实只需要测一组就可以了，与其他条件的组合可以过滤掉。

上面列出的是常见的一些筛选用例的办法，其他还可以根据实际的一些情况进行筛选。

#### 3.4.4. 用例设计

当上面的分析做完以后就可以开始写用例了，用例的模板可以参照上面的表格。

写用例时尽量将相同功能或接口的用例写在一起，相同Given条件下的写一起，然后一次改变一个条件（除非要利用组合测试）。

Given里给出前提条件，例如DataStore是对chunk的操作，前提条件就是当前chunk的状态；When就是输入参数的条件组合；Then就是预期的结果。

#### 3.4.5. 场景设计

有了上面的用例后，在考虑实际的场景来设计执行步骤。一个场景里面会包含不同接口的使用，但是每次调用必然能对应到某一条用例，如果没找到对应用例，那就是用例设计有遗漏，需要补充用例。

#### 3.4.6. 代码实现

根据场景设计中的步骤来写代码，然后检查是否与用例预期结果一致。用例跑完通过后，在对应的用例后面打钩记录。

#### 3.4.7. 迭代补充

检查用例是否都执行了，如果没有执行分析是不是场景有遗漏，需要进行补充。原则上来说，用例是需要百分百覆盖的。