谈谈如何保证测试代码的正确性 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022__E8_B0_88_E8_B0_88_E5_A6_82_E4_c104_252382.htm 本文仅就单元测试而论，虽然是说的测试，但目的是驱动开发，不过也不是谈测试驱动开发，更象是对测试驱动开发时TEST FIRST这个过程中如何保证测试代码的正确性的理解和想法，当然有一些，我认为是通用的，不管是不是测试优先。而我目前接触最多的还是JAVA的单元测试，所以谈的东西还是以JAVA为主，举的例子都是和Java有关的。 另外前些天看到一个帖子有人问这样的问题，想到当初自己刚接触JUNIT单元测试时也有类似的困惑，现在有了一些经验，所以写下来，既是对自己经验的总结，也是希望能有人相互讨论提高。 首先是我认为要做到测试代码的正确性的几个要点： 一、TEST FIRST 二、只写出需求测试（注意，是目标代码需求，这个代码的用户当然是自己了，^_^） 三、不要为了测试而测试（这句话是和一个朋友聊天时他说是他和Kent Back交流时Kent Back提醒他的，这里我也不是很确定对这句话的理解的正确与否，因为理解一句话，上下文也是关键的，而我并不很了解我朋友同Kent Back谈话的具体内容和过程，不过这里还是作为一个要点谈谈自己的想法） 四、每次写一点（原子级）测试 五、Clean code that works，（当然包括测试代码啦，^_^） 六、对于一个应用框架，最好是针对这个框架先写一个测试框架（这其实是一个很具体的内容，不过现在Java在WEB方面用得很多，测试相对来说也比较难些，所以有这点） 七、时刻提醒自己TEST FIRST的目的。（我们的目的是驱动开发，而

不是为了测试，呵呵，这点是前面第一点和第二点和起来一样，之所以还要单独列，只是再次提醒，所以这一点我后面不作详细阐述） 八、偷懒是程序员的通病，但是小偷懒就别了。（我有这样的观点：程序员的水平高低，其实从他偷懒的程度上是可以看出来的……^_^） 在开始具体来说上述要点之前，我想先写个例子，只是觉得应该写个例子，^_^，我的文采实在不是很好的，所以凭感觉的。 一个例子： 我有一个专门用于将数据库操作结果集（ResultSet）解析成一个DOM的Document对象的类，这个类可以根据给定一个XML配置模板中的一个定义节点（declare节点）的子节点（column节点）集合来解析ResultSet生成Document对象，其中每个column节点都定义了要从ResultSet中获取的某一个字段的属性，包括字段名、该字段在展示是是否可修改（editable）、是否有格式化模式属性（pattern）用于格式化该字段的数据等等；为了做得更通用，也可以依据ResultSet返回的ResultSetMetaData对象来生成column节点，再由column节点获取数据体，当然这样做有很大的局限性，比如该column节点的pattern、editable等等属性的设置都不会太灵活。这个设计，其最大的灵活性被放在XML配置模板上，由模板的定义获取数据，并且定义了数据的展示属性，而一旦column节点是根据给定的ResultSet来自动生成时，灵活性大大折扣，虽然在大多数应用中，都不会使用由ResultSet来自动生成，但是如果一开始并不能确定定义列时却是必须这样做，特别是在ResultSet输出的字段数量是变化的时候。问题终于出来了，最近有一个应用就是这样，首先是必须要使用从ResultSet获取定义节点（column），然后，在完成了所有的代码后，发现

给定ResultSet中都存在冗余字段，这个时候，没办法，只能是修改程序来适应它了。 在遇到这个麻烦，并确定必须修改自己代码后（老实说，第一反应当然是让人修改SQL来去掉冗余字段了，因为最初的设计根本就是不能有冗余数据的，不过确实是大家都有本难念的经啊，SQL是不能改了，因为数据库端的实现使用了一个稳定的公共实现，庆幸的是冗余字段是相同的），我脑袋里蹦出的第一个念头是添加一个事件监听器，来监听这个应用中生成数据部分（为了叙述方便，姑且叫“ dataBuilder ”吧）的代码，一旦数据生成就触发ResultSet解析完成事件，然后我可以写监听器来处理解析完成的结果集（当然就是将冗余的数据CUT掉啦），这样以后再出现其它类似的状况，我可以通过添加新的监听器来过滤数据而不需要动原有的代码。SWEAT，看来这个想法还算可行，至少比写一个子类看起来简单多，以后修改也容易多，动手吧。（其实要注意这个事情的发生环境，首先是码本来都OK了的，而后来突然发现这个问题，而这个问题是需要立刻修改掉的，所以没有太多时间来仔细考虑，我总是犯这样的错误。） 这个时候我有两个选择，一是直接就写代码，一是先写个测试。当然，我选择的是先写个测试，之所以摆明了这二个当然是为了比较了。先说如果我先写代码，那么我就直接进入了目标功能的角色，因为现在似乎目标很明确，我要给dataBuilder添加一个能处理监听器的功能，在使用ResultSet解析器解析完成一个Document后触发解析完成事件，通知所有注册的监听器，并将解析完成的结果通过事件对象传递给监听器进行处理。在以前，我会立刻想到如何添加事件，如何处理事件列表，还有两个必要的接口，一个是

监听器接口，一个是事件接口，一些简要的构思之后，不用多少时间就可以完成这些工作，然后就开始调试。上面那么说，主要是为了对比，现在我是先写测试的。当然，如上所述目标现在似乎也很明确的，那么无论如何写个测试类吧，在写上必要的to do list之后，我开始想如果现在代码写完了，我要怎样来使用它呢。哦，对了，测试这个之前还要写个测试使用的监听器实现，这个实现可以很简单，把解析结果Document干掉好了，呵呵，验证结果还更容易，那就把它所有节点remove掉好了，^_^。（注：这里的测试还没有到主功能，只是先做测试监听器部分）不过这个时候，我突然觉得目前这个设计似乎还是有些麻烦（懒惰是程序员的通病，sweat，每次想偷懒都想起这句），要写监听器，还要让dataBuilder处理监听列表触发事件，虽然让过滤数据操作可以很独立地添加，而监听器的注册也可以通过XML配置文件来完成，但还是显得多余，好像目前这个需求只要一个Decorator模式，写一个修饰类来修饰ResultSet解析器就差不多了，以后需要新的功能，换Decorator就可以了，也可以相互嵌套来完成多个功能，这样的话，就不需要对dataBuilder动太多手脚了。sweat，还好自己没动手瞎忙（无论如何，测试优先让我重新认识自己的设计，以及目标，于是我义无反顾地抛弃了原来的想法）。新目标出现了，看来我需要一个解析器接口实现的Decorator类（注：这个ResultSet解析器类原本就是一个接口ResultSetParser的实现），我可以先写一个扩展解析器接口的抽象类来包装下，以后的Decorator实现都从这个抽象类继承，直接实现修饰内容就可以了（实际我一直认为，在Decorator模式中所有Decorator实现类都有一个父抽象

类继承自修饰目标类的接口，其最主要的目的是使Decorator实现类功能更清晰，因为实际这个抽象类要包装的东西其实很少，这些移到子类中也完全可以，这样的话子类就是直接实现修饰目标类的接口了，效果一样，所以我认为这里有一个这样的抽象类统一由所有修饰类继承，更主要的是宣布，一个修饰目标类接口的实现类它是一个修饰类，这在阅读程序，以及使用该API上可以达到很好的效果。）。OK，就这么办（我总是很容易下决定呢，^_^）。 