1 **引言(Introduction)**

作为软件开发人员来说频繁的更改需求是一件痛苦的事情，为什么痛苦呢？一方面来自于需求的更迭，另一方面也根我们对软件的合理架构密不可分，一个合理的软件架构足够应付需求的未知变化，这体现了软件的可修改性。在考虑软件的可修改性上作出的架构很大程度上将减少需求更迭给我们带来的“痛苦”。

**2 可修改性理解（Modifiability Understanding）**

可修改性理解可理解为：指系统或软件的能够快速地以较高的性价比对系统进行变更的能力。比如说：对于一个网站，我们要修改它某一板块的UI界面，当我们对界面进行修改时是否会引起对另一个UI模块的影响，是否会引起后台控制，业务逻辑代码的变更，是否会引起整个网站的崩溃，这体现了一个网站的整个架构的是否具备可修改性。

**3 引起修改的因素（Factors causing revision）**

为什么会引起系统的修改？笔者认为最主要（甚至是唯一）的因素是：需求，这个需求会影响到系统给我们带来的收益。它可包含两个方面：1、用户需求，2、系统内在需求。首先在这阐述下“需求，成本，修改”三者的关系：需求无处不在，时时刻刻产生，判别一个需求的重要性来自于它对系统的成本产生的影响，如果严重影响了系统带来的收益，那必须对系统进行修改，如果某一部分相对于系统收益来说微不足道，甚至不会影响系统的收益，那改不改都“无可厚非”，而且有可能你对这部分修改了还会引起系统的故障。例如：一个网站的图标有人觉得不好看，有人觉得不错。场景是：使用的人们已经习惯这个图标的表示或者惯性的认为这个图标代表什么。那么是否对这个图标进行修改？我以为没有必要修改，如果你修改了反而可能会引起使用者的错误判断，比如：这个网站是停止运维了吗？好，那我去用其他的系统吧。类似错误的判断会影响到系统的收益，所以不进行修改就挺好的。

以上说明可理解为用户需求引起的修改，那么在解释下系统内在需求引起变更。

以淘宝为说明对象，根具相关资料，初期的淘宝使用的是MySQL作为它的数据库，众所周知MySQL是一个适用于小型企业的数据库，对于起初的淘宝也是适用的，因为没有产生MySQL无法处理的大量数据及相应的实时处理业务。但是近几年来淘宝的数据爆炸产生，在2012年11月30号到达全天访问用户总人数：2亿1千3百万，占中国网民40%；高峰期：每分钟成交订单89678笔，淘宝历年交易额如图1所示。

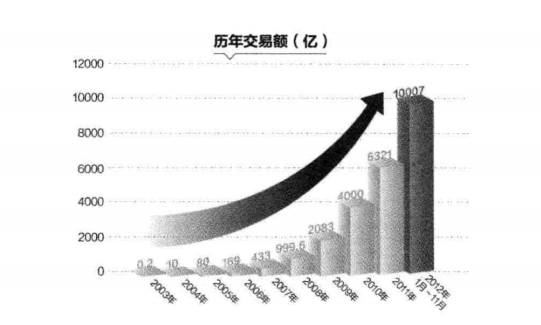


图1  淘宝历年交易额

这些大量业务数据就会引起系统“自我反馈”，如果不进行系统数据库方面的更新迭代会严重影响到淘宝未来的收益。正是在这系统本身产生的需求下，淘宝进行着对数据存储体系架构的不断升级优化。

4 **可修改性战术（Modifiable tactics）**

软件怎样具有可修改性，这里将介绍两种战术。

首先说明一下可修改性战术。可修改性战术的目标是控制实现、测试和部署变更的时间和成本。在具体的解释为根据相关战术，策略在时间和成本允许的范围内完成系统的相关变更。

以下讨论两种可修改性战术：

局部化修改和防止连锁反应

局部化修改战术，目标是把变更限制在一定范围，在设计期间为模块分配职责，把预期变更限制在一定范围内。关键点就是“限制变更范围”防止变更的内容对其他内容（或者说程序）产生影响。

防止连锁反应。我们平时编程无论是写函数还是写类，都会被其它的类或函数（方法）调用，如何实现对被调用部分的修改不会引起对调用它的部分的影响，这就是可以防止连锁反应。

无论是局部化修改还是防止连锁反应，都是基于“高内聚，低耦合”思想。那么根据软件设计模式的设计原则来分析：

（1）局部化意味着实现“模块化”思想，这里模块这样解释，一个模块只完成一个部分，这就符合设计模式中“单一职责原则”的设计原则，使每一个模块责任单一，防职责过多引起引起整体变更时的繁琐，复杂；

（2）要符合“接口隔离”设计原则，为后期的变更提供接口，当然也要规定接口调用的规范。同时根据“里氏代换”原则将接口的实例化由接口实现类完成，在此之上基于“依赖倒转原则”搭建的上层服务如果需要修改相关服务可以实现局部化修改，不用再对上层服务进行修改。在防止连锁反应的战术中“维持现有接口”就是为上层服务提供一个可使用的接口，当进行变更时对接口的实现类变更即可，无论是实现“适配器”还是“信息隐藏”都可封装在接口实现类中。

（3）降低模块之间的依赖性是防止连锁反应的必要条件。可据“迪米特法则”来实现。所谓“迪米特法则”，在C++中可理解为“友元”，java理解：在类与类间加入第三方类作为沟通，信息交流的“缓冲”，两个类间不直接相互依赖。这种第三方类可叫做“朋友”，可叫做“仲裁者”。在设计模式中可表示为“桥”“调停者”“代理”“工厂”，这些都是为了降低类间依赖。当然这又产生一个问题：大量使用第三方类使程序繁琐，复杂，这就需要架构师设计好一个明确的类图了。

以上基于软件设计模式分析了两个可修改性战术。

**5结论(Conclusion)**

我们在做软件开发时，肯定会自觉的将需求抽象成软件的功能，这是毋庸置疑的，但是我们在开发前的架构设计阶段不单单要考虑软件的将要完成的功能，还要将软件的质量属性考虑进去，因为这将影响着一个软件的长期发展。