


为啥你写的代码总是这么复杂？



华为云开发者联盟 
已认证帐号

24 人赞同了该文章

摘要：有句话说得很好，“代码质量决定生活质量”，当你把软件的复杂性降低了，bug减少了，系统可维护性更高了，自然也就带来了更好的生活质量。

本文分享自华为云社区《[写出的代码复杂度太高？看下专家怎么说](#)》，原文作者：元闰子。

前言

在进行软件开发时，我们常常会追求软件的高可维护性，高可维护性意味着当有新需求来时，系统易扩展；当出现bug时，开发人员易定位。而当我们说一个系统的可维护性太差时，往往指的是该系统太过复杂，导致给系统增加新功能时容易出现bug，而出现bug之后又难以定位。

那么，软件的复杂性又是如何定义的呢？

John Ousterhout给出的定义如下：

Complexity is anything related to the structure of a software system that makes it hard to understand and modify the system.

可见，软件的复杂性是一个很泛的概念，任何使软件难以理解和难以修改的东西，都属于软件的复杂性。为此，John Ousterhout提出了一个公式来度量一个系统的复杂性：

$$C = \sum_p c_p t_p$$

式中， p 表示系统中的模块， c_p 表示该模块的认知负担（Cognitive Load，即一个模块难以理解的程度）， t_p 表示在日常开发中在该模块花费的开发时间。

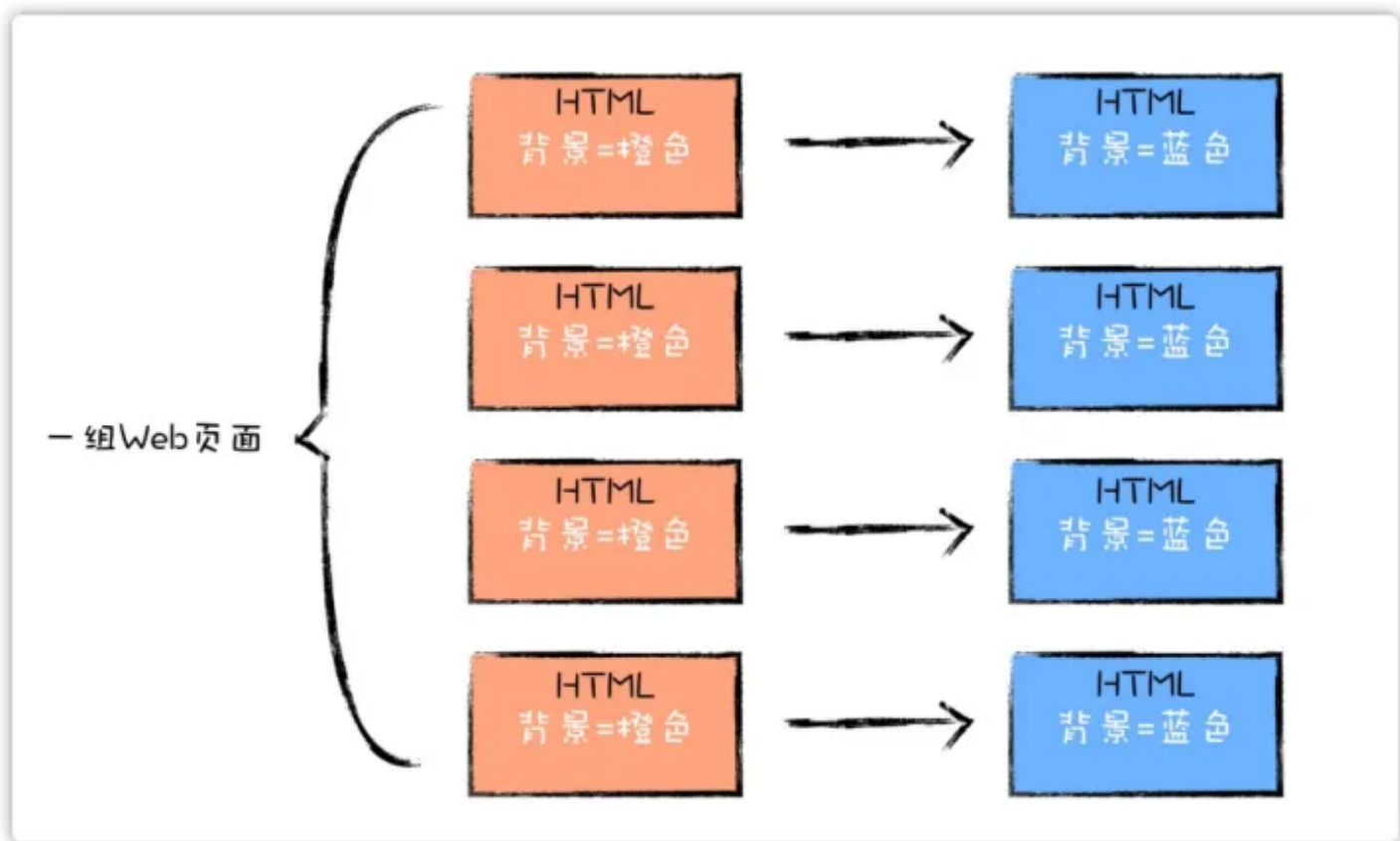
从公式上看，一个软件的复杂性由它的各个模块的复杂性累加而成，而 模块复杂性 = 模块认知负担 * 模块开发时间，也就是模块的复杂性即和模块本身有关，也跟在该模块上花费的开发时间有关。需要注意的是，如果一个模块非常难以理解，但是后续开发过程中几乎没有涉及到它，那么它的复杂性也是很低的。

导致软件复杂的原因

导致软件复杂的原因可以细分出很多种来，而概括起来莫过于两种：*依赖 (dependencies)* 和 *隐晦 (obscurity)*。前者会让修改起来很费劲而且容易出现bug，比如当修改模块1时，往往也涉及到模块2、模块3、... 的改动；后者会让软件难以理解，定位一个bug，甚至是仅仅读懂一段代码都需要花费大量的时间。

软件的复杂性往往伴随着如下几种症状：

霰弹式修改 (Change amplification)。当只需要修改一个功能，但又不得不对许多模块作出改动时，我们称之为霰弹式修改。这通常是因为模块之间耦合过重，相互依赖太多导致的。比如，有一组Web页面，每个页面都是一个HTML文件，每个HTML都有一个背景属性。由于各个HTML的背景属性都是分开定义的，因此如果需要把背景颜色从橙色修改为蓝色时，就需要改动所有的HTML文件。



认知负担 (Cognitive load)。当我们说一个模块隐晦、难以理解时，它就有过重的认知负担，这种情况下往往需要读者花费大量时间才能明白该模块的功能。比如，提供一个不带任何注释的calculate接口，它有2个int类型的入参和一个int类型的返回值。从该函数的签名上看，调用者根本无法得知函数的功能是什么，他只能通过花时间去阅读源码来确定函数功能后才敢去调用该函数。

```
int calculate(int val1, int val2);
```

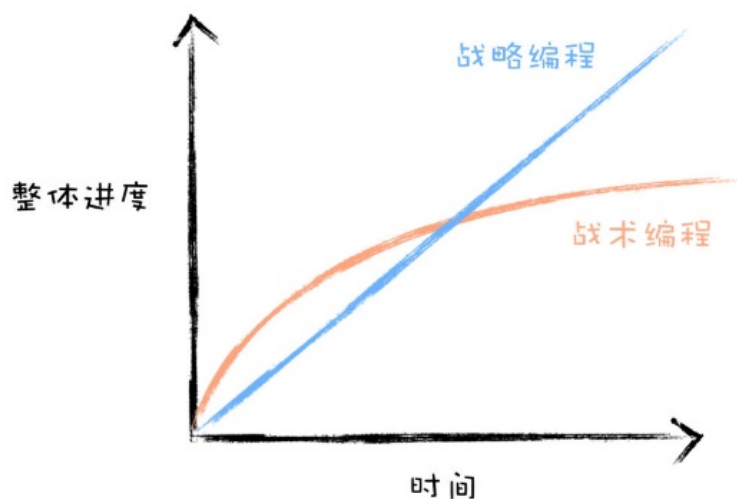
不确定性 (Unknown unknowns)。相比于前两种症状，不确定性的破坏性更大，它通常指一些在开发需求时，你必须注意的，但是又无从得知的点。它常常是因为一些隐晦的依赖导致的，会让你在开发完一个需求之后感觉心里很没谱，隐约觉得自己的代码哪里有问题，但又不清楚问题在哪，只能祈祷在测试阶段能够暴露而不要漏洞商用阶段。

如何降低软件的复杂性

对“战术编程” Say No!

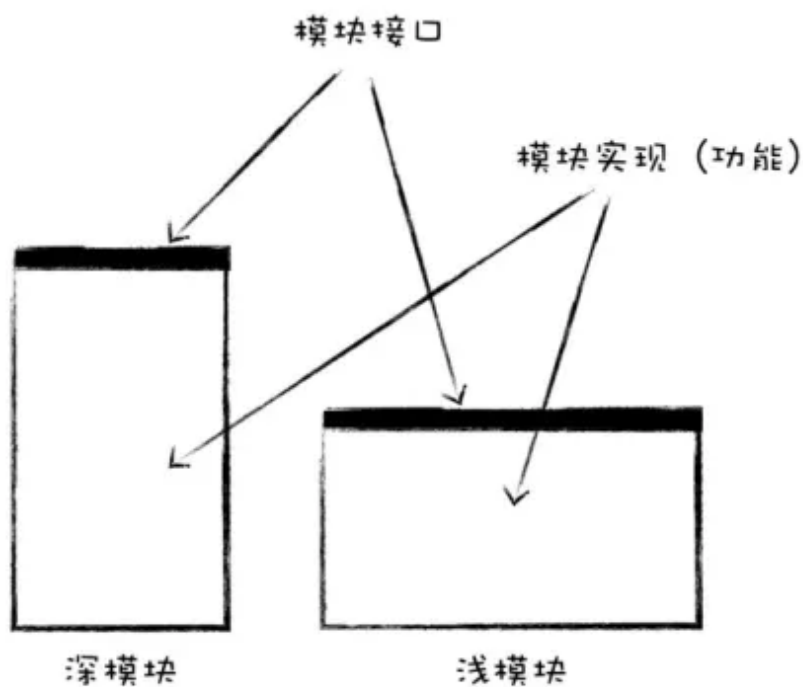
很多程序员在进行特性开发或bug修复时，关注点往往是如何简单快速让程序跑起来，这就是典型的**战术编程** (Tactical programming) 方法，它追求的是**短期的效益**——节省开发时间。战术编程最普遍的体现就是在编码之前没有进行模块设计，想到哪里就写到哪里。战术编程在系统前期可能会比较方便，一旦系统庞大起来、模块之间的耦合变重之后，添加或修改功能、修复bug都会变得寸步难行。随着系统变得越来越复杂，最后不得不对系统进行重构甚至重写。

与战术编程相对的就是**战略编程** (Strategic programming)，它追求的是**长期的效益**——增加系统可维护性。仅仅是让程序跑起来还不足以满足，还需要考虑程序的可维护性，让后续在添加或修改功能、修复bug时都能够快速响应。因为考虑的点比较多，也就注定战略编程需要花费一定的时间去进行模块设计，但相比于战术编程后期导致的问题，这一点时间也是完全值得的。



让模块更“深”一点!

一个模块由接口 (interface) 和实现 (implementation) 两部分组成, 如果把一个模块比喻成一个矩形, 那么接口就是矩形顶部的边, 而实现就是矩形的面积 (也可以把实现看成是模块提供的功能)。当一个模块提供的功能一定时, 深模块 (Deep module) 的特点就是矩形顶部的边比较短, 整体形状高瘦, 也即接口比较简单; 浅模块 (Shallow module) 的特点就是矩形顶部的边比较长, 整体形状矮胖, 也即接口比较复杂。



模块的使用者往往只看到接口, 模块越深, 模块暴露给调用者的信息就越少, 调用者与该模块的耦合性也就越低。因此, 把模块设计得更“深”一点, 有助于降低系统的复杂性。

那么, 怎样才能设计出一个深模块呢?

• 更简单的接口

简单的接口比简单的实现更重要, 更简单的接口意味着模块的易用性更好, 调用者使用起来更方便。而简单的实现 + 复杂的接口这种形式, 一方面影响了接口的易用性, 另一方面则加深了调用者与模块的耦合。因此, 在进行模块设计时, 最好遵守“把简单留给别人, 把复杂留给自己”的原则。

异常也属于接口的一部分，在编码过程中，应该杜绝没经过处理，就随意将异常往上抛的现象，这样只会增加系统的复杂性。

• 更通用的接口

在设计接口时，你往往有两种选择：（1）设计成专用的接口；（2）设计成通用的接口。前者实现起来更方便，而且完全可以满足当前的需求，但可扩展性低，属于战术编程；后者则需要花时间对系统进行抽象，但可扩展性高，属于战略编程。通用的接口意味着该接口适用的场景不止一个，典型的就是“**一个接口，多个实现**”的形式。

有些程序员可能会反驳，在无法预知未来变化的情况下，通用就意味着过度设计。过度通用确实属于过度设计，但对接口进行适度的抽象并不是，相反它可以使系统更有层次感，可维护性也更高。

• 隐藏细节

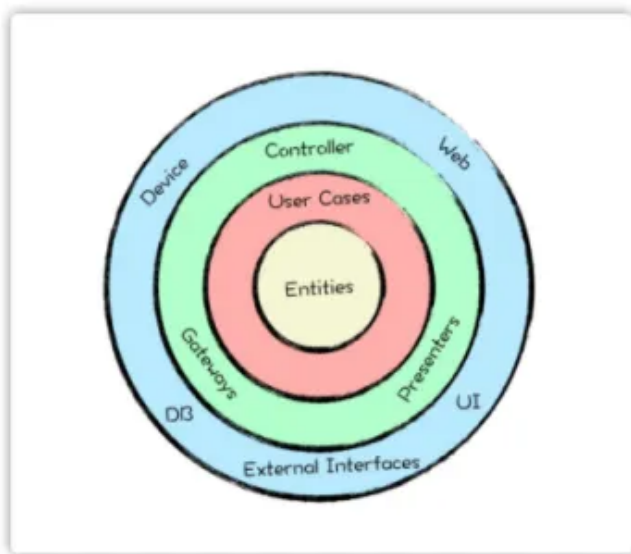
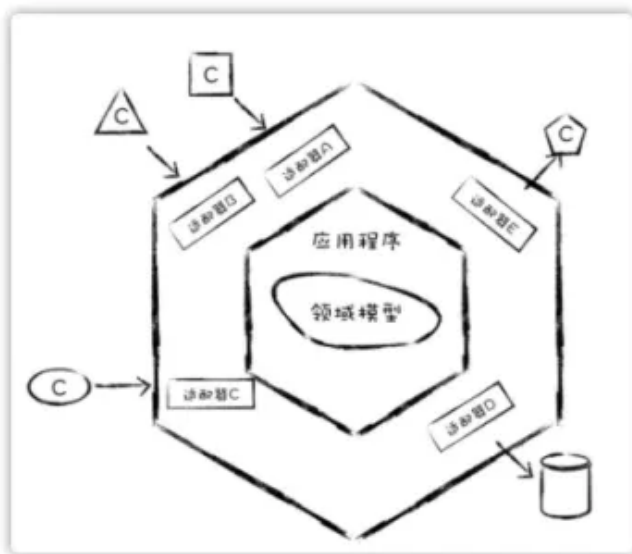
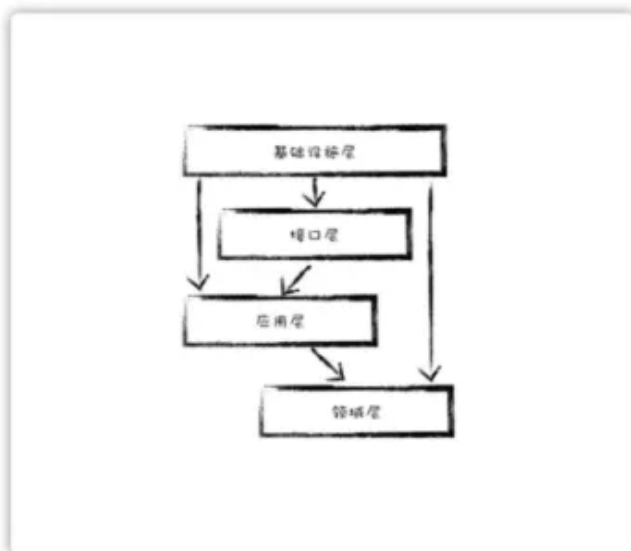
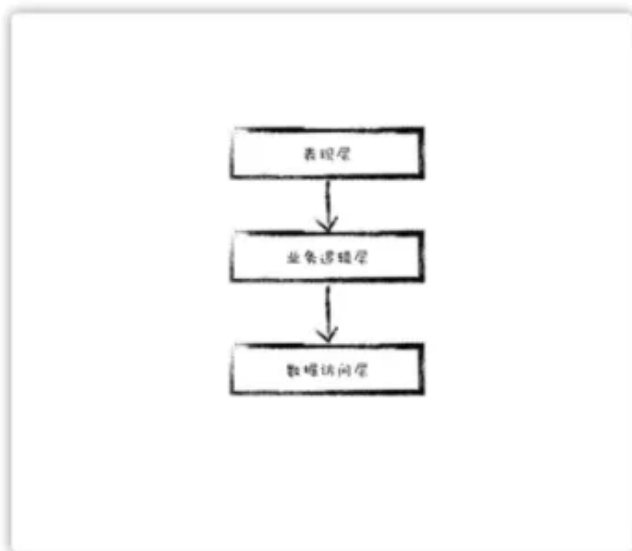
在进行模块设计时，还要学会区分对于调用者而言，哪些信息是重要的，哪些信息是不重要的。隐藏细节指的就是**只给调用者暴露重要的信息，把不重要的细节隐藏起来**。隐藏细节一则使模块接口更简单，二则使系统更易维护。

如何判断细节对于调用者是否重要？以下几个例子：

- 1、对于Java的Map接口，*重要的细节*：Map中每一个元素都是由<Key, Value>组成的；*不重要的细节*：Map底层是如何存储这些元素、如何实现线程安全等。
- 2、对于文件系统中的read函数，*重要的细节*：每次读操作从哪个文件读、读多少字节；*不重要的细节*：如何切换到内核态、如何从硬盘里读数据等。
- 3、对于多线程应用程序，*重要的细节*：如何创建一个线程；*不重要的细节*：多核CPU如何调度该线程。

进行分层设计！

设计良好的软件架构都有一个特点，就是层次清晰，每一层都提供了不同的抽象，各个层次之间的依赖明确。不管是经典的Web三层架构、DDD所提倡的四层架构以及六边形架构，抑或是所谓的Clean Architecture，都有着鲜明的层次感。



在进行分层设计时，需要注意的是，每一层都应该提供不同的抽象，并要尽量避免在一个模块中出现大量的 *Pass-Through Method*。比如在DDD的四层架构中，**领域层**提供了对领域业务逻辑的抽象，**应用层**提供了对系统用例的抽象，**接口层**提供了对系统访问接口的抽象，**基础设施层**则提供对如数据库访问这类的基础服务的抽象。

所谓的 *Pass-Through Method* 是指那些“在函数体内直接调用其他函数，而本身只做了极少的事情”的函数，通常其函数签名与被其调用的函数签名很类似。*Pass-Through Method* 所在的模块通常都是浅模块，让系统增加了无谓的层次和函数调用，会使系统更加复杂：

```
public class TextDocument ... {
    private TextArea textArea;
    private TextDocumentListener listener;
    ...
    public Character getLastTypedCharacter() {
        return textArea.getLastTypedCharacter();
    }
    public int getCursorOffset() {
        return textArea.getCursorOffset();
    }
    public void insertString(String textToInsert, int offset) {
        textArea.insertString(textToInsert, offset);
    }
    ...
}
```

学会写代码注释！

注释是软件开发过程中的性价比极高的一种手法，它只需要花费20%的时间，即可获得80%的价值。它可以**提高晦涩难懂的代码的可读性**；可以起到**隐藏代码复杂细节**的作用，比如接口注释可以帮助开发者在没有阅读代码的情况下快速了解该接口的功能和用法；如果写的好，它还可以**改善系统的设计**。

具体如何写好代码注释，参考 [《如何写出优秀的代码注释？》](#) 一文。

总结

软件的复杂性是我们程序员在日常开发中所必须面对的东西，学会如何 **“弄清楚什么是软件复杂性，找到导致软件复杂的原因，并利用各种手法去战胜软件的复杂性”** 是一门必备的能力。有句话说得很好，“代码质量决定生活质量”，当你把软件的复杂性降低了，bug减少了，系统可维护性更高了，自然也就带来了更好的生活质量。

模块设计是降低软件复杂度最有效的手段，学会使用 **“战略编程”** 的方法，并坚持下去。我们常常提倡 **“一次把事情做对”**，但这对于模块设计而言并不适用，几乎没有人可以第一次就把一个模块设计成完美的模样。二次设计是一个非常有效的手法，与其在系统腐化之后再花大量时间进行重构或重写，还不如在第一次完成模块设计后，再花点时间进行二次设计，多问问自己：是否有更简单的接口？是否有更通用的设计？是否有更简洁高效的实现？

“罗马不是一天建成的”，降低软件的复杂性也一样，贵在坚持。

[点击关注，第一时间了解华为云新鲜技术~](#)

发布于 2021-05-12 14:06

- 软件开发
- 代码质量
- Bug

2 条评论

默认 最新



临鞭被策

代码先是写给人看的，然后才是写给编译器和解析器读的。

2021-05-12

回复

2



Yi-Jiang

代码质量决定生活质量

2021-05-12

回复

赞