目錄

软件设计模式大全

设计模式不是软件设计的起点，而是终点。本书按创建型（Creational）、结构型（Structural）、行为型（Behavioral）和并发型（Concurrency）对设计模式进行分类，并使用 C++11/14 的语法描述了这些这些设计模式的具体实现。最后，讨论了现代软件架构中主的几种主流架构模型。

引言

设计模式是软件工程中对于可复用的代码现象整理得出的一套设计方案。在《设计模式：可复用面向对象软件的基础》一书作者四人帮(GoF)收录了 23 种设计模式，成为了设计模式领域的标志性著作。此书年代久远，其中的很多模式不仅古老陈旧，而且在实际的软件开发过程中，即便使用有些模式也会造成对软件的过度设计，反而不利于软件维护。

本书尝试对(所有的)设计模式进行总结，并针对各个模式的优缺点与现代软件行业实践结果进行讨论。进而达到方便查阅、快速学习的目的。

目标读者

1. 本书假定读者已经具备了一定的实际编程经验，并适合那些希望学习或快速回忆设计模式的读者；
2. 本书使用 C++11/14 语法来讨论众多设计模式的具体实现，但对于模式本身的讨论是独立于编程语言本身的，因此对于其他语言的使用者（例如 JavaScript/Python 等）来说，同样可以阅读本书。

内容一览

    - 03.抽象工厂模式 - Abstract Facotry
    - 04.原型模式 - Prototype
    - 05.建造者模式 - Builder
    - 06.单例模式 - Singleton
- 第三章 结构型模式
    - 07.装饰者模式 - Decorator
    - 08.外观模式 - Facade
    - 09.适配器模式 - Adapter
    - 10.组合模式 - Composite
    - 11.享元模式 - Flyweight
    - 12.代理模式 - Proxy
    - 13.桥接模式 - Bridge
- 第四章 行为型模式
    - 14.策略模式 - Strategy
    - 15.模板方法模式 - Template Method
    - 16.观察者模式 - Observer
    - 17.状态模式 - State
    - 18.备忘录模式 - Memento
    - 19.迭代器模式 - Iterator
    - 20.责任链模式 - Chain of Responsibility
    - 21.中介者模式 - Mediator
    - 22.命令模式 - Command
    - 23.解释器模式 - Interpreter
    - 24.访问者模式 - Visitor
- 第五章 并发型模式
- 第六章 设计模式混合与现代软件架构
    - 模型视图控制器架构 - Model View Controller
    - 模型视图视图模型架构 - Model View ViewModel
    - 发布/订阅架构 - Publish/subscribe

交流

1. 本书在每节的最下方提供了评论，由于笔者水平有限，如果读者发现教程中内容出现错误，可以通过评论或者通过发 issue 来指出，让我把这个教程做得更好；
2. 本书由于使用 C++11/14 来实现各种设计模式，因此有以下交流群，加群请注明

群名称：Modern C++ 交流群
群　号：306196433

`gitbook` ：

GitHub

本书中设计的相关代码可以在 GitHub 上查看：

版权声明

(cc) BY-NC-ND
本教程系 欧长坤 原创，采用知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议进行许可。

第一章 设计模式基本准则

复制和复用

在学习设计模式之前，我们先看看编写代码时候的复制和复用的现象。

在 C 和 C++ 两门语言的标准库就能看到复制与复用的差别。C++ 与 C 不同，C++ 提供了强大的模板，这让我们能够非常轻易的复用我们的代码，让编译器去完成所谓的『复制』工作。我们来看标准库的一个例子。

在 C 的标准库中，对于取绝对值这一简单的功能而言，就提供了十多个函数以供调用：

```
int abs(int n);
long labs(long n);
long long llabs(long long n);
intmax_t imaxabs(intmax_t n);
float fabs(double arg);
float fabsf(float arg);
long double fabsl(long double arg);
float cabsf(float complex z);
double cabs(double complex z);
long double cabsl(long double complex z);
...
```

换句话说，为了实现对某个值求绝对值这个功能，我们不得不针对不同的类型为这个函数实现若干个不同的版本，这对于希望高度复用代码的我们来说，是难以接受的。好在 C++ 具备模板这一强大的功能，因此对于 `abs` 这个简单的功能，只需定义一个模板参数就能完成：

```
template<typename T>
T abs(const T& n);
```

单一职责原则

> 单一职责原则（**Single Responsibility Principle, SRP**）：
>
> 每个类有且只负责为软件提供一个功能。

开放封闭原则

> 开放封闭原则：
>
> 软件实体（类、函数等）应可以扩展，但不能修改。

依赖翻转原则

> 依赖翻转原则：
>
> 1. 高层模块不应该依赖低层模块，两个都应该依赖抽象。
> 2. 抽象不应该依赖细节，细节应该依赖抽象。

接口隔离原则

里氏转换原则

合成聚合复用原则

迪米特原则