Morden cpp notes — design pattern

Thursday, December 10, 2020 2:43 PM

设计模式系统地描述了软件开发中常见问题,给出专家级别的设计思路和指导原则.设计模式有3大类
- 创建型模式
- 结构型模式
- 行为模式.

分别对应了开发面向对象系统的三个问题:
- 如何创建对象
- 如何组合对象
- 如何处理对象间的动态通信和职责分配.

- 设计模式
  - 描述体例
    - 名称
    - 别名
    - 动机
    - 结构
    - 示例
    - 效果
    - 相关模式
  - 创建型模式
    - 抽象工厂
    - 生成器
    - 工厂方法
    - 原型
    - 单件
  - 行为模式
    - 职责链
    - 命令
    - 解释器
    - 迭代器
    - 中介者
    - 备忘录
    - 策略
    - 状态
    - 观察者
    - 模板方法
    - 访问者
  - 结构型模式
    - 适配器
    - 桥接
    - 组成
    - 代理
    - 享元
    - 外观
    - 装饰

最常用的5个设计原则 SOLID:
1. SRP (Single Responsibility Principle) 单一职责.
2. OCP (Open Closed Principle) 开闭
3. LSP (Liskov Substitution Principle) 里氏替换.
4. ISP (Interface-Segregation Principle) 接口隔离.
5. DIP (Dependency Inversion Principle) 依赖反转/依赖倒置.

单一职责原则:即高内聚低耦合.功能明确单一.
一个反例是C++的string,集成了字符串和字符容器的双重身份,接口复杂,所以我们应只把它当作字符串,而把字符容器的工作交给 vector<char>.

开闭原则:对扩展开放,对修改关闭.关键是做好封装,隐藏内部的实现细节.开放足够的接口,可以只通过接口扩展功能,不必侵入类内部. Eg:
1. 桥接模式让接口保持稳定,另一边的实现任意变化,
2. 迭代器模式让集合保持稳定,改变访问集合的方式只需变动迭代器.

C++里的final也是实现开闭原则利器,用在类和成员函数上,可以有效防止子类的修改.

里式替换原则:子类必须能完全替换父类,子类不能改变,违反父类定义的行为.

接口隔离原则:尽量简化,归并给外界调用的接口.

依赖反转原则:上层尽量避免下层的实现细节,下层要反过来依赖上层的抽象定义.

- 设计原则
  - SRP
    - 高内聚低耦合
    - 工厂,适配器,装饰,代理
    - 反例:std::string
  - OCP
    - 对扩展开放,对修改关闭
    - 桥接,迭代器
    - C++里的final
  - LSP
    - 子类能够完全替代父类
    - 在C++里应少用继承,多用泛型
  - DIP
    - 解耦依赖关系.
    - 抽象工厂,桥接,适配器
  - ISP
    - 简化,归并对外接口
    - 适配器,装饰,外观

除了SOLID,还有
- DRY: Don't repeat yourself
- KISS: keep it simple stupid.

含义都是让代码保持简单,避免重复的代码, Eg:
- 用宏代替字面值
- 用lambda就地定义函数
- 多使用容器,算法.

C++中的应用

创建型模式
隐藏了类的实例化过程和细节,让对象的创建独立于系统的其他部分

- 创建型模式
  - 抽象工厂
  - 生成器
  - 工厂方法
  - 原型
  - 单件

单件.
控制对象的创建数量,只能有一个实例.为了避免在多线程中多次初始化单件,可以使用函数内部的static静态变量. C++会保证静态变量的初始化是线程安全的.

```
auto & instance ()   // 生产单件对象的函数.
{
  static T obj; // 静态变量
  return obj;  // 返回对象的引用
}
```

抽象工厂,工厂方法.
抽象工厂是一个类,工厂方法是一个函数.它们都用来生产对象,可以用DRY(Don't repeat yourself)来理解,避免重复的代码,是"对new的封装".隔离了客户代码和创建对象,两边只能通过工厂交互,实现了解耦,就可以在工厂模式里随意控制生产的方式,生产的时机,生产的内容.

make_unique(), make_shared()就是对工厂模式的具体应用,封装了创建的细节,看不见new,直接返回智能指针对象,接口更简洁.

```
auto ptr1 = make_unique<int>(42);
auto ptr2 = make_shared<string>("metroid");
```

使用工厂模式的关键,是理解它面对的问题和解决问题的思路,比如创建专属对象,创建成套的对象,重点是"如何创建对象,创建出什么样的对象",用函数或者类比单纯用new更灵活.

结构型模式
关注的是对象的静态联系,以灵活,可拆卸,可装配的方式组合出新的对象.

- 结构型模式
  - 适配器
  - 桥接
  - 组成
  - 装饰
  - 外观
  - 享元
  - 代理

适配器模式
目的是接口转换,不需要修改源码就能够把一个对象转换成可以在本系统中使用的形式. Eg. array包装了C++的原生数组,转换成了容器,让"裸内存数据"也可以接入标准库的泛型体系.

```
array<int, 5> arr = {0, 1, 2, 3, 4};
auto b = begin(arr);
auto e = end(arr);
for_each(b, e, [](int x) {...});
```

外观模式
封装了一组对象,简化对象的通信关系,提供一个高层次的易用接口,特点是内部会操作很多对象,然后对外表现成一个对象, Eg. async()封装了线程的创建,调度等细节.

```
auto f = std::async([](){...});
f.wait();
```

代理模式.
包装一个对象,不是为了适配插入系统,而是要控制对象,不允许外部直接与内部对象通信,可以限制,隐藏,增强,或者优化一个类, Eg. C++里的智能指针接管了原始指针,限制了某些危险操作,添加了自动生命周期管理,

行为模式:
描述了对象之间动态的消息传递,也就是对象的行为,工作方式.

- 行为模式.
  - 职责链
  - 命令
  - 解释器
  - 迭代器
  - 中介者
  - 备忘录.
  - 观察者
  - 状态
  - 策略
  - 模板方法
  - 访问者

职责链和命令经常一起使用.职责链把多个对象串成一个"链条",让里面的每一个对象都有机会处理请求,而请求通常用命令模式.把相关的数据打包成一个对象,解耦请求的发送方和接收方.
Eg. C++的异常处理机制,就是职责链+命令的应用.异常类exception就是一个命令对象, throw抛出异常就是发起了一个请求处理流程,而一系列的try-catch块就构成了异常处理的职责链,异常会自下向上地走过函数调用栈(职责链),直到在链条中找到一个能够处理的catch块.

策略模式
封装了不同算法,可在运行时灵活的互相替换.从而在外部非侵入地改变系统的行为内核.不会改变类的外部表现和内部状态,只是动态替换一个很小的算法功能模块.
Eg. 容器算法中的比较函数,散列函数, for_each里的lambda,
策略模式也非常适合应用在if-else/switch-case这样"分支决策"代码中,可以把每个分支逻辑封装成类或lambda表达式,再存进容器,让容器来查找最合适的处理策略.

	创建型模式	结构型模式	行为模式
特点.	隔离对象的生产和使用	隔离客户代码与原对象的接口.	隔离程序里稳定的主逻辑与动态变化的部分
简单(常用)	单件,工厂	适配器,外观,代理	职责链,命令,策略.
普通(常用)	生成器,原型	桥接,装饰	迭代器,观察者,状态,模板方法,访问者
困难(不常用)		组成,享元	解释器,备忘录,中介者.