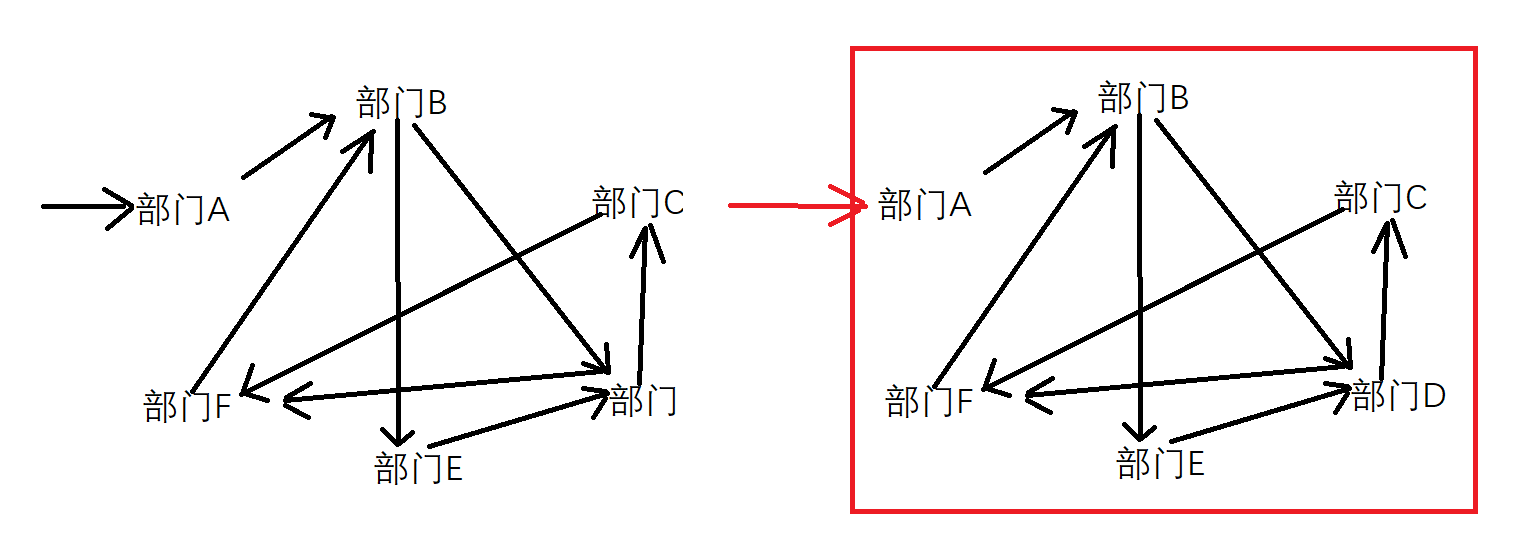
# 外观模式（Facade Pattern）

## 问题引入

现实生活中有“一站式服务”的概念，所谓一站式服务，其实就是服务的集成、整合，其目的是简化操作流程，“一人受理，内部运作，方便办事，提高效率”。在现实生活中，常常存在办事较复杂的例子，如办房产证或注册一家公司，有时要同多个部门联系，这时要是有一个综合部门能解决一切手续问题就好了。

软件设计也是这样，当一个系统的功能越来越强，子系统会越来越多，客户对系统的访问也变得越来越复杂。这时如果系统内部发生改变，客户端也要跟着改变，这违背了“开闭原则”，也违背了“迪米特法则”，所以有必要为多个子系统提供一个统一的接口，从而降低系统的耦合度，这就是外观模式的目标。



## 定义

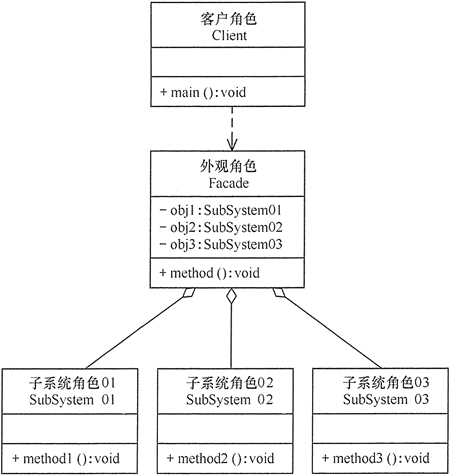
外观设计模式是一种通过为多个复杂的子系统提供一个一致的接口，使这些子系统更加容易被访问的设计模式。改模式对外提供一个统一接口，调用方不需要关心内部子系统的具体细节，这大大降低了应用程序的复杂度、耦合度，并提高了系统的可维护性、可拓展性。

## 特点

外观模式是是“迪米特法则”的典型应用，它有以下主要优点：

1. 减少系统相互依赖。降低了子系统与客户端之间的耦合度，使得子系统的变化不会影响调用它的客户类，有利于系统拓展。
2. 对客户屏蔽了子系统组件，减少了客户处理的对象数目，并使得子系统使用起来更加容易。
3. 提高系统安全性。由于客户不需要直接访问子系统，降低了复杂工作流程出错的可能性。
4. 降低了大型软件系统中的编译依赖性，简化了系统在不同平台之间的移植过程，因为编译一个子系统不会影响其他的子系统，也不会影响外观对象。

外观模式的缺点如下：

1. 不能很好地限制客户使用子系统类。
2. 增加新的子系统可能需要修改外观类或客户端的源代码，违背了“开闭原则”。

## 结构

外观模式包含以下主要角色：

1. 外观角色：为多个子系统对外提供一个共同的接口。
2. 子系统角色：实现系统的部分功能，客户可以通过外观角色访问它。
3. 客户角色：通过一个外观角色访问各个子系统。

结构图：

## 应用场景

通常在以下情况可以考虑使用外观模式：

1. 构件分层结构系统时，使用外观模式定义子系统中每层的入口点可以简化子系统之间的依赖关系。
2. 当一个复杂系统包含的子系统很多时，外观模式可以为系统设计一个简单的接口同外界访问。
3. 当客户端与多个子系统之间存在很大联系时，引入外观模式可以将它们分离，从而提高系统的独立性和可移植性。

## 模式拓展

在外观模式中，当增加或移除子系统时需要修改外观类，这违背了“开闭原则”。如果引入抽象外观类，则在一定程度上解决了该问题，其结构图如图：

