# 概述

日常生活中，我们出去吃饭都会遇到下面的场景。



定义:

快餐厨师根据订单准备餐点

将一个请求封装为一个对象，使发出请求的责任和执行请求的责任分割开。这样两者之间通过命令对象进行沟通，这样方便将命令对象进行存储、传递、调用、增加与管理。

# 结构

命令模式孢含以下主要角：

* 抽象命令类(Command)角色:定义命令的接口，声明执行的方法。
* 具体命令(concrete command)角色:具体的命令，实现命令接口;通常会持有接收者，并调用接收者的功能来完成命令要执行的操作。
* 实现者/接收者(Receiver)角色: 接收者，真正执行命令的对象。任何类都可能成为一个接收者，只要它能够实现命令要求实现的相应功能。
* 调用者/请求者(Invoker)角色:要求命令对象执行请求，通常会持有命令对象，可以持有很多的命令对象。这个是客户端真正触发命令并要求命令执行相应操作的地方，也就是说相当于使用命令对象的入口。

# 案例实现

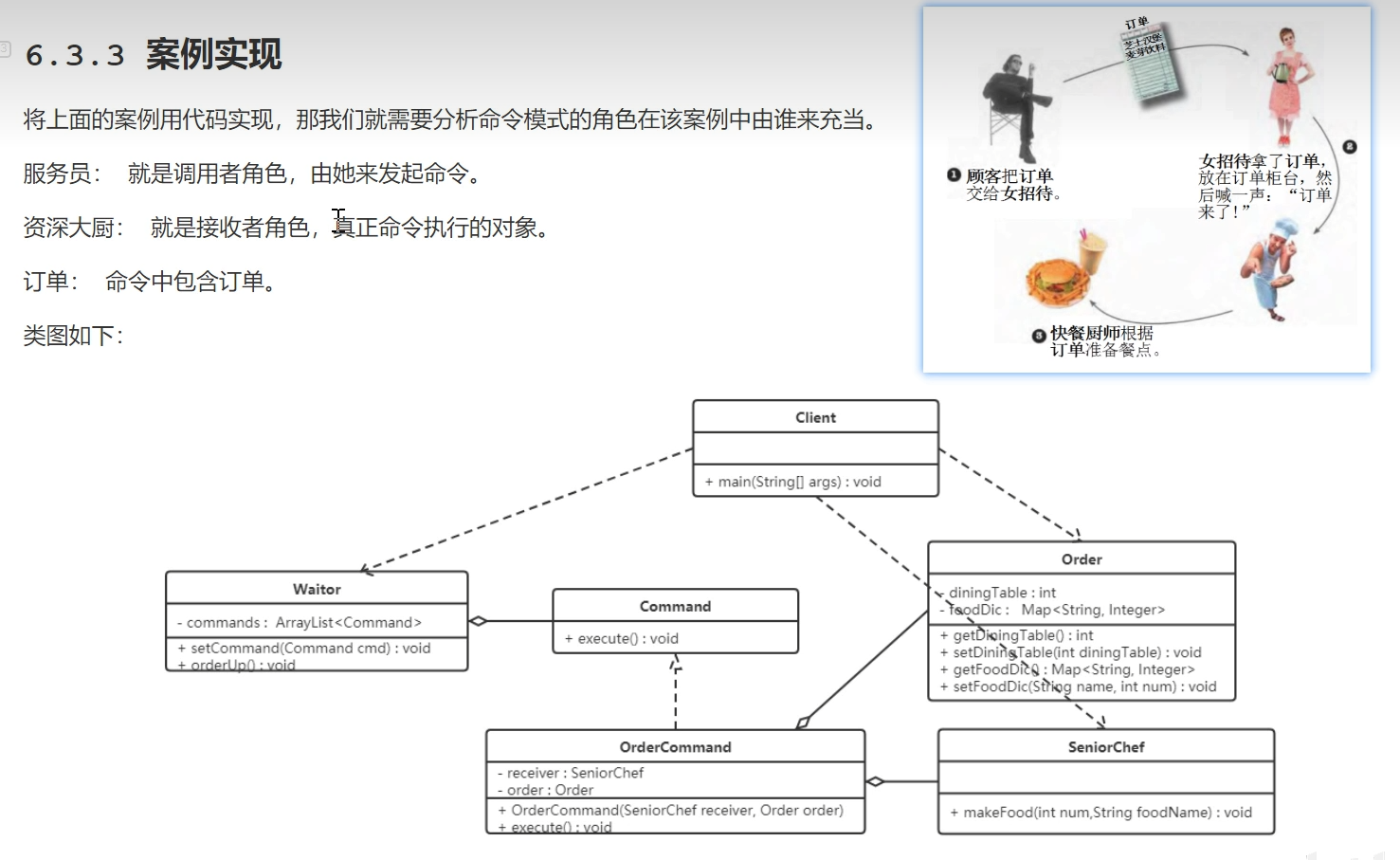
将上面的案例用代码实现，那我们就需要分析命令模式的角色在该案例中由谁来充当。

服务员: 就是调用者角色，由她来发起命令。

资深大厨:就是接收者角色，真正命令执行的对象。

订单: 命令中包含订单。

类图如下:



# 优缺点

1，优点:

* 降低系统的耦合度。命令模式能将调用操作的对象与实现该操作的对象解耦。
* 增加或删除命令非常方便。采用命令模式增加与删除命令不会影响其他类，它满足“开闭原则”，对扩展比较灵活。
* 可以实现宏命令。命令模式可以与组合模式结合，将多个命令装配成一个组合命令，即宏命令。
* 方便实现 undo 和 Redo 操作。命令模式可以与后面介绍的备忘录模式结合，实现命令的撤销与恢复。

2，缺点:

* 使用命令模式可能会导致某些系统有过多的具体命令类，
* 系统结构更加复杂。

使用场景

* 系统需要将请求调用者和请求接收者解耦，使得调用者和接收者不直接交互。
* 系统需要在不同的时间指定请求、将请求排队和执行请求。
* 系统需要支持命令的撤销(Undo)操作和恢复(Redo)操作，