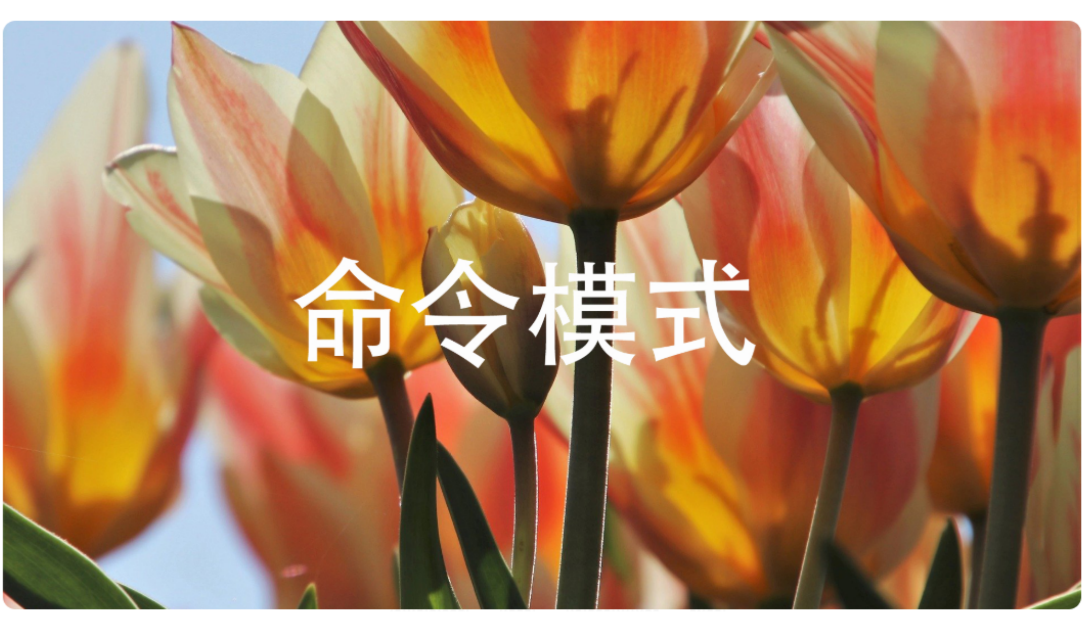
71 | 命令模式：如何利用命令模式实现一个手游后端架构？

王争 2020-04-15



00:00 1.0x
讲述：冯永吉 大小：8.51M 时长：09:17

设计模式模块已经接近尾声了，现在我们只剩下 3 个模式还没有学习，它们分别是：命令模式、解释器模式、中介模式。这 3 个模式使用频率低、理解难度大，只在非常特定的应用场景下才会用到，所以，不是我们学习的重点，你只需要稍微了解，见了能认识就可以了。

今天呢，我们来学习其中的命令模式。在学习这个模式的过程中，你可能会遇到的最大的疑惑是，感觉命令模式没啥用，是一种过度设计，有更加简单的设计思路可以替代。所以，我今天讲解的重点是这个模式的设计意图，带你搞清楚到底什么情况下才真正需要使用它。

话不多说，让我们正式开始今天的学习吧！

命令模式的原理解读

命令模式的英文翻译是 Command Design Pattern。在 GoF 的《设计模式》一书中，它是这么定义的：

> The command pattern encapsulates a request as an object, thereby letting us parameterize other objects with different requests, queue or log requests, and support undoable operations.

翻译成中文就是下面这样。为了帮助你理解，我对这个翻译稍微做了补充和解释，也一起放在了下面的括号中。

命令模式将请求（命令）封装为一个对象，这样可以使用不同的请求参数化其他对象（将不同请求依赖注入到其他对象），并且能够支持请求（命令）的排队执行、记录日志、撤销等（附加控制）功能。

对于 GoF 给出的定义，我这里再进一步解读一下。

落实到编码实现，命令模式用的最核心的实现手段，是将函数封装成对象。我们知道，C 语言支持函数指针，我们可以把函数当作变量传递来传递去。但是，在大部分编程语言中，函数没法儿作为参数传递给其他函数，也没法儿赋值给变量。借助命令模式，我们可以将函数封装成对象。具体来说就是，设计一个包含这个函数的类，实例化一个对象传来传去，这样就可以实现把函数像对象一样使用。从实现的角度来说，它类似我们之前讲过的回调。

当我们把函数封装成对象之后，对象就可以存储下来，方便控制执行。所以，命令模式的主要作用和应用场景，是用来控制命令的执行，比如，异步、延迟、排队执行命令、撤销重做命令、存储命令、给命令记录日志等等，这才是命令模式能发挥独一无二作用的地方。

命令模式的实战讲解

上面的讲解比较偏理论，比较不好理解，我这里再结合一个具体的例子来解释一下。

假设我们正在开发一个类似《天天酷跑》或者《QQ 卡丁车》这样的手游。这种游戏本身的复杂度集中在客户端。后端基本上只负责数据（比如积分、生命值、装备）的更新和查询，所以，后端逻辑相对于客户端来说，要简单很多。

考虑到你可能对游戏开发不熟悉，我这里稍微交代一些背景知识。

为了提高性能，我们会把游戏中玩家的信息保存在内存中。在游戏进行的过程中，只更新内存中的数据，游戏结束之后，再将内存中的数据存档，也就是持久化到数据库中。为了降低实现的难度，一般来说，同一个游戏场景里的玩家，会被分配到同一台服务上。这样，一个玩家拉取同一个游戏场景中的其他玩家的信息，就不需要跨服务器去查找了，实现起来就简单了很多。

一般来说，游戏客户端和服务器之间的数据交互是比较频繁的，所以，为了节省网络连接建立的开销，客户端和服务器之间一般采用长连接的方式来通信。通信的格式有多种，比如 Protocol Buffer、JSON、XML，甚至可以自定义格式。不管是什么格式，客户端发送给服务器的请求，一般都包括两部分内容：指令和数据。其中，指令我们也可以叫作事件，数据是执行这个指令所需的数据。

服务器在接收到客户端的请求之后，会解析出指令和数据，并且根据指令的不同，执行不同的处理逻辑。对于这样的一个业务场景，一般有两种架构实现思路。

常用的一种实现思路是利用多线程。一个线程接收请求，接收到请求之后，启动一个新的线程来处理请求。具体点讲，一般是通过一个主线程来接收客户端发来的请求。每当接收到一个请求之后，就从一个专门用来处理请求的线程池中，捞出一个空闲线程来处理。

另一种实现思路是在一个线程内轮询接收请求和处理请求。这种处理方式不太常见。尽管它无法利用多线程多核处理的优势，但是对于 IO 密集型的业务来说，它避免了多线程不停切换对性能的损耗，并且克服了多线程编程 Bug 比较难调试的缺点，也算是手游后端服务器开发中比较常见的架构模式了。

我们接下来就重点讲一下第二种实现方式。

整个手游后端服务器轮询获取客户端发来的请求，获取到请求之后，借助命令模式，把请求包含的数据和处理逻辑封装为命令对象，并存储在内存队列中。然后，再从队列中取出一定数量的命令来执行。执行完成之后，再重新开始新的一轮轮询。具体的示例代码如下所示，你可以结合着一块看下。

复制代码

```
public interface Command {
  void execute();
}

public class GotDiamondCommand implements Command {
  // 省略成员变量

  public GotDiamondCommand(/*数据*/) {
    //...
  }

  @Override
  public void execute() {
    // 执行相应的逻辑
  }
}
//GotStartCommand/HitObstacleCommand/ArchiveCommand类省略

public class GameApplication {
  private static final int MAX_HANDLED_REQ_COUNT_PER_LOOP = 100;
  private Queue<Command> queue = new LinkedList<>();

  public void mainloop() {
    while (true) {
      List<Request> requests = new ArrayList<>();

      //省略从epoll或者select中获取数据，并封装成Request的逻辑，
      //注意设置超时时间，如果很长时间没有接收到请求，就继续下面的逻辑处理。

      for (Request request : requests) {
        Event event = request.getEvent();
        Command command = null;
        if (event.equals(Event.GOT_DIAMOND)) {
          command = new GotDiamondCommand(/*数据*/);
        } else if (event.equals(Event.GOT_STAR)) {
          command = new GotStartCommand(/*数据*/);
        } else if (event.equals(Event.HIT_OBSTACLE)) {
          command = new HitObstacleCommand(/*数据*/);
        } else if (event.equals(Event.ARCHIVE)) {
          command = new ArchiveCommand(/*数据*/);
        } // ...一堆else if...

        queue.add(command);
      }

      int handledCount = 0;
      while (handledCount < MAX_HANDLED_REQ_COUNT_PER_LOOP) {
        if (queue.isEmpty()) {
          break;
        }
        Command command = queue.poll();
        command.execute();
      }
    }
  }
}
```

命令模式 VS 策略模式

看了刚才的讲解，你可能会觉得，命令模式跟策略模式、工厂模式非常相似啊，那它们的区别在哪里呢？不仅如此，在留言区中我还看到有不止一个同学反映，感觉学过的很多模式都很相似。不知道你有没有类似的感觉呢？

实际上，这个问题我之前简单提到过，可能没有作为重点来说，有些同学印象不是很深刻，这里我就再跟你讲一讲。

实际上，每个设计模式都应该由两部分组成：第一部分是应用场景，即这个模式可以解决哪类问题；第二部分是解决方案，即这个模式的设计思路和具体的代码实现。不过，代码实现并不是模式必须包含的。如果你单纯地只关注解决方案这一部分，甚至只关注代码实现，就会产生大部分模式看起来都很相似的错觉。

实际上，设计模式之间的主要区别还是在于设计意图，也就是应用场景。单纯地看设计思路或者代码实现，有些模式确实很相似，比如策略模式和工厂模式。

之前讲策略模式的时候，我们有讲到，策略模式包含策略的定义、创建和使用三部分，从代码结构上来，它非常像工厂模式。它们的区别在于，策略模式侧重“策略”或“算法”这个特定的应用场景，用来解决根据运行时状态从一组策略中选择不同策略的问题，而工厂模式侧重封装对象的创建过程，这里的对象没有任何业务场景的限定，可以是策略，但也可以是其他东西。从设计意图上来，这两个模式完全是两回事儿。

有了刚刚的铺垫，接下来，我们再来看命令模式跟策略模式的区别。你可能会觉得，命令的执行逻辑也可以看作策略，那它是不是就是策略模式了呢？实际上，这两者有一点细微的区别。

在策略模式中，不同的策略具有相同的目的、不同的实现、互相之间可以替换。比如，BubbleSort、SelectionSort 都是为了实现排序的，只不过一个是用冒泡排序算法来实现的，另一个是用选择排序算法来实现的。而在命令模式中，不同的命令具有不同的目的，对应不同的处理逻辑，并且互相之间不可替换。

重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

命令模式在平时工作中并不常用，你稍微了解一下就可以。今天，我重点讲解了它的设计意图，也就是能解决什么问题。

落实到编码实现，命令模式用到最核心的实现手段，就是将函数封装成对象。我们知道，在大部分编程语言中，函数是没法作为参数传递给其他函数的，也没法赋值给变量。借助命令模式，我们将函数封装成对象，这样就可以实现把函数像对象一样使用。

命令模式的主要作用和应用场景，是用来控制命令的执行，比如，异步、延迟、排队执行命令、撤销重做命令、存储命令、给命令记录日志等等，这才是命令模式能发挥独一无二作用的地方。

课堂讨论

从我们已经学过的这些设计模式中，找两个代码实现或者设计思路很相似的模式，说一说它们的不同点。

欢迎留言和我分享你的想法。如果有收获，也欢迎你把这篇文章分享给你的朋友。