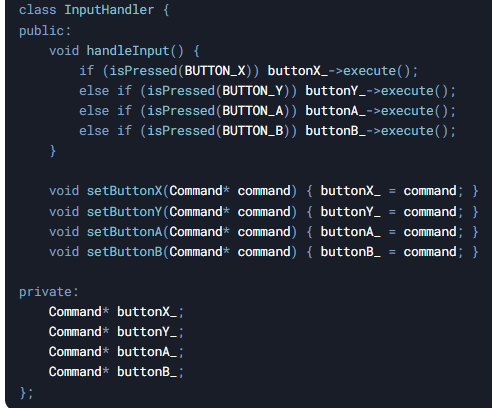
**命令模式的核心思想**

* **定义**：  
  命令模式将一个请求封装为一个对象，从而使你可以用不同的请求参数化客户端，支持请求的队列、日志记录以及撤销操作。
* **关键点**：
  + **封装请求**：将请求的具体实现细节隐藏在命令对象中。
  + **参数化客户端**：客户端可以通过命令对象来执行不同的请求，而不需要知道请求的具体细节。
  + **支持扩展**：命令模式可以轻松支持队列、日志记录和撤销操作

**2. 命令模式的应用场景**

* **游戏开发**：  
  在游戏中，命令模式常用于实现输入处理、任务队列、撤销/重做系统等。例如，玩家的每个操作（如移动、攻击）可以封装为一个命令对象。

命令是一个具体化的方法调用。

在每个游戏中的某个地方都有一段代码可以读取原始的用户输入-按钮按下，键盘事件，鼠标点击，等等。它接受每个输入并将其转换为游戏中有意义的动作：

文本, 信件

描述已自动生成

文本

描述已自动生成

图片包含 图示

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

文本, 信件

描述已自动生成

图示

中度可信度描述已自动生成

# Flyweight

图示

描述已自动生成

文本

描述已自动生成

图形用户界面

描述已自动生成

观测者模式

文本

描述已自动生成

**4. 总结**

* **观察者模式** 的性能开销主要来自于动态分发和间接调用，但在非热点代码路径中通常可以忽略。
* 这种模式适合用于解耦系统、提高代码灵活性和可扩展性。
* 如果性能成为问题，可以通过减少观察者数量、使用事件队列或优化数据局部性来改善。

手机屏幕截图

描述已自动生成

文本, 信件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

在观察者模式中，主题和观察者的生命周期管理是一个常见问题：

* **删除观察者**：如果观察者被删除，但主题仍然持有指向它的指针，会导致 **悬空指针（Dangling Pointer）**。
* **删除主题**：如果主题被删除，但观察者仍然期望接收通知，会导致 **无效通知**。

**方案 1：观察者自行注销**

* **核心思想**：观察者在析构时，主动从所有主题中注销自己。
* **实现**：
  + 在观察者的析构函数中调用 removeObserver。
  + 观察者需要知道它订阅了哪些主题。

**方案 2：主题发送“死亡通知”**

* **核心思想**：主题在析构前，发送一个“死亡通知”，告知观察者它将不再发送通知。
* **实现**：
  + 在主题的析构函数中调用 notify，发送一个特殊事件。
  + 观察者收到“死亡通知”后，可以执行清理操作。

**方案 3：使用智能指针**

* **核心思想**：使用 std::shared\_ptr 和 std::weak\_ptr 管理主题和观察者的生命周期。
* **实现**：
  + 主题和观察者互相持有 std::weak\_ptr，避免循环引用。
  + 当主题或观察者被销毁时，智能指针会自动清理。

**1. 闭包的核心特性**

* **函数**：闭包是一个函数。
* **引用环境**：闭包捕获并保存了函数定义时的上下文（即作用域中的变量）。
* **持久性**：闭包可以在其定义的作用域之外执行，但仍然可以访问定义时的变量

1. 通知您某个状态已更改。
2. 强制性地修改UI的某些块以反映新状态。
3. 图形用户界面, 文本, 应用程序, 电子邮件

   描述已自动生成
4. 文本, 信件

   描述已自动生成
5. 图形用户界面, 文本, 应用程序

   描述已自动生成

原型设计模式

文本

AI 生成的内容可能不正确。

先创建一个对象的原型，再去为其添加修饰。

图片包含 文本

AI 生成的内容可能不正确。

通用设计模式”。

文本

AI 生成的内容可能不正确。

单例模式。

文本

AI 生成的内容可能不正确。

毕竟，面向对象编程就是要让对象自己照顾自己。

图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。

状态设计模式