时间和顺序通常是游戏架构的核心部分。事情必须以正确的顺序和正确的时间发生。

开发周期被高度压缩，许多程序员需要能够快速构建和扩展一组丰富的不同行为，而不会踩到彼此的脚趾或在代码库中留下足迹。

在定义了所有这些行为之后，它开始交互。怪物咬英雄，药水混合在一起，炸弹爆炸的敌人和朋友一样。这些交互必须在代码库没有变成一个交织在一起的毛球的情况下发生。

最后，性能在游戏中至关重要。游戏开发者一直在竞争，看谁能从他们的平台中榨取最多。剃掉周期的技巧可能意味着A级游戏和数百万的销售或下降帧和愤怒的评论家之间的差异。

Intent部分提供了一个模式的快照描述，描述了它打算解决的问题。这是第一步，这样你就可以快速地通过这本书找到一个模式，这将有助于你目前的斗争。

动机部分描述了一个我们将应用该模式的示例问题。与具体的算法不同，模式通常是无形的，除非应用于某些特定的问题。没有例子就教模式就像教烘焙却不提面团一样。本部分提供了后面部分将烘烤的面团。

Pattern部分从前面的例子中提炼出模式的精髓。如果您想要对模式进行枯燥的教科书描述，这就是它。如果您已经熟悉模式并希望确保不会忘记某个成分，这也是一个很好的复习资料。

到目前为止，这种模式只通过一个例子来解释。但是你怎么知道这个模式是否对你的问题有好处呢？什么时候使用它一节提供了一些关于什么时候模式有用以及什么时候最好避免使用它的指导。记住部分指出了使用该模式的后果和风险。

如果像我一样，你需要具体的例子来真正得到一些东西，那么示例代码就是你的部分。它一步一步地介绍了模式的完整实现，因此您可以确切地看到它是如何工作的。

模式不同于单一算法，因为它们是开放式的。每次使用一个模式时，您可能会以不同的方式实现它。下一节，设计决策，将探索这一领域，并向您展示在应用模式时需要考虑的不同选项。

为了总结它，有一个简短的“请参阅”部分，展示了此模式如何与其他模式相关，并为您提供了使用它的真实开源代码。

第一个关键点是架构是关于变化的。一定有人在修改代码库。

在您可以更改代码以添加新功能、修复bug或出于任何原因启动编辑器之前，您必须了解现有代码在做什么。当然，你不必知道整个程序，但你需要将所有相关的部分加载到你的灵长类大脑中。

文本, 信件

描述已自动生成

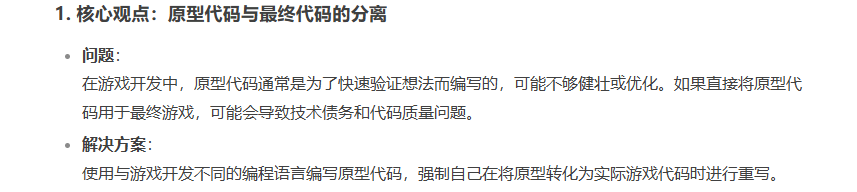
1. **GET PRESENT**：  
   这可能是指“获取当前状态”或“理解当前问题”。在编程中，第一步通常是理解你正在处理的问题或代码的当前状态。
2. **CLEARUP**：  
   这可能是指“清理”或“整理”。在理解问题后，通常需要清理代码或整理思路，以便更好地解决问题。
3. **LEARN COPE**：  
   这可能是指“学习应对”或“适应”。在编程中，你经常需要学习新的技术或适应现有的代码库，以便找到解决方案。
4. **CODE SOLUTIONS**：  
   这是实际的编码阶段，你根据前面的理解和整理，编写代码来解决问题。

对我来说，这是软件架构的一个关键目标：在你取得进步之前，最大限度地减少你需要的知识量。

1. **遵循 YAGNI 原则**：  
   不要为未来可能需要的功能提前编写代码，只有在确实需要时才添加相应的功能。

你有时会听到另一种对软件架构和抽象的批评，尤其是在游戏开发中：它会损害游戏的性能。许多使代码更加灵活的模式依赖于虚拟分派、接口、指针、消息和其他机制，这些机制都至少有一些运行时成本。

* **性能与灵活性的权衡**：  
  在游戏开发中，性能和灵活性往往是一对矛盾。开发者需要根据具体需求权衡这两者。
* **模板元编程的优势**：  
  C++ 模板元编程是一个强大的工具，可以在不牺牲性能的情况下提供抽象能力。然而，它也有一定的复杂性，需要开发者具备较高的技能水平。
* **避免过度抽象**：  
  虽然抽象可以提高代码的可维护性和可扩展性，但在性能敏感的领域（如游戏开发），过度抽象可能会带来不必要的开销。开发者需要谨慎选择合适的设计模式和技术。



抽象和解耦使开发程序更快、更容易，但不要浪费时间做这些事情，除非您确信相关代码需要这种灵活性。

在整个开发周期中考虑并设计性能，但尽可能推迟将假设锁定到代码中的低级别，实质性优化。

快速探索游戏的设计空间，但不要走得太快，以免留下一片混乱。你必须忍受它，毕竟。

如果你打算抛弃代码，不要浪费时间让它变得漂亮。摇滚明星把酒店房间弄得一团糟，因为他们知道第二天就要退房了。

但是，最重要的是，如果你想做一些有趣的事情，那就好好享受吧。