**Command（命令）**

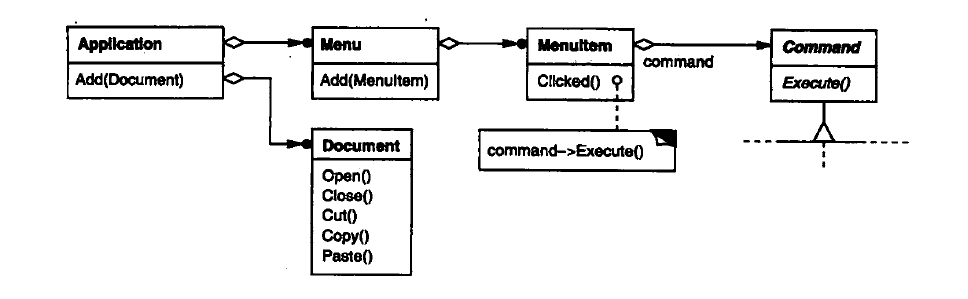
意图

将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可撤销操作。

动机

有时必须向某对象提交请求，但并不知道关于请求的操作或请求的接受者的任何信息。例如，用户界面工具箱包括按钮和菜单这样的对象，它们执行请求响应用户输入。但工具箱不能显示的在按钮或菜单中实现该请求，因为只有使用工具箱的应用知道该由哪些对象做哪个操作。而工具箱的设计者无法知道请求的接受者或执行的操作。

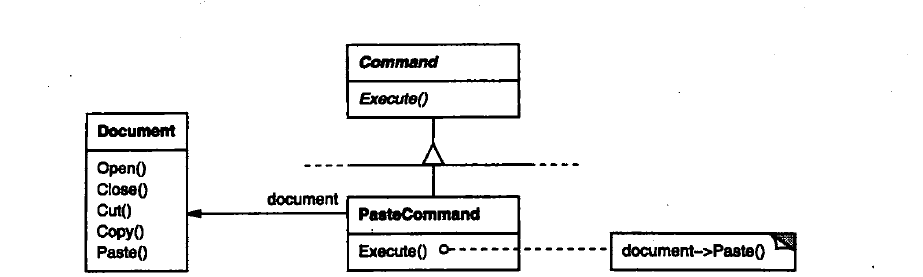
命令模式通过将请求本身变成一个对象来使工具箱对象可向未指定的应用对象提出请求。这个对象可被存储并像其他的对象一样被传递。这一模式的关键是一个抽象的Command类，它定义了一个执行操作的接口。其最简单的形式是一个抽象的Execute操作。具体的Command子类将接收者作为其一个实例变量，并实现Execute操作，指定接收者采取的动作，而接收者有执行该请求所需的具体信息。



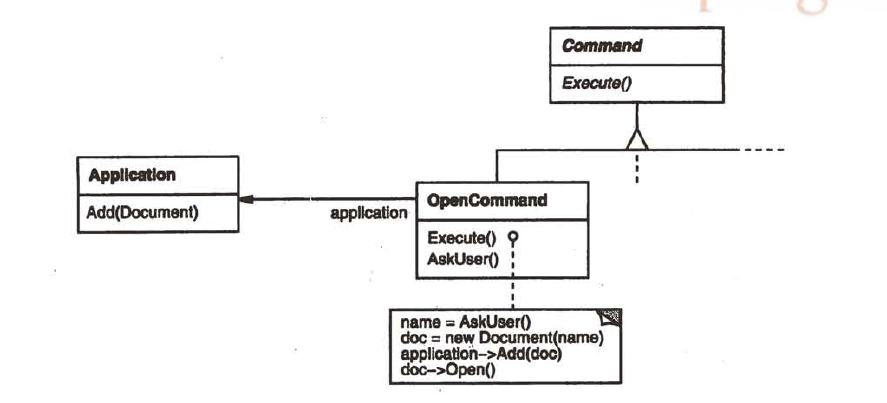
用Command对象可很容易的实现菜单，每一菜单中的选项都是一个菜单项类的实例。一个Application类创建这些菜单和它们的菜单项以及其余的用户界面。该Application类还跟踪用户已打开的Document对象。

该应用为每一个菜单项配置一个具体的Command子类的实例。当用户选择一个菜单项时，该MenuItem对象调用它的Command对象的Execute方法，而Execute执行相应操作。MenuItem对象并不知道它们使用的是Command的哪一个子类。Command子类里存放着请求的接受者，而Execute操作将调用该接受者的一个或多个操作。

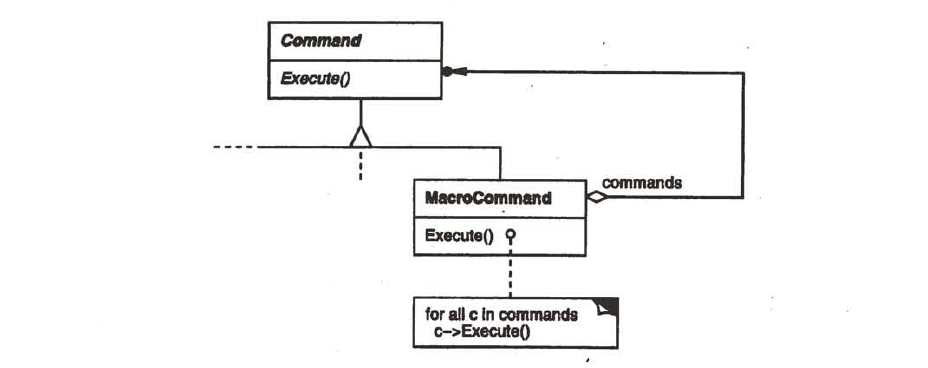
例如，PasteCommand支持从剪贴板向一个文档（Document）粘贴正文。PasteCommand的接受者是一个文档对象，该对象时实例化时提供的。Execute操作将调用该Document的Paste操作。



而OpenCommand的Execute操作却有所不同：它提示用户输入一个文档名，创建一个相应的文档对象，将其作为接收者的应用对象中，并打开该文档。



有时一个MenuItem需要执行一系列命令。例如，使一个页面按正常大小居中的MenuItem可由一个CenterDocumentCommand对象和一个NormalSizeCommand对象构建。因为这种需将多条命令串接起来的情况很常见，我们定义一个MacroCommand类来让一个MenuItem执行任意数目的命令。MacroCommand是一个具体的Command子类，它执行一个命令序列。MacroCommand没有明确的接收者，而序列中的命令各自定义其接收者。



请注意这些例子中Command模式是怎样解耦调用操作的对象和具有执行该操作所需信息的那个对象的。这使我们在设计用户界面时拥有很大灵活性。一个应用如果想让一个菜单与一个按钮代表同一项功能，只需让它们共享相应具体Command子类的同一个实例即可。我们还可以动态地替换Command对象，这可用于实现上下文有关的菜单。我们也可通过将几个命令组成更大的命令的形式来支持命令脚本。所有这些之所以成为可能乃是因为提交一个请求的对象仅需知道如何提交它，而不需知道该请求将会被如何执行。

结构图

