**Strategy（策略）**

意图

定义一系列的算法，把它们一个个封装起来，并且使它们可以相互替换。本模式使得算法可独立于使用它的客户而变化。

动机

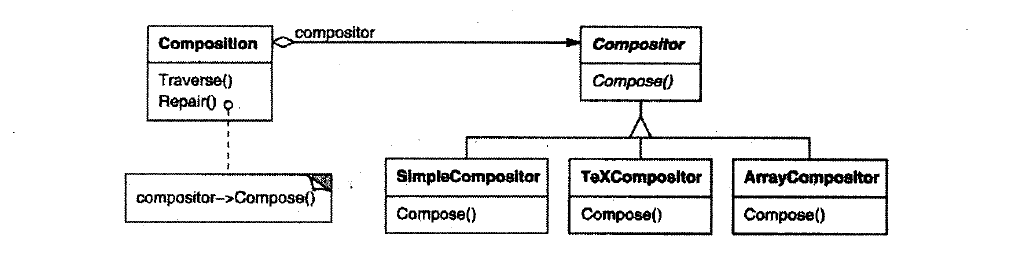
有许多算法可对一个正文流进行分行。将这些算法硬编进使用它们的类中是不可取的，其原因如下：

需要换行功能的客户程序如果直接包含换行算法代码的话将会变得复杂，这使得客户程序庞大并且难以维护，尤其当其需要支持多种换行算法时问题更严重。

不同的时候需要不同的算法，我们不想支持我们并不使用的换行算法。

当换行功能是客户程序的一个难以分割的成分时，增加新的换行算法或改变现有算法将十分困难。

我们可以定义一些类来封装不同的换行算法，从而避免这些问题。一个以这种方法封装的算法称为一个策略。如下图所示：



假设一个Composition类负责维护和更新一个正文浏览程序中显示的正文换行。换行策略不是Composition类实现的，而是由抽象的Compositor类的子类各自独立地实现的。Compositor各个子类实现不同的换行策略：

SimpleCompositor实现一个简单的策略，它一次决定一个换行位置。

TeXCompositor实现查找换行位置的TEX算法。这个策略尽量全局地优化换行，也就是，一次处理一段文字的换行。

ArrayCompositor实现一个策略，该策略使得每一个行都含有一个固定数目的项。例如，用于一系列的图表进行分行。

Composition维护对Compositor对象的一个引用。一旦Composition重新格式化它的正文，它就将这个职责转发给它的Compositor对象。Composition的客户指定应该使用哪一种Compositor的方式是直接将它想要的Compositor装入Composition中。

结构图

