**FlyWeight（享元）**

意图

运用共享技术有效地支持大量细粒度的对象。

动机

有些应用程序得益于在整个设计过程中采用对象技术，但简单化的实现代价极大。

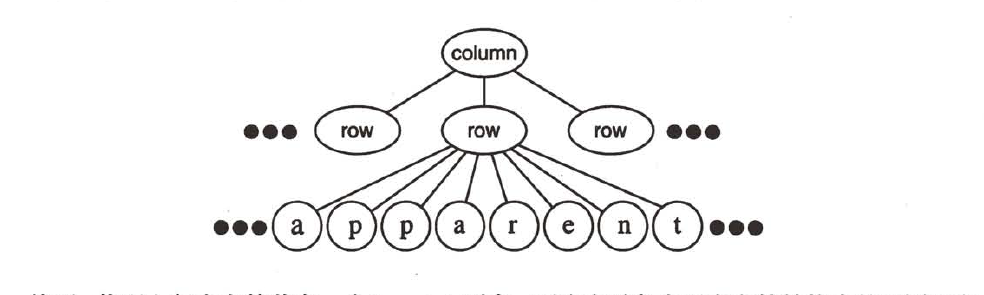
例如，大多数文档编辑器的实现都有文本格式化和编辑功能，这些功能在一定程度上是模块化的。面向对象的文档编辑器通常使用对象来表示嵌入的成分，例如表格和图形。尽管用对象来表示文档中的每个字符会极大地提高应用程序的灵活性，但是这些编辑器通常并不这样做。字符和嵌入成分可以在绘制和格式化时统一处理，从而在不影响其他功能的情况下能对应用程序进行扩展，支持新的字符集。应用程序的对象结构可以模拟文档的物理结构。

但这种设计的缺点在于代价太大。即使是一个中等大小的文档也可能要求成百上千的字符对象，这会消耗大量内存，产生难以接受的运行开销。所以通常并不是对每个字符都用一个对象来表示。FlyWeight模式描述了如何共享对象，使得可以细粒度地使用它们而无需高昂的代价。

FlyWeight是一个共享对象，它可以同时在多个场景中使用，并且在每个场景中FlyWeight都可以作为一个独立的对象—这一点与非共享对象的实例没有区别。FlyWeight不能对它所运行的场景做出任何假设，这里的关键概念是内部状态和外部状态之间的区别。内部状态存储于FlyWeight中，它包含了独立于FlyWeight场景的信息，这些信息是的FlyWeight可以被共享。而外部状态取决于FlyWeight场景，并根据场景而变化，因此不可共享。用户对象负责在必要的时候将外部状态传递给FlyWeight。

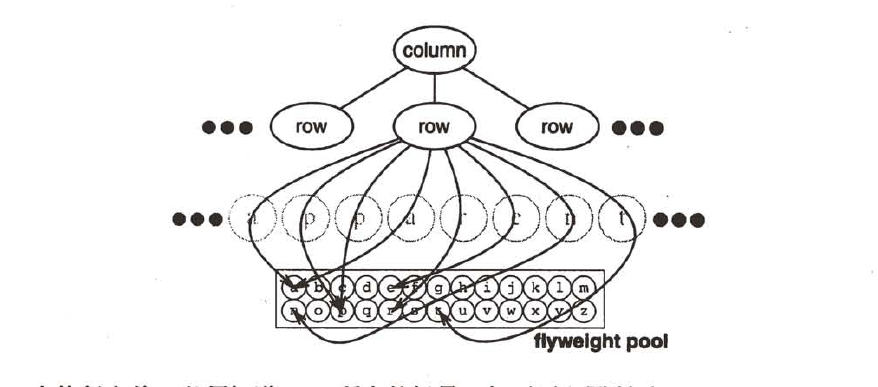
FlyWeight模式对那些通常因为数量太大而难以用对象来表示的概念或实体进行建模。例如，文档编辑器可以为字母表中的每一个字母创建一个FlyWeight。每个FlyWeight存储一个字符代码，但它在文档中的位置和排版风格可以在字符出现时由正文排版算法和使用的格式化命令决定。字符代码是内部状态，而其他的信息则是外部状态。

逻辑上，文档中的给定字符每次出现都有一个对象与其对应，如下图所示：



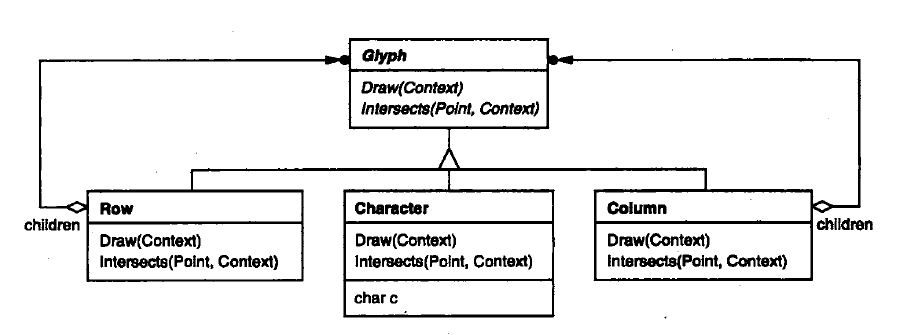
然而，物理上每个字符共享一个FlyWeight对象，而这个对象出现在文档结构中的不同地方。一个特定字符对象的每次出现都指向同一个实例，这个实例位于FlyWeight对象的共享池中。

这些对象的类结构如下图所示。Glyph是图形对象的抽象类，其中有些对象可能是FlyWeight。基于外部状态的那些操作将外部状态作为参量传递给它们。例如Draw和Intersects在执行之前，必须知道Glyph所在的场景，如图所示：



表示字母“a”的FlyWeight只存储相应的字符代码；它不需要存储字符的位置或字体。用户提供与场景相关的信息，根据此信息FlyWeight绘出它自己。例如，Row Glyph知道它的子女应该在哪儿绘制自己才能保证它们是横向排列的。因此Row Glyph可以在绘制请求中向每一个子女传递它的位置。

由于不同的字符对象数远小于文档中的字符数，因此，对象总数远小于一个初次执行的程序所使用的对象数目。对于一个所有字符都使用同样的字体和颜色的文档而言，不管这个文档有多长，需要分配100个左右的字符对象。由于大多数文档使用的字体颜色组合不超过10种，实际应用中这一数目不会明显增加，因此，对单个字符进行对象抽象是具有实际意义的。



结构图

