八哥车间BBS论坛系统包括**用户模块，版块与公共信息模块，帖子模块和管理员模块**四个主要的功能模块。

**模块化**

耦合度：

我们将整个程序的开发按照功能分为了四个模块分别是用户模块，版块与公共信息模块，帖子模块和管理员模块。

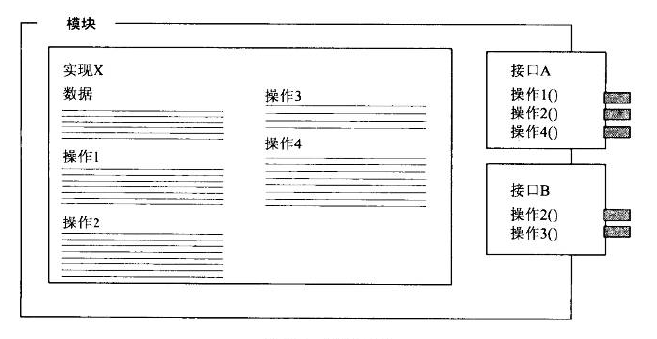
各个模块之间存在密切的联系，例如管理员模块要对其他模块进行管理，需要对其他模块进行内容的修改，像修改公告、删除不合规的帖子等，这说明各模块之间存在一定的内容耦合。

内聚度：

内聚度是指模块的内部元素（比如，数据、功能、内部模块〉的“粘合”程度。一个模块的内聚度越高，模块内部的各部分之间的相互联系就越紧密，与总体目标就越相关。一个模块如果有多个总体目标，它的元素就会有多种变化方式或变化值。例如，一个模块同时包含了数据和例程,并用以来显示那些数据，这个模块可能会频繁更改且以不同的方式变更，因为每次使用这些数据时都需要使用改变这些值的新功能和显示这些值的新方法。我们的目的是尽可能地使模块高内聚，这样各个模块才能易于理解，减少更改。

在我们的模块设计中，不同模块直接联系紧密，如帖子的id是标识帖子唯一的标志，在用户模块中记录自己发的帖子也是使用这一标识。

**接口**



软件单元接口的规格说明（specification)描述了软件单元外部可见的性质。正如需求规格说明从系统边界的角度描述系统行为一样，接口的规格说明的描述以单元的边界为依据对软件单元做出描述;该单元的6访问函数、参数、返回值和异常。一个接口的规格说明需向其他系统开发人员传达正确应用该软件单元的所有信息。

这里的版块与公共信息模块和帖子模块要提供增删改查的调用，实现相关功能。

**信息隐藏**

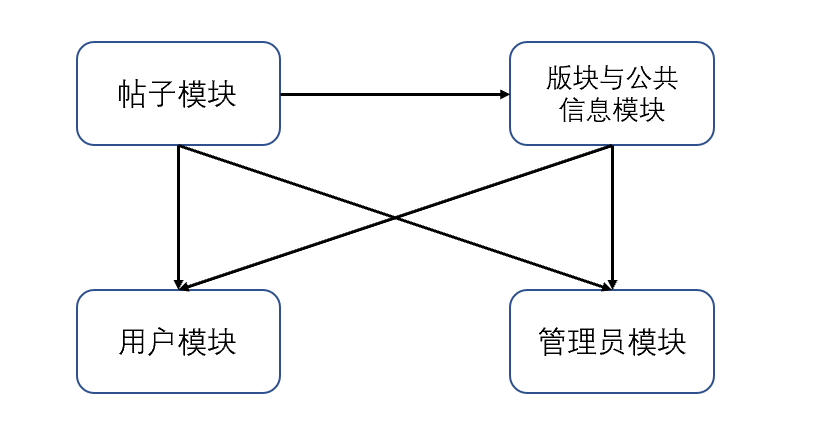
信息隐藏（information hiding)的目标是使得软件系统更加易于维护。它以系统分解为特征:每个软件单元都封装了一个将来可以改变的独立的设计决策，然后我们根据外部可见的性质，在接口和接口规格说明的帮助下描述各个软件单元。因此，这个原则的名称本身也反映了它的结果:单元的设计决策被隐藏了。

“设计决策”这种说法其实是很笼统的，它可以有很多指代，包括数据形式或数据操作、硬件设备或者其他需要和软件交互的构件、构件之间消息传递的协议，或者算法的选择。因为设计过程牵涉到软件很多方面的决策，所以最终的软件单元封装了各种类型的信息。面向对象中封装函数、数据类型或过程就很好地体现了信息隐藏。而我们这次开发就是用JavaWeb开发充分——一种纯面向对象语言，还有就是不同模块之间通过接口，而不暴露自身的数据进行通信也体现了数据隐匿。

**增量式开发**

假定一个软件设计是由软件单元和它们的接口所组成的，我们可以使用单元之间的依赖关系来设计出一个增量式设计开发进度表。我们指定单元间的使用关系它为各个软件单元和它依赖的单元之间建立关联。例如，单元A依赖单元B构造一个数据结构，并存储在一个独立的单元C中，随后A再访问C。总的说来，如果软件单元A如它接口中描述的那样“需要一个正确的B"，才能完成A的任务，那么我们说软件单元A“使用”软件单元B。

将系统的这种利用关系描述成使用图。八哥车间BBS论坛系统的使用图如下：



扇入指代使用某个软件单元的软件单元数量，扇出指代某个软件单元使用其他软件单元的数量。

|  |  |  |
| --- | --- | --- |
|  | 扇入 | 扇出 |
| 帖子模块 | 0 | 3 |
| 版块与公共信息模块 | 1 | 2 |
| 用户模块 | 2 | 0 |
| 管理员模块 | 2 | 0 |

一般来说，我们要控制有高扇出的软件单元的数量。高扇出往往表明该软件单元所做的太多，也许可以分解成更小更简单的单元。因另一方面，如果有多个单元执行相似的功能，譬如查找字符串，那么我们更倾向于把它们合并成单个的、有着更一般性的目标的单元，它可以代替原来单元中的任何一个。这样最终的单元就有了高扇入。我们设计一个系统的最终目的之一是创建一个有着高扇入、低扇出的软件单元。

设计良好的使用图应具有树型结构或者是树型结构的森林。在这样的结构中，每棵子树都是系统的一部分，所以我们可以一次一个软件单元地增量开发我们的系统，每个完成的单元都是我们系统的部分实现。在开发过程中，每一次的增量都会越来越易于测试和修改，因为错误只可能出现在新代码中，而不是在已经过测试和验证的被调用单元中。此外，我们总有一个可运行的系统版本用来展示给客户。更多地，系统频繁且可见的进展也鼓舞了开发人员的士气，和其他方法相比，增量式开发有着不可多得的优势，因为前者只有当每个单元都能工作时系统才能工作。

**抽象**

抽象是一种忽略一些细节来关注其他细节的模型或表示。而在定义中，关于模型中的哪部分细节被忽略是很模糊的，因为不同的目标会对应不同的抽象，会忽略不同的细节。

八哥车间BBS论坛系统被分解为各个子系统，每个子系统再被分解成更小的了系统，一直分解下去。其中分解的顶层给我们提供了问题系统层次上的纵览，同时对我们隐藏了那些可能会影响我们注意力的细节，有助于我们集中关注我们想要研究和理解的设计功能和特性。当我们观察低一层次的抽象时，我们会发现更多关于各软件单元的细节，它们牵涉到它的主要元素以及这些元素间的关系。各个抽象层次以这种方式隐藏了它的元素如何进一步分解的方法，而每个元素在接口规格说明中将被一一描述，这是另一种关注元素外部行为和避免元素内部设计细节被引用的抽象类型。

一个系统可能不仅仅只有一个分解方法，我们会创建若于种不同的分解来展示不同的结构，譬如，一种视图可能展示了不同运行进程以及它们内部的联系，另一种视图则展示了分解成代码单元的系统。每个视图都是一种抽象，它强调了系统结构设计的某个方面（如，运行进程）而忽略了其他结构信息（如，代码单元）和非结构细节。

对于一个特定的模型，一个好的抽象的关键是决定哪些细节是不相关的，进而可以被忽略的。

抽象的性质取决于开始时我们建立这个模型的初衷：我们想交互哪些信息，或者我们想展示哪个分析过程。

**通用性**

通用性是这样一种设计原则：在开发软件单元时，使它尽可能地能够成为通用的软件，来加强它在将来某个系统中能够被使用的可能性。我们通过增加软件单元使用的上下文环境的数量来开发更加通用的软件单元，下面是几条实现规则：

1.将特定的上下文环境信息参数化：通过把软件单元所操作的数据参数化，我们可以开发出更加通用的软件。

2.去除前置条件：去除前置条件，使软件在那些我们之前假设不可能发生的条件下工作。

3.简化后置条件：把一个复杂的软件单元分解成若干个具有不同后置条件的单元，再将它们集中起来解决原来需要解决的问题，或者当只需其中，部分后置条件时单独使用。

尽管我们希望总能够开发出可复用的单元，但有时候其他的设计目标会与该目标产生冲突。软件工程关注的是特定上下文中的软件决策。那就是说，我们要对决策做出相应调整来适应特定用户的需要。系统的需求规格说明列出了特定的设计标准（如，性能、效率），我们可以通过参照这些标准来优化设计和代码。然而，这种客户化定制往往降低了软件的通用性，这反映了我们必须在通用性（因而，还有可复用性）和客户化之间做出权衡，而我们也没有 一般性的法则可以帮助我们平衡这两个相互冲突的目标，我们的选择将取决于环境、设计标准的重要程度，以及一个更通用软件版本的实用效果。

举例：

类的通用性

论坛系统的各个实体, 不可能是相互独立的，比如发帖这个动作，必须要用到用户类和帖子类，在这种情况下，要合理划分方法，尽可能与事件独立是一个能让代码最大限度复用的好办法。具体实例: 用户发帖后，可以给用户增加积分/经验/人气等，该代码逻辑应该写在帖子类内部还是用户类内部？如果写在帖子类内部, 则用户回复帖子时也有相关操作, 则需要重新再写一遍，显然出现了冗余。所以该逻辑应该写在用户类内部。复用原则之一：公用方法与事件分离。

前端代码通用性

HTML的样式如果直接写在html文件中，同样的样式会写很多遍, 显然属于无用功，可以把样式分离到CSS文件中，使用时调用。具体实例：论坛是Web应用，则论坛的整个主题是确定的，因而颜色基调是确定的, 很多界面的风格相似，完全可以使用独立的CSS文件表示网页样式，再结合Servlet动态生成每个页面的特殊格式，可以达到最大程度的样式复用。