第31章　创建一个Web论坛

吸引网站用户的一个好方法是建立Web论坛。它可以用于哲学讨论组和产品技术支持等多种用途。在本章中，我们将使用PHP来实现Web论坛。当然，也可以选择已有的软件包，例如Phorum，来建立自己的论坛。

Web论坛有时候又叫做*讨论公告牌*或*主题讨论组*。论坛的想法就是允许论坛中的用户发表文章或提出问题，而其他人可以阅读并回复这些问题。在论坛中，每个讨论的主题叫做一个*话题*（thread）。

在这个项目中，我们要实现一个名为blah-blah的论坛，该论坛具有下面的功能。用户可以：

- 通过发表文章开始新的讨论话题
- 发表文章来回复已有的文章
- 查看发表的文章
- 查看论坛中的交谈话题
- 查看文章之间的关系，也就是，查看哪篇文章是另外文章的回复

31.1　理解流程

建立论坛事实上是一个相当有趣的流程。我们需要一些方法来在数据库中存储文章的作者、标题、发表日期和内容信息。初看觉得这似乎与Book-O-Rama数据库没有多大不同。

然而，大多数主题讨论软件都以如下方式工作：给用户显示文章的同时，还显示出这些文章之间的关系。也就是说，可以看出哪些文章是回复另外一些文章的（以及它们是哪篇文章的回复），以及哪些文章是新讨论的话题。

我们可以看到在许多地方看到实现这些功能的论坛例子，包括Slashdot:http://slashdot.org。

需要仔细考虑的地方是确定如何显示这些文章之间的关系。对于目前这个系统来说，用户应该能够查看单个的消息，一个显示有相互关系的谈论话题，或者系统中所有话题。

用户还必须能发表新主题的文章或者进行回复。这只是简单的部分。

31.2　解决方案的组成

正如我们前面介绍的，保存及检索作者和消息文本是非常容易的。该应用程序最困难的部分是如何找到一种保存所需信息的数据库结构，以及一种高效率浏览该结构的方法。

如图31-1所示就是一个可能的正在讨论中的文章结构。

在图31-1中，可以看到关于某主题的一个初始发表的文章，该文章具有3篇回复文章。这些回复文章中的某些又有回复文章。这些文章还可能回复等。

仔细查看图31-1，可以得到如下启示，我们应该如何保存并检索文章数据及文章间的连接。图中显示了一种树形结构。如果我们做过很多编程工作，会发现这是一种很常用的数据结构。

在图31-1中，有节点（或说文章）和连接（或文章之间的关系），正如任何树形结构一样。（如果对树形数据结构不熟悉，不用担心——我们会在开始主题讨论之前介绍这些基本知识。）

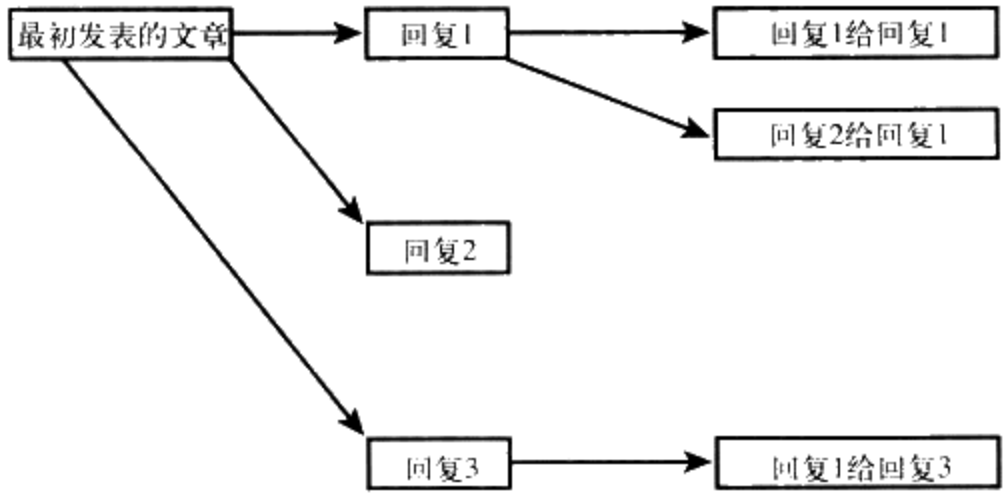

图31-1　主题讨论中的一个文章可能是一个新话题的第一篇文章，也可能是对其他文章的响应

掌握这些知识的诀窍是：

1) 找到一种将这些树形结构映射到物理存储器的方法——在这个例子中，就是映射到MySQL数据库。

2) 找到一种能够按要求重建数据的方法。

对于这个项目，我们将以实现一个能够保存使用当中的文章的MySQL数据库为开始。我们将创建一些简单的界面来实现文章的保存。

当载入待显示的文章列表时，我们会将每篇文章的标题放进一个`tree_node`的PHP类。每一个`tree_node`将包含一个文章标题以及对该文章的回复。

这些回复将保存在一个数组中，而每个回复本身又是一个`tree_node`，它可以包含由该文章的回复组成的一个数组，而这些回复本身同样又是`tree_node`，如此继续下去。直到遇到我们所说的叶子节点（leaf node），叶子节点本身不再包含回复。这样，我们就有了一个如图31-1所示的树形结构。

在这里，我们先介绍一些术语：要回复的消息称作当前节点的父节点。该消息的任何回复可以称作当前节点的子节点。如果把该树形结构想象成家庭树，就很容易记住这一点。

该树形结构的第一篇文章（没有父节点的那篇）有时候叫做根节点。

提示　将第一篇文章称作根节点可能不太直观，因为通常我们都将根节点画在图的顶部，不像实际的树根。

要建立并显示这个树形结构，我们将编写一些递归函数。（在第5章中，我们已经讨论过递归函数。）

我们决定使用一个类来表示这种数据结构，因为对于这个应用程序来说，这样更易于建立一种复杂的、动态可扩展的数据结构。它也意味着，我们可以使用简单而别致的代码来完成相当复杂的工作。

31.3 解决方案概述

为了真正理解在该项目中我们所做的工作，阅读其代码是一个不错的方法。接下来，我们就研究代码。与其他项目比起来，该项目包含冗长程序不多，都是简短的代码，但是代码却更复杂。

在该程序中，实际上仅有3个页面。它有一个主索引页，显示论坛中所有文章并指向这些文章的链接。从这里，可以增加新文章，查看已列出的文章，或通过展开和折叠树枝来改变查看文章的方法。(稍后将详细介绍)。在文章视图页面中，可以回复该文章或查看已存在的回复。在新文章页面中，可以发表一篇新文章、或者一篇对已存在文章的回复、或者一篇无关的新消息。

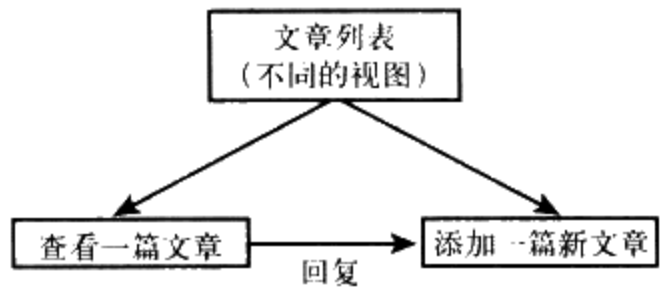

图31-2 在blah-blah论坛系统中的3个主要部分

该系统的流程图如图31-2所示。

该程序使用到的文件汇总如表31-1所示。

表31-1 Web论坛应用程序中用到的文件

文件名称	类 型	描 述
index.php	应用程序	用户进入站点看到的主页。包括该站点中所有文章的可扩展及删除的列表
new_post.php	应用程序	用来发表新文章的表单
store_new_post.php	应用程序	保存已输入到new_post.php表单中的文章
view_post.php	应用程序	显示一篇单独的文章及回复它的文章的列表
treenode_class.php	函数库	包含节点类，我们将要用它来显示文章的继承关系
include_fns.php	函数库	将该程序中用到的所有其他函数库放到一起（其他库类型文件在这里列出）
data_valid_fns.php	函数库	数据检验函数
db_fns.php	函数库	数据库连接函数
discussion_fns.php	函数库	处理存储及重新获得发表的文章的函数
output_fns.php	函数库	输出HTML的函数
create_database.sql	SQL	建立该程序所需要数据库的SQL脚本

下面，我们继续来了解程序的实现。

31.4 数据库的设计

我们需要保存一些关于发表到论坛中的文章的属性，其中包括：文章的作者（称作poster）、文章的标题、发表时间和文章正文。因此我们需要一个文章表，还必须为每一篇文章创建一个唯一的ID，称作postid。

每篇文章需要一些关于它在继承关系中所属位置的信息。我们可以将关于文章的子文章的位置随该文章一起保存起来。然而，每篇文章都会有许多回复，因此这可能会在数据库的结构上导致某些问题。由于每篇文章只能是一篇文章的回复，因此保存父文章的引用可能会更容易

些，父文章即当前文章回复的那篇文章。

这样我们需要为每篇文章保存以下数据：

- postid：每篇文章的唯一的ID
- parent：父文章的postid
- poster：该文章的作者
- title：该文章的标题
- posted：该文章发表的时间和日期
- message：该文章的正文

当然，我们还需要对以上数据进行一些优化。

当需要判断一篇文章是否有回复时，必须运行一个SQL查询来确定是否有任何的文章将该文章作为其父文章。我们需要为每一篇所列出文章生成这些信息。需要运行的查询语句越少，代码运行的速度就越快。我们可以通过增加一个表示是否有任何的回复的字段来去除这种查询的必要。我们把这个字段叫做children并将它设为查询效率较高的Boolean类型——如果节点有“children”，则为1，否则为0。

当然，优化总是要付出代价的。这里，我们选择了存储冗余数据。由于我们以两种方式保存这些数据，必须保证两种表示方法互相匹配。当增加子节点时，必须更新其父节点。如果删除子节点，需要更新父节点确保数据库的一致性。在该项目中，我们并不打算添加删除文章的功能，因此可以避免一半的问题。如果决定扩展该代码，请记住这个问题。

在这个项目中，我们还需要完成一些其他优化。我们将消息体与其他数据分离，并且将这些消息体保存在一个单独的表格中。这样做可以使消息体具有MySQL的text属性。在一个表中使用这种类型的数据将降低该表的查询速度。由于我们要完成许多小查询来建立树形结构，因此这会使该系统的速度下降很多。将消息正文放在单独的表中，我们可以在用户需要查看某个特定消息时，将它从数据库中取出。

MySQL查询固定大小记录的速度比查询可变大小记录的速度要快。如果想使用可变大小的数据，可以通过对用来检索数据库的字段上创建索引来实现。当然对于某些项目来说，将文本字段与其他字段一样放在相同的记录中，并为所有要基于这些字段的检索指定索引可能也会好些。创建索引要花费时间，而且论坛数据可能经常会发生变化，因此我们必须经常重新建立索引。

我们还增加一个area属性以备以后决定在一个程序中实现多个论坛。在这里，我们不实现这个功能，但是该方法可以保留为将来使用。

基于以上所有这些考虑，创建论坛数据库的SQL脚本如程序清单31-1所示。

程序清单31-1　create_database.sql——创建论坛数据库的SQL脚本

```
create database discussion;

use discussion;

create table header
(
  parent int not null,
```

```
  poster char(20) not null,
  title char(20) not null,
  children int default 0 not null,
  area int default 1 not null,
  posted datetime not null,
  postid int unsigned not null auto_increment primary key
);

create table body
(
  postid int unsigned not null primary key,
  message text
);

grant select, insert, update, delete
on discussion.*
to discussion@localhost identified by 'password';
```

可以在MySQL中运行如下所示的脚本来创建这个数据库结构：

```
mysql -u root -p < create_database.sql
```

需要提供root用户密码。此外，还应该更改我们为论坛用户设置的密码。

为了更好地理解该结构如何保持文章及它们相互之间存在的关系，请参阅图31-3。

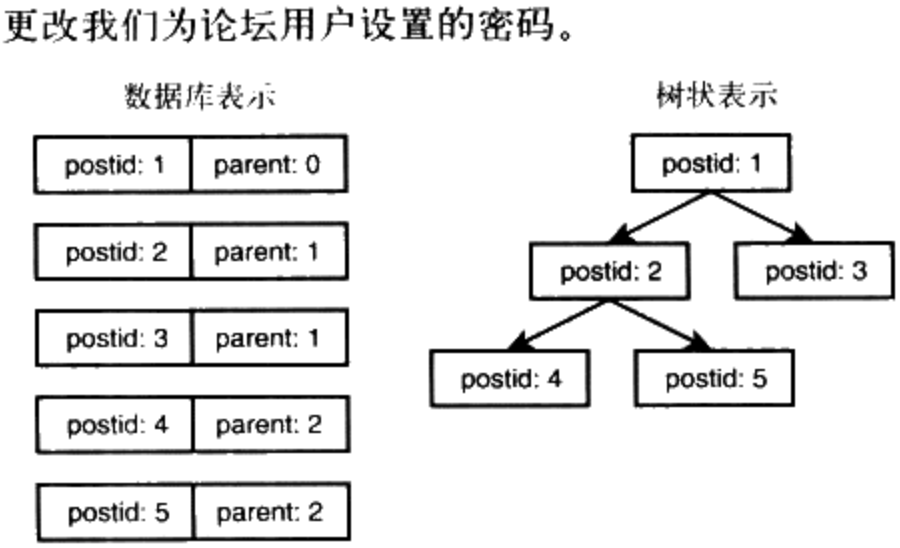

图31-3 以一个平面关系表格方式保存的树形结构数据

可以看到，数据库中每篇文章的父字段保存了树形结构中它上面的文章的postid。父文章是将被回复的文章。

我们还可以看到，根节点postid 1没有父节点，讨论组所有新主题都在这个位置。对于这种类型的文章，我们在数据中将它们的父节点保存为0。

31.5 查看文章的树形结构

接下来，我们需要一种方法将信息从数据库中取出，并将其在树形结构中表示出来。我们在主页面index.php中完成这项工作。为了将这个过程解释得更加清楚，我们通过文章粘贴脚本程序new_post.php和store_new_post.php输入了一些样本帖子。我们将在下一节介绍这两个程序。

这里，我们首先讨论文章列表，因为它是整个站点的主体。然后其他内容就容易理解了。

如图31-4所示的是用户将看到的该站点初始的文章视图。

我们这里看到的都是最初的文章，它们都没有回复文章；它们都是某特定主题的第一篇文章。

在这个例子中，还可以看到有很多选项。有一个菜单栏，我们可以用它来添加一篇新文章以及展开或折叠文章的视图。

为了理解这一点，请看这些发表的文章。其中有一些文章前面有“+”号，这表示这些文章已经被回复过。如果要查看特定文章的回复，可以点击该加号标志。如图31-5所示的就是点击这些符号产生的一个结果。

可以看到，点击加号后会显示出第一篇文章的回复。而加号则变成了减号。如果我们再点击它，在这个话题里的所有文章都将会折叠起来，又返回至初始视图。

可以看到，其中的一篇回复前面也有个加号。这表示这篇回复也有回复。如此重复操作，可以一直点击下去，通过点击相应的加号来查看每篇回复。

在菜单栏中，有两个菜单选项，Expand和Collapse，分别为展开和折叠所有可能的话题。点击Expand按钮的结果如图31-6所示。

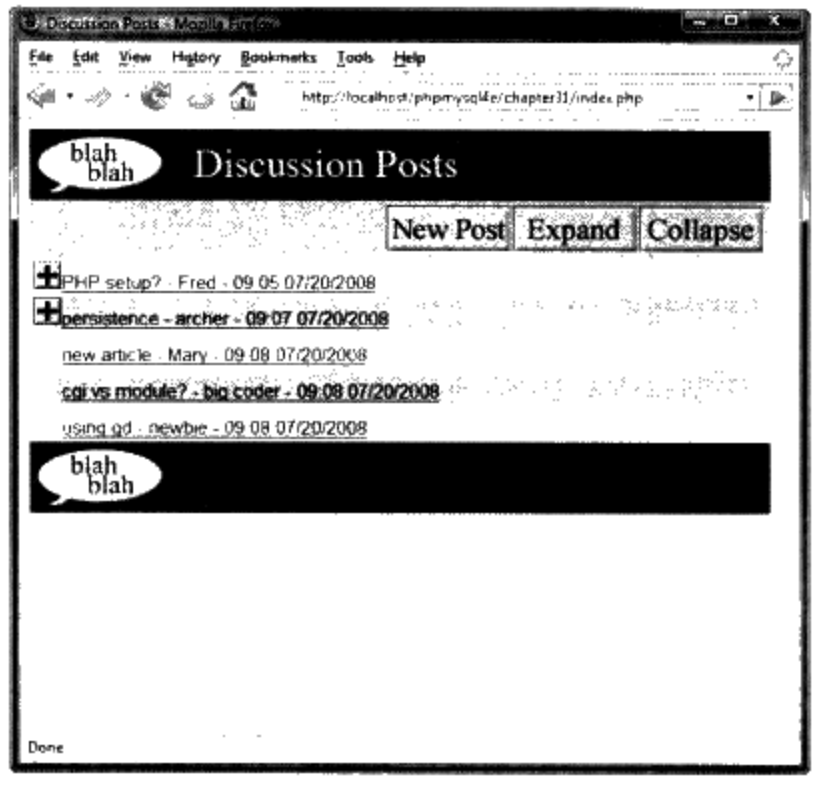

图31-4　文章列表的初始视图显示了在折叠表单中的文章

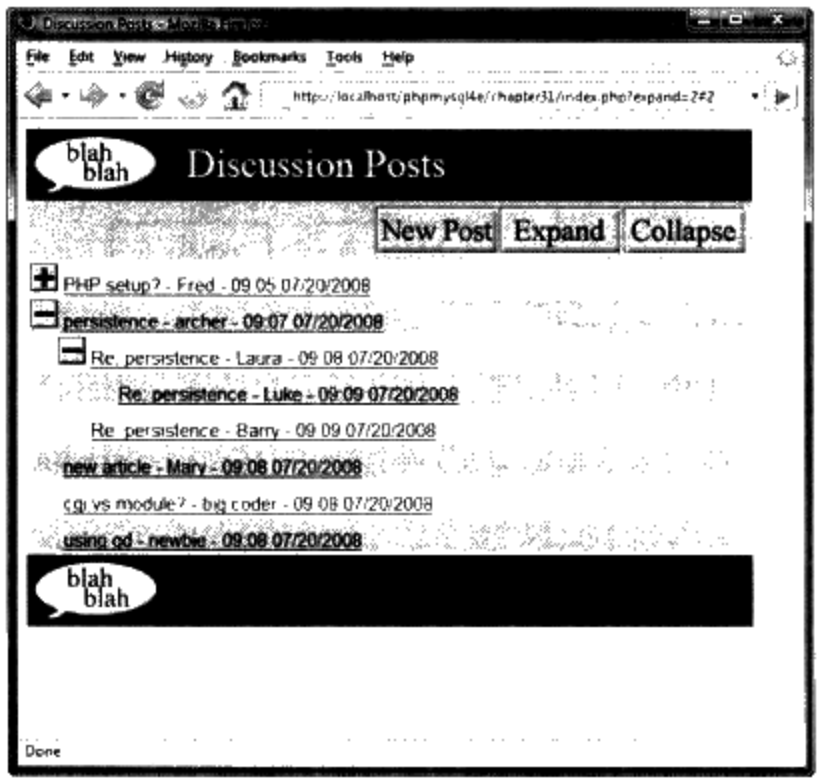

图31-5　关于PHP持续性的讨论话题已被展开

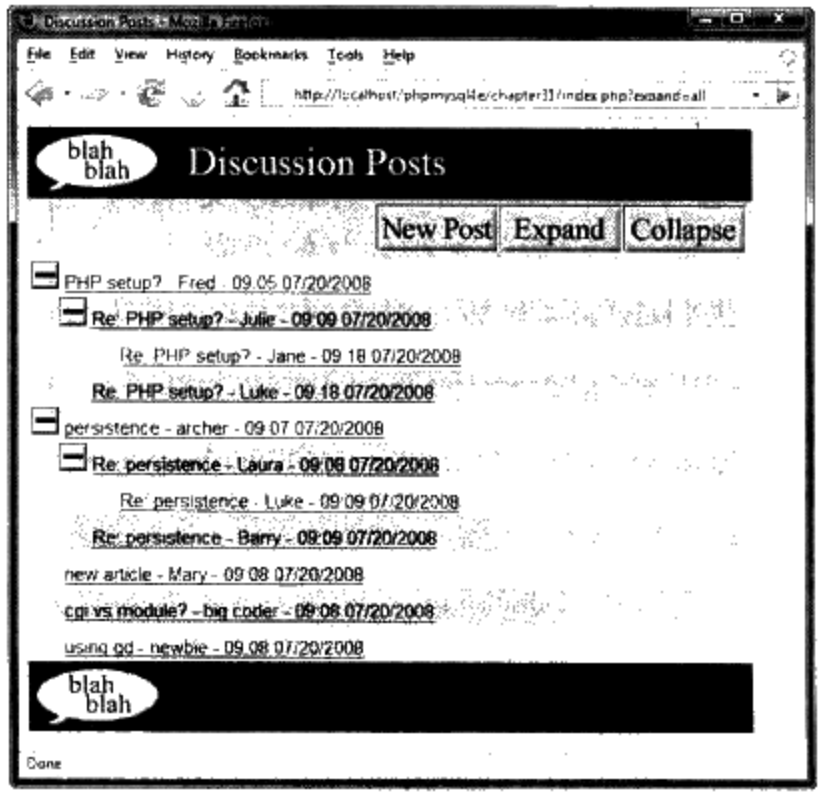

图31-6　所有话题都已被展开

如果你仔细查看图31-5和图31-6，可以看到我们在命令行中将一些参数传回了index.php。在图31-5中，URL框显示了如下所示字符串：

```
http://localhost/phpmysql4e/chapter31/index.php?expand=2#2
```

以上脚本将上面这行解释为“展开postid为2的节点”。“#”号仅是个HTML标记，用来滚动页面直到刚才展开的那个节点。

```
http://localhost/phpmysql4e/chapter31/index.php?expand=all
```

在图31-6中，URL是点击“Expand”按钮将传递一个值为all的expand参数。

31.5.1 展开和折叠

要了解如何创建文章视图，让我们来研究index.php脚本，如程序清单31-2所示。

程序清单31-2 index.php——在应用程序主页面创建文章视图的脚本

```
<?php
  include ('include_fns.php');
  session_start();

  // check if we have created our session variable
  if(!isset($_SESSION['expanded'])) {
    $_SESSION[ expanded'] = array();
  }

  // check if an expand button was pressed
  // expand might equal 'all' or a postid or not be set
  if(isset($_GET['expand'])) {
    if($_GET['expand'] == 'all') {
      expand_all($_SESSION['expanded']);
    } else {
      $_SESSION['expanded'][$_GET['expand']] = true;
    }
  }

  // check if a collapse button was pressed
  // collapse might equal all or a postid or not be set
  if(isset($_GET['collapse'])) {
    if($_GET[ collapse']=='all') {
      $_SESSION['expanded'] = array();
    } else {
      unset($_SESSION[ expanded'][$_GET['collapse']]);
    }
  }

  do_html_header('Discussion Posts');

  display_index_toolbar();

  // display the tree view of conversations
  display_tree($_SESSION['expanded']);
```

```
  do_html_footer();
?>
```

以上脚本使用3个变量来实现其功能。它们是：

- 会话变量expanded，用来记录那些已经被展开的记录。该变量可以在不同的视图之间使用，因此我们可展开多个话题。该变量是一个数组，它包含了与那些需要展开并显示其回复文章相关的postid数组。
- 参数expand，告诉脚本哪个新话题需要展开。
- 参数collapse，通知脚本哪个话题需要折叠。

当点击加号或减号，或者点击Expand或Collapse按钮时，它们将再次调用index.php脚本，不过将带有新的参数expand和collapse。我们使用expanded变量在页面间记录哪些话题在任何给定的视图中应该展开。

该脚本以初始化一个会话开始，并增加expanded变量作为会话变量，如果这个操作还没有完成的话。之后，该脚本将检查是否传递了expand或collapse参数，并相应地修改expanded数组。请看以下关于expand参数的代码：

```
if(isset($_GET['expand'])) {
  if($_GET['expand'] == 'all') {
    expand_all($_SESSION['expanded']);
  } else {
    $_SESSION['expanded'][$_GET['expand']] = true;
  }
}
```

如果点击了Expand按钮，该操作将调用expand_all()函数，这样所有有回复的话题都将加入到expanded数组中（稍后将详细介绍）。

如果想要扩展特定话题，可以通过expand变量传递一个postid。因此，我们可以在expanded数组中增加一条新记录来反映它。

expand_all()函数如程序清单31-3所示。

程序清单31-3　discussion_fns.php函数库中的expand_all()函数——对$expanded数组进行操作以展开论坛中所有话题

```
function expand_all(&$expanded) {
  // mark all threads with children as to be shown expanded
  $conn = db_connect();
  $query = 'select postid from header where children > 1';
  $result = $conn->query($query);
  $num = $result->num_rows;
  for($i = 0; $i<$num; $i++) {
    $this_row = $result->fetch_row();
    $expanded[$this_row[0]]=true;
  }
}
```

该函数运行一个数据库查询语句来找出论坛中哪些话题有回复，如下所示：

```
select postid from header where children = 1
```

每一篇返回的文章都将被添加到expanded数组中。我们运行查询语句是为了节省时间。也可以简单地将所有文章加入到expanded列表中，但是尝试处理那些不存在的回复相当浪费时间。

折叠文章节点的操作与此非常相似，不过是以相反的方法进行的，如下所示：

```
if(isset($_GET['collapse'])) {
  if($_GET['collapse']=='all') {
    $_SESSION['expanded'] = array();
  } else {
    unset($_SESSION['expanded'][$_GET['collapse']]);
  }
}
```

可以通过重置该数组，从该数组中删除所有条目。如果整个页面都需要折叠，就删除需要折叠的话题或重置整个数组。

以上所有这些操作都只是预处理，由此我们可以知道哪些文章应该显示出来而哪些不应该显示。该脚本的主要部分是对display_tree($_SESSION['expanded']);函数的调用，由它实际产生要显示的文章树形结构。

31.5.2 显示文章

下面，我们来了解display_tree()函数，如程序清单31-4所示。

程序清单31-4 output_fns.php函数库中的display_tree()函数——建立树形结构的根节点

```
function display_tree($expanded, $row = 0, $start = 0) {
  // display the tree view of conversations

  global $table_width;
  echo "<table width=\"".$table_width."\">";

  // see if we are displaying the whole list or a sublist
  if($start>0) {
    $sublist = true;
  } else {
    $sublist = false;
  }

  // construct tree structure to represent conversation summary
  $tree = new treenode($start, '', '', '', 1, true, -1, $expanded,
         $sublist);
  // tell tree to display itself
  $tree->display($row, $sublist);

  echo "</table>";
}
```

该函数的主要作用是创建树形结构的根节点。我们用它来显示整个索引，并在view_post.php页面上建立回复文章的子树。可以看到，它带有3个参数。第1个参数是$expanded，它是要以展开形式显示的文章列表。第2个参数是$row，是一个行号，用来计算在列表中该行的交替颜色。

第3个参数是$start，告知函数在哪里开始显示文章。这是将被创建并显示的树的根节点的postid。如果我们打算显示所有文章，如主页面一样，该变量将是0，表示显示所有没有父节点的文章。如果该参数是0，我们将$sublist变量设为false，并显示整个树。

如果该参数大于0，我们将它作为该树的根节点来显示，将$sublist变量设为true，可以建立和显示该树的一部分（我们将在view_post.php脚本程序中用到它）。

该函数最重要的一个功能就是初始化一个treenode类的实例，它代表该树的根节点。它实际上并不是一篇文章，但它相当于所有第一级文章的父文章，而事实上第一级文章是没有父节点的。树建立以后，我们将调用显示函数来显示文章列表。

31.5.3 使用treenode类

treenode类的代码如程序清单31-5所示。（在这里，我们会发现它非常有用，查阅第6章回忆它是如何工作的。）

程序清单31-5　treenode_class.php函数库中的treenode类——该应用程序的主体架构

```
<?php
// functions for loading, constructing and
// displaying the tree are in this file

class treenode {
  // each node in the tree has member variables containing
  // all the data for a post except the body of the message
  public $m_postid;
  public $m_title;
  public $m_poster;
  public $m_posted;
  public $m_children;
  public $m_childlist;
  public $m_depth;

  public function __construct($postid, $title, $poster, $posted,
       $children, $expand, $depth, $expanded, $sublist) {
    // the constructor sets up the member variables, but more
    // importantly recursively creates lower parts of the tree
    $this->m_postid = $postid;
    $this->m_title = $title;
    $this->m_poster = $poster;
    $this->m_posted = $posted;
    $this->m_children =$children;
    $this->m_childlist = array();
```

```
$this->m_depth = $depth;

// we only care what is below this node if it
// has children and is marked to be expanded
// sublists are always expanded
if(($sublist || $expand) && $children) {
  $conn = db_connect();

  $query = 'select * from header where
            parent = '.$postid.' order by posted';
  $result = $conn->query($query);

  for ($count=0; $row = @$result->fetch_assoc(); $count++) {
    if($sublist || $expanded[$row['postid']] == true) {
      $expand = true;
    } else {
      $expand = false;
    }
    $this->m_childlist[$count]= new treenode($row['postid'],
            $row['title'], $row['poster'],$row['posted'],
            $row['children'], $expand, $depth+1, $expanded,
            $sublist);
  }
}

function display($row, $sublist = false) {
  // as this is an object, it is responsible for displaying itself

  // $row tells us what row of the display we are up to
  // so we know what color it should be

  // $sublist tells us whether we are on the main page
  // or the message page. Message pages should have
  // $sublist = true.
  // On a sublist, all messages are expanded and there are
  // no '+' or '-' symbols.

  // if this is the empty root node skip displaying
  if($this->m_depth>-1) {
    //color alternate rows
    echo "<tr><td bgcolor=\"";
    if ($row%2) {
      echo "#cccccc\">";
    } else {
      echo "#ffffff\">";
    }
```

```
// indent replies to the depth of nesting
for($i = 0; $i < $this->m_depth; $i++) {
  echo "<img src=\"images/spacer.gif\" height=\"22\"
             width=\"22\" alt=\"\" valign=\"bottom\" />";
}

// display + or - or a spacer
if ((!$sublist) && ($this->m_children) &&
      (sizeof($this->m_childlist))) {
// we're on the main page, have some children, and they're expanded

  // we are expanded - offer button to collapse
  echo "<a href=\"index.php?collapse=".
           $this->m_postid."#".$this->m_postid."\"><img
           src=\"images/minus.gif\" valign=\"bottom\"
           height=\"22\" width=\"22\" alt=\"Collapse Thread\"
           border=\"0\" /></a>\n";
} else if(!$sublist && $this->m_children) {
  // we are collapsed - offer button to expand
  echo "<a href=\"index.php?expand=".
           $this->m_postid."#".$this->m_postid."\"><img
           src=\"images/plus.gif\" valign=\"bottom\"
           height=\"22\" width=\"22\" alt=\"Expand Thread\"
           border=\"0\" /></a>\n";
} else {
  // we have no children, or are in a sublist, do not give button
  echo "<img src=\"images/spacer.gif\" height=\"22\"
         width=\"22\" alt=\"\" valign=\"bottom\"/>\n";
}

  echo "<a name=\"".$this->m_postid."\"><a href=
           \"view_post.php?postid=".$this->m_postid."\">".
           $this->m_title." - ".$this->m_poster." - ".
           reformat_date($this->m_posted)."</a></td></tr>";

  // increment row counter to alternate colors
  $row++;
}
// call display on each of this node's children
// note a node will only have children in its list if expanded
$num_children = sizeof($this->m_childlist);
for($i = 0; $i<$num_children; $i++) {
  $row = $this->m_childlist[$i]->display($row, $sublist);
}
return $row;
  }
}

?>
```

该类包含主应用程序中用来驱动树形视图的功能。

treenode类的一个实例包括关于帖子及到所有对该帖子回复的帖子的链接细节信息。这样，我们就必须为该类定义如下所示的成员变量：

```
public $m_postid;
public $m_title;
public $m_poster;
public $m_posted;
public $m_children;
public $m_childlist;
public $m_depth;
```

请注意，treenode并不包含文章的正文。在用户访问view_post.php脚本之前，没有必要载入文章的正文。我们需要尽量使这些运行得相对较快，因为在显示树型列表时，我们还需要完成许多的数据操作，而且在页面刷新时或者按下某个按钮时需要重新计算。

这些变量的命名方法遵循在面向对象应用程序中普遍使用的命名规则——变量名称以m_开始提醒我们它们是类的成员变量。

这些变量的大部分直接对应于数据库中header表的数据行。$m_childlist和$m_depth变量是例外。我们用变量$m_childlist来保存该文章的回复。而变量$m_depth保存我们深入树的层次数——这个信息可以用来创建视图。

该类的构造函数将设置所有变量的值，如下所示：

```
public function __construct($postid, $title, $poster, $posted, $children,
                            $expand, $depth, $expanded, $sublist){
// the constructor sets up the member variables, but more
// importantly recursively creates lower parts of the tree
$this->m_postid = $postid;
$this->m_title = $title;
$this->m_poster = $poster;
$this->m_posted = $posted;
$this->m_children =$children;
$this->m_childlist = array();
$this->m_depth = $depth;
```

当我们在主页面的display_tree()函数中构造根treenode节点时，实际上创建了一个虚节点，没有文章与之相关。我们将传递一些初始值：

```
$tree = new treenode($start, '', '', '', 1, true, -1, $expanded, $sublist);
```

以上代码将创建一个$postid为0的根节点。它可用来找出所有第一级的文章，因为它们有一个postid为0的父节点。我们将它的深度设为-1是因为该节点实际上并不是视图的一部分。所有第一级的帖子的深度都为零，处于屏幕的最左端。接下来的深度慢慢向右扩展。

在该构造函数中，最重要的一点是当前节点的子节点被初始化了。该过程中我们首先检查是否需要扩展子节点。仅当某节点有子节点时才执行该过程，并已选择了显示它们：

```
if(($sublist||$expand) && $children){
$conn = db_connect();
```

接下来，我们将连接到数据库，并取出所有子文章，如下所示：

```
$query = 'select * from header where parent = '".$postid."' order by posted';
$result = $conn->query($query);
```

然后，使用treenode类的实例填充$m_childlist数组，该数组将包含存储在该节点中所有的回复，如下所示：

```
for ($count=0; $row = @$result->fetch_assoc(); $count++) {
  if($sublist || $expanded[$row['postid']] == true) {
    $expand = true;
  } else {
    $expand = false;
  }
  $this->m_childlist[$count]= new treenode($row['postid'],$row['title'],
        $row['poster'],$row['posted'], $row['children'], $expand,
        $depth+1, $expanded, $sublist);
}
```

最后一行代码将创建新的treenode节点，它完全遵照前面讨论过的过程，然而这是对于该树的下一个层次来说的，这里是递归部分。一个父节点调用treenode的构造函数，将自身的postid作为父节点传递，并在传递前将自身的深度加1。

依次创建每个树节点及它的子节点，直到不再有回复或者到达我们想要扩展的深度为止。

完成以上操作后，我们将调用根节点的显示函数（回顾display_tree()函数），如下所示：

```
$tree->display($row, $sublist);
```

display()函数首先检查它是否是虚根节点：

```
if($this->m_depth > -1)
```

通过该方法，可以将虚节点从视图中去除。然而我们并不希望完全跳过根节点。虽然我们不显示它，但它需要通知它的子节点来显示它们自己。

该函数接下来将绘制包含文章的表。它使用取模操作符（%）来判断该行的背景色应该是什么（因此两种背景色需要交替变化）：

```
//color alternate rows
echo "<tr><td bgcolor=\"";
if ($row%2) {
  echo '#cccccc\">';
} else {
  echo '#ffffff\">";
}
```

接下来，该函数使用$m_depth成员变量来计算当前条目应该缩进多少。回顾前面的图例，可以看到，回复的层次越深，缩进得越多。这是通过下面的代码来完成的：

```
// indent replies to the depth of nesting
for($i = 0; $i < $this->m_depth; $i++) {
    echo "<img src=\"images/spacer.gif\" height=\"22\"
              width=\ 22\" alt=\"\" valign=\"bottom\" />";
```

```
}
```

接下来，该函数将判断是否应提供加号或减号或者什么都没有：

```
// display + or - or a spacer
if ( !$sublist && $this->m_children && sizeof($this->m_childlist)) {
// we're on the main page, have some children, and they're expanded
  // we are expanded - offer button to collapse
  echo "<a href=\"index.php?collapse=".
           $this->m_postid."#".$this->m_postid."\"><img
           src=\"images/minus.gif\" valign=\"bottom\" height=\"22\"
           width=\"22\" alt=\'Collapse Thread\' border=\"0\" /></a>\n";
} else if(!$sublist && $this->m_children) {
   // we are collapsed - offer button to expand
   echo "<a href=\"index.php?expand=".
           $this->m_postid."#".$this->m_postid."\"><img
           src=\"images/plus.gif\" valign=\"bottom\" height=\'22\"
           width=\"22\" alt=\'Expand Thread\' border=\"0\" /></a>\n";
} else {
  // we have no children, or are in a sublist, do not give button
  echo '<img src=\'images/spacer.gif\" height=\'22\' width=\"22\"
            alt=\'\' valign=\'bottom\"/>\n';
}
```

完成以上操作后，将显示该节点的详细内容：

```
echo "<a name=".".$this->m_postid."\"><a href=
         \'view_post.php?postid=".$this->m_postid."\'>".
         $this->m_title." - ".$this->m_poster." - ".
         reformat_date($this->m_posted)."</a></td></tr>';
```

我们改变下一行的颜色：

```
// increment row counter to alternate colors
$row++;
```

之后，有一些代码将被包括根节点在内的所有节点执行，如下所示：

```
// call display on each of this node's children
// note a node will only have children in its list if expanded
$num_children   sizeof($this->m_childlist);
for($i = 0; $i<$num_children; $i++) {
  $row = $this->m_childlist[$i]->display($row, $sublist);
}
return $row;
```

在这里，又是一个递归函数调用，它将调用每一个节点的子节点来显示它们自身。将当前行颜色传递给它们，当完成显示操作后再将它传递回来，这样就可以保存交替颜色的记录。

以上就是这个类的所有功能。其代码相当复杂。可以通过运行该程序来认识这个类。当对它的功能感到满意时，再返回来研究这些代码。

31.6 查看单个的文章

display_tree()函数的调用将会给出一组文章的链接。如果我们点击其中的某篇文章，将进入view_post.php脚本程序，同时传递给该脚本这篇文章的postid。该脚本的示例输出如图31-7所示。

view_post.php脚本如程序清单31-6所示，除了显示该消息的回复外，还可以显示该消息的正文。可看到这些回复也是以树形结构显示出来的，不过这个树形结构已经被完全展开了，并且不带有加号或减号按钮。这是$sublist开关变量起作用的结果。

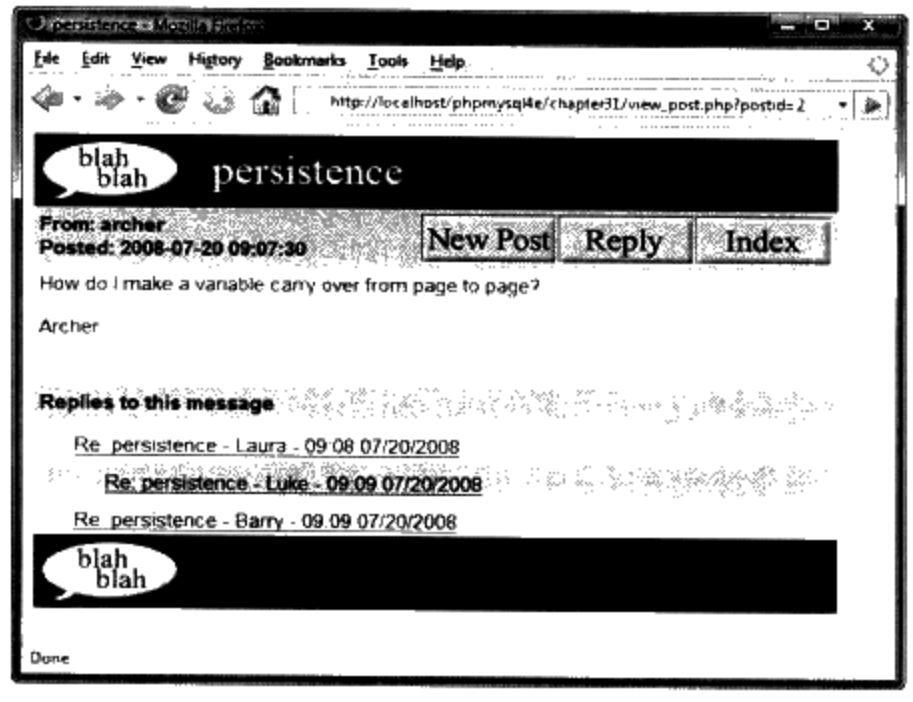

图31-7　现在，我们可以看到该帖子的消息正文

程序清单31-6　view_post.php——显示单个消息正文

```
<?php
  // include function libraries
  include ('include_fns.php');
  $postid = $_GET['postid'];
  // get post details
  $post = get_post($postid);

  do_html_header($post['title']);
  // display post
  display_post($post);

  // if post has any replies, show the tree view of them
  if($post['children']) {
    echo '<br /><br />';
    display_replies_line();
    display_tree($_SESSION['expanded'], 0, $postid);
  }

  do_html_footer();
?>
```

该脚本主要调用3个函数来实现其功能：get_post()、display_post()和display_tree()函数。get_post()函数将一条消息细节从数据库中取出来。该函数的代码如程序清单31-7所示。

程序清单31-7 discussion_fns.php函数库中的get_post()函数——从数据库中取出一则消息

```
function get_post($postid) {
  // extract one post from the database and return as an array

  if(!$postid) {
    return false;
  }

  $conn = db_connect();

  //get all header information from 'header'
  $query = "select * from header where postid = '".$postid."'";
  $result = $conn->query($query);
  if($result->num_rows!=1) {
    return false;
  }
  $post = $result->fetch_assoc();

  // get message from body and add it to the previous result
  $query = "select * from body where postid = '".$postid."'";
  $result2 = $conn->query($query);
  if($result2->num_rows>0) {
    $body = $result2->fetch_assoc();
    if($body) {
      $post['message'] = $body['message'];
    }
  }
  return $post;
}
```

该函数根据一个给定的postid执行两个查询语句，为该文章取出消息的标题和正文，并将它们一起放在一个相关的数组中返回。

接下来，该函数返回结果将传递给output_fns.php函数库中的display_post()函数，该函数仅以某种HTML格式将数组打印出来，因此这里我们不再讨论。

最后，view_post.php脚本将检查该文章是否有回复并调用display_tree()函数以子列表的格式显示它们——即不带加号或减号并且完全展开的形式。

31.7 添加新文章

了解了以上所有操作后，我们现在可以来看看如何将一篇新文章加入到论坛中去。用户可以使用两种方法来完成该项操作：第一，通过在索引页面点击“New Post”按钮，第二，通过在view_post.php页面点击“Reply”按钮。

这些动作都将触发相同的脚本，new_post.php，只是带有不同的参数。图31-8显示通过点击“Reply”按钮而得到的new_post.php脚本的输出结果。

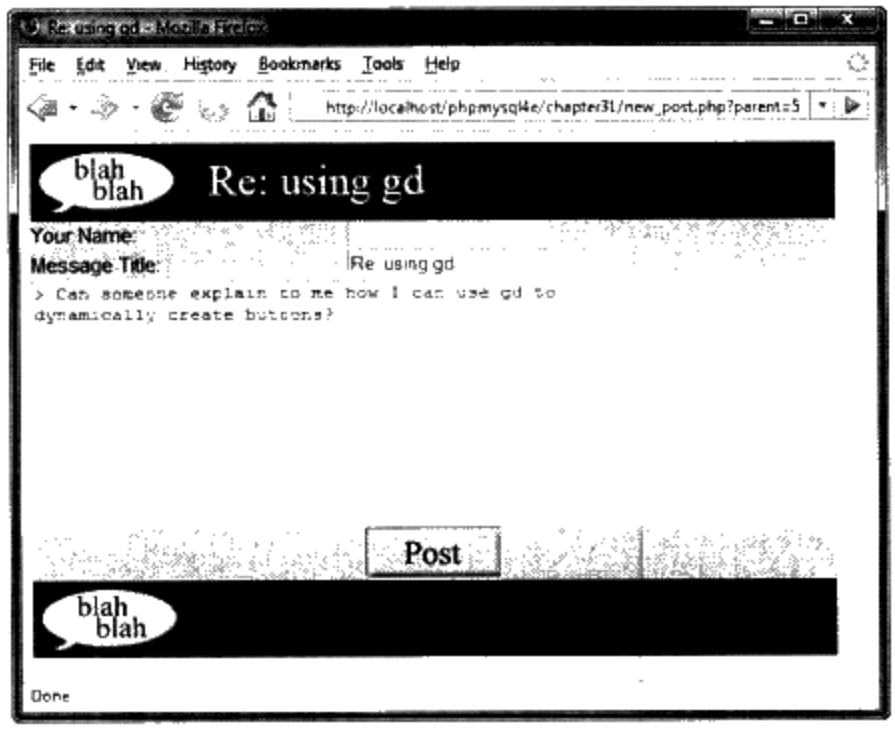

图31-8　在回复文章中，父文章的文本将自动插入和标记

首先，让我们查看上图所示的URL：

```
http://localhost/phpmysql4e/chapter31/new_post.php?parent=5
```

传递给parent参数值应该是新文章父节点的postid。如果点击了“New Post”按钮而不是“Reply”按钮，将在URL中得到parent = 0的字符串。

其次，可以看到对于一个回复，最初文章的文本被插入进来，而且前面都用“>”字符标记了，这一点与大多数的邮件和新闻阅读程序一样。

第三，可以看到回复消息的默认标题为最初标题加上前缀“Re:”。

下面，我们来看看产生该输出的代码，如程序清单31-8所示。

程序清单31-8　new_post.php——允许用户对一个已有帖子输入或创建一篇新文章

```
<?php
  include ('include_fns.php');

  $title = $_POST['title'];
  $poster = $_POST['poster'];
  $message = $_POST['message'];

  if(isset($_GET['parent'])) {
    $parent = $_GET['parent'];
  } else {
    $parent = $_POST['parent'];
  }

  if(!$area) {
    $area = 1;
  }
```

```
if(!$error) {
  if(!$parent) {
    $parent = 0;
    if(!$title) {
      $title = 'New Post';
    }
  } else {
    // get post name
    $title = get_post_title($parent);

    // append Re:
    if(strstr($title, 'Re: ') == false) {
      $title = 'Re: '.$title;
    }

    //make sure title will still fit in db
    $title = substr($title, 0, 20);

    //prepend a quoting pattern to the post you are replying to
    $message = add_quoting(get_post_message($parent));
  }
}
do_html_header($title);
  display_new_post_form($parent, $area, $title, $message, $poster);

  if($error) {
     echo "<p>Your message was not stored.</p>
           <p>Make sure you have filled in all fields and
           try again.</p>";
  }

  do_html_footer();
?>
```

在完成一些初始化设置之后，该脚本将检查父节点是否为0。如果是0，则表示该文章是一个新主题，而接下来需要完成的操作就相对少些。

如果是一个回复（$parent变量是某篇已有文章的postid），以上脚本程序将继续执行并且设置标题和最初消息的文本，如下所示：

```
// get post name
$title = get_post_title($parent);

// append Re:
if(strstr($title, 'Re: ') == false) {
   $title = 'Re: '.$title;
}
```

```
//make sure title will still fit in db
$title = substr($title, 0, 20);

//prepend a quoting pattern to the post you are replying to
$message = add_quoting(get_post_message($parent));
```

这里用到的函数包括get_post_title()、get_post_message()和add_quoting()。这些函数都来自discussion_fns.php函数库。它们分别如程序清单31-9、程序清单31-10和程序清单31-11所示。

程序清单31-9　discussion_fns.php函数库中的get_post_title()函数——从数据库中得到某消息的标题

```
function get_post_title($postid) {
  // extract one post's name from the database

  if(!$postid) {
    return '';
  }

  $conn = db_connect();

  //get all header information from 'header'
  $query = 'select title from header where postid = '.$postid.'';
  $result = $conn->query($query);
  if($result->num_rows!=1) {
    return '';
  }
  $this_row = $result->fetch_array();
  return $this_row[0] ;

}
```

程序清单31-10　discussion_fns.php函数库中的get_post_message()函数——从数据库中获得某消息的正文

```
function get_post_message($postid) {
  // extract one post's message from the database

  if(!$postid) {
    return '' ;
  }

  $conn = db_connect();

  $query = "select message from body where postid = '".$postid."'";
  $result = $conn->query($query);
```

```
    if($result->num_rows>0) {
      $this_row = $result->fetch_array();
      return $this_row[0];
    }
  }
```

这两个函数分别从数据库中获得特定文章的标题和正文。

程序清单31-11 discussion_fns.php函数库中的add_quoting()函数——用“>”符号缩进一则消息的正文

```
function add_quoting($string, $pattern = '> ') {
  // add a quoting pattern to mark text quoted in your reply
  return $pattern.str_replace("\n", "\n$pattern", $string);
}
```

add_quoting()函数将重新格式化字符串，用一个符号开始原始文本的每一行，在默认情况下，该符号为“>”。

用户在回复文章中完成所需输入并且点击“Post”按钮后，将进入store_new_post.php脚本程序。图31-9所示的是该脚本的一个示例输出。

在上图中，新发表的文章在这行下面Re: using gd?-Laura- Julie - 09:36 07/20/2008。不仅如此，该页面看起来还像普通的index.php页面。

下面，我们来看看store_new_post.php的代码，如程序清单31-12所示。

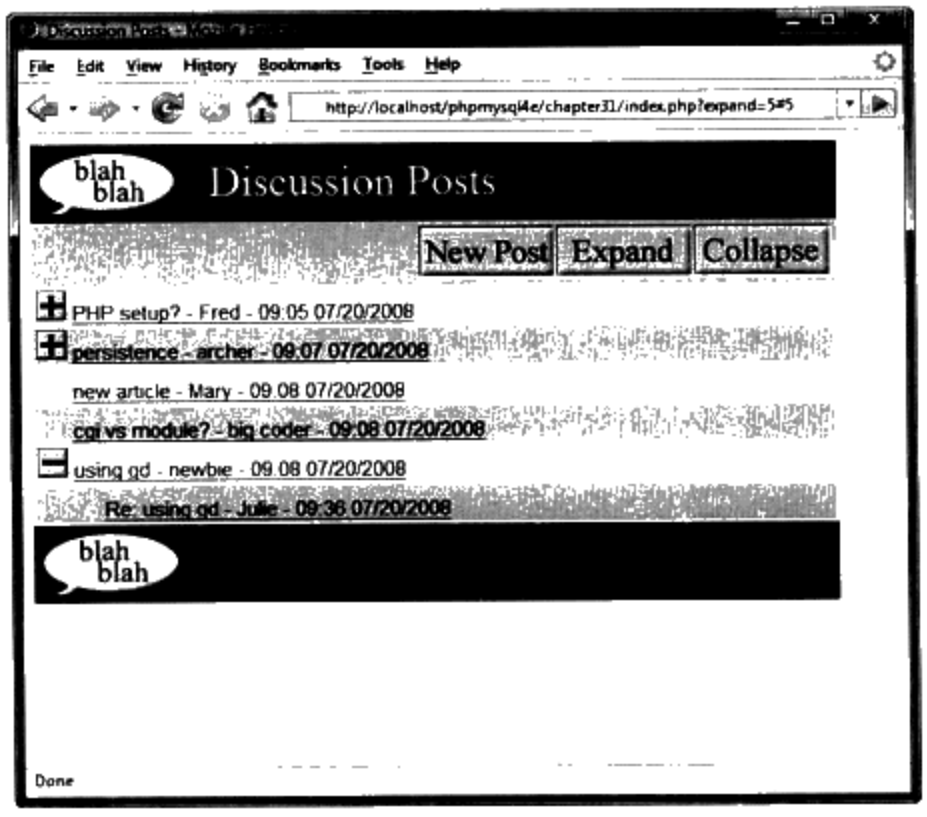

图31-9 新发表的文章在树形结构中显示出来

程序清单31-12 store_new_post.php——在数据库中存入新的文章

```
<?php
  include ('include_fns.php');
  if($id = store_new_post($_POST)) {
    include ('index.php');
  } else {
    $error = true;
    include ('new_post.php');
  }
?>
```

可以看到，这是一段很短的脚本程序。它的主要任务是调用store_new_post()函数。该页面没有它自身的可见内容。如果保存成功，我们会看到索引页面。否则，将回到new_post.php页，这样，用户可以重试。store_new_post()函数如程序清单31-13所示。

程序清单31-13　discussion_fns.php函数库中的store_new_post()函数——在数据库中保存并检验新发表的文章

```
function store_new_post($post) {
  // validate clean and store a new post

  $conn = db_connect();
  // check no fields are blank
  if(!filled_out($post)) {
    return false;
  }
  $post = clean_all($post);

  //check parent exists
  if($post['parent']!=0) {
    $query = "select postid from header where
              postid = '".$post['parent']."'";
    $result = $conn->query($query);
    if($result->num_rows!=1) {
      return false;
    }
  }

  // check not a duplicate
  $query = "select header.postid from header, body where
            header.postid = body.postid and
            header.parent = ".$post['parent']." and
            header.poster = '".$post['poster']."' and
            header.title = '".$post['title']."' and
            header.area = ".$post['area']." and
            body.message = '".$post['message']."'";

  $result = $conn->query($query);
  if (!$result) {
     return false;
  }

  if($result->num_rows>0) {
     $this_row = $result->fetch_array(1;
     return $this_row[0];
  }

$query = "insert into header values
```

```
            ('.$post['parent']." ,
             '.$post['poster ]."',
             '".$post['title']."',
             0,
             '.$post[ area']."',
             now(),
             NULL
             )";

$result = $conn->query($query);
if (!$result) {
   return false;
}

// note that our parent now has a child
$query = "update header set children = 1 where postid = '.$post[ parent']."'";
$result = $conn->query($query);
if (!$result) {
   return false;
}

// find our post id, note that there could be multiple headers
// that are the same except for id and probably posted time
$query = "select header.postid from header left join body
            on header.postid = body.postid
            where parent = '.$post['parent']."'
               and poster = '.$post['poster']."'
               and title = '".$post['title']."'
               and body.postid is NULL';

$result = $conn->query($query);
if (!$result) {
   return false;
}

if($result->num_rows>0) {
  $this_row = $result->fetch_array();
  $id = $this_row[0];
}

if($id) {
   $query = 'insert into body values
                ($id,  ".$post['message']." )';
   $result = $conn->query($query);
   if (!$result) {
     return false;
   }
```

```
        return $id;
    }

}
```

这是个很长的函数，但它并不太复杂。它之所以长，是因为插入一篇文章就意味着要标题表和正文表进行插入操作，并要在标题表中更新父文章行以表明它现在有子文章了。

以上就是该Web论坛应用程序的所有代码。

31.8 添加扩充

可以在该项目中增加如下所示的几项扩充功能：

- 可以在视图选项中增加导航功能，这样就可以从一篇文章跳到下一篇、前一篇、下一个话题的文章，或前一个话题的文章。
- 可以增加一个界面来建立新的论坛和删除旧的文章。
- 可以增加用户验证以确保只有注册用户可以发表文章。
- 可以增加某些限制和审查机制。

学习一些已有的系统，将从中受到一些启发。

31.9 使用一个已有的系统

Phorum是一个值得注意的已有系统，它是开放源代码的Web论坛项目。它与我们的系统有不同的导航和语义，但是它的结构相对来说易于定制，从而适合我们的站点要求。Phorum的一个显著特性是它能由用户自主确定配置，从而决定是以集中视图还是普通视图来显示文章。可以在下面的站点找到更多关于它的内容：http://www.phorum.org。

31.10 下一章

在第32章中，我们将用PDF格式来发布极具吸引力的、排列一致的、并且可以在某种程度上防止破坏的文档。这对于基于服务范畴的应用程序来说是相当有用的，比如生成在线合同。