第3章　应用程序设计：一个实际的例子

我们做事就应该防患于未然。
未雨绸缪则事半功倍。
一颗参天大树是从一颗幼苗长成的；
千里之行，
始于足下。

应用程序的开发是一个非常广泛的话题，即使我们用一整本书也不能全面的将其介绍详尽。术语应用程序设计（application design）几乎包含程序开发的每一个独立的部分——从数据结构的规划到信息流程图，到实体关联框图，到文档编制以及这之间的所有方面。既然它这么重要，我们就不能不在这本书中提到它。但我们不是就应用程序设计而进行研究讨论，而只是把话题局限于一个“沾一点边”的例子，即 phpChat。这一章将使读者详细而深入地了解这个用 PHP实现的实际的聊天服务程序，就像一次扩展的软件实例学习。我们希望读者能够吸取有用的信息和方法，在自己设计应用程序时能够用得着。

这一章中许多框图都包含了很多对应用程序设计都适用的技术注释。你应该记住这些注释，直接在我们建议的聊天服务程序（这里是 phpChat）中使用这些技巧，而且你也可以把这些技巧间接的应用到自己的开发项目中。

注意　另一个更加理论化的、但要简短一些的关于应用程序设计的讨论，可以参看第7章“尖端应用设计”。

3.1　项目概观

设计一个应用程序时，你先考虑的是这个应用程序要做什么。在 phpChat中，我们希望这个应用程序能够提供基于网络浏览器的聊天服务。

聊天应该具备以下的特征：

- 实时的同步聊天：没有信息的中断，也不需要刷新
- 不需要客户端的程序：浏览器面对的应该只是纯 HTML（最后也可能是JavaScript）。
- 有网络功能：它应该能够连接到用于聊天的对话框。
- 通用性：对目标系统的配置作尽可能低的假设，对系统的需求要尽可能少。
- 设计没有强制性：程序代码和页面布局是分离的。
- 易于使用和管理。
- 对用户数目和聊天室数目没有限制。

一旦你了解了这些要求，而且也知道了你的应用程序需要做些什么了，你就要估计一下情况，设计一个更加详细的关于应用程序该如何布局的概观表。

花一点时间把所有的要求都写下来。这是非常有用的，特别是作为以后的备忘录。

替用户设计一个应用程序时的步骤叫建立技术规格（ creating the specification）。这个时候，用户仍然可以影响应用程序的整个布局。这是非常重要的，因为设计的应用程序必须满足用户的所有要求。要不然你的程序就不会被用户所接受。

用户永远是对的，即使他们错了

需要应用程序的用户，通常他们自己没有多少设计类似应用程序的专业知识，这也是他们为什么要雇佣你的原因。当你和用户讨论各种要求时，如果他们提出了错误的方案，你要耐心的引导他们。例如，如果用户说：“ 我希望这个聊天程序能够全屏显示每一个聊天者的图像，至少每秒钟刷新一次 ”。你可能会有一个这样的相反的建议：“ 试着在每一行后面使用非常小的视图是不是会更好一点呢？大部分聊天的用户根本没有足够大的网络带宽去显示全屏的图像 ”。

但是要小心一点，绝对不要坚持你的观点（除非用户要求你实现根本不现实的功能）。总之，用户是花钱请你实现他们的想法。为了避免失去与用户的业务联系，你可能不得不接受去执行一个不太好的解决方案（当你发现不能引导用户同意使用正确的方法时）。以后当用户发现用他们的方法不能够实现的时候，你再来改变它。

在这个项目中，你将扮演项目经理的角色，但董事长却是你的用户。既然我们都是很不错的用户，我们就不应该坚持要详述这个应用程序更深的细节问题了；我们就把设计的剩余部分交给读者你了。无论什么时候，对于本章涉及到要作出选择和决定的地方，你都要静下来尽量作出自己的选择，仔细地估计所有的情况，然后把你的结果和本书讨论的结论相比较。

3.2 比较技术环节

在开始考虑程序代码的布局之前，这中间有一个我们不知道怎么称呼的步骤，只知道它“ 将所有的东西聚集到一起 ”。这是一个介于整体构思和技术规格 /代码布局阶段的步骤 —— 指出了内部的工作情况以及基于这些构思和技术规格的是什么。

为了把这一点说清楚，让我们回到最开始的部分，提出以下问题：

- 我们想要建立什么东西？
- 我们如何才能建立他们呢？
- 是否已经存在能够实现我们构思的方法？
- 是否有相似的几乎具有相同功能的系统存在？
- 如果是这样的话，我们能不能从别人的设计那里借鉴什么东西？
- 我们是否能够再利用外国的技术？有没有可能把它们添加到自己的系统中去？

第一条非常容易回答。现在我们想要创建一个聊天系统。怎样创建呢？用 PHP和某种方式的服务器端 —— 此时此刻，我们知道的就这么多了。

有没有现成的聊天系统或者某些相似的东西？事实上有。你可以马上翻你的书架 ——“ talk ”命令允许你和网上的其他人聊天，它创建了一个有效的网络连接通路（或者一个局域连接），如

图3-1所示。

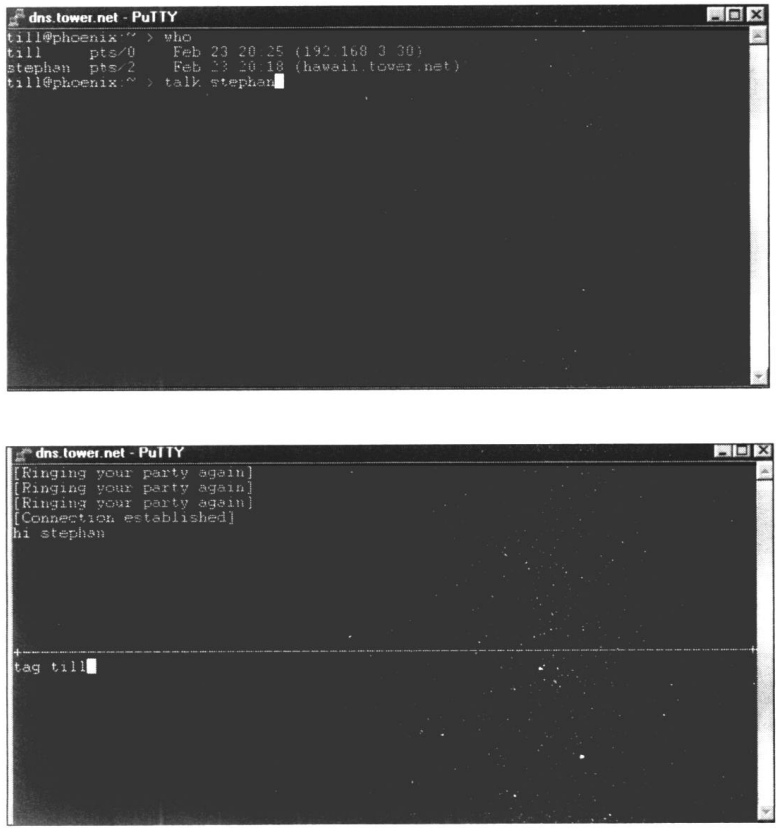

图3-1　传统的“talk”命令

当然，这还没有我们所期望的那样完美，但这只是一个开始。下一步我们可以到网上冲浪，寻找带有一个网络聊天链接的网页（这对聊天网络来说，几乎是必不可少的）。虽然它们表面和实际执行有很大的差异，但是它们大部分可以被概括成这样：

- 特别的程序接口一般使用Java语言，有的也使用一般的HTML语言。
- 用于单个服务器的（或仅仅是用于由数据库支持的）内部所属协议。
- 一些预定义的聊天室。
- 一些预定义的命令。

我们暂且不谈聊天初始化配置。现在已经有许多聊天应用程序和网络系统，例如：Mirabilis’ ICQ和多种多样的即时信息传播系统——系统通常不提供实时的服务，并且一般要求在每一个参与聊天的用户系统上安装附加的客户端软件。

然而，在以上清单中的系统中有一个系统出类拔萃。IRC（Internet Relay Chat）是非常著名的，而且长期被许多网络使用的聊天协议。它能同时负载成千上万的用户。IRC协议是基于文本的——在高强度负载情况下有一个缺点（长字符串命令比简单的二进制字符产生的网络堵塞严重的多），但是这也使得它处理过程变得非常简单了。大部分现有的IRC服务器支持压缩主干链

接，这大大地减少了网络堵塞。

虽然IRC要求在每一个参与聊天的用户系统上安装特殊的用户软件，但是我们可以利用这种要求，使其转化为我们软件的优势：我们为什么不在服务器端提供一个用户软件，然后使用一个HTML接口把它抽象化，并且允许每一个用户通过 HTML的客户端获得这些软件呢？这将使我们可以控制用户所做的事情（每个用户都要求使用我们的 HTML客户端）。另外，我们拥有现有网络系统的所有优点：可靠的用户软件、已经得到验证的理论、成百上千的工具等等。我们甚至可以允许用户使用他们自己的客户端软件 —— 这是在大多数情况下都是应该避免的，因为我们想创建一个“封闭”的聊天网络。在一个封闭的聊天网络中，我们知道每个用户能够访问你的网络的所有方法。通过在详细的初始化设置中限制这些访问的方式，可以大大的减少被黑客攻击的危险。

这直接导致了一个问题，我们到底需不需要像 IRC这样的协议呢？或者仅仅只使用一个由数据库驱动、有远程同步处理特征、能够满足所需网络功能的协议就已足够呢？

> 这样的问题在每次规划一个应用程序的时都会出现，而且还会经常出现。你要确信已经把所有类似问题都解决了，而且确信在开发的以后阶段都不会出现问题。你必须能找到出现这些问题的地方；否则以后你也许不能解决他们（还可能使你的项目最后变成一堆垃圾）。一个好的项目应该是一个没有垃圾、没有不确定性、没有不一致性和不可无法预料结果的项目。确保在规划环节之后，你可以获得一个稳定的而且是完全符合预想的解决方案。

所以，让我们回过头来回答这个问题：我们需要一个像 IRC这样开放的（可能也是复杂的）协议吗？还是应该坚持使用一个传统的数据库方法？找出答案的最简单的同时也是最有逻辑的方法 —— 进行正反比较，然后选择有最好结果的一项。

将IRC作为一个协议执行到聊天系统中，将引入极其复杂的事情，因为协议过程——网络协议过程需要非线性的程序代码，而实际上 PHP并不支持非线性代码（为了对网络信息起反应，我们需要一个基于事件处理的系统）。紧接下来的问题就是我们需要一种有效的处理信息交换的方法。也就是说处理来自用户的信息和替用户发送信息（遗憾的是收发信息通常不是用同一种方式处理的）。当然使用由数据库支持的解决方法也同样存在这个问题。但是由数据库支持的解决方法不需要协议处理。许多数据库本来就被 PHP支持，同时那些不被PHP支持的数据库也很可能会被ODBC间接地支持。为了获得能用于网络聊天的对话框的功能，我们只需要创建一个能够在两个对话框之间同步工作的工具即可（除非你只想运行一个能被所有对话框同时访问的中心数据库服务器）。

你选择哪一种方法？

最后的选择：phpChat是基于IRC的，下面是其原因：

- 使用一个数据库，相当于引入了一种所属类型，私有协议不能连接其他的规范系统。在讲究互用性和互连性的时代，这不是一件好事情。
- 一个功能完好的IRC库（也就是phpIRC，请参看www.phpwizard.net/phpIRC），把对IRC网络系统的访问抽象成一套易于使用的应用程序接口的函数 —— 在代码的复杂性方面使得IRC的处

理与数据库的处理相当。

• 现存的IRC服务器软件处理所有用户管理程序的琐碎问题，还管理可靠信息转送和路由选择等问题。这些软件已经被传播了很长一段时间，并被证实能够很好的工作。另外，它还能用于所有的系统类型。

• IRC非常容易升级。在网络使用的颠峰时刻，如果服务器 A加载时出现了问题，或者出现了未知的事件，仅仅只需连接到服务器 B上，动态地建立一个服务器连接，然后进入现有的聊天室（IRC允许你这样做，而且是全自动的） —— 现在你就可以给额外的用户提供另外一个拥有足够空间的服务器。

3.3　IRC网络基础

现在我们已经为聊天室的设计选择了一个交流标准，下面看一看 IRC网络究竟是如何建立的。

理想的情况是，你应该在选择IRC之前，评价这一节所讨论的IRC网络基础。因为在已经决定使用IRC之后，才发现IRC引入了非常复杂的结构，这是非常不好的。然而到这里为止，为了应用程序规划的目的，我们已经使用有关IRC网络的“共用知识”来工作了。现在我们已经介绍了一个可以用在应用程序（IRC）上的“正确的”方法。这部分提供了你执行这个规划所需的详细信息。

IRC网络对于客户端和服务器端是有区别的。用户只需要使用一个特殊的客户端软件就可以参与网络活动，同时，这就建立了一个到服务器到客户端的连接。通过特殊服务器的连接，所有网络上的服务器之间是连通的。当前的 IRC服务器执行只支持分层结构，意味着连接服务器时绝对不能有多余的方法。这形成了一个像树一样的倾向于形成网状分支的结构，但是这同时也大大的简化了路径：所有的服务器只需要把输入数据传送给所有其他连接就可以了，不用担心给一个服务器传送多余的信息。

每一个服务器可以拥有很多用户，用户数的最大值取决于这个服务器愿意接受的连接数量（当然，就网络容量和服务器负载而言，也存在限制）。如图3-2所示，每个服务器都可以通过穿梭或多或少的服务器中继段而连接到其他所有服务器。例如：服务器 C和服务器F可以带动在同一通道中的用户（通道是 IRC的聊天室 —— 人们可以见面和聊天的地方）。在这个例子中，服务器C可以通过它的唯一的连接将数据传送给服务器 B。服务器B再把数据分布给它的其他连接，即服务器A和服务器D。服务器A没有任何其他的连接，所以它不干任何事情，但是服务器 D将把数据又传给服务器E，然后服务器E又传给服务器F。这非常容易实现，但有一个缺点：如果服务器A没有任何连接进来的用户参与到服务器 C传送数据的通道中，那么在这个通道中的以服务器A为目标的所有数据都会浪费带宽。

IRC的RFC（如图3-2所示）

和互联网上所有开放的标准相似， IRC协议的基础是在一个 RFC（Request For Comments）中指定的。IRC使用的RFC是RFC 1459，例如， www.irchelp.org网站上有许多关于IRC的的信息。

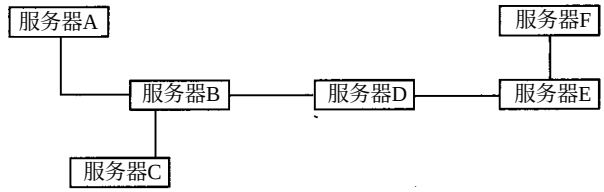

图3-2 一个IRC网络结构的例子

这是一个被设计所限制的网络的一个主要问题：所有的共用信息流都必须到达所有的服务器。但是，在我们打算执行IRC网络的时候，这些问题在连接时真的会出现吗？当然不会，因为在一个标准的服务器主机中，我们要处理的用户数量永远也不会大到有害的程度。在国际互联网中，这样的问题应该根本不会产生。

为了减少关键连接的总数量，网络可以按照它的物理拓扑结构来布局。如果一个服务器的连接超过它的剩余容量，它可以采用比其他服务器多的叶结构（连接许多叶节点到只有一个小的中枢服务器上是没有意义的）。另一个可选的方法是安装一个适合这个网络的路径。例如：美国的服务器设置在美国，而德国的服务器设置在德国等等。从法兰克福有一个到纽约的海外连接，在法兰克福的IRC服务器就应该连接到美国的服务器上（依据这个网络的物理布局）。这可以通过另一种方式来做到：法兰克福可以连接到波兰，随便举的一个例子。但是如果波兰自己没有海外连接的话，从法兰克福到波兰的这个路径的连接就需要找到其他的出口到达大洋彼岸——它可以绕到其他国家（甚至是两个，三个或者更多的国家）直到它找到一个自由的海外连接为止。这些附加的路径浪费了许多网络带宽。因而所作的尝试是为了使 IRC网络结构最好地适应这个内部的物理网络结构。

这些设计问题只有在最大型的、负载有成千上万的用户的网络中才会出现。这些网络才真正地需要为它的中枢系统找到可靠的连接。典型的基于网络聊天室或者网络不太可能同时负载1 000个用户，所以你不会一开始就碰到严重的问题。然而为了避免复杂性，围绕这些最终还是会出现的各种各样的问题，作一些规划是个不错的想法。

从服务器的角度来看，这个网络结构如图 3-3所示。

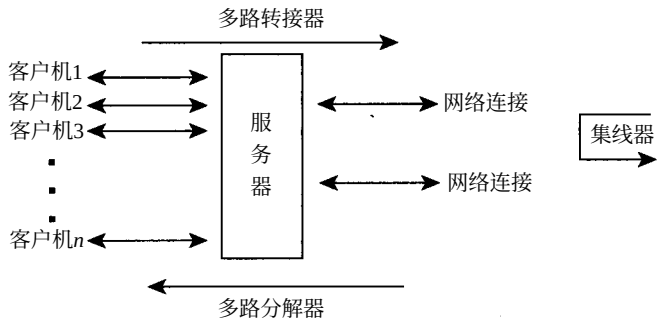

图3-3 从服务器角度看到的网络结构

这里执行的结构类似于一个多路转接器、多路分解器和一个网络交换机的混合物。从客户

端到网络端的方向来看，服务器压缩了从客户端传来的所有数据，然后把它传送到网络连接上。在另外一个方向上，它决定网络端的哪一个信息对哪一个用户来说是非常重要的，然后把它传送到合适的连接中。所有网络传来的要被继续传递的其他数据就直接被送走了。

这基本上就是我们需要对聊天系统所作的初始化设置。现在花一点时间试着想一下我们该如何实现我们的目标。我们需要一个正在工作的符合下列条件的服务器环境：

- 允许IRC网络连接。
- 允许IRC的用户连接。
- 提供基于互联网的用户接口。
- 尽可能容易的执行。

3.4　使应用程序适用于网络

如果你正好计划用 PHP开发你自己的服务程序（或者其他一些相似的东西），请重新思考一下。你可能已经对这些思想有些迷惑了：实现一个聊天服务程序意味着实现一个网络服务程序。这是我们实际上介绍给大家的东西，但这并不是要大家去做，因为这些是不必要的，现在各个系统已经有了许多很好的服务器软件。那么怎样使用现有的服务器？怎样把它作为一个客户端连接到网络上呢？我们唯一需要做的就是给网络添加另一个抽象化的布局，如图 3-4所示。

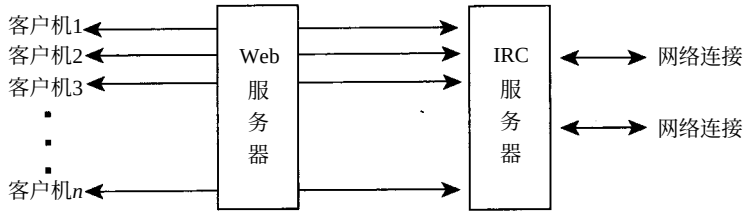

图3-4　作为一个抽象层连接到服务器上的phpChat

互联网服务器将运行PHP聊天服务程序。对于其接收的每一个用户连接，它都将创建一个从客户端到IRC服务器的连接。每一个聊天进程只负载单独的一个用户，并不需要担心其他用户。用户的坐标、信息流的控制等等可以由 IRC服务程序完成，这样我们只需要使用一台空闲的服务器就行了。

这项技术也有这样的优点：聊天服务应用程序可以被用作 IRC网络的一个安全的网关（参看图3-5）。许多公司和私有网络利用网络防火墙过滤 IRC端口。既然这个聊天程序只是通过 HTTP与它的用户交流（没有被过滤），那么只有这个聊天服务程序本身需要一个到 IRC服务器的开放连接。

因此，我们要做的唯一事情就是去编写这个客户端的软件，另外，在我们的互联网服务中客户端也需要这个软件。IRC知道所有设置一个完美的聊天程序所必需的命令，这个网络问题可以通过使用标准的服务软件来解决，这些软件就像架上的书一样唾手可得。因此，假设我们的接口能以很方便的方式支持所有 IRC特性的话，那么我们就完成了任务。

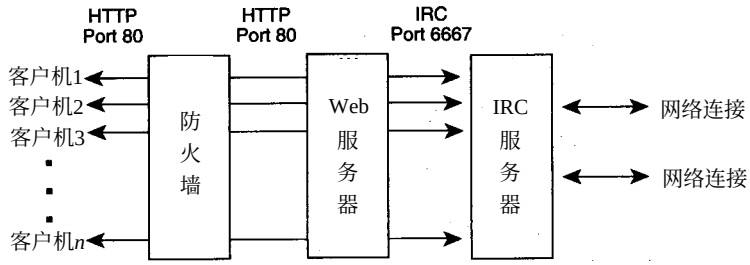

图3-5 phpChat作为一个安全的IRC网关

3.5 连接网络的接口

正如我们前面所提到的那样， IRC需要一些前置处理。为连接 IRC网络接口开辟一个完整的协议句柄的任务有一点艰巨。但是我们面对的是 IRC而不是由数据库支持的解决办法，这是因为一个应用程序接口已经存在了，它可以为我们完成这项任务。

一定要了解市场！每一个程序项目都必须知道程序的哪部分被其他人做好了，哪一些还需要我们去做。千万不要重新设计那些已经设计好的程序！特别是一些作为商业项目地程序，我们宁可花大量的钱为那些特别的任务购买国外安全的解决方案，也不要自己设计和开发它们。因为后者有时候会更贵，更耗时间。紧接的是，外来的解决办法通常是要经常被改进的—一个不受你自己项目进展情况制约的过程。从外来公司获得升级程序，你仅仅只需要把你应用程序中对应的那一部分用更新版本替代就可以了。通过这种方法，你可以将你应用程序的某部分升级，却不需要在这些工作中投入你自己的精力。另外，当使用现有库的时候，你会自动把你的项目建成通用的标准化的应用程序接口，这通常会带来很多好处。

另一方面，如果外来的产品不能够改善自身或者不能跟得上时代，你自己受制于外来的产品已经被证实是非常消极的，就像项目中有BUG一样是不正确的。

根据我们的经验，代码开放的产品成为最成功的合成外来部分。代码开放的产品的改进和扩展速度非常快，通常面向于通用的和开放的很有潜力的标准。

读者练习

搜索用PHP编写的、能够利用IRC的应用程序/库，然后就设计、灵活性和易于使用这几点将它们比较一番。当然，做这件事情也是非常有趣的（但是不应成为你的主要目的）。设计通常是最重要的开发部分。设计完成之后，剩下的工作通常就非常的直接，做起来也就很简单了（即使很多程序设计员想的不一样）。

我们为这个项目选择的库之所以是 phpIRC（www.phpwizard.net/phpIRC），原因如下：

- 使用起来很方便。
- 它是一个完美而且完整的应用程序接口。
- 它使用了基于事件的处理过程。

基于事件处理过程的使用在这里非常有趣，它是一个通常只有在传统应用程序中才能实现的技术。例如：所有的 Windows应用程序都是基于事件的程序。基于事件的程序在不停的循环中运行，等待事件的出现。事件可以包括用户输入、鼠标移动、网络事件（输入数据包）等等。只要有一个事件被发送过来，所有的要处理要刚才发生的这个事件的过程就被调用，调用时使用这个事件指定的参数（例如：输入的网络信息流的打包数据）。

确切的说，使用“传统”的编程方法，一个引入的 ping将会被处理，如清单3-1所示：

清单3-1　处理ping的伪代码

```
again:

wait_for_network_data();

if(incoming_data == ping)
{
    send_pong();
    update_traffic_counter();
}

goto again;
```

这段代码一直都在等待，直到它从网络上接收到数据，然后试着找出这个数据是否为一个ping。如果是，那么这段代码就把 pong送回，由于统计学的原因，它要更新一个信息流计数器。接下来，它就跳回开始处。假设这种情况有成百上千的事件，其中一些可能还依赖于别的事件，有的不依赖，有的又只是出现在某种特定的情况中……这真是太痛苦了。

然而，基于事件的程序使得处理一个 ping变得特别简单，如清单3-2所示。

清单3-2　基于事件的处理ping的伪代码

```
event_handler ping()
{

    send_pong();
}

event_handler incoming_data()
{

    update_traffic_counter();

    case of ping: handle_event(ping);
}

while(not_done())
{

    wait_for_event();

    case of network_data: handle_event(incoming_data);

}
```

这段代码看起来好像多一点，但是却清晰多了。这段代码的主循环等待一个事件的发生。如果它发现一个发生的事件，该事件的发生是由于网络数据的输入而触发的。它就使用内部的过程处理函数handle_event（）来调度这个事件。该函数为此事件选择一个处理程序然后调用它。这个处理程序依次更新信息流计数器；如果第一个事件也是一个 ping的话，处理程序将装入这个事件进行处理。在再次使用 handle_event（）调度这个事件之后，一个 pong就被发送了。

ping（）和incoming_data（）中的任一个都可以在事件“ incoming_data ”中注册自己。然而创建两个不同的事件能够使事件的种类更具多样性，而且允许更多详细的有目标性的处理过程。

刚开始的时候，要习惯于使用基于事件的信息处理是有一点点难度的（它使用起来和一台有限状态的机器差不多），但是它有很多优点：

- 一个模块化的结构安装在这个应用程序中。每一个模块都独立于其他模块工作，它们可以很容易地被改变、交换或者扩展。
- 程序的任何部分都可以引发任何种类的事件，因而加强了在这个应用程序中对任何类型的事件的处理能力（换一句话说，你可以从你的代码的任何部分来控制其他部分）。
- 从程序中心的一个位置，所有的数据都可以清楚而明确的被调配到所有的容器当中去。你不必担心需要自己编写拷贝和传送的结构：每个事件的处理程序都自己负责接收数据。
- 新的代码可以很容易的插入到应用程序中：你只要创建一个能把自己注册到合适事件中的过程就可以了。

因此，一旦主要的事件调配框架被创建，整个应用程序就可以通过分别编写一个又一个的处理程序来创建。

请熟练的掌握这个用于实现有限状态的机器的技术。这些技术一般在程序设计和信息处理中都是基础性的东西。

幸运的是，这些事件调配框架都已包含在 phpIRC当中。于是我们不再需要为这个项目编写类似的程序了。

3.5.1 接口结构

phpIRC构成IRC用户端的部分应用程序，它还负责所有的网络信息交流。这意味着它也需要始终在我们的控制之下，以便能实时地对网络消息起作用。如果 phpIRC的信息处理函数只是偶尔才起作用，那么安全、保密和快速的信息交流就不能保证了。就是因为这个原因， phpIRC强行指定了一个程序布局，如图 3-6所示。

在初始化和设置完之后，应用程序就不得不把控制权交给 phpIRC。然后phpIRC进入它的主事件循环，并且等待事件的发生。在设置的过程中，这个应用程序不得不为每一个要处理的事件（例如：输入的私有信息、输入的服务器信息等等）注册一个回调动作。这些回调动作是这些应用程序重新获得控制的唯一途径。 phpIRC然后就把所有的事件调配到所有使用库进行自身注册过的函数中。这些函数可以依次进入 phpIRC中的另外一个空闲循环中，以等待另一次事件的发生，或者它们可以使用 phpIRC的应用程序接口在网络上执行某种特定的功能（发送私有信

息、加入/离开通道等等)。

这个非常基本的布局允许下游的信息交流，这意味着 phpIRC能够接收来自于其他用户的信息。人们就可以真正地通过你的脚本程序交谈了。

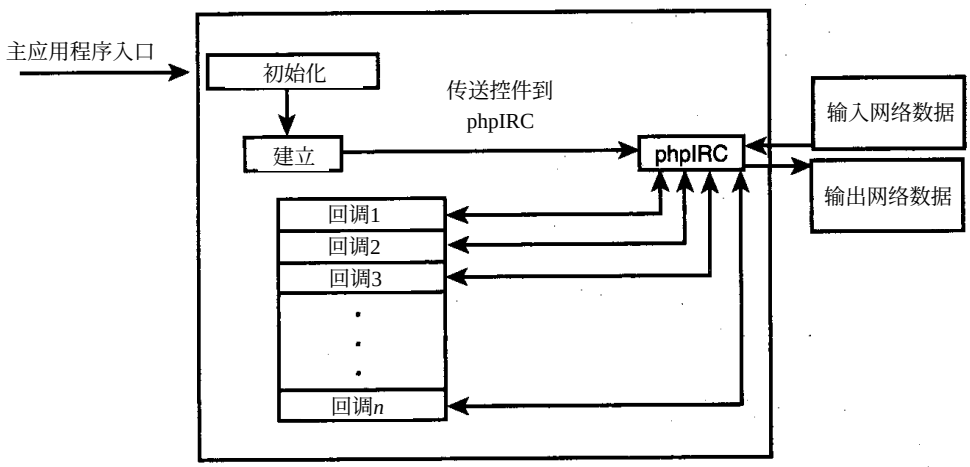

图3-6　在phpIRC中强行指定的应用程序布局

注意　下游的意思是从网络到用户。上游的意思就恰好相反，是从用户到网络。

读者练习

构建一个利用phpIRC特性的下游接口。在稿纸上实现它，使它与phpIRC的应用程序接口相近。然后建立一个简单的下游接口，这个接口能注册到IRC，以显示从特定通道传来的信息。

3.5.2　下游信息交流

既然聊天是一个实时的任务，意思是说当你做某事的时候，某事就发生，并且引发立即反应，那么我们不希望给这个接口引入延迟效应。延迟效应描述的是接口的反应时间。例如：从用户按下Enter键提交一个信息这个点开始，直到该信息在聊天窗口中显示出来的时间间隔。即使少于一秒的时间延迟客观上可能是一小会儿等待，但是它对于用户来说却好像是非常的漫长而且令人讨厌。因此，输入的信息必须马上就显示出来（或者至少要尽可能的快）。HTTP是一个通用的协议，但是它不允许没有重新加载一整套文档就马上刷新网页。当然，也有多成分文档和自动刷新，但是这些选项在网页再次加载时都引入一个非常令人讨厌的跳动，它需要一个输出的数据库缓冲区，而且由于不断的从服务器传来的重新连接和数据而造成了延迟。

一个解决办法就是“分流 HTML”，它在任何地方都不被正式支持，但是仍然能起作用：这个接口的脚本程序只是在一个不停的循环中输出等待状态，而且不中止当前浏览器正在接收的HTML页面。当有些信息必须发送给用户时，它就被打印出来，马上在服务器的缓冲区刷新。通过这种方式，浏览器不断的提交和显示最新的数据。然而这种方法存在的一个问题就是没有复

杂的HTML实体能够凭空移交。例如：你不能一个一个的输出列表行，因为浏览器要求完全提供列表的所有行和列，以便于决定最后列表的大小。只要你限制自己一个接一个的输出文本行，而且只有在你能够一下子把它们全都打印出来的时候使用列表，所有的事情才都会很顺利。

像“分流HTML”这样的古怪名称是你必须知道的通用技巧。请一定经常关注这些东西。

分流HTML也有一些看起来像缺点又像优点的执行过程：既然用户的连接一直是开的，那就必须用一个服务器程序来处理它。这就意味着每个用户需要至少一个只为自己运行的 Web服务进程。其优点就是不会有过量的访问。通常，当一个用户需要一个文档的时候，一个新的进程就必须被开辟；产生那个文档的脚本程序必须被加载、分析并执行；最后这个数据还必须被发送出去。然而既然这个进程现在仍然留在内存中，开辟进程、加载脚本程序和解释对每个用户都必须做一次。对于那些每秒钟就有成百上千的用户访问的站点来讲，这有说不尽的好处。然而，每个进程现在都常驻在内存中，它们需要占用 RAM空间——在装有Linux、Apache和PHP 4.0的Intel x86系统上，这些进程每个都要占到2MB大小的内存。结果是运行着的一个只有很小容量内存的小型服务器将很快开始远离正常的内存与映像间的数据交换——这意味着死机。

注意 映像内存就是一种虚拟内存，虚拟内存能够扩展内存——计算机上物理内存的容量。映像内存被存储在硬盘上，其速度是非常的慢。当物理内存都用光时，现代的操作系统就开始在缓慢的映像内存中分配新的内存空间。如果一个聊天服务器在同一时间被许多用户访问，耗掉所有的物理内存并开始使用映像内存，这时操作系统必须不断的在部分物理内存和映像内存之间交换数据（因为程序不能在映像内存中执行），这就造成了一个死循环：操作系统通知在映像内存中的一个进程需要运行，并把它加载到物理内存中，但是又要把另外一个运行着的进程从物理内存中放到映像内存中。它在物理内存中运行这个进程的时候，又发现前面的进程（现在驻留在映像内存中）需要被运行，于是它又把它加载到物理内存中，如此不断的循环。你可以用这种方法很快地破坏一个服务器，强迫它重起并切断网络。顺便说一下，这也是一个很常用的“拒绝服务”的攻击，和“雅虎”受的攻击相似，其他的一些情况在今年的早些时候也都出现了。

你已经考虑过主流进程的相关结果吗？如果没有的话，下次请一定要考虑。每次都要对所有的情况进行完整的估计。

3.5.3 上游信息交流

上游信息交流（意思是接收用户的输入然后把它发送到网络中）是下一阶段要考虑的。

这是最难懂的部分：我们不能只通过所有的进程发送数据到 IRC网络中去。为什么不能呢？因为IRC是一个对各种状态非常敏感的协议，信息交流被限制在一个特定的用户连接上。 PHP不允许从其他进程中接过外部网络接口。因而，这个也处理下游信息交流的主进程（这个进程作为IRC的用户）独立于所有其他进程而运行。问题现在就是我们如何才能打开传递数据给主用户之门。

你将如何去实现上游信息交换？至少要做一个理论处理方法。把数据流写到草稿上。

如果你已经这样做了，那至少要写下三个运行时的数据交换的可能性。

下游进程必须一直运行，不能中断。我们不能只用“ POST ”或者是“ GET ”重新调用它们来传递数据。既然这样，那就意味着要调用另外一个进程，需要重新登录，重新安装等等。使用该处理方法的结果将是持续的登录 /退出序列将会在一次聊天中被不断的打断。而且它还会造成数据丢失，因为在退出和再登录的这段时间中，可能有许多信息已被传送了（这对于刚登录的用户来说是看不见的）。

聊天可以基于一个单独的一直都保持在线的“ 磁体 ”，可以把所有用户的所有信息都记录到数据库中。这个用户接口接下来只需要从这个数据库中释放所有有意义的数据就可以了。然而有两个与该可能性相抵触的问题： a）：聊天程序主要由数据库支持（这是我们应该避免的）；b）：它不能让其他IRC用户看见这个用户，因为“ 磁体 ”是这个网络中唯一的“ 真实 ”用户。这将使得IRC网络的使用变得滑稽可笑。

因此，我们至少需要两个独立的进程：一个处理不允许干扰的 IRC信息交流，另一个从用户那里接收输入信息。某种特定的容器必须使用在这两个进程之间的接口上，图 3-7说明了这个问题。

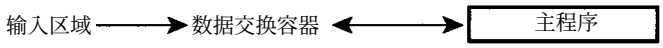

图3-7　上游信息交换

这种情况可以用一次汽车赛来比拟。在赛道上飞驰的驾驶员就好比是“ 主用户 ”，后排竞赛团队就好像是用户输入区域。这个驾驶员被限制在他所在的赛道中：他不能离开赛道去看发生了什么事情。无论什么时候竞赛团队指示他要为后排的人停一下，他们是在和驾驶员“ 接口 ”—— 给他一个信号，要他下一圈停下来。

所做的事情就是（把信息交流放到一边），每次当驾驶员路过终点线的时候发送信号。这个信号就像驾驶员的“ 接口 ”。基本上这也就是我们需要做的事情 —— 给我们主进程的信号。既然主进程是基于事件的，我们要不断的获取应用程序的控制权，做我们希望做的事情。这就是说我们可以安装一个处理程序，它不断的从外部寻找信号。这个方法叫做轮流检测，它周期性地停下来检查输入数据。它将是 phpIRC的优先选择的方法。 phpIRC具有空闲的回调特性，这个特性是在phpIRC没有事情可做的时候才调用的，它仅仅是在等待网络传来消息。给这个事件指定一个处理程序可以使我们能够监视信号。现在，我们怎样才能给一些事件发信号呢？这实际上非常简单，使用下面方法之一就可以了：

- 在数据库中设置一个标志。
- 在文件系统中创建一个锁定文件。
- 使用信号量。
- 在共享的存储器中设置一个标志。

这基本上这就是我们使用 PHP所拥有的方法，目的是“ 留下一则消息 ”。

下面部分描述了每一种方法。

在这里，进程的管道不能用于内部进程的信息交流，因为一个管道需要两个进程同

时运行。我们的情况就是需要能连接一个运行的进程和另一个短程进程的接口。

注意 当然有更多更巧妙的方法存在，比如在不同的进程之间发送电子邮件。我们已经看到有人这么干了。但是这里不选择这个方法，因为缺点应该让读者清楚地了解。

1. 在数据库中设置一个标志

在数据库中设置一个标志可能就是PHP用户使用的de facto的标准方法：连接到一个数据库，然后在数据库中留下一些数据，将其留给其他的进程去处理。这个方法非常容易实现，而且在所有的系统中都是可利用的，但是它也有一个缺点。你可以说出缺点是什么吗？

这个缺点不是来自于数据库的填充器（插入用户信息的进程），而是来自于数据库的读取器（从数据库中取出所有用户信息的主进程）。为了实现一个很好的“聊天感觉”，需要尽量的减少延迟——而且这样也是一个非常好的反应时间。反应时间对于基于互联网的聊天来说是非常关键的，因为这恰好就是用户真正的感觉是自己加入到这个行动当中去了。当发送信息的速度变得越来越慢，用户很快就变得非常沮丧，马上就会退出。我们的测试显示多于一秒钟的延迟就非常地过分了。为了能够保证低于这个值，主进程从数据库读取信息的查询时间必须非常短。phpChat缺省值是0.5秒（在一秒钟之内检查两次）。现在，只要有许多用户需要聊天系统来处理，那么数据库就非常的忙，这样就占用越来越多的资源。在每秒钟 40～50次的查询条件下，我们的测试服务器花掉了三分之一的过程时间仅仅是用来执行数据库查询。就算是这能作为数据库系统的基准（它应该能够处理更多的查询），一些优化选择很显然也是必要的，况且这非最佳设置。

2. 创建文件锁

我们的下一个想法就是当处理内部过程信息交换的时候，如果数据库占用了太多的内存，一个文件系统可能还需要更加有效率的工作。

但是很明显文件系统会在内存的竞争中失败。同样的，填充器并不是问题——文件锁的创建就会很轻松的解决这个问题。然而为了侦测文件锁是否已经被安装了，必须做很多关于clearstatcache（）的调用，这样是为了正确的侦测文件锁是否已经被删掉了或者还依然存在。clearstatcache（）对系统性能有如此强烈的效果以至于我们不需要再尽力去研究这一项了。它的聊天程序占用的系统资源只占使用数据库支持的方法所占用的四分之一。

创建我们自己的标准。做一个脚本程序，使它能够高频率的存取数据库和文件系统。记录下运行结果，然后比较它们。当评估数据交换方法的时候，这通常是一个非常好的办法——不要相信理论上关于系统可以胜任的那些话。在实际生活中，许多事情都不一样了。

3. 使用信号量

当然，前述方法不佳的原因可以很容易找到。

那么原因是什么呢？试着把它们找出来然后记录下来。试着找出关键点——当以后必须作出优化选择的时候它们也是非常关键的。“一条链子只和它最脆弱的连接点一样长，”同样地，软件也只和它内部最慢的循环一样快。这个寻找瓶颈的过程就叫做剖视，它是非常重要的。

当使用一个数据库时，瓶颈就是数据库：调用数据库、执行查询、取出结果和下一步要干什么（叫做内部操作）所耗费的时间和我们取得结果的时间相比是很长的。换句话说，我们现在用一个大型的为复杂数据的存储而设计的软件系统去交换简单的布尔值 —— 如果说数据库的设计中还有没有涉及到的东西的话，那就是布尔值。毫无疑问它执行起来并不理想。它的瓶颈就是内部操作，设置和取消初始化所需的时间。

文件系统运行得不好是因为它不是为这个用途而设计的，也因为其他的一些限制： PHP没有包含理想的文件系统存取方法。决定一个文件存在的因素是需要不断的使缓冲区失效和重新分配缓冲区。这使得我们再次在琐碎的任务上使用了大量的内部操作。

那么为什么不使用完全不同的东西呢？我们肯定不是第一个处理内部过程信息交流的人。其他人肯定已经找到了好的解决方案。于是我们实现了另外一个可能的解决方案：信号量。

信号量（Semaphores）能够很完好地达到我们的要求：它们像信号一样工作。信号量是存储在共享内存中的计数器。你可以获得一个信号量并且增加它的计数器的值。你也可以释放这个信号量并且减少其计数器的值。另外，也有等待某个信号量空闲的可能性，这意味着计数器的值将重新回到零。然而这个选择也有缺点：信号量倾向于锁定资源，创建某种类型的有时序安排的机制，允许许多进程等待某个装置上可利用的时间，或者相似的东西。无论什么时候你等待一个信号量变得空闲，这个等待的进程就被设置为睡眠状态，不能去做其他的事情。如果一个主进程正在等待用户输入区域发送一个新信息，那么它将进入睡眠状态，不能够处理输入的网络信息流。

然而，永不放弃。因为人们已经获得其他的解决办法。

3.5.4　在共享的存储器中设置一个标志

共享存储器（Shared memory）和信号标志相似，但是更加通用。共享存储器是系统中每一个进程都可以获得的存储器。多任务系统通常是这样设计的：因为安全方面的原因，每个进程都是彻底与其他进程相独立地运行。不同的进程可以通过设置和连接特殊的内存块来共享数据，也就是共享数据块。这些块可以包含变量（或者其他任何类型的数据，但是 PHP只支持确定变量的存储）。

这些恰好就是我们所希望的：能够在内存中某个地方存储布尔值而该内存单元可被每一个进程访问。共享存储器只在内存中起作用，它非常的快，几乎不需要内部操作。通过这一点，每一个聊天进程在共享存储器中寻找自己的变量，无论什么时候它找到用户输入区域设置的变量，它就指定一个数据库的查询。

为什么在结尾处数据交换还是基于数据库的呢？试着找出一些答案。

数据库还是为了某个主要的原因而使用的。共享存储器不被 PHP的缺省设置所支持。你只需要在PHP中编辑支持它就可以了。然而，许多能够访问装备有 PHP服务器的人没有重新编辑 PHP的权限，因为他们只是能够使用服务器的空间，所以还没有足够的权限，或者另外可能的原因是PHP的某种特定设置。把数据库留作最终的数据交换的路径，这样把共享存储器作为一个可选的优化选择加以利用。不能利用它的人只有使其无效，但是仍然拥有一个完全起作用的聊天服务程序版本 —— 运行起来性能较为差一点，但是确实能够运行。

当创建一个为广泛传播而设计的应用程序时，要时时提醒自己并非每个人都有和你相同的设置 —— 可能也没有重新创建你的非常特殊设置的可能性。虽然PHP99%是不依赖于系统的，但是仍然有一些是依赖于系统的。仔细考虑是否应实施一定的情况处理，如果处理不好，可能会损失一大批用户。

3.5.5 用户接口

现在我们把带有数据交换的所有棘手的部分明朗化，事实上提供给用户的 HTML接口是非常小的。我们知道如何从用户那里接受输入，如何处理网络信息交流。最后一个问题就是通过一个方便的方式为用户包装由此产生的输出。 HTML只提供一种在浏览器窗口中独立使用两个不同视窗的运作的方式：框架设置。典型的接口是由这些组成的：用户输入区域、聊天输出区域、一个显示在其他的参与同一个聊天室人的昵称名清单（或者只是一个标记清单）和动作按钮，动作按钮允许聊天时单击一次就能实现一个动作，例如更改昵称、加入、离开、退出等等。这些动作可以都用单独一个进程来处理，这个进程的输出将被集成到一个框架设置中。

负责聊天输出流的主进程在数据库中将保持状态信息的更新，使所有的其他接口组件都可以用合适的方式存入、取出和显示（参见图 3-8）。

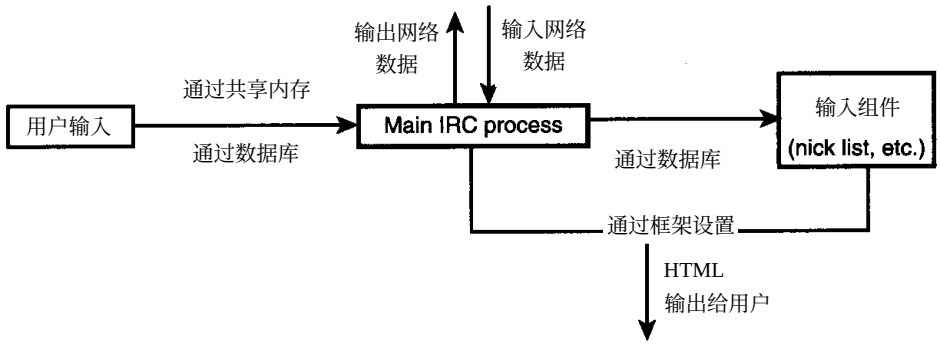

图3-8 最后的应用程序布局

3.5.6 开发者的接口

开发者的接口？这个接口对聊天程序要做什么？又期望它怎样工作呢？典型的，大多数应用程序都苦于不能“ 固化 ”。固化的意思是说要么彻底的不能修改，要么不能被陌生的开发者修改。就最终面向用户的软件来说（例如桌面环境，像 Windows、KDE、MacOS等等），几乎没有人能够找到理想的解决方案。和聊天系统相似，大多数下载它的人都说：“ 嗨，真不错，它们又少这又少那。” 或者是：“ 很酷，但是我却不喜欢它们做这做那的方式 ”。

没有一个容易、清楚的暴露方式提供给使用他们的用户，大多数应用程序最终的结果是成为垃圾。很多人甚至不会使用不是他们自己开发的程序 —— 如果这种简单的做法没有让他们受到沉重的打击的话。

这就意味着要为聊天应用程序一致地强制代码和接口布局的独立性（给 HTML开发者留下一

个接口），而且要一致地强制实现数据处理步骤的独立性，我们需要创建一个固化的应用程序核心（应用程序中没有人需要改变的部分），这个核心可作为一套分布式插件的接口（这是应用程序中大多数人都想改变的部分）。

再次思考这些强制的重要性。你愿意一个应用程序像这样被设计出来吗？你需要它吗？思考一下怎样才能认识其重要性。

3.5.7　HTML开发者的接口

就HTML的接口而言，代码和布局的抽象化是通过使用模板实现的。这是把一个应用程序设计为你需要的最简单的办法，同时它也是最完美的。在几秒钟之内，你可以改变其外观和感觉性能——不用修改一行程序。每一个有基本HTML知识的人都可以彻底地重新构建一个能够让应用程序把自己显示给用户的方法。因为这种方法在本书的其他地方讨论过，所以我们在这里将不再做深入的讨论。关于使用模板的详细信息，请阅读第 5章“ 基本网络应用策略 ”。

3.5.8　代码开发者的接口

给其他的开发者提供一个接口通常伴随着 API（应用程序接口）这个术语。 API一般是由库提供的（例如phpIRC），而不是由完整的应用程序提供的。但是具有能被程序员扩展功能的应用程序比只能按它原有状况使用的应用程序要成功得多。当然，就 PHP应用程序而言，任何人都可以修改源程序，但是很多人是不会分析一个复杂的系统然后进行所需的修改的。因而，应用程序本身也需要透露某些可以被扩展的方法。

注意　我们在这里要区别应用程序和库。库是由应用程序使用的，但不能独立的运行，它一般比应用程序更容易扩展。应用程序由一个完全封闭的系统组成。

试着找出通用应用程序如何才能被扩展。例如：对于你偏爱的处理文本的工具来说，你要看一看开发者是否提供了扩展工具功能的能力。

两个主要的扩展应用程序的可能性已经被开发出来：要么应用程序提供脚本程序功能（与宏相似），要么应用程序可以使用插件。对于 PHP来说，在部分系统运行的关键时候实现一个脚本语言……我们不必更深入的考虑这个问题了。紧接的是，创建一个羽翼丰满的分析程序的方法，其复杂性是难以言尽的。但是用插件实现就容易多了，也有很多优点。一个插件就是一段非常小的代码，它能够把自己注册到应用程序中，然后从中捕捉事件，取得访问内部数据的权限等等。天衣无缝的集成到主系统中时，插件始终保持是分离的文件，它可以单独的分离和传播。它们可以不需修改一行代码就连接到系统当中，这使得一个完全不了解 PHP的系统管理员可以使用其他的代码扩展应用程序。通过图 3-9可以确切的认识这一点。

图3-9　有插件的聊天系统

设计你自己的插件系统，至少在理论上将它构思出来。创建一个小程序，使它能够给自己注册插件，并且能够执行这些插件。

开始的时候，phpChat包括一个包含文件，这个文件依次包含了全部所需的插件。清单 3-3显示了这些包含文件是如何工作的：

清单3-3 插件包含文件

```
////////////////////////////////////////////////////////////////////////////
//
// Plug-in Integrator
//
////////////////////////////////////////////////////////////////////////////

include("chat_plugin_out_htmlspecialchars.php3");

include("chat_plugin_out_link_transform.php3");

include("chat_plugin_out_colorcodes.php3");

include("chat_plugin_clock.php3");

include("chat_plugin_cmd_basic.php3");
include("chat_plugin_out_basic.php3");

////////////////////////////////////////////////////////////////////////////
```

每一个插件都用同样的方式建立：由主要部分和时间部分组成。主要部分调用 phpChat的两个函数，这两个函数的名称如下：chat_register_plugin_init（）和chat_register_plugin_deinit（）。每个函数都将另一个函数的名字作为一个参数，这被称为插件程序的初始化和反初始化。

phpChat把这些函数名称添加到一个内部列表中。聊天程序初始化的时候，只要 phpChat被完全安装好，它就遍历初始化列表，然后自己调用每一个注册插件的初始化函数。相似的，当关机时，它遍历反初始化列表。这个方法允许给插件发送信号，激活和关闭它们。

为了在应用程序中有效， phpChat提供了一套事件，每一个插件可以把自己连接到这些事件中去。在插件初始化过程中，每一个插件都告诉 phpChat去发送一套所需的事件。事件可能包括聊天程序是空闲的、用户提交一个新的信息、用户在昵称列表中单击一个昵称名，一个从网络传来的输入信息等等。

运行的时候，插件可以截取这些事件并完成一定的任务。例如：时钟插件把自己注册到“空闲”事件里面，不断检查当前系统的时间。经过预定义的一段时间之后，它把这个当前的系统时间通知给用户。

对大多数用户来说， phpChat也传递参数（例如输入的文本信息），这就允许更改插件。例如：清单3-3中的插件清单中包含了以下插件： htmlspecialchars和link_transform。这些插件改变信息的输出， htmlspecialchars把对htmlspecialchars（）的调用应用到所有打印的文本（这是出于安全方面的考虑，没有人可以把有害的 HTML代码插入到这个聊天程序中）和侦测所有 URL地址和电子邮件地址的连接转换器中去，并且分别使用 或mailto:预定义它们。

于是用户只需要在聊天窗口中单击连接就可以了（参看图 3-10）。

图3-10　工作中的插件

正如你所看到的那样，插件提供了一个非常可靠的扩展复杂系统的方法。因此， phpChat把大多数内部程序都抽象成插件。完整的命令行解释器也成了插件，一整套文本格式和打印处理过程也都使用插件。这就是说只有一个固化的内核不需要修改，因为内核中没有东西需要修改——其余部分都可以自由的修改、扩展、甚至删除，而对系统性能和操作性没有任何影响。你曾见过应用程序不怕它的文件被删除吗？使用这种方法，你就可以做到上点了。

插件的应用很广泛，并不只是用在聊天程序上。例如：你也可以建立一个入口站点，由一般的新闻页、电子邮件接口等组成。你可以设计一个“站点内核”，处理所有的基本问题，例如提供网页布局，数据库终端等等。基于这个站点内核，你可以创建一些插件来显示新闻、发送和收取电子邮件，甚至提供不同的登录方法等等。虽然使用插件可能有点困难，但是我们还是鼓励你能够创建一个使用插件的应用程序作为练习。这是值得一做的。

清单3-4显示的是一个插件模板，实现了“虚拟”的能够插入新插件的代码。

清单3-4　一个插件模板

```
<?

//
// Use these variables to tell the plug-in installer how you named your
// initialization and deinitialization functions. This is done to eliminate
// the need for changing the installer code, which would ask for errors.
//
$plugin_init_function = "myplugin_init";
```

```
$plugin_deinit_function = "myplugin_deinit";
//
//
//

////////////////////////////////////////////////////////////////////////////
//
// myplugin_idle_callback(int code, mixed parameter) - example callback
//
////////////////////////////////////////////////////////////////////////////
//

// This is an example for a callback function. See below on how to register
// and remove it from the call chain.
//
// $code specifies the reason for invocation, $parameter contains all callback
// information.
//
// The return value should always consist of a modified or unmodified version
// of the input parameter $parameter. The return value is used as input
// parameter for the next callback. This allows for multi-stage message
// processing and such.
//
////////////////////////////////////////////////////////////////////////////

function myplugin_idle_callback($code, $parameter)
{

    return($parameter);

}

////////////////////////////////////////////////////////////////////////////
//
// myplugin_init() - initializes this plug-in
//
////////////////////////////////////////////////////////////////////////////
//
// Put all your initialization code in here. This code will be called as soon
// as the main bot is all set up with connecting and callback installation;
// thus, you can rely on a safe environment.
//
// Although the return value is currently not used, "0" should indicate
// initialization failure and "1" initialization success. This might be used
// later on to enable plug-ins to stop the current chat session right after
// login.
//
////////////////////////////////////////////////////////////////////////////
//
// Return value:
//    0 - error
//    1 - success
//
////////////////////////////////////////////////////////////////////////////

function myplugin_init()
{
```

```
    // register callbacks here
    chat_register_callback(CHATCB_IDLE, "myplugin_idle_callback");

    return(1);

}

////////////////////////////////////////////////////////////////////////////
//
// myplugin_deinit() - deinitializes this plug-in
//
////////////////////////////////////////////////////////////////////////////
//
// All deinitialization code should go here. This function is called before
// the bot goes down; thus, all network connections are still active.
//
// Although the return value is currently not used, "0" should indicate
// deinitialization failure and "1" deinitialization success. This might be
// used later on to force delayed shutdowns.
//
////////////////////////////////////////////////////////////////////////////

function myplugin_deinit()
{

    // remove callbacks here
    chat_remove_callback(CHATCB_IDLE, "myplugin_idle_callback");

    return(1);

}

////////////////////////////////////////////////////////////////////////////
//
// NOTE: DO NOT CHANGE ANYTHING BELOW THIS POINT!
//
////////////////////////////////////////////////////////////////////////////

// installer code starts here

// register initialization function
chat_register_plugin_init($plugin_init_function);

// register deinitialization function
chat_register_plugin_deinit($plugin_deinit_function);

// installer code done

////////////////////////////////////////////////////////////////////////////

?>
```

主代码为这个插件注册了初始化和反初始化的例行程序。插件的初始化程序随后安装一些插件想截取的回收信号，然后由反初始化程序删除这些回收信号。

3.6 管理和安全

如果一个系统不能被管理，它就不是一个好系统。以前的“网络礼节”将礼貌作为一种荣誉，而且人们都把自己和团体紧密结合起来，这样的日子已经远去了。现在系统被黑客入侵、骚扰、或其他形式的攻击，不幸的是，大多数系统都停留在初级等级上。在一个应用程序或者网络系统中，对于挖掘安全漏洞等缺陷是没有什么好说的。不断探索才是他们值得认真考虑的，然而有很多人却只是为了寻找“乐趣”才这么做。这需要一个外部接口程序，它独立于主系统运行，能够完全控制应用程序的数据和用户。就聊天系统而言我们要能够踢出用户、使用户的信息适当、使聊天室安全等。

注意 并不是所有列出的特性都能在CD-ROM的代码中实现基础的管理系统是完整的操作函数。但是我们希望你能够用自己觉得合适的特性练习和扩展基础代码。如果你还没有对大程序进行过扩展工作，那么现在我们就鼓励你获得这样的一个实战经验。

这里聊天程序的一个问题就是：我们如何定格管理程序？有如下方法：

- 网络等级：我们可以过滤连接的用户。
- PHP等级：我们可以阻止用户登录到聊天室中。
- 数据库等级：我们可以丢弃用户数据库中的信息。
- IRC等级：我们可以使用IRC的本地网络管理特性。

3.6.1 网络等级

网络等级上的安全模式只允许两种可能性：允许或不允许一个连接。这可以通过使用防火墙或者是其他的IP检测来识别。这种方法的功能很有限，而且复杂，不安全，所以不是我们所期望的。

3.6.2 PHP/Web服务器等级

Web服务等级上的安全模式允许用户连接到服务器，但是可以通过密码来限制登录（或者其他不同的方法）。它基本上解决了是否允许一个连接的问题，但也不是非常令人满意。

但是这种方法可以仿效用户禁令。IRC通用的禁令也就是K-lines和G-lines（局部和全局用户禁令），不能用在基于Web的聊天系统上。唯一可被禁止的地址是Web服务器的地址，它能够彻底地禁止整个网络接口。为了能够过滤特殊用户，应该在PHP等级上评估连接。

3.6.3 数据库等级

数据库等级是一种完全不同的解决方案。用户可以登录和聊天，但是他们的通话和信息在数据库中过滤。或者是一个外部的工具或者是聊天的源代码检查是否允许用户“说话”和做一些事情，基于这些推断，决定是否允许他或者她的信息插入到数据库中。但是这个策略要求主聊天代码非常严密而且具有综合性，它也缺乏韧性（有一些笨拙），不易实现。

3.6.4　IRC等级

IRC提供了建立在服务器代码中的局部管理特性和网络协议（我们希望你能够阅读 RFC，熟悉这些可能性）。一般的用户都可以进行管理。有三个 IRC等级是可以利用的：

• 通道处理程序。这些处理程序有管理通道的控制权。它们可以踢出用户、不让他们聊天、禁止他们登录等等（这个等级对所有用户都是开放的）。

• IRC处理程序。这些处理程序有管理网络的控制权（但控制不了通道）。它们可以让用户断线、禁止他们登录、建立网络连接等等（这个等级只对特殊用户开放）。

• 服务。服务能够管理通道但不能管理网络，也不能由一般用户执行。它们还需要特殊的登录程序（这个等级只对特殊用户开放）。

可以看到，IRC等级的管理可以通过用户独立于主聊天系统运行来实现。拥有 IRC处理程序和通道处理程序的独立用户状态是我们所需管理系统应具特性的完美结合。基本上，最初只有IRC处理程序状态是必需的。因此，只要管理用户获得 IRC处理程序状态，就可以通过从通道中切断所有的用户来获得任何地方的通道处理程序状态。这不算是一个非常好的方法，但比修改IRC服务代码以使IRC处理程序具有与通道处理程序相同的权限更有效，也更通用。

3.7　执行

聊天管理的执行被设计得与主聊天脚本程序非常相似。一个初始程序启动时带有 phpIRC的帮助，它记录到IRC中，而且试着把自己作为一个 IRC处理程序注册。然后它等待从 Web接口传来的进一步的命令。这些命令与由数据库支持的 RPC中的命令相似（远程过程调用）。初始程序将不断的查询一个数据库中包含输入命令的列表。这些命令被 Web接口放在数据库中，由一个函数名、一个通话 ID和一个参数数组组成。一旦这个初始程序在数据库中找到一个新的命令时，它就执行这个命令，然后把这个命令的执行结果写入到通话 ID的输出列表中。因而， Web接口只需要用一个自产生的通话 ID写入一个命令，然后等待具有和通话 ID相同ID的结果数据在输出列表中弹出即可（如图3-11所示）。

这种方法允许对初始程序进行弹性远程控制。

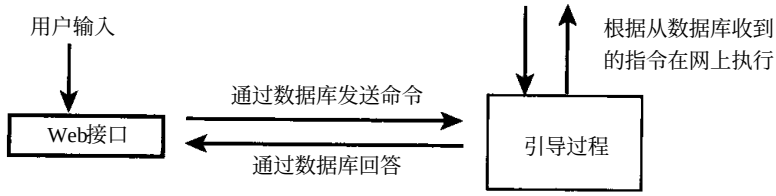

图3-11　由数据库支持的RPC控制管理初始程序

3.8　小结

这一章中，你学到了一个实际的例子，关于如何计划一个开发项目。我们略述一下典型的开发阶段：

- 分析要求。
- 选择合适的技术。
- 定义接口和API。
- 执行。

在我们的讲解下你了解了整个开发过程。从对于大多数软件项目来说都是可行的例子中我们得到了一些结论。有了这些背景，我们就可以准备本书下一部分的学习了。在下一章中我们将介绍一些网络应用程序的重要概念。