第9章
Database System Concepts, 6E

应用设计和开发

实际上，对数据库的所有应用都发生在应用程序内。相应地，几乎所有的用户与数据库之间的交互都是通过应用程序间接发生的。因此毫不奇怪，数据库系统长期以来都支持诸如表单和 GUI 构建器等工具，用于快速开发与用户交互的应用程序。近几年来，Web 已成为最广泛使用的数据库用户界面。

在本章中，我们学习建立应用程序所需要的工具和技术，集中关注那些使用数据库存储数据的交互式应用程序。

在 9.1 节对应用程序和用户界面的介绍后，我们集中关注基于 Web 界面的应用的开发。首先 9.2 节对 Web 技术进行概述，然后 9.3 节讨论广泛用于构建 Web 应用的 Java Servlet 技术。9.4 节简要介绍 Web 应用程序体系架构。9.5 节讨论快速应用程序开发工具，而 9.6 节介绍构建大型 Web 应用程序中的性能问题。9.7 节讨论应用程序安全的问题。用 9.8 节结束本章，其中对加密及它在应用程序中的使用进行了介绍。

9.1 应用程序和用户界面

尽管许多人和数据库打交道，但很少有人用查询语言直接跟数据库系统交互。用户与数据库交互时最常用的方法是通过一个应用程序，它在前端提供用户界面，并在后端提供数据库接口。这种应用程序通常是通过一个基于表格的界面从用户得到输入，然后根据用户输入或将数据输入数据库或从数据库中抽取信息，并产生输出展示给用户。

375 举个应用程序的例子，考虑一个大学注册系统。和其他这种应用程序类似，注册系统首先需要你标识并认证你自己，通常使用用户名和密码。然后，应用程序使用你的身份从数据库中抽取信息，比如你的姓名和你已注册的课程，并展示该信息。应用程序提供了若干接口，让你可以注册课程及查询多种其他的信息，比如课程信息和教师信息。组织机构使用这种应用程序以自动完成多种任务，例如出售、购买、会计和薪资、人力资源管理以及库存管理等。

应用程序有可能正在使用，即使不能明显地看出来它们正在使用。例如，一个新闻网站可以提供对个人用户透明定制的页面，即使该用户在和网站交互时并没有显式填写任何表单。为了做到这点，它实际上运行了一个对每个用户生成定制页面的应用程序，例如，定制可以基于用户浏览文章的历史。

一个典型的应用程序包括一个处理用户界面的前端部分、一个和数据库通信的后端部分，以及一个包含“业务逻辑”的中间层，即执行特定的信息请求或更新的代码，强制执行诸如为了执行给定任务应该采取什么行动，或者谁可以执行什么任务的业务规则。

如图 9-1 所示，应用程序体系架构随时间而演变。诸如航班预订的应用程序自 20 世纪 60 年代以来一直在使用。在计算机应用早期，应用程序运行在一台大的“主机”计算机上，用户通过终端与应用程序交互，其中一些终端甚至支持表格。

随着个人计算机使用的普及，许多机构对内部应用程序使用了一个不同的体系架构，其中应用程序在用户的计算机上运行并访问中央数据库。这个架构通常称作“客户 - 服务器”架构，它允许创建强大的图形用户界面，这是早期基于终端的应用所不支持的。然而，软件必须安装在每个用户的机器上以运行应用程序，这就使得诸如升级的任务比较困难。即使在客户 - 服务器架构流行的个人计算机时
376 代，诸如航班预订这种从大量地理上分布的位置使用的应用程序仍旧选择主机架构。

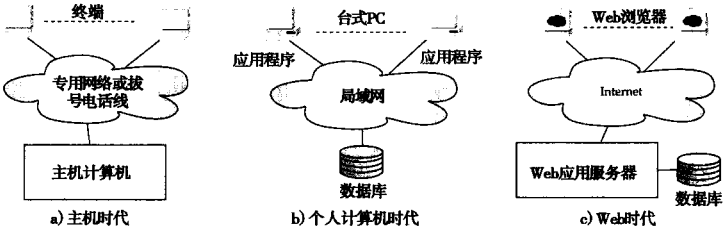

图 9-1 不同时期的应用架构

在过去的 15 年中，Web 浏览器已成为数据库应用程序的通用前端，它通过互联网连接后端。浏览器使用一种标准的语法，**超文本标记语言**(HyperText Markup Language，HTML)标准，它既支持信息的格式化显示，又支持基于表格的界面的创建。HTML 标准是独立于操作系统或浏览器的，并且目前几乎每台计算机都安装了 Web 浏览器。因此一个基于 Web 的应用程序可被任何一台连接到互联网的计算机访问。

与客户 - 服务器架构不同，要使用基于 Web 的应用程序不需要在客户机上安装任何特定于应用的软件。然而，支持远超过普通 HTML 所能提供的功能的复杂用户界面在目前广泛使用，并且是用大多数 Web 浏览器都支持的脚本语言 JavaScript 构建的。与用 C 写的程序不同，JavaScript 程序可以运行在安全模式下，保证它们不会导致安全问题。JavaScript 程序被透明地下载到浏览器上，且不需要用户的计算机上任何显式的软件安装。

Web 浏览器提供用户交互前端的同时，应用程序构成了后端。通常，来自浏览器的请求发送到 Web 服务器，它依次执行应用程序以处理请求。有多种技术都可以用于创建运行在后端的应用程序，包括 Java servlet、Java Server Pages(JSP)、Active Server Pages(ASP)，或者脚本语言，比如 PHP、Perl 或 Python。

本章其余部分讲述如何构建这些应用。首先介绍用于构建 Web 界面的 Web 技术和工具及构建应用程序的技术，然后介绍应用架构，以及应用程序构建中的性能和安全问题。

9.2 Web 基础

在本节中，我们为了那些不熟悉 Web 底层技术的读者，回顾一下万维网背后的一些基本技术。

9.2.1 统一资源定位符

统一资源定位符(Uniform Resource Locator，URL)是 Web 上每个可访问文档在全球唯一的名字。URL 的一个例子如下：

```
http://www.acm.org/sigmod
```

URL 的第一部分表明文档如何被访问："http"表明文档将用**超文本传输协议**(HyperText Transfer Protocol，HTTP)访问，这是一个传输 HTML 文档的协议。第二部分给出一台具有 Web 服务器的机器的 377
名称。URL 的其余部分是文件在机器上的路径名，或者文档在机器中其他的唯一标识符。

URL 可以包含位于 Web 服务器上程序的标识符，以及传递给程序的参数。这样的 URL 的一个例子是：

```
http://www.google.com/search?q=silberschatz
```

它指出，www.google.com 服务器上的程序 search 应该执行，并带有参数 q = silberschatz。接收到这个 URL 请求时，Web 服务器使用给定的参数执行该程序。该程序返回一个 HTML 文档给 Web 服务器，它将该文档传送到前端。

9.2.2 超文本标记语言

图 9-2 是一个使用 HTML 格式表示的表的例子，而图 9-3 表示浏览器根据 HTML 表示的表而产生的显示图像。HTML 源文本显示了一些 HTML 标签。每个 HTML 页应包含在 html 标签内，该页的主体包

含在 body 标签中。表用 table 标签表示，其中包含的行用 tr 标签表示。表的表头行有用标签 th 说明的表单元，而普通行有用标签 td 说明的表单元。这里不再对标签进行更详细的介绍，关于 HTML 的更多信息，请参看文献注解中的参考文献。

图 9-4 展示了如何表示一个允许用户从菜单中选择人员类型(学生或教师)以及允许用户向文本框中输入一个数的 HTML 表单。图 9-5 展示了以上表单如何在 Web 浏览器中显示。表单中表示了两种接受输入的方式，但是 HTML 还支持几种其他的输入方式。form 标签的 action 属性表示当提交表单时(通过点击提交按键)，表单数据应发送到 URL PersonQuery(该 URL 是相对于页面的 URL)。Web 服务器被设置成当该 URL 被访问时，便调用相应的带有用户提供的参数值 persontype 和 name(在 select 和 input 域中指定)的应用程序。该应用程序生成一个 HTML 文档，该文档随即传送回去并显示给用户；我们将在本章的后面看到如何创建这种程序。

```
<html>
<body>
<table border>
<tr> <th>ID</th>      <th>Name</th>    <th>Department</th> </tr>
<tr> <td>00128</td> <td>Zhang</td>    <td>Comp. Sci.</td> </tr>
<tr> <td>12345</td> <td>Shankar</td> <td>Comp. Sci.</td> </tr>
<tr> <td>19991</td> <td>Brandt</td>   <td>History</td> </tr>
</table>
</body>
</html>
```

图 9-2 HTML 格式的表格数据

ID	Name	Department
00128	Zhang	Comp. Sci.
12345	Shankar	Comp. Sci.
19991	Brandt	History

图 9-3 图 9-2 中 HTML 源文本的显示

```
<html>
<body>
<form action="PersonQuery" method=get>
Search for:
<select name="persontype">
      <option value="student" selected>Student </option>
      <option value="instructor"> Instructor </option>
</select> <br>
Name: <input type=text size=20 name="name">
<input type=submit value="submit">
</form>
</body>
</html>
```

图 9-4 一个 HTML 表单

HTTP 定义了两种将用户在 Web 浏览器端输入的值发送到 Web 服务器的方式。get 方法将值编码为 URL 的一部分。例如，如果 Google 搜索页使用一个表单，包含一个名为 q 的 get 方法的输入参数，并且用户键入字符串“silberschatz”并提交该表单，浏览器将会向 Web 服务器请求如下的 URL：

Search for: Student
Name:
submit

图 9-5 图 9-4 中 HTML 源文本的显示

http://www.google.com/search?q=silberschatz

而 post 方法将会向 www.google.com 页面发送一个请求，并将参数的值作为 Web 服务器和浏览器间交换的 HTTP 协议的一部分发送。图 9-4 中的表单表示该表单使用 get 方法。

尽管 HTML 代码可以用普通的文本编辑器创建，但是存在若干种可以使用图形界面直接创建 HTML 文本的编辑器。这些编辑器允许从菜单中选择诸如表单、菜单和表这样的结构加入到 HTML 文档中，而不是手工输入生成这些结构的代码。

HTML 支持*样式表*(*stylesheet*)，它可以改变如何显示 HTML 格式化结构的默认定义，以及其他显示属性，如页面的背景颜色等。*层叠式样式表*(Cascading StyleSheet，CSS)标准允许同一个样式表用于多个 HTML 文档，使一个 Web 站点上的所有页面具有独特而统一的样式。

9.2.3 Web 服务器和会话

Web 服务器(Web Server)是运行于服务器上的程序，它接收来自于 Web 浏览器的请求并以 HTML 文档的形式返回结果。浏览器和 Web 服务器通过 HTTP 通信。Web 服务器提供了强大的功能，而不只是简单的文档传输。最重要的功能是具有带用户提供的参数执行程序，并以 HTML 文档形式传送回结果的能力。

所以，Web 服务器可以作为中介提供对多种信息服务的访问。一个新的服务可以通过创建并安装提供服务的应用程序而创建。**公共网关接口**(Common Gateway Interface，CGI)标准定义了 Web 服务器如何与应用程序通信。应用程序通常通过 ODBC、JDBC 或者其他协议与数据库服务器通信，以获取或存储数据。

图 9-6 显示了一个使用三层体系结构搭建的 Web 应用，包括一个 Web 服务器、一个应用服务器和一个数据库服务器。使用多级服务器增加了系统的开销；CGI 接口为每个请求都开启一个新的进程为之服务，这导致甚至更大的开销。

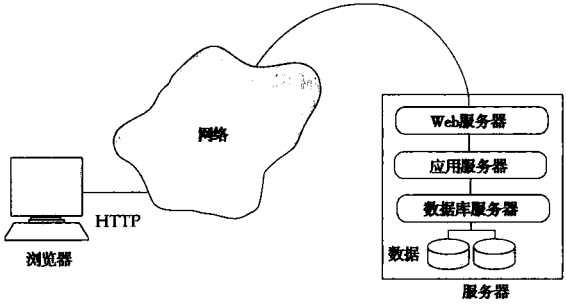

图 9-6 三层 Web 应用体系结构

因此目前大多数 Web 应用使用一种两层的 Web 应用体系结构，其中应用程序运行在 Web 服务器中，如图 9-7 所示。下面的章节将更加详细地研究基于两层体系结构的系统。

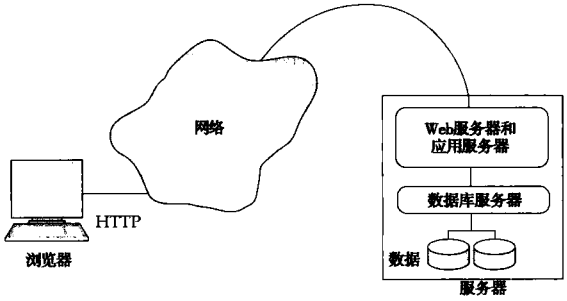

图 9-7 两层 Web 应用体系结构

在客户端与 Web 服务器之间不存在持续的连接；当 Web 服务器接收到一个请求时，临时创建一个连接以发送请求并接收来自 Web 服务器的响应。但是该连接可能会关闭，且下一个请求可以生成一个

新的连接。与此相反，当一个用户登录计算机，或是使用 ODBC 或 JDBC 连接到数据库时，会创建一个会话，且会话信息保留在服务器和客户端，直到会话结束——信息包括诸如该用户的用户标识符和用户已设置的会话参数等。HTTP 协议是**无连接**(connectionless)的一个重要原因在于，大多数计算机能同时容纳的连接数目是有限的。如果 Web 中大量的站点对单台服务器建立连接，就会超过限制，拒绝后续用户的服务。而使用无连接协议，当满足了请求时连接就可以马上断开，为其他请求 ⊖ 留出可用连接。

但是，大多数 Web 应用需要会话信息，以允许有意义的用户交互。例如，应用程序通常限制对信息的访问，且因此需要认证用户。每次会话应进行一次认证，会话中进一步的交互应不需重新认证。

380 | 381

尽管连接会关闭，但为了实现会话，需要在客户端存储额外的信息，并随会话中的每个请求返回；服务器使用这个信息来辨别出请求是用户会话的一部分。关于会话的额外信息同样必须在服务器端维护。

这种额外信息通常以**网络跟踪器**(cookie)的形式保存在客户端；简单来说，一个 cookie 是一小段包含标识信息的文本，并关联一个名字。例如，google. com 可能设置一个名为 prefs 的 cookie，它包含用户设置的偏好信息，比如语言偏好和每页显示的结果数目。对于每个搜索请求，google. com 都能从用户的浏览器中得到这个名为 prefs 的 cookie，然后根据指定的偏好显示结果。一个域(Web 站点)只能获取它自己设置的 cookie，而不能得到别的站点设置的 cookie，而且 cookie 名可以跨域复用。

为了跟踪用户会话的目的，应用程序可能会产生一个会话标识符(通常是一个当前没有用作会话标识符的随机数)，然后发送一个包含这个会话标识符的名为(比如)seessionid 的 cookie。该会话标识符也保存在服务器本地。当一个请求进来时，应用服务器向客户端请求名为 sessionid 的 cookie。如果客户端没有存储该 cookie，或者返回的值不是当前在服务器中记录的有效会话标识符，应用程序就认为该请求不是当前会话的一部分。如果 cookie 的值与一个存储的会话标识符匹配，则该请求就被识别为一个进行中的会话的一部分。

如果一个应用需要安全地鉴别用户，则它只能在对用户认证之后设置 cookie；例如，用户只有在提交了有效的用户名和密码之后才能通过认证。 ⊜

对于不需要高安全性的应用，比如公开的新闻站点，cookie 可以永久存储在浏览器和服务器中；它们识别出用户对同一个网站的后续访问，而不需要输入任何认证信息。对于要求更高安全性的应用，服务器可能会在有效期过后或用户注销时使会话无效(丢弃)。(通常用户通过点击一个注销按钮来注销，它会提交一个注销表单，其动作就是使当前会话失效。)使一个会话失效仅仅是在应用服务器中的活动会话列表里丢弃该会话标识符。

382

9.3 servlet 和 JSP

在两层 Web 体系结构中，应用程序作为 Web 服务器本身的一部分运行。实现这种体系结构的一个方法是将 Java 程序加载到 Web 服务器中。Java servlet 规格说明定义了一种 Web 服务器与应用程序间通信的应用程序编程接口。Java 中的 HttpServlet 类实现了 servlet API 规格要求；实现特定功能的 servlet 类被定义为这个类的子类。 ⊜ 通常 *servlet* 一词指实现了 servlet 接口的 Java 程序(和类)。图 9-8 是一个 servlet 的例子；我们简短地介绍它的一些细节。

当服务器启动或者服务器接收到远程的 HTTP 请求执行某个特定 servlet 时，servlet 的代码被加载到

⊖ 为了性能方面的原因，连接可能会在短时间内保持，以允许子请求复用该连接。然而，不能保证连接将会保持，因此在设计应用时应当假设一旦处理请求连接就有可能关闭。

⊜ 用户标识符可以存储在客户端，比如一个叫做 userid 的 cookie 中。这样的 cookie 可以用于低安全要求的应用中，比如免费的 Web 站点识别它们的用户。但是，对于要求更高级别安全性的应用，这样的方法就会造成安全隐患：cookie 中的值可能在浏览器中被用户恶意篡改，该用户就能够伪装成另一个用户。将 cookie(比如名为 sessionid)设置为一个随机产生的会话标识符(一个很大的数字空间)，可以使得一个用户冒充另一个用户的可能性变得极小。而一个顺序生成的会话标识符，则很容易冒充。

⊜ 虽然我们的例子中只使用了 HTTP，但是 servlet 接口也能支持非 HTTP 请求。

Web 服务器中。servlet 的任务就是处理这种可能涉及访问数据库撷取必要信息，并动态生成 HTML 页面返回给客户端浏览器的请求。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PersonQueryServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE> Query Result</TITLE></HEAD>");
        out.println("<BODY>");

        String persontype = request.getParameter("persontype");
        String number = request.getParameter("name");
        if(persontype.equals("student")) {
            ... 寻找具有指定姓名的学生的代码 ...
            ... 使用JDBC与数据库通信 ...
            out.println("<table BORDER COLS=3>");
            out.println(" <tr> <td>ID</td> <td>Name: </td>" +
                " <td>Department</td> </tr>");
            for(... each result ...){
                ... 获取 ID、name 和 dept_name
                ... 放入变量 ID、name 和 deptname
                out.println("<tr> <td>" + ID + "</td>" +
                    "<td>" + name + "</td>" +
                    "<td>" + deptname + "</td></tr>");
            };
            out.println("</table>");
        }
        else {
            ... 同上，但是是对于教师 ...
        }
        out.println("</BODY>");
        out.close();
    }
}
```

图 9-8 servlet 代码示例

9.3.1 一个 servlet 的例子

servlet 通常用于对 HTTP 请求动态生成响应。它们可以访问 HTML 表单提供的输入，执行“业务逻辑”以决定给出什么样的响应，然后生成 HTML 输出并发送回浏览器。

图 9-8 所示为实现图 9-4 中表单的 servlet 代码的例子。这个 servlet 叫做 PersonQueryServlet，同时表单指明了 action = “PersonQuery”。必须告知 Web 服务器这个 servlet 是用来处理 PersonQuery 请求的。该表单指定使用 HTTP 的 get 机制进行参数传递，这样 servlet 代码中定义的 doGet( ) 方法被调用。

每次请求都会生成一个新的线程，在线程中执行调用，因此多个请求就可以并行处理。任何来自 Web 网页上的表单菜单和输入域中的值，和 cookie 一样，由一个为请求创建的 HttpServletRequest 类对象传入，然后请求的应答由一个 HttpServletResponse 类对象返回。

该例子中的 doGet( ) 方法通过 request. getParamter( ) 抽取参数的 type 和 number 的值，并使用它们的值执行数据库查询。用于访问数据库以及从查询结果获取属性值的代码没有表示出来，有关如何使用 JDBC 访问数据库的细节请参考 5. 1. 1. 4 节。servlet 代码将查询结果输出到 HttpServletResponse 类的对象

response 中返回给请求者。将结果输出到 response 是通过首先从 response 中取得 PrintWriter 对象 out，然后向 out 以 HTML 格式输出结果而实现的。

9.3.2 servlet 会话

回想一下，浏览器和 Web 服务器之间的交互是无状态的。也就是说，每次浏览器向服务器发出一个请求，浏览器都要连接服务器，请求某些信息，然后从服务器断开连接。cookie 可以用来识别一个请求与前一个请求是来自于同一个浏览器会话。但是，cookie 构成一个低级别机制，程序员需要一个更好的抽象概念来处理会话。

383
384

servlet API 提供了一种跟踪会话并存储会话相关信息的方法。HttpServletRequest 类的 getSession(false)方法的调用搜取出发出请求的浏览器对应的 HttpSession 对象。参数值为 true 将表示当请求是新请求时，必须创建新的会话对象。

当 getSession()方法被调用时，服务器会先要求客户端返回具有指定名字的 cookie。如果客户没有该名字的 cookie，或者返回的值与任何进行中的会话都不匹配，则请求不是进行中的会话的一部分。在这种情况下，getSession()将返回一个空值，并且 servlet 会将用户指引到一个登录页面。

登录页面可以允许用户提供用户名和密码。登录页面对应的 servlet 可以验证密码与用户匹配(例如，通过在数据库中查找认证信息)。如果该用户正常通过认证，登录 servlet 将会执行 getSession(true)，返回一个新的会话对象。为了创建一个新的会话，Web 服务器内部执行以下任务：在客户的浏览器中设置一个 cookie(比如名为 sessionId)，用会话标识符作为它所关联的值，创建一个新的会话对象，并将会话标识符的值与会话对象关联。

servlet 代码还能够在 HttpSession 对象中存储和查找(属性名、值)对，以便在一个会话内的多个请求间维持状态。例如，在用户被认证并创建了会话对象之后，登录 servlet 可以通过在 getSession()所返回的会话对象上执行方法

```
session. setAttribute("userid", userid)
```

将用户的 user-id 存储为会话的一个参数；假定在 Java 变量 userid 中包含用户标识符。

如果请求属于一个进行中的会话，浏览器就会返回该 cookie 的值，且 getSession()会返回对应的会话对象。servlet 通过在上述所返回的会话对象上执行方法

```
session. getAttribute("userid")
```

就能够从会话对象中获取会话参数，比如 user-id。如果属性 userid 没有设置，该函数会返回一个空值，表明该客户端的用户还没有通过认证。

9.3.3 servlet 的生命周期

385

servlet 的生命周期由它部署于的 Web 服务器控制。当有客户端请求一个特定的 servlet 时，服务器首先检查是否存在该 servlet 的实例。如果不存在，Web 服务器就将 servlet 类加载到 Java 虚拟机(Java Virtual Machine，JVM)中，并创建该 servlet 类的一个实例。另外，服务器调用 init()方法将该 servlet 实例初始化。注意，每个 servlet 实例仅在加载的时候初始化一次。

在确定 servlet 实例的确存在之后，服务器调用 servlet 的 service()方法，并以一个 request 对象和一个 response 对象作为参数。在默认情况下，服务器创建一个新的线程以执行 service()方法；因此，对一个 servlet 的多个请求就能够并行处理，而不必等待上一个请求执行完。service()方法适当调用 doGet()或 doPost()方法。

当不再需要的时候，可以通过调用 destroy()方法停止一个 servlet。服务器可以设置为如果在一段时间内没有对某个 servlet 的请求，则自动停止这个 servlet；该时间段是服务器的一个参数，可以根据应用适当地设置。

9.3.4 servlet 支持

许多应用服务器都提供 servlet 的内嵌支持。最流行的应用服务器之一是 Apache Jakarta 项目中的 Tomcat 服务器。其他支持 servlet 的应用服务器包括 Glassfish、JBoss、BEA Weblogic 应用服务器、Oracle 应用服务器以及 IBM 的 WebSphere 应用服务器。

开发 servlet 应用程序最好的方式是利用一个 IDE，比如 Tomcat 或 Glassfish 服务器内嵌的 Eclipse 或者 NetBeans。

除了基本的 servlet 支持之外，应用服务器通常还提供多种有用的服务。它们允许应用程序部署或者停止，并提供监视应用服务器状态的功能，包括性能统计。如果一个 servlet 文件修改，某些应用服务器能够监测到，然后透明地重新编译并重新加载该 servlet。许多应用服务器还允许服务器在多台机器上并行运行以提高性能，并将请求分发至适当的机器上。许多应用服务器还支持 Java 2 Enterprise Edition(J2EE)平台，它针对多种任务都提供支持和 API，比如处理对象、跨多个应用服务器并行处理以及处理 XML 数据(XML 将在后面第 23 章介绍)。

9.3.5　服务器端脚本

用一种诸如 Java 或 C 的程序设计语言编写一段甚至很简单的 Web 应用程序也是很花时间的一项任务，它需要很多行代码以及熟悉该语言中错综复杂的事物的程序员。另一种方法，也就是**服务器端脚本**(server-side scripting)，为创建许多应用程序提供了一种简单得多的方式。脚本语言提供了可嵌入 HTML 文档中的结构。在服务器端脚本中，服务器会在传送 Web 页面之前执行嵌入在 HTML 页面内容中的脚本。每段脚本在执行时会生成加入到页面中的文本(或可能甚至从页面中删除内容)。脚本的源代码将从页面中删除，因此客户端可能根本没察觉页面原先含有任何代码。执行的脚本可能包含数据 386
库上执行的 SQL 代码。

一些广泛使用的脚本语言包括 Sun 的 Java Server Pages(JSP)、Microsoft 的 Active Server Pages(ASP)和后续的 ASP. NET、PHP 脚本语言、ColdFusion 标记语言(ColdFusion Markup Language，CFML)以及 Ruby on Rails。许多脚本语言也允许用诸如 Java、C#、VBScript、Perl 以及 Python 语言写的代码嵌入到 HTML 页面中或被 HTML 页面调用。例如，JSP 允许 Java 代码嵌入 HTML 页面中，而 Microsoft 的 ASP. NET 和 ASP 支持内嵌的 C#和 VBScript。这些语言中的大部分都带有库和工具，它们一起构成了 Web 应用开发的框架。

接下来简要介绍 **Java Server Pages(JSP)**，一种允许 HTML 程序员将静态的 HTML 和动态生成的 HTML 混合在一起的脚本语言。其动机在于，对于许多动态的 Web 页面，其中大多数的内容仍然是静态的(也就是说，不论该页面何时生成，总是显示相同的内容)。Web 页面的动态内容(比如，根据表单的参数生成)通常是页面的一小部分。通过写 servlet 代码来创建这样的页面导致大量的 HTML 被写作 Java 字符串。JSP 则允许 Java 代码嵌入静态的 HTML 中；嵌入的 Java 代码生成该页面的动态部分。JSP 脚本实际上转换为 servlet 代码进行编译，但是应用程序员却从撰写大量 Java 代码以创建 servlet 的困境中解脱出来了。

图 9-9 所示为一个包含了内嵌的 Java 代码的 JSP 页面的源代码文本。脚本中的 Java 代码用 <% …% > 括起来以区别于周围的 HTML 代码。代码使用 request. getParameter( )以获取属性 name 的值。

```
<html>
<head> <title> Hello </title> </head>
<body>
 < % if (request.getParameter("name") == null)
  { out.println("Hello World"); }
  else { out.println("Hello, " + request.getParameter("name")); }
 %>
</body>
</html>
```

图 9-9　一个包含 Java 代码的 JSP 页面

当浏览器请求一个 JSP 页面时，应用服务器根据该页面生成 HTML 输出，并发送回浏览器。JSP 页面中的 HTML 部分正如所输出的。⊖在 HTML 输出中，在 <%…% > 中嵌入的所有 Java 代码都用它向 387
out 对象输出的文本代替。在图 9-9 的 JSP 代码中，如果没有值输入给表单参数 name，则脚本输出

⊖ JSP 允许一种更复杂的嵌入，即 HTML 代码在 Java 的 if-else 语句内部，它根据 if 条件等于真或非真而有条件地得到输出。我们在这里省略细节。

"Hello World"；如果输入了一个值，则脚本输出"Hello"，后面跟着名字。

PHP

PHP 是一种广泛用于服务器端脚本的脚本语言。PHP 代码可以像 JSP 中的方式一样与 HTML 混合使用。字母串"<? php"表示 PHP 代码的开始，字母串"? >"表示 PHP 代码的结束。下列代码与图 9-9 中的 JSP 代码执行相同的动作。

```
<html>
<head> <title> Hello </title> </head>
<body>
 <?php if (isset($_REQUEST['name']))
  { echo 'Hello World'; }
  else { echo 'Hello,' . $_REQUEST['name']; }
 ?>
</body>
</html>
```

数组 $_REQUEST 包含请求的参数。注意，数组用参数名索引；在 PHP 中数组可以用任意字符串索引，而不仅是数字。函数 isset( ) 检查数据元素是否已初始化。echo( ) 函数将它的参数输出到 HTML。两个字符串间的操作符"."将字符串连起来。

一个适当配置的 Web 服务器将把任意以".php"结尾的文件解释为 PHP 文件。如果请求该文件，Web 服务器将以与处理 JSP 文件类似的方法处理该文件，并将生成的 HTML 返回给浏览器。

PHP 语言有许多库，包括使用 ODBC(类似于 Java 中的 JDBC)访问数据库的库。

一个更实际的例子可能执行更复杂的操作，例如使用 JDBC 从数据库中查询值。

JSP 也支持标签库(*tag library*)的概念，它允许使用标签，这些标签看起来非常像 HTML 标签，却是在服务器端解释，并用相应生成的 HTML 代码取代。JSP 提供了一个标准标签集，其中定义了变量和控制流(迭代器、if-then-else)，以及基于 Javascript 的表达式语言(但在服务器端解释)。标签集是可扩展的，并且许多标签库都已实现。例如，存在一个标签库支持对大数据集的分页显示，还有一个库
388 简化了对日期和时间的显示和解析。关于 JSP 标签库的更多信息可以参看文献注解中的参考资料。

9.3.6 客户端脚本

文档中程序代码的嵌入允许 Web 页是**活动的**(active)，通过在本地执行程序以实现活动，例如动画，而不仅仅是展示被动的文字和图片。这种程序主要用于在 HTML 与 HTML 表单提供的有限的交互能力之上，与用户进行灵活的交互。此外，与每次交互都发送给服务器端处理相比，在客户端执行程序极大加快了交互的速度。

支持这种程序的一个危险在于，如果系统设计得不小心，Web 页中(或等同地，在电子邮件消息中)内嵌的程序代码能够在用户的计算机上执行恶意操作。恶意操作会包括读取私人信息，在计算机上删除或修改信息，甚至控制计算机并将代码传播到其他计算机(例如通过电子邮件)。近年来许多邮件病毒都是通过这种方式广泛传播的。

Java 语言日趋流行的一个原因是它为在用户的计算机上执行程序提供了一种安全模式。Java 代码能够编译成独立于平台的"字节码"，它可以在任意支持 Java 的浏览器上执行。与本地程序不同的是，作为网页的一部分所下载的 Java 程序(小应用程序)无权执行任何可能是破坏性的操作。它们可以在屏幕上显示数据，或者与下载网页的服务器进行网络通信以获取更多的信息。然而，它们不能访问本地文件，执行任何系统程序，或和任何其他计算机进行网络通信。

Java 是一种完全成熟的程序设计语言，另外有一些更简单的语言，称为**脚本语言**(scripting language)，在提供与 Java 相同的保护的同时，能够丰富用户的交互。这些语言提供能够嵌入到 HTML 文档中的构件。**客户端脚本语言**(client-side scripting language)是用于在客户端的 Web 浏览器上执行的语言。

其中，*JavaScript* 语言在目前使用最为广泛。目前这代 Web 界面广泛使用 JavaScript 脚本语言构造复杂的用户界面。JavaScript 用于多种任务。例如，用 JavaScript 写的函数可以用于对用户输入执行错误

检查(验证)，例如日期字串是否符合格式，或者输入的值(例如年龄)是否在适当的范围内。在输入数据时，甚至在数据发送往 Web 服务器之前，这些检查就已在浏览器上执行。

图 9-10 所示为用于验证表单输入的 JavaScript 函数的一个示例。该函数在 HTML 文档的 head 段声明。该函数检查所输入的课程学分是一个大于 0 并且小于 16 的数字。form 标签表示该验证函数在表单提交时调用。如果验证失败，则将一个警告框显示给用户，而如果验证成功，则将表单提交给服务器。

JavaScript 可以用于动态修改所显示的 HTML 代码。浏览器将 HTML 代码解析为内存中的一个树结构，它是由称为**文档对象模型**(Document Object Model，DOM)标准定义的。JavaScript 代码能够修改该树结构以执行某些操作。例如，假设一个用户需要输入几行数据，例如一个清单中的几项。一个包含文本框以及其他形式输入方法的表格可以用来收集用户输入。表格可能有一个默认大小，但是如果需要更多行，用户可以点击标有(比如)"添加项目"的按钮。可以将这个按钮设置为调用修改 DOM 树以向表格额外增加一行的 JavaScript 函数。

```
<html>
<head>
<script type="text/javascript">
  function validate() {
    var credits=document.getElementById("credits").value;
    if (isNaN(credits)|| credits<=0 || credits>=16) {
      alert("Credits must be a number greater than 0 and less than 16");
      return false
    }
  }
</script>
</head>

<body>
<form action="createCourse" onsubmit="return validate()">
  Title: <input type="text" id="title" size="20"><br />
  Credits: <input type="text" id="credits" size="2"><br />
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

图 9-10 用于验证表单输入的 JavaScript 示例

虽然 JavaScript 语言已经标准化，但是浏览器之间仍存在差别，特别是 DOM 模型的一些细节。因此，在一个浏览器上运行的 JavaScript 代码有可能在另一个浏览器上无法运行。为了避免这种问题，最好使用一个 JavaScript 库，例如 Yahoo 的 YUI 库，它使得代码的编写独立于浏览器。库中的函数在内部能够找出正在使用哪种浏览器，并向浏览器发送相应生成的 JavaScript。关于 YUI 和其他库的更多信息可以参看本章结尾的 9.5.1 节。

今天，JavaScript 广泛用于创建动态网页，它使用统称为 Ajax 的几种技术。用 JavaScript 编写的程序和 Web 服务器异步通信(即在后台，不阻断用户和 Web 浏览器的交互)，并能够获取数据并显示。

389
390

作为使用 Ajax 的一个示例，考虑一个网站表单，允许你选择国家，并且一旦选择了一个国家，你就可以从这个国家的州列表中选择一个州。在国家选定之前，州的下拉列表是空的。Ajax 框架允许在选定一个国家时，在后台从该网站下载州的列表，并且一获取到这个列表，它就添加到下拉列表中，使你可以选择州。

还有一些专用脚本语言用于专门的任务，比如动画(例如，Flash 和 Shockwave)以及三维建模(虚拟现实标记语言(Virtual Reality Markup Language，VRML))。目前，Flash 不仅被广泛用于动画，还用于处理流媒体视频内容。

9.4 应用架构

为了处理大型应用的复杂性，通常将它们分为若干层：

- *展示层*或*用户界面层*，它处理用户交互。单个应用程序可能有若干个不同版本的展示层，对应于不同类型的界面，例如 Web 浏览器，以及手机用户界面，它的屏幕相比较小很多。

  在很多实现中，基于**模型 - 视图 - 控制器**(Model-View-Controller，MVC)架构，展示/用户界面层本身在概念上分为多层。**模型**(model)对应于下文中所述的业务 - 逻辑层。**视图**(view)定义数据的显示；单个底层模型根据用于访问应用所指定的软件/设备可以具有不同的视图。**控制器**(controller)接收事件(用户操作)，在模型上执行操作，并返回一个视图给用户。MVC 架构用于多个 Web 应用框架，这将在 9.5.2 节讨论。

- **业务逻辑**(bussiness-logic)层，提供对数据和数据操作的高级视图。9.4.1 节详细讨论业务逻辑层。

- **数据访问**(data access)层，提供业务逻辑层和底层数据库间的接口。当底层数据库是关系数据库时，许多应用使用面向对象语言编写业务逻辑层，并使用面向对象数据模型。在这种情况中，数据访问层还提供从业务逻辑所使用的面向对象数据模型到数据库所支持的关系模型的映射。9.4.2 节详细讨论这种映射。

图 9-11 所示为上述各层，以及处理来自 Web 浏览器的一条请求所采取的一系列步骤。图 9-11 中箭头上的标记表示步骤的顺序。当应用服务器接收到请求时，控制器向模型发送一条请求。模型使用
391 业务逻辑处理请求，其中可能涉及更新模型中的对象，接着创建一个结果对象。模型依次使用数据访问层更新数据中的信息或从数据库获取信息。模型生成的结果对象发送到视图模块，它对结果生成一个 HTML 视图以在 Web 浏览器上显示。该视图可能根据用于查看结果的设备的特点来定制，例如它是否是一个具有大屏幕的计算机显示器，或者是手机上的小屏幕。

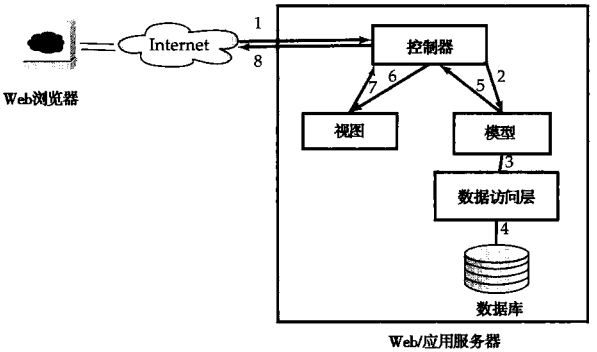

图 9-11 Web 应用程序框架

9.4.1 业务逻辑层

一个用于管理大学的应用程序的业务逻辑层可能会提供实体的抽象，比如学生、教师、课程、课程段等，以及操作，比如大学录取学生、学生注册课程等。实现这些操作的代码保证**业务规则**(bussiness rule)被满足；例如，代码要保证只有当学生已经完成课程先修课并且已经交付学费时才能注册课程。

另外，业务逻辑包含**工作流**(workflow)，它描述如何处理一个涉及多个参与者的特定任务。例如，如果一名候选人申请大学，存在一个工作流规定首先由谁查看并批准申请，并且如果第一个阶段批准后，接下来应该由谁查看申请，如此下去，直到给学生发出入学邀请或者寄出拒信。工作流管理也需要处理错误情况；例如，如果接收/拒绝的最后期限还没到，则可能需要通知主管，使她能够干预并确
392 保申请已处理。工作流在 26.2 节中有详细的讨论。

9.4.2 数据访问层和对象 - 关系映射

在最简单的场景中，业务逻辑层和数据库使用相同的数据模型，数据访问层只是隐藏了与数据库连接的细节。然而，当用面向对象程序设计语言编写业务 - 逻辑层时，很自然地会将数据建模为对象以及对象上调用的方法。

在早期实现中，程序员不得不为了从数据库中获取数据以创建对象以及将更新后的对象存回数据库而编写代码。然而，这种数据模型间的人工转换很麻烦并且容易出错。解决这个问题的一种方法是开发一个本身存储对象和对象之间关系的数据库系统。这种数据库称作**面向对象数据库**(object-oriented database)，在第 22 章将对其详细介绍。然而，由于许多技术和商业原因，面向对象数据库没有取得商业上的成功。

另一种方法是使用传统关系数据库保存数据，但自动建立从关系中的数据到内存中按需创建(由

于通常没有足够的内存存储数据库中所有的数据）的对象的映射，以及反向映射，从而将更新后的对象以关系的形式保存回数据库。

HIBERNATE 示例

作为 Hibernate 使用示例，创建一个对应于 student 关系的 Java 类如下：

```
public class Student {
    String ID;
    String name;
    String department;
    int tot.cred;
    Student(String id, String name, String dept, int totcreds); //构造函数
}
```

为了准确，类属性应该声明为私有，并应该提供 getter/setter 方法以访问属性，但是我们省略这些细节。

从 Student 的类属性到 student 关系属性的映射在映射文件中以 XML 格式表示。我们再次省略细节。

下面的代码段创建了一个 Student 对象并将它保存到数据库中。

```
Session session = getSessionFactory().openSession();
Transaction txn = session.beginTransaction();
Student stud = new Student("12328", "John Smith", "Comp. Sci.", 0);
session.save(stud);
txn.commit();
session.close();
```

Hibernate 自动生成所需的 SQL insert 语句，以在数据库中创建一个 student 元组。

为了检索学生，可以用以下代码段：

```
Session session = getSessionFactory().openSession();
Transaction txn = session.beginTransaction();
List students =
        session.find("from Student as s order by s.ID asc");
for ( Iterator iter = students.iterator(); iter.hasNext(); ) {
    Student stud = (Student) iter.next();
    .. 输出学生信息 ..
}
txn.commit();
session.close();
```

以上代码段使用了用 Hibernate 的 HQL 查询语言表达的一个查询。HQL 查询被 Hibernate 自动翻译成 SQL 并执行，然后把结果转换成一个 Student 对象对象。for 循环通历该列表中的对象并将输出它们。

已经有好几个实现这种对象 - 关系映射（object-relational mapping）的系统被开发出来。Hibernate 系统广泛用于将 Java 对象映射到关系。在 Hibernate 中，从每个 Java 类到一个或多个关系的映射都记录在一个映射文件中。例如，该映射文件可以记录一个叫做 *Student* 的 Java 类映射到关系 *student*，同时 Java 属性 ID 映射到属性 *student*. ID 等。*properties* 文件记录了数据库的信息，例如数据库运行所在的主机，以及用于连接数据库的用户名和密码。程序需要打开一个会话，建立到数据库的连接。一旦会话建立，Java 中创建的 *Student* 对象 *stud* 就可以通过调用 session. save(*stud*)保存到数据库中。Hibernate 代码生成用于将数据存储到 *student* 关系中的 SQL 命令。

一组对象可以通过执行用 Hibernate 查询语言所写的查询从数据库中获取；这与用 JDBC 执行查询并返回包含一组元组的 ResultSet 是类似的。或者，单个对象可以通过提供它的主码来获取。获取到的对象可以在内存中更新；当运行中的 Hibernate 会话上的事务被提交时，Hibernate 通过在数据库中的关系上进行相应的更新，自动保存更新的对象。

虽然 E-R 模型中的实体自然地对应诸如 Java 的面向对象语言中的对象，但联系通常并不对应。Hibernate 支持将这种联系映射为关联对象的集合的能力。例如，*student* 和 *section* 之间的 *takes* 联系可以通过将 *section* 的集合和每个 *student* 关联，并将 *student* 的集合和每个 *section* 关联来建模。一旦指定好适

393
|
394 当的映射，Hibernate 就会根据数据库关系 *takes* 自动填充这些集合，并且对于集合的更新也会在提交时反映回数据库关系中。

上述特点帮助提供给程序员一个不需要考虑关系存储细节的高级数据模型。然而，Hibernate 和其他对象 - 关系映射系统一样，也允许程序员直接用 SQL 访问底层关系。这种直接访问对于编写复杂查询以生成报表至关重要。

Microsoft 开发了一种数据模型，称作**实体数据模型**(Entity Data Model)，可以将它看成多种实体 - 联系模型，以及一个相关的框架，叫作 ADO. NET 实体框架，它能在实体数据模型和关系数据库间映射数据。该框架还提供一种类 SQL 查询语言，称作**实体 SQL**(Entity SQL)，它直接在实体数据模型上操作。

9.4.3 Web 服务

在过去，大部分 Web 应用只使用应用服务器及其相关的数据库中的数据。近年来，Web 中存在许多种类的数据，需要通过程序来处理，而不是直接显示给用户；程序可能作为后端应用的一部分运行，或可能是在浏览器中运行的脚本。通常，通过实际的 Web 应用编程接口访问这些数据，即使用 HTTP 协议发送一个函数调用请求，在应用服务器上执行，然后将结果发送回主调程序。支持这种接口的系统被称为 **Web 服务**(Web service)。

有两种方法广泛用于实现 Web 服务。较简单的方法为**代表性状态传输**(REpresentation State Transfer，REST)，它通过在应用服务器对 URL 的标准 HTTP 请求执行 Web 服务函数调用，参数作为标准 HTTP 请求的参数发送。应用服务器执行该请求(有可能涉及服务器端数据库的更新)，生成结果并对其编码，然后将结果作为 HTTP 请求的结果返回。服务器对一个特定请求的 URL 可以使用任意编码；XML 和一种叫作 **JavaScript 对象符号**(JavaScript Object Notation，JSON)的 JavaScript 对象编码广泛使用。请求方解析返回的页面以访问所返回的数据。

在许多 REST 风格的 Web 服务(即使用 REST 的 Web 服务)的应用中，请求方是运行在 Web 浏览器中的 JavaScript 代码；该代码使用函数调用的结果更新浏览器屏幕。例如，当你在 Web 上的地图界面上滚动显示时，地图中需要新增的显示部分可以使用 REST 风格的接口通过 JavaScript 代码获取，然后显示在屏幕上。

一个更加复杂的方法有时称作“大 Web 服务”，它对参数和结果使用 XML 编码，使用一种专门的语言正规定义 Web API，并在 HTTP 协议上构建了一个协议层。23.7.3 节有对这个方法的详细描述。

9.4.4 断连操作

许多应用都希望即使当一个客户端从应用服务器断开连接时仍支持某些操作。例如，一个学生可
395 能希望填写申请表即使她的笔记本电脑从网络断开，但在笔记本电脑重新连接网络的时候将它保存回去。另一个例子是，如果将一个电子邮件客户端构建为一个 Web 应用，一个用户可能希望书写电子邮件即使她的笔记本电脑从网络断开，并在重新连接到网络时将邮件发送出去。构建这种应用需要在客户端机器中本地存储，最好是以数据库的形式。Gears 软件(最初由 Google 开发)是一个浏览器插件，它提供一个数据库、一个本地 Web 服务器，并支持在客户端并行执行 JavaScript。该软件在多种操作系统/浏览器平台上同样运行，允许应用程序支持丰富的功能而并不需要安装任何的软件(除了 Gears 自己以外)。Adobe 的 AIR 软件还给能够为运行在 Web 浏览器之外的 Internet 应用的构建提供类似的功能。

9.5 快速应用开发

如果构建 Web 应用时没有使用工具或库构建用户界面，则构建用户界面所需的编程工作可能远大于业务逻辑和数据库访问所需。有几种方法可以用以减少构建应用程序所需的工作：

- 提供一个函数库，以用最小的编程量生成用户界面元素。
- 在集成开发环境中提供拖放功能，允许将用户界面元素从菜单中拖至页面的设计视图。该集成开发环境通过调用库函数生成创建用户界面元素的代码。

- 根据声明规范自动生成用户界面的代码。

在创建 Web 之前，所有这些方法就都已经作为**快速应用开发**(Rapid Application Development，RAD)工具的一部分用于创建用户界面了，并且目前也广泛用于创建 Web 应用。

用来快速开发数据库应用接口的工具的例子包括 Oracle Forms、Sybase PowerBuilder 以及 Oracle Application EXpress(APEX)。另外，为 Web 应用开发所设计的工具，例如 Visual Studio 和 Netbeans VisualWeb，支持用于快速开发基于数据库的应用的 Web 界面的一些功能。

我们在 9.5.1 节中学习创建用户界面的工具，并在 9.5.2 节中学习支持自动根据系统模型生成代码的框架。

9.5.1 构建用户界面的工具

许多 HTML 结构最好是由适当定义的函数生成，而不是作为网页代码的一部分来编写。例如，地 396
址表单通常需要一个包含国家或州名的菜单。最好定义一个输出该菜单的函数并在需要的时候调用该函数，而不是每次用到的时候编写冗长的 HTML 代码以生成所需的菜单。

菜单通常最好由数据库中的数据生成，比如一个包含国家名或州名的表。生成菜单的函数执行数据库查询并用查询结果填写菜单。增添一个国家或州则只需要改变数据库，而不需要改变应用程序代码。这个方法有个潜在的缺陷，即需要增加数据库交互，不过可以通过在应用服务器端缓存查询结果以最小化这个开销。

类似地，输入日期和时间或需要验证的输入的表单最好通过调用适当定义的函数生成。这些函数能够输出在浏览器端执行验证的 JavaScript 代码。

对于许多数据库应用，显示查询的结果集是一项共有的任务。可以构建一个泛型函数，将 SQL 查询(或 ResultSet)作为参数，并以表格形式显示查询结果(或 ResultSet)中的元组。可以使用 JDBC 元数据调用从查询结果中找到诸如列数以及列的名称和类型的信息；然后该信息用于显示查询结果。

为了处理查询结果非常大的情况，查询结果显示功能可以提供对结果的分页。该功能可以在一页中显示固定条数的记录，并提供控制以跳到下一页或前一页或者跳到结果的指定页。

遗憾的是，没有(广泛使用的)用于实现上述用户界面任务的标准 Java API 函数。构建这样的一个库将会是一个有趣的编程项目。

不过，存在其他一些工具，比如 JavaServer Faces(JSF)框架，它就支持上面所列功能。JSF 框架包含一个实现这些功能的 JSP 标签库。Netbeans IDE 具有一个叫做 VisualWeb 的组件，它构建在 JSF 上，提供一个可以将属性自定义的用户界面组件拖放到一个页面的可视化开发环境。例如，JSF 提供创建下拉菜单的组件，或者显示从数据库查询中获取数据的表格的组件。JSF 还支持在组件上验证规范，例如做选择或强制输入，或者限制一个数或一个日期在指定范围内。

Microsoft 的 Active Server Pages(ASP)及其最近的版本 Active Server Pages. NET(ASP. NET)，是 JSP/Java 的一种广泛使用的替代品。ASP. NET 与 JSP 类似，其中可以在 HTML 代码中嵌入 Visual Basic 或者 C#语言的代码。另外，ASP. NET 提供多种控件(脚本命令)，在服务器端解释并生成 HTML 返回给客户端。这些控件能显著地简化 Web 界面的构建。我们对这些控件带来的好处给出一个简要的概览。 397

例如，诸如下拉菜单和列表框这样的控件可以与一个 DataSet 对象关联。DataSet 对象与 JDBC 的 ResultSet 对象相似，通常也是通过在数据库上执行查询而创建的。HTML 菜单内容从 DataSet 对象的内容生成；例如，一个查询可能将一个机构中所有部门的名字检索到 DataSet 中，并且关联的菜单将会包含这些名字。这样，基于数据库内容的菜单就能以极少量的编码方便地创建出来。

验证控件可以添加到表单的输入域中；这声明性地定义了有效性约束，诸如值的范围或者输入是否是必需的(即用户必须提供的值)。服务器创建适当的结合 JavaScript 的 HTML 代码，以在用户的浏览器中执行有效性验证。对于无效输入所显示的错误消息可以与每个验证控件相关联。

可以指定用户操作在服务器端具有相关联的操作。例如，一个菜单控件可以指定从菜单中选择一个值具有一个相关联的服务器端动作(生成 JavaScript 代码以探查选择事件并初始化服务器端操作)。显示属于所选值的数据的 Visual Basic/C#代码可以与服务器上的该动作相关联。因此，从菜单中选择一个值会引起页面上相关联的数据的更新，而不需要用户点击提交按钮。

DataGrid 控件提供了一个非常方便的方法显示查询结果。一个 DataGrid 和一个 DataSet 对象相关联，DataSet 对象通常是一个查询的结果。服务器生成将查询结果显示为表的 HTML 代码。列标题根据查询结果的元数据自动生成。另外，DataGrid 提供了一些功能，比如分页，以及允许用户在所选列上对结果排序。所有实现这些特性的 HTML 代码以及服务器端功能都是由服务器自动生成的。DataGrid 甚至允许用户编辑数据并将更改提交回服务器。应用开发人员可以指定一个函数在一行数据被编辑时执行，这就可以实现数据库上的更新。

Microsoft Visual Studio 为创建使用这些特性的 ASP 页面提供了图形用户界面，进一步减少了编程的工作。

关于 ASP. NET 的更多信息，请参看文献注解中的参考文献。

9.5.2 Web 应用框架

有多种 Web 应用开发框架都提供一些常用的特性，比如：

- 一个包含对象 - 关系映射的面向对象模型，从而在关系数据库中保存数据(如 9.4.2 节所述)。
- 一种(相对)声明式的方式，说明一个表单具有在用户输入上的验证约束，系统据此生成 HTML 和 JavaScript/Ajax 代码以实现表单。
- 一个模板脚本系统(与 JSP 类似)。
- 一个控制器，将诸如表单提交的用户交互事件映射到处理该事件的适当函数上。控制器还管理身份验证和会话。一些框架还提供管理身份验证的工具。

398

这样，这些框架就以一种集成的方式提供了多种构建 Web 应用所需的特性。通过根据声明规范生成表单，以及透明地管理数据访问，这些框架最小化了 Web 应用程序员所需完成的代码量。

这样的框架有许多，它们基于不同语言。其中一些较为广泛使用的框架包括 Ruby on Rails，它是基于 Ruby 程序设计语言的，还有 JBoss Seam、Apache Struts、Swing、Tapestry 以及 WebObjects，它们都是基于 Java/JSP 的。其中的一些(比如 Ruby on Rails 和 JBoss Seam)提供自动生成简单 CRUD Web 界面的工具；也就是支持对象/元组的创建、阅读、更新和删除的界面，它是通过对象模型或数据库生成代码的。这种工具在使简单应用快速运行时特别有用，并且可以编辑生成的代码以构建更复杂的 Web 界面。

9.5.3 报表生成器

报表生成器是从数据库生成人们可读的概要报告的工具。它将生成格式化文本和概要图表(例如条形图或饼状图)与查询数据库集成在一起。例如，一个报表可能会显示过去的两个月中每个月每个销售地区的总销售额。

应用开发人员可以利用报表生成器的格式化工具来指定报表的格式。变量可以用来存储参数(如月、年)以及定义报表中的域。表、图、条形图或其他的图可以通过对数据库的查询来定义。查询定义可以利用存储在变量里的参数值。

一旦在报表生成器工具上定义了一个报表的结构，就可以保存它并可以在任何时候执行它来产生报表。报表生成器系统提供了多种工具来组织表格式输出，如定义表和列标题，显示表中每一组的小计，自动将一个长的表格分成若干页，在每一页末尾显示本页小计等。

图 9-12 是一个格式化报表的例子。报表中的数据是按顺序对信息进行聚集产生的。

许多厂商都提供报表生成工具，例如 Crystal Reports 和 Microsoft(SQL Server Reporting Services)。一些应用套件(例如 Microsoft Office)提供了将来自数据库的格式化查询结果直接嵌入到文档中的方法。Crystal Reports 提供的图表生成工具或者诸如 Excel 提供的电子表格都可以用于从数据库访问数据，并生成表格化的数据展现或者使用图表或图的图形化方式展现数据。这种图表可以嵌入文本文档中。图表最初由执行数据库查询生成的数据创建；查询在需要的时候可以重新执行并重新生成图表，以获得概要报表的当前版本。

除了生成静态报表，报表生成工具也支持创建交互式报表。例如，用户可以“下钻”到感兴趣的区域，比如从一个显示全年总销量的聚集视图转移到某一年的月销售图表。在前面 5.6 节已讨论过这种操作。

Acme供应有限公司
季度销售报告

日期: 2009年1月1日至3月31日

地 区	类 别	销 量	小 计
北部	计算机硬件	1 000 000	
	计算机软件	500 000	
	所有类别		1 500 000
南部	计算机硬件	200 000	
	计算机软件	400 000	
	所有类别		600 000
		总计	2 100 000

图 9-12 一个格式化报表

9.6 应用程序性能

Web 站点可能被来自全球的数百万人访问，每秒钟要处理数千次的请求，对于最流行的站点甚至更多。确保对请求的服务有较短的响应时间是 Web 站点开发人员面临的一个主要挑战。为了做到这点，应用开发人员通过利用诸如缓存等技术尽可能加快处理单个请求的速度，并通过使用多个应用服务器进行并行处理。我们在下面简要描述这些技术。数据库应用调优在后面的第 24 章(24.1 节)详细讲述。

9.6.1 利用缓存减少开销

多种缓存技术用于充分利用事务之间的共性。例如，假设服务于每个用户请求的应用程序代码都需要通过 JDBC 连接数据库。因为创建一个新的 JDBC 连接需要几个毫秒的时间，所以在要支持很高的事务处理速度时，为每次请求创建一个新连接是不合适的。

399
400

连接池(connection pooling)用来减轻这种开销；它是这样工作的：连接池管理器(应用服务器的一部分)创建一个打开的 ODBC/JDBC 连接的池。与打开一个新的数据库连接不同，对用户请求提供服务的代码(通常是一个 servlet)向连接池申请(请求)一个连接，然后在代码(servlet)完成处理时将连接归还给连接池。如果在请求连接时连接池没有未使用的连接，则打开一个新的数据库连接(要小心不要超过数据库能同时支持的最大连接数目)。如果有很多打开的连接在一段时间内没有使用，连接池管理器可能会关闭一些打开的数据库连接。许多应用服务器以及较新的 ODBC/JDBC 驱动程序提供内置的连接池管理器。

在创建 Web 应用时，许多程序员常犯的一个错误是忘记关闭一个打开的 JDBC 连接(或者等同地，在用连接池的时候，忘记把连接归还给连接池)。每个请求则打开一个新的数据库连接，而且数据库很快就达到了同时能支持的连接数目的上限。这样的问题通常在小规模测试的时候不会暴露出来，因为数据库通常能允许数百个打开的连接，而仅会在频繁使用的时候显现出来。某些程序员以为连接像 Java 程序分配的内存，会自动回收。遗憾的是，这不会发生，并且程序员有责任关闭他们所打开的连接。

某些请求可能导致向数据库重新提交完全相同的查询。只要数据库中的查询结果没有改变，通过缓存先前的查询结果并重用它们，数据库通信的开销可以大大减少。一些 Web 服务器支持这样的查询结果缓存；如果不支持的话，可以在应用程序代码中显式实现缓存。

通过缓存响应一个请求所发送的最终 Web 页面可以进一步减少开销。如果新请求和前面的一个请求参数完全一致，请求不需要执行更新，并且结果 Web 页面在缓存中，那么它就可以重用，从而避免重新计算该页面的开销。缓存可以在 Web 页的片段级别实现，然后它们被组装以创建完整的 Web 页。

缓存的查询结果和缓存的 Web 页面是物化视图的形式。如果底层数据库数据改动，缓存的结果必须废弃，或重新计算，或甚至增量更新，就如物化视图的维护(在后面 13.5 节介绍)那样。某些数据库系统(比如 Microsoft SQL Server)为应用服务器提供了一种方式在数据库注册查询，并且在查询结果改变时从数据库得到**通知**(notification)。这种通知机制可以用于确保缓存在应用服务器中的查询结果是最新的。

9.6.2 并行处理

处理非常重的负载的一个常用的方法是并行运行许多台应用服务器，每个处理一小部分请求。一
401 个 Web 服务器或者一个网络路由器可以用于将每个客户端的请求路由到一台应用服务器。来自一个特定客户端会话的所有请求都必须送到同一个应用服务器，因为服务器要维护客户端会话的状态。比如，可以通过将所有来自一个 IP 地址的请求都路由到相同的应用服务器来确保这个性质。然而，底层数据库被所有应用服务器共享，因此用户看到的数据库是一致的。

由于在以上架构中数据库是共享的，因此它很容易成为瓶颈。应用程序设计人员通过在应用服务器端缓存查询结果，特别地注意最小化数据库请求的数量。另外，当需要时则使用并行数据库系统，这将在第 18 章讲述。

9.7 应用程序安全性

应用程序安全性要在 SQL 授权处理的范围之外对几种安全威胁和问题进行处理。

第一个必须要强制实施安全性的地方是在应用程序里。为此，应用程序必须对用户进行身份验证，并确保用户只允许完成授权的任务。

存在许多方法使一个应用程序的安全性能够受到威胁，即使数据库系统本身是安全的，不好的应用程序代码也会使应用程序的安全性受到威胁。本节首先介绍几种安全漏洞，它们能够允许黑客采取行动绕过应用程序采取的身份认证和授权检查，并解释如何防止这种漏洞。本节后面介绍安全认证和细粒度授权的技术。然后我们介绍审计追踪，它能够帮助从未经授权的访问和错误更新中恢复。我们通过介绍数据隐私问题结束本节。

9.7.1 SQL 注入

在 **SQL 注入**(SQL injection)攻击中，攻击者设法获取一个应用程序来执行攻击者生成的 SQL 查询。在 5.1.1.4 节中，我们看到了一个 SQL 注入漏洞的例子，如果用户输入直接与 SQL 查询连成一串并提交给数据库。作为 SQL 注入漏洞的另一个例子，考虑图 9-4 中所示的表单源文本。假设在图 9-8 所示的相应的 servlet 中用以下 Java 表达式创建了一个 SQL 查询串：

```
String query = "select * from student where name like '%" + name + "%'"
```

402 其中 name 是用户输入的包含字符串的变量，并继而在数据库上执行该查询。利用 Web 表单的恶意攻击者则可以键入诸如"'; < some SQL statement >; __ "的字符串，替代一个有效的学生姓名，其中 < some SQL statement > 表示任何攻击者希望的 SQL 语句。servlet 则将执行以下字符串：

```
select * from student where name like''; < some SQL statement >; - -'
```

攻击者插入的引号结束该字符串，后面的分号终止该查询，然后攻击者接下来所插入的文本被解析为第二条 SQL 查询，而右引号已被注释掉。这样，该恶意用户成功插入被应用程序执行的任意 SQL 语句。该语句可以导致重大的损害，因为它能够绕过应用程序代码中实现的所有安全措施，在数据库上进行任意的操作。

如 5.1.1.4 节所述，为了避免这种攻击，最好使用预备语句来执行 SQL 查询。当设置预备语句的一个参数时，JDBC 自动添加转义字符，从而使得用户提供的引号无法再终止字符串。等价地，添加这种转义字符的操作也可以在与 SQL 查询连接之前用在输入字符串上，而不使用预备语句。

SQL 注入危险的另一个来源是基于表单中指定的选择条件和排序属性动态创建查询的应用。例如，一个应用可能允许用户指定用哪个属性对查询结果排序。假设该应用从表单变量 orderAttribute 中获取属性名，并创建了一个查询串如下：

```
String query = "select * from takes order by " + orderAttribute;
```

即使用于获取输入的 HTML 表单通过提供表单试图限定所允许的值，一个恶意用户仍可以发送一个任意字符串替代一个有意义的 orderAttribute 值。为了避免这种 SQL 注入，应用程序应该在追加 orderAttribute 变量值之前确保它是所允许的值(在例子中为属性名)。

9.7.2 跨站点脚本和请求伪造

一个允许用户输入诸如评论或姓名的文本，然后将其保存并在以后显示给其他用户的网站，有可能会受到一种攻击，叫**做跨站点脚本**(Cross-Site scripting，XSS)攻击。在这种攻击中，一个恶意用户并不输入一个有效的姓名或评论，而是输入用诸如 JavaScript 或 Flash 的客户端脚本语言编写的代码。当另一个用户阅览输入的文本时，浏览器将会执行脚本，它可能会进行一些操作，比如将私人的 cookie 信息发送回恶意用户，或甚至在一个用户可能登录的另一个不同的 Web 服务器中执行操作。 403

例如，假设该脚本执行的时候用户恰巧登录了她的银行账户。该脚本可以将有关银行账户登录的 cookie 信息发送回恶意用户，他就可以用该信息连接到银行的 Web 服务器，欺骗它使它相信该连接来自原用户。或者，该脚本可以适当设置参数并访问银行网站上的适当网页，以执行转账。事实上即使不写脚本而只是用如下的一行代码就能够使这个问题发生：

```
< img src =
    "http: //mybank. com/transfermoney? amount = 1000&toaccount = 14523" >
```

假定 URL mybank. com/transfermoney 接受指定的参数，并进行了转账。这后一种漏洞又称作**跨站点请求伪造**(Cross-Site Request Forgery 或 XSRF，有时也称作 CSRF))。

XSS 可以用其他一些方式实现，比如引诱用户访问有恶意脚本嵌入其网页的网站。存在其他更加复杂的 XSS 或 XSRF 攻击，我们在此不再细讲。为了防止此类攻击，需要完成两件事：

- **防止你的网站被用来发动 XSS 或 XSRF 攻击。**
  最简单的技术就是禁止用户输入的任何文本中的任何 HTML 标签。存在检测或去除这些标签的函数。这些函数可以用来防止 HTML 标签，并就此防止脚本展示给其他用户。在有些情况下 HTML 格式是有用的，并且在这种情况下可以使用那种解析文本并允许有限的 HTML 结构，但不允许其他危险结构的函数。这些必须小心地设计，因为有时包含一个跟图片一样无害的结构也可能是危险的，如果图片显示软件中有一个能发现的错误。
- **防止你的网站被其他站点发动的 XSS 或 XSRF 攻击。**
  如果该用户已经登录你的网站，并访问了另一个易受 XSS 攻击的网站，在该用户的浏览器上执行的恶意代码则可能在你的网站上执行操作，或将与你的网站关联的会话信息发送回试图利用它的恶意用户。无法完全防止这种攻击，但是你可以采取几个步骤最小化风险。
  - □ HTTP 协议允许一个服务器检查访问页的**引用页**(referrer)，即用户为了初始化访问页而点击的链接所在网页的 URL。通过检查引用页是否有效，例如，引用页 URL 是同一个网站上的网页，源自用户访问的不同网页上的 XSS 攻击则可以防止。
  - □ 除了只使用 cookie 来标识一个会话外，还可以将会话限制在原始验证它的 IP 地址上。这样，即使一个恶意用户得到了 cookie，他也可能无法从另一台计算机登录。 404
  - □ 决不要使用 GET 方法执行任何更新。这防止了利用 < img src . . > 的攻击，如我们前面所看到的。事实上，由于其他原因，比如页面的刷新重复了本应只发生一次的操作，HTTP 标准建议 GET 方法不应该执行任何更新。

9.7.3 密码泄露

应用程序开发人员必须处理的另一个问题是在应用程序代码的明文中保存密码。例如，诸如 JSP 脚本的程序通常在明文中包含密码。如果这种脚本保存在一个 Web 服务器可访问的目录中，一个外部用户就可能能够访问脚本的源码，并获取应用程序使用的数据库账户的密码。为了避免这种问题，许多应用服务器提供用编码的形式保存密码的机制，在传送给数据库之前服务器对其解码。该功能去除了在应用程序中将密码另存为明文的需要。然而，如果解码密钥也容易暴露的话，这种方法就并不完

全有效了。

作为处理易受泄露的数据库密码的另一种措施，许多数据库系统对于数据库的访问只限制在给定网络地址集合中，该机制通常运行在应用服务器端。从其他网络地址企图连接数据库会被拒绝。这样，除非恶意用户能够登录应用服务器，否则即使她获取了数据库密码的访问权限她也无法造成任何损害。

9.7.4 应用程序认证

认证是指验证连接到应用程序的人/软件的身份。认证最简单的形式由一个密码构成，当一个用户连接到应用程序时必须出示该密码。遗憾的是，密码容易泄露，例如通过试猜，或者通过嗅探网络中的数据包，如果密码没有加密就传送的话。对于关键应用需要更健壮的方案，比如网上银行账户。加密是更健壮的认证方案的基础。通过加密的认证在 9.8.3 节讲述。

许多应用使用**双因素认证**(two-factor authentication)，其中两个独立的因素(即信息或程序的片段)用于识别一个用户。这两个因素不应该具有相同的弱点；例如，如果一个系统只需要两个密码，二者都可能以同样的方式泄露(例如通过网络嗅查，或通过用户使用的计算机上的病毒)。虽然诸如指纹或虹膜扫描仪的生物识别技术可以用于在认证点上用户是物理存在的情况，但是跨网络时它们就不是很
405 有用了。

在大部分这种双因素认证方案中，密码用作第一个因素。通过 USB 接口连接的智能卡或其他加密设备，可基于加密技术用于认证(见 9.8.3 节)，它们广泛用于第二个因素。

一次性密码设备，每分钟生成一个新的伪随机数，也广泛用于第二个因素。给每个用户一个设备，为了进行认证，用户必须输入认证时设备上所显示的数字以及密码。每个设备生成不同的伪随机数序列。应用服务器能够生成与用户的设备相同的伪随机数序列，在认证时在将显示的数字处停下来，并验证数字是否匹配。该方案需要设备中与服务器端的时钟始终是合理紧密同步的。

而第二个因素的另一种方法是当一个用户想登录一个应用时，给用户的手机(其号码是早先已注册的)发送一条包含(随机生成的)一次性密码的短信。用户必须拥有一部具有该号码的手机以接收短信，然后将一次性密码和她的密码一起输入用以认证。

然而即使使用双因素认证，用户可能仍很容易受到**中间人**(man-in-the-middle)攻击。在这种攻击中，一个试图连接应用的用户被转向一个虚假网站，它接受用户的密码(包括第二因素密码)，并立即用该密码到原始应用中认证。9.8.3.2 节描述的 HTTPS 协议用于为用户认证网站(使得用户不会连接到虚假网站)。HTTPS 协议还对数据加密，并防止中间人攻击。

当用户访问多个网站时，用户通常会因为不得不分别在每个网站上认证自己而感到不快，通常每个网站的密码是不同的。有系统允许用户向一个中央认证服务进行认证，并且其他网站和应用可以通过中央认证服务对用户进行认证；于是相同的密码可以用于访问多个站点。LDAP 协议广泛用于实现这种中央认证点；一些机构实现了包含用户名和密码信息的 LDAP 服务器，并且应用程序使用 LDAP 服务器对用户进行认证。

除了认证用户外，中央认证服务还能够提供其他的服务，例如，向应用程序提供用户的信息，比如姓名、E-mail 以及地址信息。这消除了在每个应用中分别输入这些信息的需要。如 19.10.2 节所述，LDAP 可以用于这个任务。其他的目录系统，如微软的活动目录，也提供认证用户以及提供用户信息的机制。

单点登录(single sign-on)系统还允许用户只认证一次，且多个应用通过一个认证服务对用户的身份
406 进行验证，而不需要再次认证。也就是说，一旦用户登录一个站点，他就不需要在其他使用相同单点登录服务的站点输入他的用户名和密码。这种单点登录机制已长期用于网络认证协议，比如 Kerberos，并且目前已经有面向 Web 应用的实现。

安全断言标记语言(Security Assertion Markup Language，SAML)是一个在不同安全域间交换认证和授权信息的标准，以提供跨机构单点登录。例如，假设一个应用需要提供对一所特定学校(比如说耶鲁)所有学生的访问。该学校可以建立一个基于 Web 的服务实施认证。假设一个连接到该应用的用户具有用户名，比如"joe@ yale. edu"。该应用将该用户转向耶鲁大学的认证服务对用户认证，而并不直接对用户认证，并告诉应用该用户是谁且可能提供一些额外的信息，比如用户的类别(学生或教师)或

者其他相关的信息。该用户的密码以及其他认证因素不会显示给应用，并且用户也不需要对应用程序显式地注册。不过，当认证用户时，该应用程序必须信任学校的认证服务。

OpenID 标准是另一种跨机构单点登录的标准，并在近年来逐渐被接受。许多流行的网站，比如 Google、Microsoft、Yahoo！等，是 OpenID 认证提供方。任何作为 OpenID 客户端的应用程序则可以使用这些提供方中的任意一个来认证用户；例如，一个具有 Yahoo！账户的用户可以选择 Yahoo！作为认证提供方。该用户被重定向到 Yahoo！进行认证，并且成功认证后被透明地重定向回应用程序，并能够继续使用应用程序。

9.7.5 应用级授权

虽然 SQL 标准支持一种比较灵活的基于角色的授权系统（如 4.6 节所述），但 SQL 授权模型在一个典型应用中对于用户授权的管理还是非常有限的。例如，假设你想要所有学生都可以看到他们自己的成绩，但是看不到其他人的成绩。这样的授权就无法在 SQL 中表示，原因至少有两点：

1. **缺乏最终用户信息。**随着 Web 规模的增长，数据库访问主要来自 Web 应用服务器。最终用户在数据库本身上通常没有个人用户标识，并且数据库中可能只存在单个用户标识对应于应用服务器的所有用户。因此，SQL 中的授权规范在上述情景中无法使用。

应用服务器能够认证最终用户，并继而将认证信息传递给数据库。在本节中我们将假设函数 *syscontext. user_id*( ) 返回一个正在执行的查询所代表的应用程序用户的标识。⊖ 407

2. **缺乏细粒度的授权。**如果我们授权学生只查看他们自己的成绩的话，授权必须是在单条元组的级别上。在目前的 SQL 标准中这种授权是不可能的，SQL 标准只允许在整个关系或视图上，或者关系或视图的指定属性上授权。

我们可以通过为每个学生在 *takes* 关系上创建一个只显示该学生成绩的视图，绕过这个限制。虽然这样在理论上可行，但是这将会非常繁琐，因为我们需要为学校中每一个注册的学生创建一个这样的视图，这是非常不实际的。⊜

另一种方法是创建一个视图形如：

```
create view studentTakes as
select *
from takes
where takes. ID = syscontext. user_id( )
```

随即向用户授权这个视图，而不是底层的 *takes* 关系。然而，代表学生所执行的查询现在就必须是在视图 *studentTakes* 上，而不是原始 *takes* 关系上，而代表教师所执行的查询就可能需要使用不同的视图。结果开发应用程序的任务变得更为复杂。

目前，授权的任务通常全部由应用程序执行，绕过 SQL 授权机制。在应用级别，授权用户访问特定接口，并可能进一步被限制只能查看或更新某些数据项。

虽然在应用程序中执行授权给应用程序开发人员带来很大的灵活性，但也存在一些问题：

- 检查授权的代码和应用程序的其他代码混合在一起。
- 通过应用程序的代码实现授权，而不是在 SQL 声明中授权，使得难以确保没有漏洞。由于一个疏忽，一个应用程序可能没有检查权限，而允许未权限的用户访问机密数据。 408

验证所有应用程序都完成所有需要的授权检查，涉及通读所有应用服务器的代码，在一个大的系统中是个非常艰巨的任务。也就是说，应用程序有一个非常大的“表面积”，这使保护应用程序的任务异常艰难。并且事实上，安全漏洞在各种实际生活应用中都有发现。

⊖ 在 Oracle 中，一个使用 Oracle 的 JDBC 驱动的 JDBC 连接可以用 OracleConnection. setClientIdentifier(userId) 函数设置最终用户标识，并且一个 SQL 查询可以用 sys_context('USERENV', 'CLIENT IDENTIFIER') 函数检索用户标识。

⊜ 数据库系统用于管理大关系，但是管理诸如视图的模式信息时会假定数据量较小，从而提高整体性能。

与此相反，如果一个数据库直接支持细粒度的授权，则可以在 SQL 级别指定并强制执行授权规则，这样表面积就会小很多。即使一些应用接口无意中省略了所需的授权检查，SQL 级授权也能够防止非授权操作的执行。

一些数据库系统提供了细粒度的授权机制。例如，Oracle 的**虚拟私有数据库**（Virtual Private Database，VPD）允许系统管理员将一个函数关联一个关系；该函数返回一个谓词，该谓词必须加入到任何一个使用该关系的查询中（对于被更新的关系可以定义不同的函数）。例如，通过使用我们获取应用程序用户标识的语法，关系 *takes* 的函数可以返回一个谓词，比如：

$$ID = sys\_context.\ user\_id()$$

该谓词加入到每个使用关系 *takes* 的查询的 **where** 子句中。结果（假定应用程序将 *user_id* 的值设置为学生的 *ID*），每个学生只能看见她选择的课程所对应的元组。

这样，VPD 提供了在关系的指定元组（或行）上的授权，并因此称为*行级授权*（row-level authorization）机制。上述添加谓词的方法中一个潜在的问题是，它可能会显著地改变一个查询的含义。例如，如果一个用户写了一条查询来查找所有课程的平均成绩，她最后将只能得到*她的*成绩的平均值，而非所有课程的平均成绩。虽然系统对于重写的查询会给出“正确的”答案，但是这个答案与用户可能认为她所提交的查询并不对应。

关于 Oracle VPD 的更多信息请参见文献注解。

9.7.6 审计追踪

审计追踪（audit trail）是关于应用程序数据的所有更改（插入/删除/更新）的日志，以及一些信息，如哪个用户执行了更改和什么时候执行的更改。如果应用程序安全性被破坏，或者即使安全性没有被破坏但是一些更新错误执行，一个审计追踪能够（a）帮助找出发生了什么，以及可能是由谁执行的操作，并（b）帮助修复安全漏洞或错误更新造成的损害。

例如，如果发现一个学生的成绩不正确，则可以检查审计日志，以定位该成绩是什么时候以及如
409 何被更新，并找出执行这个更新的用户。该学校还可以利用审计追踪来跟踪这个用户所做的所有更新，从而找到其他错误或欺骗性的更新，并将它们更正。

审计追踪还可以用于探查安全漏洞，在安全漏洞中用户账户易泄露并被入侵者访问。例如，在用户每次登录时，她可能被通知审计追踪中从最近一次登录开始所做的所有更新，如果用户发现一个更新并不是由她执行的，则有可能该账户泄露。

可以通过在关系更新操作上定义适当的触发器来创建一个数据库级审计追踪（利用标识用户名和时间的系统定义变量）。然而，很多的数据库系统提供了内置机制创建审计追踪，使用更加方便。具体如何创建审计追踪的细节随数据库系统的不同而不同，具体细节应参考数据库系统用户手册。

数据库级审计追踪对于应用程序来说通常是不够的，因为它们通常无法追踪应用程序的最终用户是谁。另外，它在一个低级别记录更新，即关系中元组的更新，而并不是在较高的级别，即业务逻辑级别。因此，应用程序通常创建一个较高级别的审计追踪，例如，记录执行了什么操作、何人、何时，以及请求源自哪个 IP 地址。

一个相关的问题是防止审计追踪本身被威胁应用程序安全的用户修改或删除。一个可能的解决方法是将审计追踪备份到入侵者无法访问的另一台机器中，每条追踪记录一旦生成立即复制。

9.7.7 隐私

目前，越来越多的个人数据都可以在线获取，人们也越来越担心他们的数据隐私。例如，大多数人都希望他们的私人医疗数据保持私密而不会公开暴露。但是，这些医疗数据又必须开放给治疗病人的医生和急救人员。许多国家都具有针对这种数据隐私的法律，定义了数据在什么时候以及对谁可以显示。违反隐私法在某些国家能够导致刑事处罚。访问这种隐私数据的应用程序的创建必须小心，谨记隐私法律。

另一方面，聚集后的隐私数据在许多任务中可以扮演重要的角色，比如检测药物的副作用，或者探查流行病的蔓延。如何使这些数据可以为执行这种任务的研究人员所用，而又不侵犯个人的隐私，

这是一个重要的现实问题。例如，假定一家医院隐藏了患者的名字，但是给研究人员提供了患者的出生日期和邮政编码(这二者可能对研究人员都有用)。在很多情况下，仅使用这两个信息就可以唯一标识患者(使用外部数据库中的信息)，侵犯了他的隐私。在这个特定情况下，一个解决方法是随邮政编码一起只提供出生年份而不提供出生日期，这对于研究人员可能就足够了。对大多数人，这将不足以唯一确定一个人。[⊖] 410

另举一个例子，Web 站点通常会收集个人数据，比如地址、电话、电子邮件以及信用卡信息。这种信息在执行交易时可能会需要，比如从商店购买商品。但是，顾客可能不希望这些信息公开给其他组织，或者可能希望其中部分信息(比如信用卡号码)能够在一段时间之后清除，避免它们在发生安全问题时落入未授权的人手中。许多 Web 站点允许客户指定他们的隐私偏好，并且必须确保遵守这些偏好设置。

9.8 加密及其应用

加密是指将数据转换成一种除非使用反过程解密否则不可读的形式的过程。加密算法使用一个加密密钥执行加密，并需要一个解密密钥(它可以和加密密钥相同，这是由使用的加密算法决定的)来执行解密。

加密最早是用于发送消息，用一个只有发送者和接收者知道的密钥加密消息。即使消息被敌方截获，不知道密钥的敌方也将无法解密并理解该消息。今天，加密广泛使用，它在多种应用中保护传输中的数据，比如互联网中的数据传输，以及移动电话网络中的数据传输。我们将在 9.8.3 节中看到，加密还用于执行其他一些任务(比如身份验证)。

在数据库方面，加密用来以一种安全的方式存储数据，从而即使数据被一个未授权的用户获取(例如，一个包含数据的笔记本电脑被偷)，如果没有解密密钥的话数据也无法访问。

目前，许多数据库都存储敏感客户信息，比如信用卡号码、姓名、指纹、签名以及身份证号码，例如在美国就是社会保险号。一个获取了这种数据的访问权限的罪犯能够将其用于多种非法活动，比如使用一个信用卡号码购买物品，或者甚至用他人的名字申请信用卡。诸如信用卡公司的机构利用个人信息来识别谁在请求服务或物品。这种个人信息的泄露使罪犯可以假冒其他人并获取服务和物品；这种假冒称作**身份盗窃**(identity theft)。因此，存储这种敏感数据的应用程序必须非常小心地保护它们不被盗用。 411

为了减少敏感信息被罪犯获取的机会，目前许多国家和州通过法律要求任何存储这种敏感信息的数据库必须以加密形式存储信息。因此，一个不保护其数据的企业一旦发生数据盗用，就可以被追究刑事责任。因而，加密是任何存储这种敏感信息的应用程序的重要部分。

9.8.1 加密技术

加密数据的技术数不胜数。简单的加密技术可能无法提供足够的安全性，因为未授权用户可能会轻易地就将编码破译。作为弱加密技术的例子，考虑用字母表中的下一个字母替代每个字母的情况。这样，

Perryridge

就变成了

Qfsszsjehf

如果未授权用户仅仅看到“Qfsszsjehf”，她可能还不具有充足的信息破译编码。但是，如果侵入者看到大量加密后的支行名称，她就能够使用关于字符相对频率的统计数据来猜测所做的替代是如何的(例如，*E* 是英语中最常用的字母，接下来是 *T*、*A*、*O*、*N*、*I* 等)。

一个好的加密技术具有如下性质：

⊖ 对于年纪特别大的人，由于比较稀少，即使只是出生年份加上邮政编码都足以唯一确定一个人，因此对于 80 岁以上的人，可以提供值的一个范围，比如 80 岁及以上，而不是其实际年龄。

- 对于授权用户，加密数据和解密数据相对简单。
- 加密模式不应依赖于算法的保密，而应依赖于被称作加密密钥的算法参数，该密钥用于加密数据。在**对称密钥**(symmetric-key)加密技术中，加密密钥也用于解密数据。相反，在**公钥加密**(public-key)(也称作**非对称密钥**(asymmetric-key))加密技术中，存在两个不同的密钥，公钥和私钥，分别用于加密和解密数据。
- 对入侵者来说，即使当他已经获得加密数据的访问权限了，确定解密密钥仍是极其困难的。在非对称密钥加密的情况中，即使已有公钥，推断出私钥也是极其困难的。

扩展加密标准(Advanced Encryption Standard，AES)是一种对称密钥加密算法，它作为一种加密标准于2000年被美国政府所采用，并且目前广泛使用。该标准基于**Rijndael算法**(Rijndael algorithm)(根
412 据发明人V. Rijmen和J. Daemen命名)。该算法每次对一个128位的数据块操作，密钥的长度可以是128、192或256位。该算法运行一系列步骤，以一种解码时可逆的方式将数据块中的位打乱，并将其与一个来自加密密钥的128位的"回合金钥"做异或操作。对于每一个加密的数据块都由加密密钥生成一个新的回合金钥。在解密过程中，再次根据加密密钥生成回合金钥，并且逆转加密过程从而恢复原始数据。一个称作数据加密标准(Data Encryption Standard，DES)的早期标准于1977年采用，并在早期广泛使用。

对于任意对称密钥加密模式的使用，授权用户通过一个安全机制得到加密密钥。这个要求是它的一大弱点，因为模式的安全不高于加密密钥传输机制的安全。

公钥加密(public-key encryption)是另一种模式，它避免了对称密钥加密技术面临的一些问题。它基于两个密钥：一个公钥和一个私钥。每个用户U_i有一个公钥E_i和一个私钥D_i。所有公钥都是公开的：任何人都可以看到。每个私钥都只由拥有它的用户知道。如果用户U_1想要存储加密数据，U_1就使用公钥E_1加密数据。解密需要私钥D_1。

因为加密密钥对每个用户公开，所以我们有可能利用这一模式安全地交换信息。如果用户U_1希望与U_2共享数据，那么U_1就用U_2的公钥E_2来加密数据。由于只有用户U_2知道如何对数据解密，因此信息可以安全地传输。

要使公钥加密发挥作用，在给定公钥后，必须有一个让人很难推断出私钥的加密模式。这样的模式确实存在，并且建立在如下条件基础之上：

- 存在一个高效算法，测试某个数字是否为素数。
- 对于求解一个数的素数因子，没有高效算法。

为了这一模式，数据被看作一组整数。我们通过计算两个大素数P_1和P_2的积来创建公钥。私钥由(P_1，P_2)对构成。如果只知道乘积P_1P_2，解密算法无法使用成功；它需要P_1和P_2各自单独的值。由于公开的只是乘积P_1P_2，因此未授权用户为了窃取数据就需要对P_1P_2做因数分解。通过将P_1和P_2选得足够大(超过100位)，我们可以使对P_1P_2做因数分解的代价极高(即使在最快的计算机上，计算时间也需要以年来计算)。

关于公钥加密的细节以及该技术性质的数学证明请参见文献注解。

尽管使用上述模式的公钥加密是安全的，但是它的计算代价很高。一种广泛用于安全通信的混合
413 模式如下：随机产生一个对称加密密钥(例如基于AES)，使用公钥加密模式以一种安全的方式交换，并使用该密钥对随后传输的数据进行对称密钥加密。

词典攻击(dictionary attack)使得对诸如标识符或名字等小值的加密变得复杂，特别是当加密密钥公开时。例如，如果生日域被加密了，当一个攻击者试图对一个特定的加密值e解密时，他可以试着对任何可能的生日加密，直到他找到一个生日，其加密后的值与e匹配。即使当加密密钥未公开，也可以利用数据分布的统计信息找出一个加密的值在一些情况下代表什么，比如年龄或邮政编码。例如，如果在数据库中18岁是最普遍的年龄，则加密后的年龄值中出现最多的通常可推断出代表18。

可以通过在加密之前往值的末尾添加额外随机位(并在解码后将其删除)来防止词典攻击。这种额外位在AES中称为**初始化矢量**(initialization vector)，或在其他情况下称为*salt*位，它能面对词典攻击提供良好的保护。

9.8.2 数据库中的加密支持

当前，许多文件系统和数据库系统都支持数据加密。这种加密使数据免受那些能够访问数据却无法访问解密密钥的人的攻击。在文件系统加密的情况下，要加密的数据通常是大文件以及包含文件信息的目录。

而在数据库的情况下，加密可以在多个不同级别上进行。在最低的级别上，使用数据库系统软件的密钥，可以加密包含数据库数据的磁盘块。当从磁盘上获取一个磁盘块时，它先被解密然后以平常的方式使用。这种磁盘块级别的加密抵御了那些能访问磁盘内容但不能访问密钥的攻击者。

在一个高级别上，关系中指定的(或所有的)属性可以用加密的形式存储。在这种情况中，关系的每个属性可以具有不同的加密密钥。目前，许多数据库支持指定属性级别的加密以及整个关系所有级别或者数据库中所有关系的加密。指定属性的加密通过让应用程序只对诸如信用卡号码等包含敏感值的属性加密，最小化解密的开销。然而，虽然单个属性或关系可以加密，但是数据库通常不允许主码和外码属性加密，并不支持加密属性上的索引。如前文所述，加密还需要使用额外随机位来防止词典攻击。

为了访问加密的数据，显然需要一个解密密钥。单个主加密密钥可能用于所有的加密数据；在属性级别的加密中，可以对不同的属性使用不同的加密密钥。在这种情况下，可以将不同属性的解密密 414
钥保存在一个文件或关系中(通常称为"钱夹")，它本身也用主密钥加密。

一个需要访问加密属性的数据库的连接必须提供主密钥；除非提供了主密钥，否则该连接将无法访问加密数据。主密钥要保存在应用程序中(通常在一台不同的计算机上)，或者由数据库用户记住，并在用户连接到数据库时提供。

数据库级别的加密具有需要相对小的时空开销的优点，并且不需要对应用程序进行修改。例如，如果笔记本电脑数据库中的数据需要防范来自计算机本身的窃贼，就可以使用这种加密。类似地，访问数据库备份磁带的人，如果没有解密密钥则将不能访问磁带中包含的数据。

在数据库中执行加密的另一个方法是在数据发送到数据库之前对其加密。于是，应用程序则必须在将数据发送给数据库之前对其加密，并当获取到数据时对其解密。与在数据库系统中执行加密不同，这种数据加密方法需要对应用程序进行大量的修改。

9.8.3 加密和认证

基于密码的认证广泛用于操作系统和数据库。然而，密码的使用具有一些缺陷，特别是在网络上的时候。如果一个窃听者能够"窃听"网络上传送的数据，她就可能能够当密码在网络上传送时找到密码。一旦窃听者具有用户名和密码，她就可以假装成合法用户连接到数据库。

一个较为安全的机制包括一个**询问 - 回答**(challengE-Response)系统。数据库系统发送一个询问字符串给用户，用户用一个密码作为加密密钥对询问字符串加密，然后返回结果。数据库系统可以通过用同样的密码把字符串解密并检查结果是不是和原始询问字符串一样来验证用户的身份。这个机制确保没有密码在网络上传输。

公钥系统可以在询问 - 回答系统中用于加密。数据库系统使用用户的公钥加密询问字符串，并把它发送给用户。用户用她的私钥对字符串解密，并把结果返回给数据库系统。数据库系统随后检查该应答。这个方案具有新增的优势，它不在可能被系统管理员看到的数据库中存储密码。

将一个用户的私钥存储在计算机(甚至是个人计算机)中是有风险的，如果该计算机受到攻击，密钥可能会暴露给攻击者，该攻击者就可以假冒该用户。**智能卡**(smart card)对于该问题提供了一种解决方案。在一张智能卡中，密码可以存储在一块嵌入的芯片中；智能卡的操作系统保证该密码绝对不会 415
被读取，但是允许数据被发送至卡上，使用私钥[⊖]进行加密或解密。

9.8.3.1 数字签名

公钥加密的另一个有趣的应用是**数字签名**(digital signature)，它用来验证数据的真实性；数字签名

⊖ 智能卡也提供了其他功能，比如数字化现金存储和支付，不过这与我们的讨论无关。

扮演文档上物理签名的电子角色。私钥用来对数据“签名”，即加密，且签名后的数据可以公开。所有人都可以通过用公钥解密数据来验证签名，但没有私钥的人无法生成签名的数据。(注意在这个方案中公钥和私钥角色的互换)这样，我们就可以认证(authenticate)该数据；也就是我们可以验证数据确实是由声称创建这些数据的人所创建。

另外，数字签名也可以用来确保认可(nonrepudiation)。也就是，在一个人创建了数据过后声称她没有创建它(声称没有签支票的电子等价)的情况下，我们可以证明那个人肯定创建了该数据(除非她的私钥被泄露给其他人)。

9.8.3.2 数字证书

通常认证是一个双向的过程，交互实体的双方都要向对方认证自己的身份。这种成对的认证是很有必要的即使当客户端访问一个 Web 站点时，可以防止一个恶意站点冒充成一个合法的 Web 站点。例如，如果网络路由器被攻击了，数据被重新路由到恶意站点的时候，这种冒充就可能发生。

为了确保用户与真实的 Web 站点交互，她必须拥有该站点的公钥。这带来了一个问题：该用户如何获取公钥——如果公钥存储在该 Web 站点上，恶意站点也可以提供一个不同的密钥，用户将无法验证提供的公钥是否是真实的。认证可以通过**数字证书**(digital certificate)系统来处理，公钥由一个公钥公开的认证机构签名。例如，根认证机构的公钥保存在标准的 Web 浏览器中。它们签署的证书可以使用保存的公钥来验证。

两级系统将会给根认证机构带来创建证书这一巨大负担，因此采用多级系统取而代之，该系统有一个或多个根认证机构，并在每个根认证机构下有一棵认证机构树。每个机构(除了根机构)都有其父机构签署的一个数字证书。

416 认证机构 A 签署的一个数字证书由一个公钥 K_A 和一个可以用公钥 K_A 解密的加密文本 E 构成。该加密文本包括该证书签发给的团体名称以及它的公钥 K_C。在认证机构 A 不是根认证机构的情况下，该加密文本还包含由其父认证机构签发给 A 的数字证书；这个证书认证了密钥 K_A 本身。(该证书可能依次包含上一级父机构的证书，直至根机构)

要验证一个证书，加密文本 E 通过使用公钥 K_A 解密从而获得团体名称(即拥有该网站的机构的名称)；另外，如果 A 不是一个根机构，其公钥 K_A 会递归地使用 E 中包含的数字证书进行验证；递归在由根机构签发的证书到达时终止。对证书的验证建立起一个认证特定站点的链，并且提供了站点的名称以及认证的公钥。

数字证书广泛用于为用户认证 Web 站点，以防止恶意站点冒充成其他 Web 站点。在 HTTPS 协议(HTTP 协议的安全版本)中，站点向浏览器提供它的数字证书，然后浏览器将证书显示给用户。如果用户接受该证书，浏览器则使用提供的公钥来加密数据。一个恶意的站点将能够访问该证书，但是不能访问私钥，因此无法解密浏览器发送的数据。只有拥有相应私钥的真实站点，能够解密浏览器发送的数据。我们注意到公钥/私钥加密和解密的代价远远高于使用对称私钥进行加密/解密的代价。为了减少加密代价，HTTPS 实际上在认证之后创建一个一次性的对称密钥，并在随后的会话中采用该对称密钥加密数据。

数字证书也可以用于认证用户。用户必须向站点提交一个包含她的公钥的数字证书，该站点验证该证书是由一个可信的机构签发。而后该用户的公钥则可以用在询问 - 应答系统中，以确保用户拥有相应的私钥，并以此认证用户。

9.9 总结

- 在后端使用数据库并与用户交互的应用程序自 20 世纪 60 年代以来一直在使用。应用程序架构在此期间不断发展。目前，大多数应用程序都使用 Web 浏览器作为它们的前端，数据库作为它们的后端，以及一个应用服务器介于其间。
- HTML 提供了定义将超链接与表单功能相结合的界面的能力。Web 浏览器通过 HTTP 协议与 Web 服务器通信。Web 服务器可以将请求传递给应用程序，并将结果返回浏览器。

417
- Web 服务器执行应用程序以实现所需的功能。servlet 是一个广泛使用的机制，它用于编写能够作为

Web 服务器进程的一部分运行的应用程序，从而减少开销。另外还存在几种由 Web 服务器解释的服务器端脚本语言，作为 Web 服务器的一部分提供应用程序功能。

- 有多种客户端脚本语言——Javascript 使用最为广泛——在浏览器端提供更丰富的用户交互。
- 复杂的应用程序通常具有一个多层的架构，包括一个实现业务逻辑的模型、一个控制器，以及一个用于显示结果的查看机制。它们可能还包括一个实现了对象 - 关系映射的数据访问层。许多应用程序实现并使用 Web 服务，允许在 HTTP 上调用函数。
- 目前已经开发出了许多工具用于快速应用开发，特别是用来减少构建用户界面所需的工作。
- 诸如多种形式的缓存(包括查询结果缓存和连接池)以及并行处理的技术用于提高应用程序性能。
- 应用程序开发人员必须小心注意安全问题，以防止受到攻击，比如 SQL 注入攻击及跨站点脚本攻击。
- SQL 授权机制是粗粒度的，并且对于处理大量用户的应用只具有有限的价值。目前，应用程序完全在数据库系统之外实现了细粒度的、元组级别的授权，以处理大量应用程序用户。提供元组级访问控制和处理大量应用程序用户的数据库扩展已开发出，但还没有成为标准。
- 保护数据的隐私是数据库应用的一项重要任务。许多国家都有法律规定保护某些类型的数据，比如信用卡信息或医疗数据。
- 加密在保护信息以及认证用户和 Web 站点中扮演了关键角色。对称密钥加密和公钥加密是两种相对立但广泛应用的加密方法。在许多国家和州，对存储在数据库中的某些敏感数据的加密是法律规定的。
- 加密还在为应用程序认证用户、为用户认证 Web 站点以及数字签名中扮演了关键角色。

术语回顾

- 应用程序
- 数据库的 Web 界面
- 超链接
- 统一资源定位符(URL)
- 表单
- 超文本传输协议(HTTP)
- 公共网关接口(CGI)
- 无连接协议
- cookie
- 会话
- servlet 及 servlet 会话
- 服务器端脚本
- JSP
- PHP
- ASP. NET
- 客户端脚本
- Javascript
- 文档对象模型(DOM)
- 小应用程序
- 应用程序架构
- 展示层
- 模型 - 视图 - 控制器(MVC)架构
- 业务逻辑层
- 数据访问层
- 对象 - 关系映射
- Hibernate
- 超文本标记语言(HTML)
- Web 服务
- REST 服务
- 快速应用开发
- Web 应用框架
- 报表生成器
- 连接池
- 查询结果缓存
- 应用程序安全性
- SQL 注入
- 跨站点脚本(XSS)
- 跨站点请求伪造(XSRF)
- 认证
- 双因素认证
- 中间人攻击
- 中央认证
- 单点登录
- OpenID
- 虚拟私有数据库(VPD)
- 审计追踪
- 加密
- 对称密钥加密
- 公钥加密
- 词典攻击
- 询问 - 回答
- 数字签名
- 数字认证

实践习题

9.1 虽然 Java 程序通常要比 C 或 C ++ 程序运行得慢，而 servlet 却比使用公共网关接口(CGI)的程序性能好的主要原因是什么？

9.2 列出无连接协议相比较有连接协议的优缺点。

9.3 考虑为在线购物系统写的一个未经仔细编写的 Web 应用程序，该应用程序将每件商品的价格以隐藏的表单变量保存在发送给顾客的页面中；当顾客提交表单时，隐藏的表单变量中的信息用于计算顾客的账单。这个设计中的漏洞在哪里？(有一个真正的实例，其中在探测并解决问题之前，在线购物系统

418 / 419

的顾客使该漏洞暴露了出来。)

9.4 考虑未经仔细编写的另一个 Web 应用程序，它使用 servlet 检查是否有活动的会话，但并不检查用户是否授权访问某页面，而依靠指向页面的连接只显示给已授权用户的事实。这种机制下的缺陷是什么？(有一个真正的实例，其中某个学校招生站点的申请者在登录网站后可以利用这个漏洞，并且可以查看他们没有授权查看的信息；然而，此未授权的访问被探测出来了，访问该信息的人由于违反招生规定而被惩罚。)

9.5 列出使用缓存提高 Web 服务器性能的三种方式。

9.6 netstat 命令(在 Linux 和 Windows 中可用)显示计算机上活动的网络连接。试解释如何使用该命令以发现是否某网页将不关闭它所打开的连接，或者是否使用了连接池并且不将连接返回连接池。你应该考虑到使用连接池的时候，连接可以不立即关闭。

9.7 SQL 注入漏洞的检测：

a. 提出一个检测应用程序的方法，以发现在文本输入上是否容易受到 SQL 注入攻击。

b. SQL 注入能够与其他形式的输入一同出现吗？如果可以，你要如何检测漏洞呢？

9.8 为了安全，一个数据库关系可能对某些属性的值加密。数据库系统为什么不支持加密属性上的索引呢？用你对这个问题的答案解释为什么数据库系统不允许主码属性上的加密。

9.9 练习 9.8 提出了在某些属性上加密的问题。然而，一些数据库系统支持整个数据库的加密。试解释练习 9.8 中提出的问题在加密整个数据库时该如何避免。

9.10 假设某人假冒一个公司并得到了一个证书授予机构颁发的证书。这对于被假冒公司认证的事物(例如购买订单或者程序等)，以及对于其他公司认证的事物有什么影响？

420 9.11 在任何数据库系统中，最重要的数据项或许都是用来控制数据库访问的密码。为密码的安全存储设计一个机制，确保你的机制允许系统检测由试图登录系统的用户所提供的密码。

习题

9.12 为下面这个非常简单的应用写一个 servlet 以及关联的 HTML 代码：允许用户提交一个表单，该表单包含一个值，比如说 n，然后得到的响应包含 n 个“*”号。

9.13 为下面这个简单应用写一个 servlet 以及关联的 HTML 代码：允许用户提交一个表单，该表单包含一个数字，比如说 n，然后得到一个响应说出数值 n 在此前提交的次数。每个数值此前提交的次数应当存储在数据库中。

9.14 写一个对用户进行认证的 servlet(基于存储在数据库关系中的用户名和密码)，并在验证通过之后设置一个名为 *userid* 的会话变量。

9.15 什么是 SQL 注入攻击？解释它的工作原理，以及必须采取什么措施以预防 SQL 注入攻击。

9.16 编写管理连接池的伪码。你的伪码必须包括一个创建连接池的函数(以数据库连接字符串、数据库用户名和密码作为输入参数)，一个向连接池请求连接的函数，一个将连接归还连接池的函数以及关闭连接池的函数。

9.17 解释术语 CRUD 和 REST。

9.18 目前，许多网站都使用 Ajax 提供丰富的用户界面。列出两个不需要看源码即可表明站点使用 Ajax 的特点。利用上述特点，找到三个使用 Ajax 的站点；你可以查看网页的 HTML 源码以检验该网站是否真正使用了 Ajax。

9.19 XSS 攻击：

a. XSS 攻击是什么？

b. 如何使用引用页域以探查 XSS 攻击？

9.20 什么是多因素认证？它是如何帮助防止密码被盗的？

9.21 考虑 9.7.5 节所述的 Oracle **虚拟私有数据库**(VPD)功能，以及一个基于大学模式的应用。

421
- 应该生成什么谓词(使用子查询)以使每个教员只看见和他们所教课程相关联的 *takes* 元组。
- 给出一个 SQL 查询，使得添加了谓词的查询的结果是未添加谓词的原始查询的结果的子集。
- 给出一个 SQL 查询，使得添加了谓词的查询的结果包含一条不在未添加谓词的原始查询的结果中的元组。

9.22　对存储在数据库中的数据加密有哪两个好处？

9.23　假定你希望对关系 *takes* 的变化建立审计追踪：

a. 定义触发器来建立审计追踪，把日志信息记入一个关系，例如 *takes_trail*。日志信息应包括用户 id（假设函数 *user_id*( ) 提供该信息）和一个时间戳，以及旧的和新的值。你还要提供 *takes_trail* 关系的模式。

b. 以上实现方法是否能保证由恶意数据库管理员（或者那些设法得到管理员密码的人）所做的更新也会被审计追踪。解释你的答案。

9.24　黑客能够欺骗你相信他们的 Web 站点实际上是你所信任的一个 Web 站点（比如一家银行或者信用卡 Web 站点）。他们可以使用误导性的电子邮件，或者甚至通过入侵网络基础设施将目的地为（比如）mybank. com 的网络传输重新路由到黑客的网站。如果你在黑客的站点上输入用户名和密码，该站点就会把它记录下来，以后就能用它来侵入你在真实站点的账户。当你使用一个 URL，比如 https: // mybank. com，HTTPS 协议可以用来防止这样的攻击。解释该协议如何使用数字证书来验证站点的合法性。

9.25　解释什么是用于验证的询问 - 应答系统。为什么它比传统的基于密码的系统更安全？

项目建议

下面都是较大的项目，可以由一组学生在一个学期内完成。项目的难度可以通过增加或者减少功能来调整。

项目 9.1　选一个你最喜爱的交互网站，比如 Bebo、Blogger、Facebook、Flickr、Last. FM、Twitter、WikiPedia；这只是一些例子，还有更多其他的交互网站。这些网站中的大部分都管理大量的数据，并使用数据库存储并处理这些数据。实现你挑选的一个网站的功能的子集。需要明确的是，即使实现这种网站的功能的一个有意义的子集也远超过了一个课程设计要求的工作量，不过可以找一些有趣的功能来实现，足够满足课程设计的工作量即可。 422

目前流行的网站大部分都大量使用 JavaScript 创建丰富的界面。搭建这种界面非常耗时，所以至少在最初你可能会希望用它实现你的项目的界面，然后在时间允许的情况下再添加更多的功能。利用 Web 应用开发框架或网上的 Javascript 库，比如 Yahoo 用户界面库，来加快你的开发。

项目 9.2　创建一个“混合网站”，即用诸如 Google 或 Yahoo 地图 API 的 Web 服务创建一个交互网站。例如，这些地图 API 提供一种在网页上展示地图及在地图上重叠的其他信息的方法。你可以实现一个餐厅推荐系统，使用用户提交的餐厅信息，比如位置、美食、价格范围以及评分信息。用户搜索的结果可以在地图上显示。你可以允许类 Wikipedia 的功能，比如让用户可以添加信息并编辑其他用户添加的信息，并安排可以删除恶意更新的审阅人。你还可以实现社区功能，比如给你的朋友提供的评分更高的权重。

项目 9.3　你的学校可能使用了一个课程管理系统，比如 Moodle、Blackboard 或 WebCT。实现这样一种课程管理系统的功能的子集。例如，你可以提供作业提交和评分功能，包括为学生和老师/助教提供讨论作业的评分的机制。你还可以提供问卷调查或其他机制以获得反馈。

项目 9.4　考虑实践习题 7.3（第 7 章）中的 E-R 模型，它表示一个联赛中各队的信息。设计并实现一个基于 Web 的系统来输入、更新和查看这些数据。

项目 9.5　设计并实现一个购物车系统，该系统可以让购买者将商品放置到购物车中（你可以决定为每件商品提供的信息），最后一起购买。你可以扩展并使用第 7 章中习题 7.20 中的 E-R 模式。你应该检查商品是否有货并用你觉得适合的方式处理商品缺货的情况。

项目 9.6　设计并实现一个基于 Web 的系统为一所大学的课程记录学生注册信息和成绩信息。

项目 9.7　设计并实现一个系统记录课程表现信息——具体来说，即每个学生在每门课程中的每次作业或考试中所获得的分数，以及对各项成绩进行（加权）求和以得到课程总成绩。作业/考试的数量不可以预先定义；即，任何时候都可以加入更多的作业或考试。该系统还应该可以支持等级评定，允许对于不同的级别指定分隔点。 423

你也可以将它与项目 9.6 中的学生注册系统（可能由另一个项目组实现）相结合。

项目 9.8 设计和实现一个基于 Web 的系统，用来在你的学校中预订教室。它必须支持周期性的预订（整个学期中每周固定的天/时间），也要支持在周期性预订中取消某个指定的讲座。

你也可以将它与项目 9.6 中的学生注册系统（可能由另一个项目组实现）相结合，这样可以为课程预订教室，取消讲座或新增额外讲座可以通过一个单独的界面来记录，并且要将其反映到教室预订情况中，然后通过电子邮件通知给所有的学生。

项目 9.9 设计并实现一个系统，管理在线的多选题考试。你应该支持分布式的考题提交（由助教提交），负责该课程的人可以编辑题目，并且可以从已有的考题集合中创建试卷。你还应该能够在线管理考试，可以让所有学生都在一个固定的时间参加考试；或者让学生在任何时间都可以开始考试，但是从开始到结束有一个时间限制（支持一种或者两种方案都支持），并且规定时间结束时将成绩反馈给学生。

项目 9.10 设计并实现一个系统，管理电子邮件顾客服务。到达的电子邮件进入一个公用的缓冲池中。一组顾客服务代理人负责答复电子邮件。如果某封电子邮件是正在答复的序列的一部分（用电子邮件的 in-reply-to 域来跟踪），这个邮件应该由早先答复它的那个代理人来答复。系统应该追踪所有到达的邮件和答复，这样代理人可以在答复一封电子邮件之前看到顾客以前问过的所有问题。

项目 9.11 设计并实现一个简单的电子商场，该电子商场可以按照不同的类别（应该形成一个层次结构）列出用于销售或购买的商品。你可能还希望支持提醒服务：一个用户可以在某一个特定类别中注册自己感兴趣的商品，也许还有一些其他限制，而不公开她的兴趣，当这样的商品促销时，系统会通知该用户。

项目 9.12 设计并实现一个基于 Web 的新闻组系统，用户应该可以订阅新闻组，并且浏览新闻组中的文章。该系统跟踪用户阅读过的文章使它们不会再次显示。该系统还提供对旧文章的搜索支持。你可能还希望提供文章的评分服务，这样，高分的文章高亮度显示，从而使繁忙的用户可以跳过低分的文章。

424 项目 9.13 设计并实现一个基于 Web 的系统，管理一个运动“梯子”。许多人来注册，并给予初始的排名（可能基于以前的表现）。任何人可以通过比赛挑战其他人，而排名按照比赛结果进行相应的调整。一个调整排名的简单系统仅是当胜者原先排在败者后面时，在排名中将胜者移到败者的前面。你也可以尝试实现更加复杂的排名调整系统。

项目 9.14 设计并实现一个出版物列表服务。这个服务应该允许输入有关出版物的信息，例如书号、作者、年份、出版物出现地点、页数等。作者应该是一个具有属性（如名字、机构、部门、电子邮件、地址和主页）的单独实体。

你的应用应该支持同一数据的多个视图。例如，你应该提供某个给定作者的所有出版物（比如按年份排序），或者提供来自某个给定机构或部门的作者的所有出版物。你也应该支持在整个数据库或每个视图上按关键词搜索。

项目 9.15 所有组织中的一项常见任务是收集一组人的结构化信息。例如，一个经理可能需要要求职员输入他们的休假计划；一个教授可能希望从学生中收集关于某个特定主题的反馈；或者一个组织活动的学生希望允许其他学生注册活动，或者某人希望针对某个主题进行一个在线投票。

建立一个允许用户很容易创建信息收集活动的系统。当创建一个活动时，该活动的创建人必须定义谁是有资格参加的，为了达到这个目的，你的系统必须维护用户信息，并且允许定义一个用户子集的谓词。活动的创建人应当能够指定一组用户需要提供的输入（有类型、默认值和有效性检查）。活动应当关联一个截止时间，并能够向尚未提交信息的用户发送通知。活动创建者可以选择是在指定的日期或时间自动执行截止，还是登录系统然后宣布截止。提交信息的统计信息能够生成——为了达到这个目的，应当允许活动创建者对输入的信息建立简单的汇总。活动创建者可以选择将部分汇总信息公开以让所有用户都能查看，要么持续地（比如有多少人已经响应），要么在截止期之后（比如平均回馈分值）。

项目 9.16 建立一个函数库以简化 Web 界面的生成。你必须至少实现以下函数：显示 JDBC 结果集（以表格形式）的函数，创建不同种类的文本和数字输入（带有验证规则，比如输入类型和可选

范围，在客户端用适当的 Javascript 代码执行）的函数，输入日期和时间值（带有默认值）的函数，基于结果集生成菜单项的函数。要得到加分，可以允许用户设置格式参数，比如颜色和字体，以及在表格中提供分页支持（隐藏的表单参数可以用来指示需要显示的页）。建立一个数据库应用样例以展示这些函数的使用。 425

项目 9.17 设计并实现一个基于 Web 的多用户日历系统。该系统需追踪每个用户的约会，包括多发事件，如周会，及共享事件（事件创建者更新事件将反映至所有分享该事件的用户）。系统需提供安排多用户事件的界面，该界面允许事件创建者添加受邀请的用户。系统需提供事件的电子邮件通知。要得到加分，可以实现一个 Web 服务，该服务可以用在客户端上运行的提醒程序中。

工具

开发一个 Web 应用需要多个软件工具，比如应用服务器、编译器、像 Java 或 C#这样的程序设计语言的编辑器以及其他可选的工具，比如 Web 服务器。有一些对 Web 应用开发提供支持的集成开发环境。两款最流行的开源 IDE 为 IBM 开发的 Eclipse 以及 Sun 微系统开发的 Netbeans。微软的 Visual Studio 是在 Windows 环境中使用最为广泛的 IDE。

支持 servlet 和 JSP 的应用服务器包括 Apache Tomcat（jakarta. apache. org）、Glassfish（glassfish. dev. java. net）、JBoss（jboss. org）以及 Caucho 的 Resin（www. caucho. com）。Apache Web 服务器（apache. org）是目前使用最为广泛的 Web 服务器。微软的 IIS（Internet Information Services）是一款广泛用于微软 Windows 平台并支持微软的 ASP. NET（msdn. microsoft. com/asp. net/）的 Web 兼应用服务器。

IBM 的 WebSphere（www. software. ibm. com）为 Web 应用开发和部署提供了许多软件工具，包括应用服务器、IDE、应用集成中间件、业务流程管理软件以及系统管理工具。

上述工具中有些是开源软件可以免费使用，有些对于非商业用途或个人使用是免费的，而另外一些则需要付费。更多信息可查看各相关网站。

Yahoo！用户界面（YUI）的 JavaScript 库（developer. yahoo. com/yui）广泛用于创建跨浏览器的 JavaScript 程序。

文献注解

有关 servlet 的信息，包括指南、标准说明和软件，可以在 java. sun. con/products/servlet 上找到。有关 JSP 的信息可以在 java. sun. com/products/jsp 上找到。关于 JSP 标签库的信息也能在该 URL 上找到。关于 . NET 框架的信息以及关于使用 ASP. NET 进行 Web 应用开发的信息可以在 msdn. microsoft. com 上找到。

Atreya 等[2002]提供了涵盖数字签名的教科书，其中包括 X. 509 数字证书以及公钥基础设施。 426