Velocity 网页程序设计

李晓黎　张晓辉　编著

人 民 邮 电 出 版 社

图书在版编目(CIP)数据

Velocity 网页程序设计/李晓黎，张晓辉编著．—北京：人民邮电出版社，2001.11
ISBN 7-115-09828-X

Ⅰ．V… Ⅱ．①李… ②张… Ⅲ．Java 语言—程序设计 Ⅳ．TP312

中国版本图书馆 CIP 数据核字(2001)第 077686 号

内 容 提 要

Velocity 是一个基于 Java 的模板引擎，它可以独立产生源代码、HTML 和报告，也可以与其他系统（或应用）相结合提供模板服务。

本书重点介绍了 Velocity 模板语言 VTL（Velocity Template Language）和 Velocity 应用程序开发技术。Velocity 的模板语言非常简单，它并没有复杂的数据类型和语法结构，即使没有编程经验的读者也可以轻松地掌握。但是，要学习 Velocity 应用程序开发技术，必须对 Java 程序设计有所了解，因此本书对 Java 程序设计也进行了简单的介绍。

本书适合从事网页设计和网站开发建设工作的人员使用，也可以作为对此感兴趣的读者了解这一领域新技术的参考书。

Velocity 网页程序设计

◆ 编　　著　李晓黎　张晓辉
责任编辑　邹文波

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号

邮编　100061　　电子函件　315@pptph.com.cn
网址　http://www.pptph.com.cn
读者热线：010-67129212　010-67129211（传真）

北京汉魂图文设计有限公司制作

北京顺义向阳胶印厂印刷

新华书店总店北京发行所经销

◆ 开本：787×1092　1/16

印张：22

字数：535 千字　　2001 年 11 月第 1 版

印数：1 – 0 000 册　　2001 年 11 月北京第 1 次印刷

ISBN 7-115-09828-X/TP· 2577

定价：32.00 元

本书如有印装质量问题，请与本社联系　电话：（**010**）**67129223**

前　言

Velocity 是 Jakarta Project 的一部分，Jakarta Project 的目标是提供基于 Java 平台的商用的服务器解决方案，因此这些方案将具有开放和协作的开发风格。

Velocity 是一个基于 Java 的模板引擎，它可以独立产生源代码、HTML 和报告，也可以与其他系统（或应用）相结合提供模板服务。由于 Web 页面设计者可以使用简单而功能强大的 Velocity 模板语言引用 Java 代码中定义的对象，因此 Velocity 实际上是充当 Java 代码和网页之间的一个分离层。这样网页设计者就可以把注意力集中在美化网页上，无需考虑网页如何工作；而程序员则着重于编写 Java 代码，支持站点的功能。这样分工协作的模式很适合于构建较大规模的网站，而对于个人用户而言，这使得程序设计的条理更加清晰，也同样是一种明智的选择。

Velocity 把 Java 代码和页面分离开来，Web 设计者能够和 Java 程序员根据 MVC 模块（Model-View Controller）并行设计一个 Web 站点，从而使整个网站更加容易维护，是 JSP 和 PHP 更好的替代品。

在笔者着手编写这本书的时候，Velocity 1.0 刚刚发布不久，在国外已经拥有了相当规模的用户群，但是在我国，绝大多数网页设计人员对 Velocity 并不了解，也就更无从体验它的强大功能了。正是出于这样的考虑，特编写了此书，向广大读者介绍 Velocity 网页程序设计和管理技术，从中也可以了解到这一领域发展的最新方向。

本书从很多方面将 Velocity 和 JSP、PHP 进行比较，正在使用 JSP 或 PHP 进行网页程序设计的读者，可以更加直观地了解 Velocity 的特点，更方便地学习掌握 Velocity 网页程序设计技术。

本书重点介绍了 Velocity 模板语言 VTL（Velocity Template Language）和 Velocity 应用程序开发技术。网页设计者不但可以使用 Velocity 模板语言设计网页的外观，而且可以方便地访问 Java 程序中的数据；Java 程序员则可利用 Velocity 提供的 Java 开发包开发 Java 程序，实现网站的功能，并将数据结果通过 Velocity 传送给网页设计者，从而在网页中以适当的格式反映出来，呈现给浏览者。

要在网站中实际应用 Velocity，还需要 Java Servlet 的支持。Java Servlet 是协议和平台独立的服务器端 Java 应用程序，可用于创建可扩展的 Web 应用程序。为了便于读者在实际工作中更加方便地使用 Velocity，本书还以 Tomcat 为例，详细地介绍了 Velocity 在 Java Servlet 环境下的应用。

学习 Velocity 模板语言非常简单，它并没有复杂的数据类型和语法结构，即使没有编程经验的读者也可以轻松地掌握。但是，要学习 Velocity 应用程序开发技术，还必须对 Java 程序设计有所了解，因为 Velocity 是基于 Java 的模板引擎。本书对 Java 程序设计进行了简单的介绍，但是由于这不是一本专门介绍 Java 编程的书籍，因此希望深入了解这部分内容的读者请参阅其他相关的书籍。

此为试读，需要完整PDF请访问：www.ertongbook.com

本书所使用的软件都可以在指定的网站中免费下载，相应的软件授权协议可以参见附录，请读者在使用这些软件时遵守相关协议。

由于作者水平有限，本书中难免有不足之处，敬请广大读者批评指正。

编　者

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com

第1章　Velocity入门

1.1　Velocity简介

对于热衷于在Internet上筑巢建站的用户来说，使用Java技术构建网站已经成为当今的时尚，而JSP则是最受青睐的工具了。现在，又多了一种选择，就是新推出的Velocity v1.1。

Velocity是Jakarta Project的一部分，Jakarta Project的目标是提供基于Java平台的商业质量的服务器解决方案，这些方案具有开放和协作的开发风格。

Velocity是一个基于Java的模板引擎，它能让Web页面设计者使用简单而功能强大的模板语言来引用Java代码中定义的对象。Web设计者能够和Java程序员根据MVC模块（Model-View Controller）并行设计一个Web站点。这样，页面设计者可专注于页面的外观，而程序员可以专注于写出高质量的代码来。Velocity把Java代码和页面分离开来，使整个网站更加容易维护，是JSP和PHP更好的替代品。

1.1.1　在网络中使用Velocity

例如，如果要创建一个网页来表述一家公司的金融状况，包括股票价格和公司新闻等信息。Java程序员可以通过Velocity以Java对象的形式向网页设计者提供股票和新闻等信息。Velocity还为设计者提供一种简单的模板语言VTL（Velocity Template Language），从而使数据访问和格式化变得更加简便。

Java程序员编写程序代码，把股票价格和最新的公司新闻从数据库中提取到Servlet，然后再按照设计者需要的格式对网页模板进行替换。

Velocity可以独立产生源代码、HTML和报告，也可以与其他系统或应用相结合提供模板服务。因为Velocity充当Java代码和网页之间的一个分离层，网页设计者就可以把注意力集中在美化网页的外观上，而无需考虑网页如何工作；而程序员着重于编写Java代码，支持站点的功能。这样分工协作的模式很适合构建较大规模的网站，而对于个人用户也同样是一种明智的选择。

1.1.2　Velocity如何工作

程序员收集数据和对象，并且把它们放到一个Context对象中，这样就可以被设计者访问。在使用Velocity时，Java程序员和网页设计者首先在一个模板中商定一组数据，设计者就可以在各自的程序或网页源代码中分别对它们进行操作。

然后，设计者就可以建立模板，添加VTL(Velocity Template Language)元素。从Context对象中获取的数据将被放置在其中，以便处理后输出。这里，Context充当中间人的作用，在Java层（程序员）和模板层（设计者）之间传递数据。

Velocity 允许网页设计者将元素嵌入在网页中，与在 Java 代码中定义的一个 Context 对象一道工作。Java 代码把数据放到 Context 对象中，然后为此数据选择一个适当的模板(格式)。网页设计者从 Context 中获取数据元素，并把这些数据元素与模板的内容相结合，从而产生设计者的需要输出。

对于模型视图控制器(MVC，Model-View-Controller)的支持是 Velocity 的优势之一，这样可以支持更多易于维护且设计很好的网页。如果把 Java 代码看作是 MVC 模型中的模型控制器层，HTML 模板代码看作是 MVC 模型中的视图层，则 Velocity 就充当了它们之间的一个分离层。与 JSP 不同，Velocity 不允许在网页之内嵌入 Java 代码，因为这将破坏 MVC 模型。

因为模板语言仅仅对 Context 中的对象提供控制机制和数据访问，所以设计者可以访问由程序员提供的数据，而不必担心影响到程序代码。同样，因为网页输出是由设计者在模板中完全指定的，程序员也不能影响数据的显示。

1.1.3 Velocity 与 Turbine 结合使用

Turbine 是使用 Velocity 作为其模板引擎的一个 Jakarta 网络应用框架。Velocity 和 Turbine 提供一种模板服务，通过这种服务，网络应用可以根据一个真实的 MVC 模型进行开发。

关于 Velocity 与 Web 应用框架 Turbine 结合的情况，将在本书第 9 章中介绍。

1.1.4 平台兼容性

Velocity 可以在支持 Java 2 软件开发包（JSDK，Java 2 Software Development Kit）的任何平台运行，与 Java Platform 2 企业版本（J2EE™）环境也兼容。支持 Windows 平台和 UNIX 平台，具有非常好的兼容性。本书所表述的内容是在 Windows 2000 环境下调试运行的。

1.2 Velocity 与 JSP 的比较

既然有人把 Velocity 说成是 JSP 和 PHP 的替代品，那么在这一节中，我们就以 JSP 为例，与 Velocity 进行比较。

Velocity 和 JSP 都有可重用代码的框架程序，提供 MVC 规范的“模型”和“控制器”部分（Turbine 和 Struts），这使生成网络程序变得更加方便了。同时，它们还都提供作为“视图”部分的模板语言（VTL 和 JSP）。

当然，Velocity 和 JSP 存在着许多不同之处，这里我们并不想从技术的角度来对比它们所能实现的功能，而只是比较它们是如何完成任务的。

这一节中所涉及的 Velocity 的内容，都将在以后各章中做进一步的讨论，如有疑问，请到本书的相关章节寻找答案。

1.2.1 MVC 设计模型简介

为了便于读者对 Velocity 与 JSP 进行比较，本节将首先介绍 MVC 设计模型，也可以称其为 MVC 规范。这是当前非常流行的软件设计模型，以此为标准，衡量 Velocity 与 JSP 哪一个

此为试读，需要完整PDF请访问：www.ertongbook.com

更符合 MVC 规范。

MVC 设计模型把软件组件分成模型（model）、视图（view）和控制器（controller）3 个部分，其结构图如图 1.1 所示。

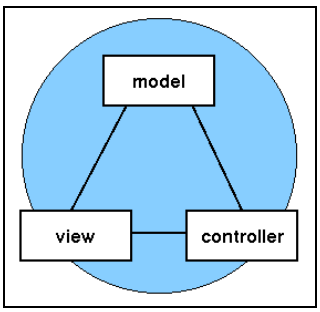

图 1.1　MVC 设计模型

“模型”是代表组件状态和低级行为的部分。它管理组件的状态，并执行状态的变换。“模型”作为中间层，并不区分“控制器”和“视图”。系统自身维持着“模型”和“视图”之间的连接，当“模型”的状态发生变化时，系统会自动通知视图。

“视图”部分负责将“模型”代表的状态可视化地显示出来。一个“模型”可以存在多个“视图”。

“控制器”部分负责管理用户与“模型”之间的通信。它提供一种控制“模型”状态改变的机制。

在实际应用中，用户输入现实世界的模型，按照 MVC 规范处理后，得到分割清晰的、有 3 种对象处理的可视化反馈，每一部分都对应指定的任务。在一个应用程序中，“视图”负责将图像及文本输出到应用程序指定的位图部分；“控制器”负责解释用户输入的鼠标和键盘事件，并使“模型”和“视图”做出相应的变化。最后，“模型”负责应用程序部分的动作和数据，它主要对以下两种请求做出响应。

- 来自“视图”：对指定信息的状态查询。
- 来自“控制器”：改变状态的指令。

为了更有效地使用 MVC 规范，读者应该了解以下内容：

- 在 MVC 规范中，现实世界中的对象是如何被分割成这 3 个部分的。
- MVC 规范中的这 3 个部分之间，以及它们和其他活动的“视图”和“控制器”之间是如何通信的。例如，在多个应用程序之间共享一个鼠标、键盘和显示器等都需要相互通信和协作。

MVC 规范并不是本书所要介绍的重点内容，感兴趣的读者可以查阅相关书籍。在下面 Velocity 与 JSP 的比较中，读者将对 MVC 规范有进一步地了解。

1.2.2　一个简单的例子——Hello World

现在，从一个简单的例子开始。刚开始接触一种语言，首先想到的例子就是 Hello World 了。

通过这个小例子，我们可以比较 JSP 和 Velocity 在处理同一件事情时的不同方法。JSP

输出参数的例子如下。

```
<Html>
<Head><title>Hello</title></Head>
<Body>
<H1>
<%
if (request.getParameter("name") == null) {
   out.println("Hello World");
}
Else {
  Out.println ("Hello, " + request.getParameter ("name"));
}
%>
</H1>
</Body>
</Html>
```

如果创建一个 HTML 文档 hello.html，其内容如上，然后在 Internet Explore 5.0 中浏览，结果如图 1.2 所示。

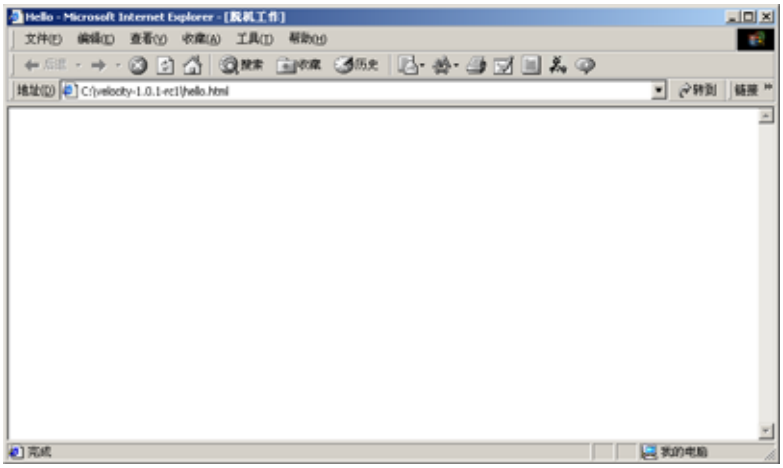

图 1.2　JSP 编写的 hello.html 的运行结果

同样的事情，如果用 Velocity 来实现，其代码如下。

```
<Html>
<Head><title>Hello</title></head>
<Body>
<H1>
#If ($request.getParameter ("name") == null)
```

```
    Hello World
#Else
    Hello, $request.getParameter ("name")
#End
</H1>
</Body>
</Html>
```

将上述代码替换为 hello.html 的内容，然后在 Internet Explore 5.0 中浏览，结果如图 1.3 所示。

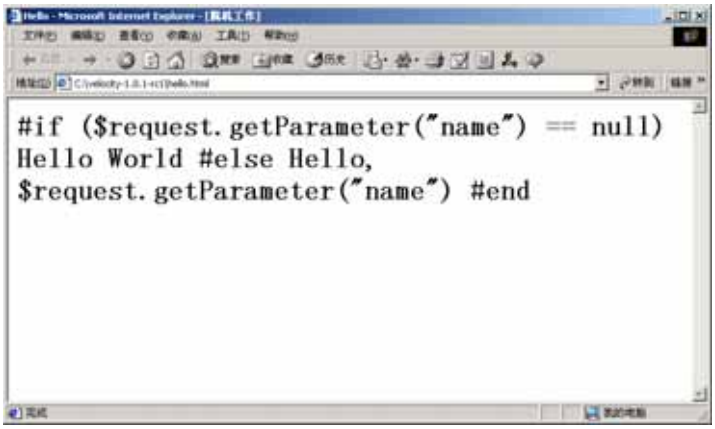

图 1.3　Velocity 编写的 hello.html 的运行结果

这两个例子的比较表明，当使用 JSP 编写 HTML 文档的时候，通过浏览器并不容易观察代码所要完成的工作。

JSP 和 Velocity 最主要的区别在于输出的方法上。在 JSP 中，需要把代码嵌入到<% %>标记中；而 Velocity 则不需要这样做。嵌入代码的特点正如上面的例子所示，当网页文档直接装入浏览器的时候，其中的 JSP 代码不会显示出来。而在有些情况下，例如调试的时候，我们恰恰需要将代码显示出来。

另外，即使是最简单的 JSP 程序也会与整个 MVC 规范发生冲突。原因在于，把 Java 代码嵌入到 HTML 编码中实际上是一种不好的设计风格。这样做将使日后修改应用程序的外观及风格变得非常困难。这破坏了 MVC 中分割的概念，也就是说，把网页的“视图”（HTML）显示从“模型”和“控制器”中分离出来。例如，如果只是想把“Hello”这个单词设置成黑体，则需要把 …标记嵌入到 out.println()语句中。

当然，JSP 也可以按如下方法编写代码。

```
<Html>
<Head><title>Hello</title></head>
<Body>
<H1>
```

```
<% If (request.getParameter ("name") == null) %>
          Hello World
<% Else %>
          Hello, <% request.getParameter ("name") %>
</H1>
</Body>
</Html>
```

在 Internet Explore 5.0 中浏览，其结果如图 1.4 所示。

图 1.4　JSP 另一种编码的直接显示结果

正如上面例子的结果所示，JSP 编写的代码也可以在网页中显示一些信息。但是在显示的信息中，用来生成网页的逻辑关系都丢失了。这只是一个非常简单的例子，也许并不能非常清晰地显示出 Velocity 的优势。但是，如果在比较复杂的应用中（如代码中包含 for/foreach 循环），Velocity 的这一特性将对程序的编写及调试有很大的好处。

读者在看到以上 3 个例子的显示时，也许会感到迷惑，因为显示的结果与程序中的逻辑关系并不相符。当然这些程序在逻辑上并不存在问题，这些图示都是将相关代码在本地直接装入浏览器显示的结果，这种方法主要用于程序的调试。

在上面的例子中，JSP 看上去与 Velocity 非常相似。但是，JSP 还需要在很多地方嵌入必要的<% %>标记。输入的代码越多，就意味着失误的几率越大。

1.2.3　编译与生成

JSP 有这样一个特点，它可以读取已存在的.jsp 网页，然后把它编译到 Servlet 中。这意味着 JSP 首先把.jsp 网页解析成中间结果，然后使用 javac 或者您喜欢的其他 Java 编译器（如 Jikes）把 Servlet 编译成为.class 文件。最后，由 Servlet 引擎装入.class 文件，并在浏览器中显示。 JSP 的工作流程如图 1.5 所示。

使用 JSP 实际上是一个多步骤的过程。JSP 的作者已经成功地将这些过程隐藏起来，普通用户很难注意到。用户只需要在编辑器中编写.jsp 网页，然后使用浏览器通过 URL 调用此网页，就开始将.jsp 转换成.class 文件的过程。

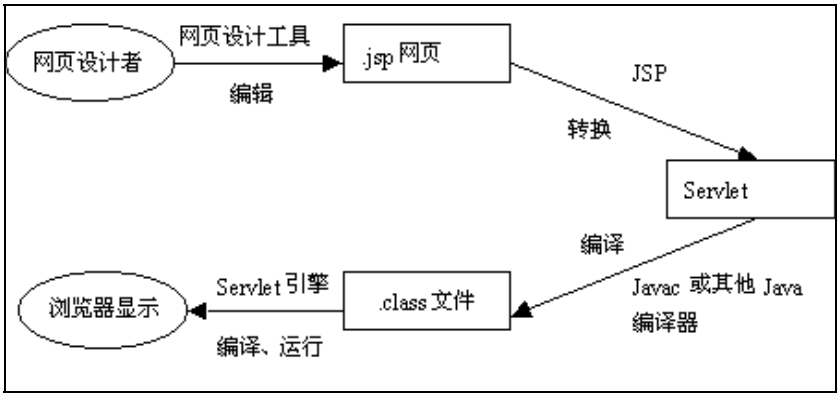

图 1.5　JSP 的工作流程图

在生成.jsp 模板的时候，有几个基本的问题要处理。首先是类的名称。Java 引擎需要生成一个唯一的名字，从而与类载入器同时工作。这样就可能出现问题，为了实现上述的功能，在每次修改.jsp 文件的时候，都会在硬盘的临时目录下生成一个新的文件。这样，这个目录占用的空间就会越来越大，除非用户亲自来整理它。当然，引擎自身也可以来管理这个目录，控制文件的大小和数量，但是有时候也会出现错误删除文件的情况。

“编辑->转换->编译->装入->运行”的整个过程实际上是很不必要的，甚至可以说是不好的设计。从另一个角度来看，Velocity 只是装入模板，将模板解析一次，然后存储为抽象语法树（AST，Abstract Syntax Tree)。抽象语法树代表在内存中的模板，它可以一次一次地重复使用，这个过程只是简单的“编辑->解析->运行”。这样做的好处是显而易见的，使用 Velocity 模板将明显地提高速度，同时还减少了对 Javac 编译器和其他临时目录的需求。在 Velocity 中，如果模板发生了变化，存在缓冲存储器（cache）中的模板将被最新解析生成的版本替换掉，从而保证了正确性。

Velocity 使用模板方法的另一个好处在于，实际的模板数据可以存储在任何地方，包括数据库和远程主机。通过对模板载入器进行配置，可以创建一个完全个性化的、满足自己需要的模板载入器。

类似 Struts 和 Turbine 这样的框架软件都会有很多处理错误的恰当方法。即使没有 Turbine，Velocity 也可以提供很多种处理错误的方法。但是，由于 Struts 是建立在 JSP 的基础上，它也继承了一些与 JSP 相关的问题。在下一节中，我们将进行详细讨论。

JSP 生成的代码存在一个主要的问题，JSP 在向输出流端口写数据的时候，并不检查输出流端口是不是开放的。当输出流端口是关闭的，或者在向输出流端口写数据时发生了异常错误，JSP 没有办法捕捉到它，除非手工定义特定的错误句柄。这些都可能导致异常错误的出现。

JSP 在设计方面还存在一个问题，就是在 JSP 页中，JSP 只能捕捉到 Exception 类型的错误。如果在 JSP 页的代码中出现了另外一种错误，如 OutOfMemoryError，就会出现网页的混乱。因为它是基于 Throwable 类型的错误，而不是 Exception 类型的，因此在 JSP 页中捕捉这种错误是非常困难的。

在连续地向输出流端口写数据时，缓冲也是一个重要的问题。在下面的例子中，这两行代码要 JSP 向缓冲写 12KB 的数据，并且将网页的 autoFlush 参数打开。Struts + JSP 可以提供 JSP 模板作为“视图”部分，从而实现了 MVC 模型，那么代码中的哪些标记应该属于 MVC 模型中的哪一部分呢？

```
<%@ page buffer="12kb" %>
<%@ page autoFlush="true" %>输出
```

Error: 500

Location: /studies/study_main.jsp

Internal Servlet Error:

```
javax.servlet.ServletException: JZ006: Caught IOException: java.io.IOException: Broken pipe
 at org.apache.jasper.runtime.PageContextImpl.handlePageException(PageContextImpl.java:386)
 at studies._0002fstudies_0002fstudy_0005fmain_0002ejspstudy_0005fmain_jsp_0._jspService(_0002fstudi
 at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:126)
 at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
 at org.apache.jasper.runtime.JspServlet$JspServletWrapper.service(JspServlet.java:174)
 at org.apache.jasper.runtime.JspServlet.serviceJspFile(JspServlet.java:261)
 at org.apache.jasper.runtime.JspServlet.service(JspServlet.java:369)
 at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
 at org.apache.tomcat.core.ServletWrapper.handleRequest(ServletWrapper.java:503)
 at org.apache.tomcat.core.ContextManager.service(ContextManager.java:559)
 at org.apache.tomcat.service.connector.Ajp12ConnectionHandler.processConnection(Ajp12ConnectionHand
 at org.apache.tomcat.service.TcpWorkerThread.run(PoolTcpEndpoint.java:366)
 at org.apache.tomcat.util.ThreadPool$ControlRunnable.run(ThreadPool.java:411)
 at java.lang.Thread.run(Thread.java:484)
```

Root cause:

```
java.sql.SQLException: JZ006: Caught IOException: java.io.IOException: Broken pipe
 at com.sybase.jdbc2.jdbc.ErrorMessage.raiseError(ErrorMessage.java:426)
 at com.sybase.jdbc2.tds.Tds.handleIOE(Tds.java:2794)
 at com.sybase.jdbc2.tds.Tds.language(Tds.java:635)
 at com.sybase.jdbc2.jdbc.SybStatement.sendQuery(SybStatement.java:1315)
 at com.sybase.jdbc2.jdbc.SybStatement.executeQuery(SybStatement.java:1385)
 at com.sybase.jdbc2.jdbc.SybStatement.executeQuery(SybStatement.java:399)
 at niac.util.DBHandler.dbQuery(DBHandler.java:37)
 at niac.www.studies.StudyMain.buildPage(StudyMain.java:70)
 at niac.www.studies.StudyMain.getPage(StudyMain.java:251)
 at studies._0002fstudies_0002fstudy_0005fmain_0002ejspstudy_0005fmain_jsp_0._jspService(_0002fstudi
 at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:126)
 at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
 at org.apache.jasper.runtime.JspServlet$JspServletWrapper.service(JspServlet.java:174)
 at org.apache.jasper.runtime.JspServlet.serviceJspFile(JspServlet.java:261)
 at org.apache.jasper.runtime.JspServlet.service(JspServlet.java:369)
 at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
 at org.apache.tomcat.core.ServletWrapper.handleRequest(ServletWrapper.java:503)
 at org.apache.tomcat.core.ContextManager.service(ContextManager.java:559)
 at org.apache.tomcat.service.connector.Ajp12ConnectionHandler.processConnection(Ajp12ConnectionHand
 at org.apache.tomcat.service.TcpWorkerThread.run(PoolTcpEndpoint.java:366)
 at org.apache.tomcat.util.ThreadPool$ControlRunnable.run(ThreadPool.java:411)
 at java.lang.Thread.run(Thread.java:484)
```

在处理上述问题时，Velocity 的方法是，允许开发者把输出流传送到引擎中。如果在这个过程中有异常错误出现，这个错误将被捕捉到，并同时处理掉。Velocity 通过传送恰当大小的数据流到解析器的方法，解决了缓冲的问题。如果出现错误，另一个数据流会自动地作为替代输出。

以下是 Velocity 如何实现输出控制的代码。

此为试读，需要完整PDF请访问：www.ertongbook.com