REVISED
6:18 pm, Jun 30, 2005

DRAFT

JavaServer Faces in Action

Kito D. Mann

Foreword by Ed Burns

MANNING

JavaServer Faces in Action

中文版

KITO D. MANN 著

铁手 译

MANNING

Greenwich (74° w. long.)

For online information and ordering of this and other Manning books, please go to www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact:

Special Sales Department

Manning Publications Co.

209 Bruce Park Avenue Fax: (203) 661-9018

Greenwich, CT 06830 email: orders@manning.com

Manning Publications Co. Copyeditor: Liz Welch 209 Bruce Park Avenue Typesetter: Denis Dalinnik Greenwich, CT 06830 Cover designer: Leslie Haimes

ISBN 1-932394-11-7

Printed in the United States of America 1 2 3 4 5 6 7 8 9 10 – VHG – 08 07 06 05 04

谨以此书献给我的妻子和挚友，Tracey。

没有你便没有此书，我将永远感激你对我的生活之积极影响，并且一直推动着我做得更好。

前言

在我写这个前言的时候，我正在为 2004 JavaOne℠ 大会的一个陈述与四个领先的用户接口 (UI) 组件厂商合作。在我们的陈述中，各厂商都将展示它们如何将 JavaServer Faces 技术应用在其各自的产品中。在准备该陈述的时候，我逐渐认识了过去 3 年中我们一直在 JavaServer Faces 上面所做的工作。每个厂商对于它们的产品如何采用 JavaServer Faces 技术都有其自身独特的考虑，但是都发出一个声音，宏大而清晰：它们非十分坚信最终能有一个关于 Web 用户接口的标准。

基于 Web 的 UI 标准的缺失迫使这些组件厂商不得不针对它们想要插入的集成开发环境（IDE）分别编写特定的代码。既然现在有了 JavaServer Faces 标准，任何 IDE 都可以宣称符合此标准，而符合此标准的组件厂商也可能以很少的工作就可以将其组件插入到这些 IDE 之中。当然，这意味着你开发的组件也可以插入到开发工具中而不用太多的额外的工作。

JavaServer Faces 规范是在一个在 Web UI 领域具有领先理念的团队开发的。我们将吸取自各种不同方法的最佳理念引入到 UI 框架中，并将它们集成到一个有机的整体之中。标准的问题是，宁肯复杂一点也要解决其所针对的问题。对 JavaServer Faces 而言，问题是在那些根本不适合 UI 的技术之上提供一种易用的 UI 框架。这导致了一个实现特别复杂的规范。感谢上帝，具体实现此规范的人数要远远少于那些使用该实现的人数，但是在它出场的时候，详细了解该规范对于用好它也是很有帮助的。

作为开发下一版本 JavaServer Pages 专家组成员之一，Kito 对技术标准并不陌生。Kito 抓住了 JavaServer Faces 的关键增值点，并在一本易读和很有深度的书中解释了它们。你将看到是什么使得 JavaServer Faces 有别于其他的 web UI 框架，包括其一流的组件模型，其良好定义的状态管理系统，以及它与 JavaBean 之间的概念相似性。Kito 非常熟悉规范所使用的抽象，并且，更为重要的是，他***理解*** 我们为什么要使用这些抽象。理解规范的这种*为什么*还会给你，读者，带来更有高度的解释。例如，看看第 2 章的“概念关系”图。这是理解 JavaServer Faces 设计原理的最好方式。

Kito 也理解技术所适合的市场。这意味着你首先获得最重要的信息，所以你可以快速和完整的完成你的工作。它花了足够的时间来构筑了一个 JavaServer Faces 所依赖的技术基础，使得此书对从头开始的人来说非常理想。

最后，此书具有你所期盼的顶级的软件技术书籍所具有的一切东西：一个可用于对框架理解的描述性目录，可以为你节省时间的章节目标和小结，可以直接用在你的项目中的大量实际示例。还有一个我在此书中见到而未在其它书中所见的是，对当前支持 JavaServer Faces 技术的 IDE 的深入描述。这非常有价值，因为一旦你理解了它们所支持的基础技术之后，这些工具会节省你很多时间。

除了本书对正在交付的 IDE 的独特视角之外，Kito 作为 JSFCentral.com 的负责人，带来了他很多的实际经验。该站点是一个高质量的文章，访谈的汇集之地，并且，更重要的是，它提供关于了 JavaServer Faces 技术相关的社区和业界的最新发展动态。Kito 设置了一些单独

的区域来讨论组件，渲染包，实现，等等。我想你将会发现该站点—以及本书—对你的 JavaServer Faces 编程具有莫大价值。

ED BURNS

JavaServer Faces Specification Lead

序

我一直喜欢写作。我是一个计算机的滑稽演员，在某些方面，我发现人文科学要比计算机科学容易的多—--前提是，如果我能够设法让我的教授信服我已经阅读了他给我开出的那些成打的书了的话。在上世纪后期，我终于将我对写作的痴迷和对软件开发的困惑，通过撰写杂志文章和进行技术书评融合在一起。后来，在 2000 年，在结束了对一家财富 500 公司的财务咨询之后，我开始创业了。

好，不过有些迟了。但是我还是被我那些加入我的的朋友很是嫉妒，并且他们也为我那些精彩的杂志文件所吸引。我选择的起点是做一个具有一个真正的业务计划的教育应用服务提供商(最近被 McGraw-Hill 收购)。我们产品的主要部分是一个基于 web 的应用，而作为首席架构师，我的任务之一就是构建它 (当然，还有一些其它的微弱主意的帮助)。我们没有使用我自己开发的 web 应用框架，而是选择了 Struts，这在当时是一个很好的决定。在我指导项目经过了几个发布周期后，我逐渐明瞭了 Struts 的优势和弱点，以及在一个其成员具有不同背景的团队中开发一个任务紧急的 web 应用所涉及的各种问题。

在经过两年的煎熬和对女友 Tracey 的忽视之后，我辞职了，并且花了一段时间来重新熟悉已经陌生许久的一个叫做睡觉的东西。经过我们充分的重新熟悉后，我又尽情地吸收新技术的信息，并且徘徊于 Java Specification Request (JSR) 127—JavaServer Faces (JSF)。JSF 似乎是日渐增长的框架热的一个终极答案。此外，因为 JSF 抽象了如此之多的 Web 开发中令人厌烦的细节，它似乎在很多方面都领先于 Struts。我很清楚 JSF 是 Java Web 开发中下一件大事情。

因为我急于想要回头去写些架构规范和备忘录之外的东西，所以编写一本有关 JSF 的书似乎是个好主意。我评论过 Manning 的好几本书，所以我联系了出版人以讨论撰写一本关于 JSF 的书籍。经过几轮游说， Manning 同意了，这样 *JavaServer Faces in Action* 项目才得以诞生。对于规范在其 2004 年 3 月的最终发布之前是否要经历根本性的变化我几乎一无所知。 (如果我有个水晶球，我根本不会在 2003 年早期就写这么多；要知道重写可不是一件好玩的事。)

在过去的一年半中，发生了很多事情。2003 年 5 月，我和 Tracey 结婚(我想毕竟离开创业公司是一件好事)。那年晚些时候，我启动了 JSF Central，一个专著于 JSF 社区的网站，塞满了各种资源和一个便捷 FAQ。终于，在 2004 年 3 月， JSF 1.0 发布(5 月发布了 1.1 维护版(maintenance release))。我相信其结果会给如火如荼的 Java web 开发领域再加一把火。它不但可以使日常开发工作更加容易，而且还开启了一个第三方用户接口(UI) 组件产业，就象 Microsoft 的 ASP.NET Web Form 一样。

从那之后，我努力的工作，以保证这本书成为两方面的催化剂：不但帮助你明白 JSF 是什么，它如何运作，以及如何使用它，而且还教你如何编写你自己的 UI 组件。我也和 Oracle, IBM, 和 Sun 的代表一起勾画了如何将 JSF 与不同的 IDE 集成。另外，这些文字也受到我作为 JSF Central 的首席编辑角色的影响，那里是我得到日益发展的 JSF 生态系统的独特优势点之地。

所以，有了你看到的这本书。我希望 *JavaServer Faces in Action* 能够激发你如我一般对此技

术的热情，并且成为你自己项目开发的有用工具。同时，我又想要重新来熟悉我的旧朋友：睡觉。尽管将你关于本书的建议发送到 kmann@virtua.com 或者提交到本书的作者在线 www.manning.com/mann；一旦醒来，我很高兴能够阅读它们。

鸣谢

大多数项目都不是一人之力，即使它们是从一个人开始的。技术书籍需要各种人员的才能，他们中很多人都献出了他们的空余时间。(尽管，我自己很自豪这些特殊的牺牲。) 我首先感谢 Manning 的员工 Clay Anders，是他相信这本书是一个好主意，并且在其它人之前开始讨论 JavaServer Faces。然后是 Marjan Bace，Manning 的出版人，是他授权了这个项目，并且迅速将我的第一稿撕成碎片。 没有 Marjan，这本书可能会使你昏昏欲睡。(如果你正用火柴棍撑起你的眼皮，那就责备他好了。)

接下来是 Jackie Carter—我不能在梦想有更好的开发编辑了。Jackie 并不只是批评和指导我的工作；她是整个过程中的真正伙伴。我还想感谢众多的技术评论者，是他们让我的文字消耗了他们的夜晚时间而没有任何报酬：Roland Barcia, Todd Cunningham, Jeff Duska, Carl Hume, Will Forster, Aleksandar Kolundzija, Jason LeCount, Josh Oberwetter, Michael Nash, Russ Pearlman, Mark Pippins, Matthew Schmidt, Keyur Shah, Sang Shin, 和 Henri Yandell。Ted Kennedy (已经安息了) 和 Dave Roberson 完全值得提及，他们做了一件值得赞扬的工作，就是协调整个评论过程。

JSF 专家组的多名成员(Eric Lazarus, Brian Murray, Adam Winer, 特别是 Michael Nash) 帮助确保本书的技术准确性，并且 Ed Burns (规范的领导) 还为本书编写了导言。

同时，几个厂商也让我在它们的产品公开发布之前见识它们。Beverly Dewitt 提供了 IBM WebSphere Studio 的总体信息，Roland Barcia 编写了在线扩展的附录 B 中关于该产品的部分 (还提供了一个十分彻底的技术评论)。Sun Microsystems 的 Jim Inscore 使我能够接触到 Java Studio Creator 的早期发布版。

Jonas Jacobi 和其在 Oracle 的团队(Brian Albers, John Fowler, 和 Adam Winer) 也提供了以上帮助，并且额外还不但提供了关于 JDeveloper 的详细信息，而且还提供了正式的评论并作为一个通用的 JSF 资源。Jonas 也编写了在线扩展附录 B 中有关 JDeveloper 的 JSF 支持和 Oracle 的 ADF Faces 组件的内容。

另外，我还请求我的好朋友 Ed LaCalle 帮忙进行了西班牙语的翻译，以及我那全世界旅游的兄弟，John A. Mann II，进行了俄语的翻译。我的妻子 Tracey Burroughs 提供了最全面的技术指导和支持。它也编写了附录 E 的大部分内容—她对细节的处理有着奇妙的诀窍，她以她的才华很好的完成了她的工作。

还有就是生产团队，是他们创建了本书的早期版本 (PDF 格式) 和印刷版本。Henri Yandell 是个极端小心仔细的编辑，它确保我所写的每一个字都是合理的。Liz Welch，版面编辑，纠正了我所有的语法错误并确保我遵循 Manning 的风格。Susan Forsyth 校对了每一个字，Denis Dalinnik 排版了每一页，而 Susan Edwards 则应付了极端繁琐的索引工作。最后，Mary Piergies 协调了整个生产过程。

如果你在网站上看到本书，或者收到一个免费拷贝或者摘录，你应该感谢 Helen Trimes—她做了非常杰出的营销工作。我还想感谢 Manning 的另外几个人：Lianna J. Wlasiuk, Susan Capparelle, Syd Brown, 以及 Iain Shigeoka。

我还应该感谢我的父母—我所知最坚强和善良的人。当我在生活中苦闷彷徨时，我就会看到

他们的爱和鼓励所留下的深深烙印。

所有这些人的努力工作才得以使这本 *JavaServer Faces in Action* 成为一本优秀的书籍。最后我还得感谢你，读者，购买了此书。(OK，也许你只是随便翻翻，就这也已是值得感谢的了。)

关于此书

本书是针对那些正在开发 Java web 应用的人— 架构师，应用开发人员和前端开发人员。在我看来，架构师关心的是应用的设计，使用哪种技术，以及开发过程如何运作。开发人员则构建模型对象，应用逻辑等等，而前端开发人员则具体构建 GUI，通常使用象 JSP 或者 Velocity 之类的显示技术。在某些团队中，这些角色可能由相同的人承担，或者不同的角色具有重叠的技能要求。JSF 是一个 web 应用框架，类似于 Struts, WebWork, 或者 Tapestry，但是本书即使你之前从没有用过一种 Web 框架也是可读的。

JavaServer Faces in Action 分为 5 部分。第一部分涉及 JSF 基础。第 1 章阐述了 JSF 背后的动机，以及它如何适应当前的情况和应用环境，以及必不可少的 Hello, world! 示例。本章也提供了 JSF 所使用的基础技术，HTTP、servlet,、portlet、以及象 JSP 之类的显示技术的简述。第 2 章通过考察其核心概念进一步研究 JSF，并详细解释 JSF 如何施展其魔力。第 3 章涵盖了日常主题，如配置，JSP 集成，JavaBean 初始化和导航等。第 4 章和第 5 章包括标准的 UI 组件，而第 6 章则研究国际化，校验和类型转换。所有这些章节都将 JSF 作为一种技术解释，但是也展示了在工具中如何使用它们。

第 2 部分集中于使用第 1 部分所描述的概念构建 UI 。从第 7 章开始，介绍了一个贯穿第 2 和第 3 部分的案例。第 8, 9, 和第 10 章使用 UI 组件和 JSP 构建了一个工作原型，没有使用任何 Java 代码。

第 3 部分关注将原型转化为实际应用所需的应用代码。第 11 章从开发人员的角度简述了 JSF 的 Java API ，紧跟着第 12 和 13 章则遍历了应用的设计和开发。第 14 章研究了 JSF 和现有的框架，如 Struts，之间的集成。

第 4 部分从另一个角度来研究 JSF：从 UI 组件，渲染，校验和转换来扩展 JSF。第 15 章从组件开发人员的角度研究 JSF API 。本书书面版本结束于附录 A，在其中，有关于在不用 JSP 的条件下使用 JSF 的内容。

紧跟附录 A 之后是一个在线扩展 (第 5 部分) ，它可以从 http://www.manning.com/mann 免费下载。在线扩展中有大量的基于第 4 部分所讲述的基础的示例。第 16 章展示了如何开发基本的 UI 组件，第 17 章则研究渲染器。第 18 和 19 章展示如何构建更综合的 UI 组件，而第 20 章则涉及校验器和转换器的开发。所有这些章节都使用了适合于日常 web 开发的实际示例。

在线扩展最后有四个附录，涵盖了大量的附加主题。附录 B 提供了 Oracle JDeveloper, IBM WebSphere Studio, 和 Sun Java Studio Creator 对 JSF 的支持。附录 C 更加深入的研究了 JSF 的架构，并且展示如何使用可插入类来扩展它。最后两个附录是参考附录：附录 D 包括每一个配置元素，而附录 E 则列出了时间区，语言和国家代码。

如何使用此书

本书集教程，案例和参考于一身。其写作方式是为了能够持续阅读，但是我试图确保每个单独小节之间的隔离。即是说，如果你有时间，你可以从开头阅读，尽可跳过不相干的部分。要当心跳过整个章节，因为在每一章你都可能发现其中自有黄金屋。

当然，有些人相信他们有一些时间 (让时间一边呆着去吧)，所以这里没有更多指导。如果是一个架构师，你应该细读本书的绝大部分，但要仔细关注第 1, 2, 3, 6, 8, 12, 13, 和 15 章。你也可能希望细读附录 A，以及在线扩展附录 B 和 C。

应用开发人员应该阅读第 1-3 部分的大部分内容，但是可以跳过第 4 和第 5 章。如果你正在开发一个现有的 Web 应用，或者当前你在使用 Struts，你仅需要阅读第 14 章。高级开发人员应该仔细阅读第 4 和第 5 部分（在线），以及附录 A，扩展附录 B 和 C。

前端开发人员应该阅读第 1 和第 2 部分的所有内容，第 2 章可以例外。总而言之，本书越靠近结尾更加复杂。

引用和参考

引用的网站，书籍和文章用方括号([]) 括住，可以在本书末尾的引用参考一节找到。例如，作者的社区网站，JSF Central [JSF Central] 是一个找到更多关于 JSF 新闻，产品和资源信息的好地方。在参考小节中，方括号标注的文本对应到实际的 URL:

[JSF Central] JSF Central 社区网站，http://www.jsfcentral.com.

括起来的名称看起来都是一样，而不管它是网站，产品，书籍还是文章。

约定

像那些好书一样，文本应该是尽量自解释的。因而，我使用了一些约定，逐项解释如下：

黑体

我使用黑体来强调代码片断 (可能是 Java, XML, 或 JSP)中的某些部分。通常我试图指出代码的关键之处，或者引起对添加到前一代码清单中的文本的注意。

斜体

在我定义某个术语词汇时使用斜体。通常我也用它来强调特别的词语。

Courier 类型

Courier 字体用于代码(Java, XML, 或 JSP)。我在代码清单中使用它，但是也在 Java 类名，XML 标记，和你可能在开发工具中输入的其他东西中使用它。

组件表

在第 4 章和第 5 章，我使用一种特别的表格来描述 JSF UI 组件。首先是一个组件概述表；示例如下：

HtmlOutputText summary

组件	HtmlOutputText

类别	`javax.faces.Output`
IDE 可能显示名	Output Text
显示行为	Converts the value to a string and displays it with optional support CSS styles. (If the `id` or `style` property is set, encloses the text in a `<span>` element.)
标签库	HTML
JSP 标签	`<h:outputText>`
直通属性	`style, title`
公共属性	`id, value, rendered, converter, styleClass, binding` (see table 4.2)

属性	类型	默认值	要求?	说明
`escape`	`boolean`	`true`	No	Controls whether or not HTML or XML characters are escaped (displayed literally in a browser).

不要着急其到底是什么意义—只是所有的标准 JSF UI 组件都用这种方式描述。其初衷是在一个表格中向你总结一个关于组件的基本细节。

UI 组件示例处理为下面的这种表格：

HtmlOutputText 示例：默认文本转义

HTML	`What are &lt;i&gt;you&lt;/i&gt; looking at?`
组件标签	`<h:outputText value="What are <i>you</i> looking at?"/>`
浏览器显示	

在这里，我在一个表格中展示了 HTML 输出， JSP 组件标签，以及浏览器显示。通过这种方式，你可以很容易的看到这 3 个部分是如何联系在一起的。HTML 首先被显示给那些按照 HTML 方式思考的人。

代码标解

我使用代码标注，因为它看起来比注释更酷。例如：

```
public String myAction()          ← ① 这是一个 Action 方法
{
// Do something
}
```

有时，我将在代码清单的后面分段落来展开这些注解，并使用这样的带圈数字作为项目编号：①。

注解

我在全书中使用**备注，警告，定义**等典型的注解来强调那些可能淹没在普通文本中的特定点。如下所示：

定义 *UI 组件，或者控件* 是提供与最终用户交互的特定功能的组件。典型的例子包括工具条，按钮和日历等。

另外，我也使用大量独特的方式：

顺便提及 指出并不是当前文本本质，但是可能会在某些地方有用，的一些要点。通常我会试图解释你可能遇到的一些相关问题。

SOAPBOX 我自己的个人观点。可视为是一些调味点缀。

源代码和在线扩展

本书的所有源代码都可以从本书的网站下载：http://www.manning.com/mann。下载文件中包含有安装和编排指导。从该站点，本书的拥有者还可以下载额外的 300 内容，称为在线扩展，格式是 PDF。.

作者在线

Manning 为我们作者方便收集来自于心爱读者的评论，而维护了一个在线论坛。针对 *JavaServer Faces in Action* 的地址是：http:// www.manning.com/mann.。如果你对 JSF 有任何问题或者对本书有任何建议，请尽管访问它。我自己会随时检查这个论坛，而且我对你会说些什么很感兴趣。

关于作者

Kito D. Mann 是一个关于企业架构，顾问和开发方面的咨询专家。从 12 岁开始编程，它已经撰写了大量的 Java 相关的技术文章，并活跃在各种用户组和会议上。它为多个财富 500 的大公司做过咨询，并且曾是一个教育性应用服务提供商的首席架构师。Kito 也是 JSF Central 社区站点的创始人，以及 JSF 1.2 和 JSP 2.1 专家组成员。它拥有约翰．霍普金斯大学的计算机学士学位，现在和他的妻子，四只猫以及两只鹦鹉生活在康涅狄格州的斯坦福德。闲暇时，它喜欢组装电子音乐设备。

关于标题和封面

关于标题

通过介绍，总论和教程实例的综合，*In Action* 系列旨在帮助学习和记忆。根据对认知科学的研究，人们记住的东西总是那些自发探索所得到的东西。

虽然在 Manning 没有一个人是认知科学家，我们深信学习并成为永久记忆必须经过探索，动手，以及对正在所学的东西进行重新区分这些阶段。人们理解和记住新事物，即是说他们只有在主动探索之后，才会掌握了它们。人类的学习是主动的。*In Action* 系列书籍的基本点之一就是它是实例驱动的。它鼓励读者去尝试，去运行新的代码，并且探索新的理念。

这里还有一个更通俗的原因：我们的读者总是很忙。他们通常使用书籍来进行工作和解决问题。他们需要能使他们容易进入和退出的书，并且仅在需要的时候学习他们需要的部分。他们需要能帮助他们干活的书。这个系列的书就是针对这些读者。

关于封面图片

JavaServer Faces in Action 的封面图片是一个 “萨卡人（Saka）,”或者称土耳其送水人。该图取自 1802 年 1 月 1 日由 London 的 Old Bond Street 出版的土耳其帝国服装集。标题页已经缺失，所以我们不能确定其日期。该书的目录以英文和法文标识这些图片，每张图都有两个艺术家作者的名字，他们一定会惊讶他们的艺术作品居然是作为一本计算机编程书籍的封面...在两百年之后。

该画册被 Manning 的一个编辑在设在曼哈顿西 26 大街的一个车库中的古文物跳蚤市场购得。卖主是一名居住于土耳其安卡拉的美国人，那时候他正在进行日常的货架整理。Manning 的编辑身上并没有足够的现金购买该书，并且信用卡和支票也被客气地拒绝。

如果当晚，那个卖主飞回安卡拉，那就没指望了。怎么办呢？除了用古老的口头协议和握手之外，没有其他办法。卖主提议钱可以通过电汇转给他，这样当我们的编辑离开时，手中除了拿着那本画册之外，还有卖主相关的银行信息。不用说，我们在第 2 天汇出了全部款项。我们心存感激，而且对这位不知姓名的人给予的信任留下了深刻的印象。它唤起了我们对那些许久之前曾经发生过的什么东西的悠远回忆。

从土耳其帝国画册中得到的图片，和出现在我们其他书籍上面的图片一样，给我们带来了两个世纪以前丰富多样的服装风格。它唤起了我们对那个时代以及除现在这个过渡亢奋的时代之外的其他每个时代的孤立感和距离感。

服装符号从那时候起发生了改变，因地域的差异化和丰富性随着时间慢慢消逝。现在你已经很难从服装上来分辨一个大陆上的原著民和其他人了。或许，乐观点来看，我们换得了对各种各样个人生活的文化和视觉的差异性。或者更加多样和有趣的知识和技术生活。

在 Manning，我们赞扬独创性，主动性，当然还有那些使用能够带给我们 200 年前区域生活的丰富多样性感受的画册图片作为封面的计算机书籍所带来的乐趣。

第1部分

JavaServer Faces 探秘

第1 部分介绍和探索 JavaServer Faces 编程领域。我们提供了一个技术性的总体介绍并解释了 JSF 如何适合当前 web 开发状况。接下来，我们讨论了一个实例应用，研究 JSF 是如何工作的，并且讲述了一些基本概念。这一部分的结尾总结了所有标准的 JSF 组件，以及诸如国际化和校验之类的特征。

1.JavaServer Faces 介绍

本章内容

- JavaServer Faces 是什么，不是什么
- 基础技术 (HTTP，servlet，portlet，JavaBean，以及 JSP)
- JavaServer Faces 与现有的 web 开发框架的关系
- 构建一个简单的应用

欢迎阅读 ***JavaServer Faces in Action***。JavaServer Faces (JSF, 或简称“Faces”) 通过为 Web 开发领域带来对丰富的，强大的用户接口组件 (如文本框，列表框，分页面板和数据网格等等)的支持来使 Web 开发更加容易。作为 Java Community Process[1]的一个产物，JSF 有望成为 J2EE 的一个组成部分。本书将帮助你了解 JSF 到底是什么，它如何工作，以及目前你可以如何将它应用到你的项目当中。

1.1. 这是个 RAD 化的世界

在前 Web 时代的一个流行词汇就是***快速应用开发（Rapid Application Development）(RAD)***。RAD 的主要目标是使你能够使用大量可重用的组件来构建强大的应用。如果你曾用过诸如 Visual Basic, PowerBuilder, 或者 Delphi 之类的工具，你就知道它们对软件开发的生产率提高是一个大大的飞跃。因为它们是第一次，能够很容易地开发复杂的用户接口(UI) 并且将其与数据源进行集成。

你可以从组件面板拖出应用部件—UI 控件或者其它组件— 然后放到你的应用中。这些组件每一个都有其自己的影响其自身行为的属性。(例如，`font` 是所有显示文本的控件的一个公共属性；而数据网格则具有一个表达其数据存储源的 `dataSource` 属性。) 这些组件会产生一组事件，而事件处理器则定义 UI 和应用的其它部分之间的交互。你可以在一个集成开发环境（IDE）中直接访问这些好东西，并且你可以很方便的在设计视图和代码视图之间进行切换。

RAD 工具最长于开发全面完整的应用，但是它们对开发快速原型也是非常有用的，因为它们可以快速的创建 UI，基本上甚至完全不用编程。另外，较低的门槛也使得不管对经验丰富的程序员还是新手来说都能立竿见影地得到开发结果。

这些工具典型地分为四层：

- 一个基础组件架构
- 一套标准部件
- 一个应用基础架构(Application Infrasturcture)
- 工具自身

基础组件架构具有足够的可扩展性，也带出了一个第 3 方的组件开发产业，比如 Infragistics 和 Developer Express 这些专业的组件开发商。

当然，RAD 的理念并未消失—它仅是被其它更强势的词汇所代替。它在某些 Java IDE 和其它一些开发环境如 Borland Delphi 和 C++Builder 中也植根很深。然而，这些环境，已经对 Web 开发停止使用 RAD 的概念了。Web 开发领域对 RAD 的采用也非常的缓慢。

这种停滞部分原因在于对一个并不简单和连贯的应用需要创建一个简单、连贯的视图本身的复杂性。如果和标准的桌面应用相比，Web 应用要复杂得多。你有太多不同的资源需要管理—页面, 配置文件，图像，以及代码。你的用户可能使用运行于不同的操作系统之上的不同类型的浏览器。你不得不对付 HTTP，一个非常难以构建复杂系统的协议。

软件产业对于掩饰复杂性倒是做得很好，所以毫不奇怪过去几年出现了许多 RAD 的 web 解决方案。这些方案都将可视化，面向组件的强大威力带到了复杂的 Web 开发领域。它们

1 Java 社区流程（Java Community Process --JCP) 是一个通过新的应用编程接口（API）和其它平台增强来扩展 Java 的公共流程。新的建议称为是 Java 规范请求（Java Specification Requests -JSR)。

的祖先是 Apple 的 WebObjects[2] ，而 Microsoft 则将这种概念通过 Visual Studio.NET 和 ASP.NET Web Form 带入了主流开发。在 Java 世界，出现了许多框架，它们大多是开源的。某些框架有工具支持，而有些则没有。

但是*标准的* Java RAD web 框架的缺乏使 Java 方案的魔方缺了一块—而这一块正是 Microsoft 的.NET 框架所涵盖已久的范围。JavaServer Faces 就是来填补这个洞的。

1.1.1. 那，什么是 JavaServer Faces 呢?

对于 RAD 工具的四个层次， JavaServer Faces 定义了其中 3 个：一个基础组件架构，一个标准的 UI 部件集，以及一个应用基础架构。JSF 的组件架构定义了一个通用的方式来建立 UI 部件。此架构能驱动标准的 JSF UI 组件(按钮，超链接，复选框，文本框等等)，也为第 3 方组件留出了空间。组件是面向事件的，所以 JSF 可以允许你处理客户端产生的事件 (比如，文本框中的值发生了变化或者用户点击了某个按钮)。

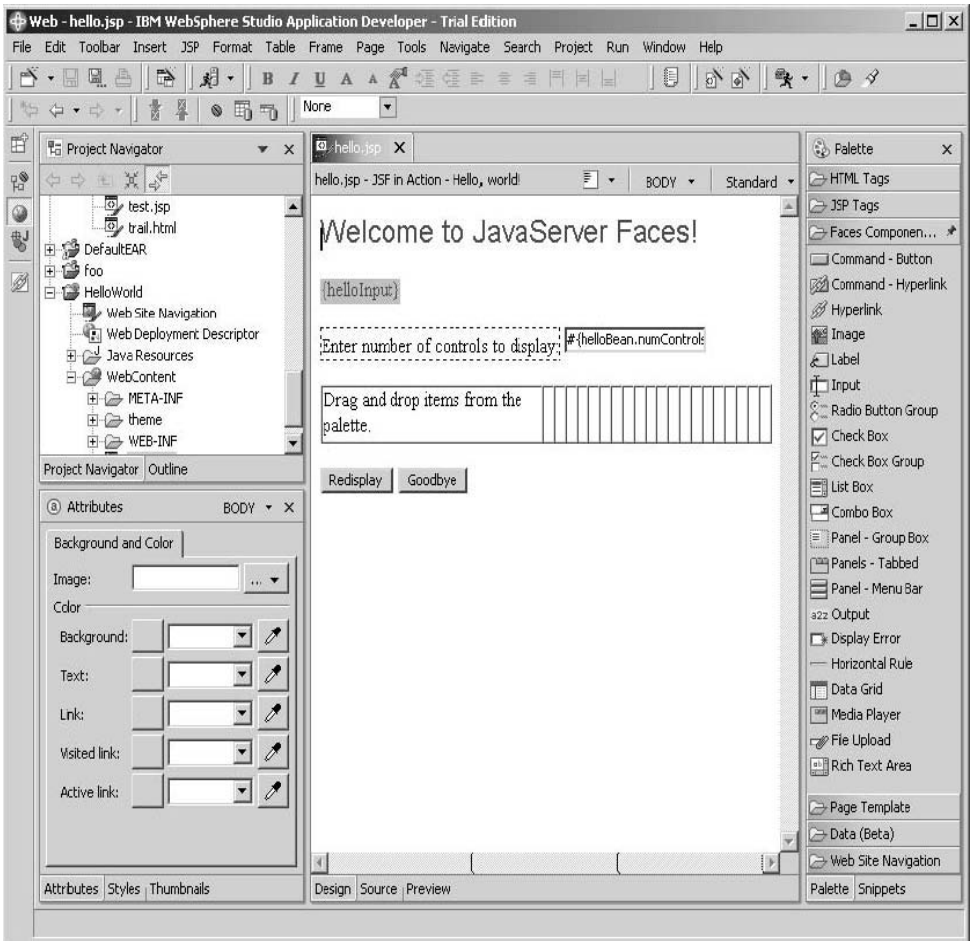


图 1.1 IBM 的 WebSphere Application Developer (WSAD) 已经扩展来在其所支持的无数种技术之外又支持 JSF 应用了。你可以使用 WSAD 中熟悉的 Eclipse 环境可视化地构建

[2] WebObjects 是一个包含了 J2EE server，web services 支持，对象持久性，以及其它一些特征的完整开发环境。

JSF 应用，并且可以混合使用其他 JSP 标签库。

因为对 Web 应用来说，和其桌面应用堂兄弟不同，它必须总是要满足多个客户端 (比如桌面浏览器，移动电话和 PDA 等), JSF 有一个可以以不同方式显示组件的强大架构。它还具有一个可扩展的机制来做输入校验 (如字段长度) 以及在显示字符串和对象之间进行转换的工作。

Faces 也能够自动保持你的 UI 组件和收集用户输入值的 Java 对象之间的同步，并且通过调用 Backing Bean 来对事件进行响应。另外，它有一个强大的导航系统并且全面支持多语言特征。这些特征构成了 JSF 的应用基础架构—即对构建一个新应用系统来说必不可少的基本构建块。

JavaServer Faces 定义了工具支持的基础，但是将实现留给了工具厂商，这是 Java 的习惯。你可以自行选择业界领先者提供的工具，它们都可以使你能够象使用其它 RAD 开发工具，如 Visual Studio. NET，一样可视化地布局和设计你的 Web UI。 (如图 1.1, 1.2, 和 1.3 就分别展示了在 IBM, Oracle, 和 Sun 的 IDE 开发工具中开发 JSF 的情形。)或者，如果你愿意，也可以在没有设计工具的情况下开发 Faces 应用。

图 1.2 Oracle 的 JDeveloper [Oracle, JDeveloper] 具有对 JSF 的全面支持，并通过大量的 UIX 组件来完善，而这些都可以集成到标准的 JSF 应用之中。它也支持在其应用开发框

架（ADF）[Oracle，ADF]中使用 JSF 组件。（这个屏幕 JDeveloper 10g 中的 UIX 组件，这是下一版本 JDeveloper 中对 JSF 的基本支持）

I

在这些赞美之词之后，我们应该指出 JavaServer Faces 和桌面 UI 框架如 Swing 或者 SWT 之间的关键不同之处： JSF 是运行在服务器之上。因此， Faces 应用将运行在一个标准的 Java web 容器之中，比如 Apache Tomcat [ASF, Tomcat]，Oracle Application Server [Oracle, AS]，或者 IBM WebSphere Application Server [IBM, WAS]，然后向客户显示 HTML 或者其它标记语言。

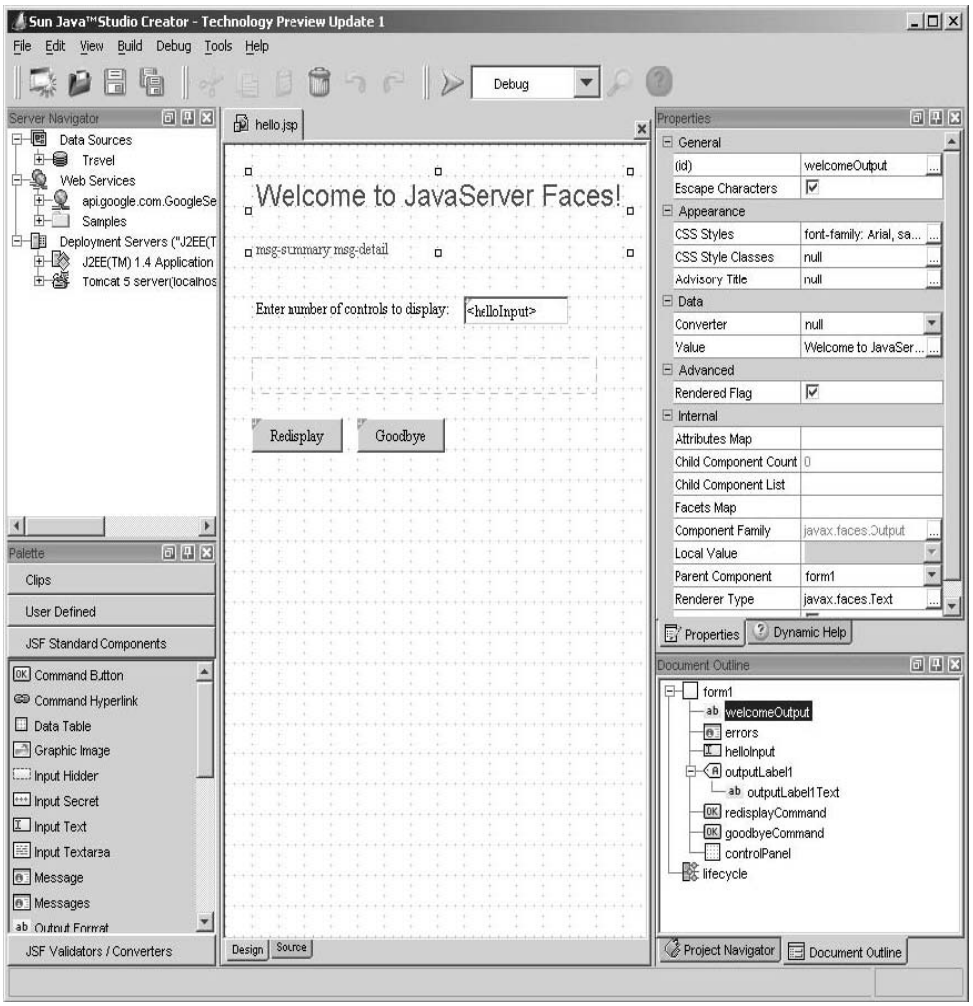


图 1.3 Sun 的 Java Studio Creator [Sun, Creator] 是一个易用的，基于可视化的开发 JSF 应用的环境。你可以在一个对 Visaul Studio.NET, Visual Basic, 或者 Delphi 用户来说很熟悉的环境中可视化的在 JSF 页面设计，JSP 源代码编辑，相关 Java 代码编写之间轻易地切换。

如果你点击 Swing 应用中的一个按钮，它将产生一个事件，而你可以直接在驻留于桌面中的代码来处理该事件。相反， web 浏览器并不知道有关 JSF 组件和其事件的任何东西；

它仅仅知道显示 HTML[3]而已。所以当你点击一个 Faces 应用按钮，它将产生一个请求，由浏览器发送到服务器。Faces 负责将该请求转换成一个可以被服务器中的应用逻辑所处理的事件。它也负责保证你在服务器所定义的每一个 UI 部件都正确显示给浏览器。

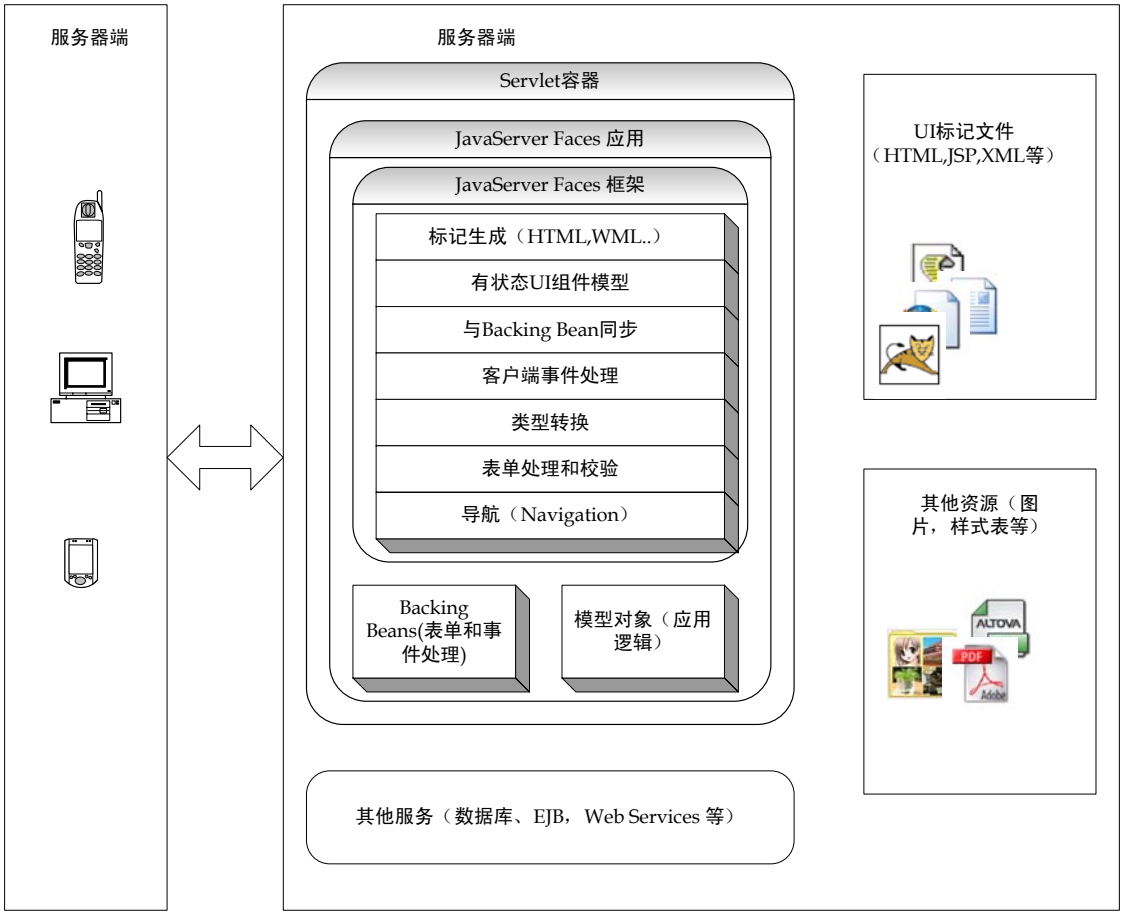


图 1.4 JavaServer Faces 应用的一个高阶视图。JSF 通过提供 UI 组件的支持，以及处理大量通用的 Web 开发事务而使 Web 开发更加容易。

图 1.4 展示了 Faces 应用的一个高阶视图。你可以看到，应用运行在服务器上，并且可以集成其它子系统，比如 EJB 服务或者数据库服务。当然，JSF 还提供许多其它一些服务，可以帮助你以更小的代价构建强大的 Web 应用。

JavaServer Faces 有一个特定的目标：使 web 开发更快更容易。它允许开发人员以组件，事件，Backing Bean 以及它们之间的交互来进行思考，而不是基于请求，响应和标记。换句话说，它掩盖了 Web 开发的大量的复杂性，使开发人员能够集中于他们最擅长的事情：开发应用系统。

注 JSF 是一种技术，本书使用了数百页的内容尽量彻底的介绍它。工具通过产生 JSP 的 GUI 设计器，编辑配置文件的屏幕，来包装了大量的开发难题。整本书中，我们都会展示 JSF 所使用的真实的 Java, JSP, 和 XML 代码以及配置，虽然这样难以使你感觉到工具可以使你

3 技术上将，它们还做了许多其他事情，比如执行 JavaScript 或 VBScript，显示 XML 和 XHTML，等等。

的开发工作容易一些。通过这种方法，你将完全理解 IDE 到底在幕后干了些什么，这对你的应用构建之后的维护，以及在你想从一个工具厂商转移到另一个厂商的情况，非常有用。当然，如果你根本不喜欢 IDE，知道事情实际是如何工作的也是最基本的要求。(如果你是一个 IDE 的发烧友，别担心，– 我们在书中会展示各种工具的屏幕截图，并且在线扩展的附录 B 中还包括对 3 个工具的详细介绍)。

1.1.2. 工业支持

对 JCP 社区和 Sun 扩展 Java 的方式来说，最好的事情莫过于有大量的公司，组织和个人投身其中。通过 JCP 产生一种规范实际并算不上快速，但结果却非常之好。JavaServer Faces 在 2001 年 5 月通过 Java 规范请求(JSR) 127 引入；规范的最终版本，JSF 1.0，在 2004 年 3 月 3 日才发布。而 JSF 1.1 (维护发布版) 则是 2004 年 5 月 27 日发布的。 参与开发 Faces 的公司和组织 (除 Sun 之外)包括 Apache 软件基金，BEA 系统，Borland 软件，IBM，Oracle，Macromedia，等等。

这些公司开发的产品可分为三类(某些可能适合不只一类)：J2EE 容器，开发工具，和 UI 框架。因为 JavaServer Faces 是一个与工具一起工作并运行于 J2EE 容器中的 UI 框架，这样做是很合情理的。最重要的是，这些公司中包括许多业界巨头。这意味着你可以期望 JSF 具有大量的工业支持。并且，如果你的供应商不支持 JSF，你也可以免费下载 Sun 的参考实现 [Sun, JSF RI]。

要跟踪最新的 JSF 新闻，文章，产品和厂商，请访问 JSF Central [JSF Central]，它是笔者运作的一个社区站点。

1.2. 引擎盖下面的技术

所有 JSF 应用都是标准的 Java web 应用。Java web 应用通过 Servlet API 和其他显示技术，如 JSP，来运行超文本传输协议 (HTTP)，如图 1.4。 显示技术用来定义与 Java 代码交互的组件所组成的 UI。Faces 应用也可以运行于 portlet 之中，它类似于 servlet。 JSF 组件架构使用 JavaBean 来暴露属性和处理事件。

在这一节，我们将简述这些技术，以及解释它们和 JSF 之间的关系。如果你已经熟悉 Java web 开发并且理解了它们和 JSF 之间的关系，你可以跳过这一节。

1.2.1. 超文本传输协议(HTTP)

外交官和政府首脑们来自于不同的文化地区，讲着不同的语言。为了沟通，他们遵循特定规则的礼仪和礼节，这就是协议。遵循协议可以使他们能够有效的沟通，即便是他们来自完全不同的背景。

计算机也使用协议来进行通信。遵循已经建立的协议使得程序能够进行沟通，而不用管是何种特定的软件，硬件或者操作系统。

万维网(WWW) 开启了一个共享文档的机制。这些文档通过超文本标记语言 (HTML) 来标记，允许人们查看文档并且可以非常容易地通过点击链接在文档间进行切换。为了适应这种文档以及支持这种超链接能力，开发了超文本传输协议 (HTTP)。它使得各种 web 浏览器能够以一种标准的方式从服务器获取文档。

定义 Web 原本是针对静态内容设计的，比如学术文档，它不会经常变更。相反，动态内容，比如股票信息或者产品订单，却是经常改变的。动态内容也是应用通常会产生的。

HTTP 是一个简单的协议—它基于文本头。客户发送一个请求给服务器，服务器发送一个响应给浏览器并附带所请求的文档。服务器是个哑巴[4]—如果请求了另一个文档，它并不记住客户的任何东西。这种失忆意味着 HTTP 是一个“无状态”协议；它自己并不在请求之间维护客户的信息。

HTTP 的无状态性意味着它能够很好的伸缩(毕竟它是一个 Internet 协议，而 Internet 则是一个巨大的场所)。这种特性对于 HTTP 原本要服务的静态文档并不是一个问题。

当时想象一下代客泊车的情况，并且没有给你收据也没有记住你的样子。当你返回的时候，它一定会花很多时间来找到到底那一辆车是你的。这就是在无状态环境中开发这种应用的问题。要解决这个问题有两种方式：cookies 和 URL 重写。这有点象泊车人给你一个收据，它自己也留着一联。

不管你使用何种语言，如果你要编写一个 web 应用，它都会使用 HTTP。Servlet 和 JSP 的开发使得在该协议之上构建应用更加容易。JavaServer Faces 的引入更使得开发人员能够彻底忘记他们所使用的协议。

1.2.2. Servlets

HTTP 对静态内容来说非常之好， web 本身都自带有这个功能。但是创建动态的内容需要编写代码。即便 HTTP 是很简单的，仍然要花许多工作来编写程序与之一同工作。你不得不解析头部，理解它们的意义，然后以正确的格式创建新的头。这就是 Java Servlet API 所关心的东西：提供面向对象的视图来使开发 Web 应用更加容易[5] 。HTTP 请求和响应被封装为对象，你可以访问输入和输出流，以便能够读取用户请求和写入动态内容。请求由 servlet 处理—Servlet 是处理特定 HTTP 请求的对象。

根据定义，一个标准的 J2EE web 应用基于 Servlet API。 Servlet 运行在一个容器中，容器本质上是一个 Java 应用，它执行所有幕后工作，比如与运行多个 servlet 相关的工作，关联组织为一个 Web 应用的资源的工作，以及管理所有其他相关服务的工作。最流行的 servlet 容器是 Tomcat [ASF, Tomcat]，但是有些 J2EE 应用服务器比如 IBM WebSphere [IBM, WAS] 和 Sun Java System Application Server [Sun, JSAS] 也提供 servlet 容器。

我们前一节说过， HTTP 的一个大问题是其无状态性。Web 应用解决这种问题的方法是使用会话—它们使得用户好像总是在，即使他们实际上并不在。会话（Session）是 Servlet API 提供的一个最大的优点。虽然幕后它还是使用 cookies 或者 URL 重写，但对程序员已经屏蔽了这种复杂性。

Servlet API 也提供其它很多好处，比如安全性，日志，生命周期事件，打包和部署，等等。这些特征也形成了 JavaServer Faces 的基础。事实上，JSF 本身就是实现为一个 servlet，而所有的 JSF 应用都是标准的 J2EE web 应用。

JSF 比 Servlet API 稍微更进了一步。Servlets 包含了构建 Web 应用的基本的必要的基础架构。但是这之后，你仍然不得不自己处理下层协议 HTTP 的属性：请求和响应。JSF 应用具有 UI 组件，它们和 backing bean 相关联，并产生应用逻辑所使用的事件。Faces 在其开发中使用 Servlet API，但是应用开发人员却可以在更高阶的抽象来获得这些优点：你可以

---

4 Web 服务器已经变得非常的综合了，但是最初它们是很简单的。

5 从技术上说，Servlet API 可用来提供各种请求响应环境的服务功能。它并不是一定要使用 HTTP 协议。在这一节，我们引用了 java.servlet.http 包，它是专门设计来处理 HTTP 请求的。

开发你的 web 应用而不用过多关心 HTTP 或者 Servlet API 自身的特定之处。

1.2.3. Portlets

大多数 web 应用都需要从数据存储中—通常是从数据库，获得动态数据。(即便业务逻辑是运行在另一种服务之上，如 EJB 或者 CORBA 服务，最后也是有一些代码和数据库打交道) 然而，从 Web 的早期开始，就有从各种数据源汇聚信息到易于使用的用户接口之中的需求。这些类型的应用，称之为***门户（Portals）***，最初属于某些公司的领域，如 Netscape 和 Yahoo! 。然而，越来越多的公司意识到，这种概念也非常适合于公司将来自于各种内部数据源的信息汇聚到一起给员工使用。

所以各种厂商，包括像 IBM, BEA, 和 Oracle 这样的巨头，都纷纷提供门户产品来简化这种任务。每个数据源都显示在 Web 页面中的一个区域，有点像一个窗口—你可以关闭该区域，定制其行为，或者与页面中的其它独立部分进行交互。每个这种区域就称为是***portlet***。

这些产商都有各自的 API 来编写工作于其门户产品中的 portlet。为了更加容易的开发能工作与不同门户中的 portlet， JCP 开发了 Portlet 规范 [Sun, Portlet]，它于 2003 年晚些时候发布了。所有主要的门户厂商 (包括 Sun, BEA, IBM, 和 Oracle) 以及开源组织如 Apache 软件基金都宣布在其门户产品中支持该规范。

Portlet 规范定义了 Portlet API。象 Servlet API 一样，它定义了大量的底层细节但是并没有简化 UI 开发或者屏蔽 HTTP。那是 JSF 的用武之地；它的开发能使它能够工作于 Portlet API。 (它在很多方式下非常类似于 Servlet API)。你可以在 Portlet 中使用常规的 JSF 组件，事件处理和其他特征，就像在 servlets 中一样。

注 贯穿本书，我们都将与 Servlet 一起进行相对讨论。然而，我们的大多数讨论同样也适合于 portlet。

.

1.2.4. JavaBeans

相当一部分 Java web 开发人员都以为 JavaBean 就是一个简单类，具有一些可通过 getter 和 setter 方法(访问器和修改器)来暴露的属性。例如，有一个 Java 类，具有方法 `getName` 和 `setName`，就表示暴露了一个可读写的属性 `name`。然而，属性仅仅是其冰山一角，JavaBean 是一个全能的组件架构，其设计本着工具支持。

这是很重要的，因为这意味着对它来说除属性之外还有很多东西。JavaBeans 遵循一定的模式，以便其他 Java 类能够动态发现事件和其它一些除属性之外的元数据。事实上，JavaBeans 正是驱动 Swing 的技术，并使得 IDE 能够提供 GUI builders 来构建桌面应用和 applet。使用 JavaBeans，你可以开发一个组件，不仅可以很精确地与可视化的 GUI builder 一起工作，也可以提供一个特定的向导 (或者*定制器*) 以引领用户进行配置流程。JavaBeans 也包括一个强大的事件模型(和 Swing 和 JSF 组件所使用的一样)，持久化服务，以及其它一些优雅的特征。

理解 JavaBeans 的强大之处将有助你领会 JSF 的全部魔力所在。象 Swing 组件一样，每个 JSF 组件都是一个全功能的 JavaBean。另外，Faces 组件设计来就是和 backing bean— 实现为 JavaBean 并且也处理事件的对象，一起工作。

如果你只是计划编写应用代码或者构建 UI，那么基本的 JavaBeans 的知识 (修改器和访问器) 已经足够。如果你想开发定制组件，对 Java Bean 的更深了解将会使你的开发工作更加容易。

1.2.5. JSP 和其他显示技术

Sample

Servlets 对 Web 开发来说是很低阶的构建块，但是它们并不能充分的简化显示动态内容的任务。你必须手动处理每一个请求的响应。

我们来假定你所发送的每一行 HTML 都由一行 Java 代码来处理。如果你的应用有 30 个页面，每一个页面有 80 行 HTML。你将有 2400 行类似于下面这样的语句：

```
out.println("This is a <b>really</b> repetitive task, and \"escaping\"" +
" text is a pain. ");
```

这真是一个单调乏味的工作，特别是你还不得不转移大量的字符，而且很难对它们进行快速的修改。毫无疑问，必须要有更好的解决办法。

为了解决这个问题，Sun 引入了 JSP 来作为标准的模版机制。JavaServer Pages 看起来像是 HTML 页面，但是它具有可以进行定制处理和显示 JavaBean 的值的特殊的标签，而且还可以在其中嵌入 Java 代码[6]。 从根本上说，其行为类似于一个看起来极象上面所示的代码片段的 servlet。JSP 翻译器承担了令人厌烦的工作，你自己可以不必亲为。

你可以创建你自己的定制标签[7] 来执行额外的处理(比如访问数据库)，并且已经有一些很有用的标准标签集，称为是 JSP 标准标签库(JSTL) [Sun, JSTL]。这样，你就可以通过类似于 HTML 的标签来定义 UI ，而不是使用麻烦的 Java 代码。

虽然 JSP 是一个工业标准的显示技术，你也可以有很多其他选择。你可以完全使用扩展标记语言/扩展样式表语言转换 (XML/XSLT) 方法，使用如 Cocoon [ASF, Cocoon]之类的框架，或者更为严谨的基于模版的方法，如 Velocity [ASF, Velocity] 或者 WebMacro [WebMacro]。还有其它更多的选择。

JSF 的设计目标之一就是避免对某种显示技术的依赖。所以 JSF 提供了可插入接口以允许开发人员能够将其集成到各种显示技术之中。然而，因为 JSF 是一个标准的 Java 技术，并且 JSP 也是，毫不奇怪 Faces 出场时是使用 JSP 实现(通过定制标签)。并且因为 JSP 是唯一的规范要求必须和 JSF 集成的显示技术，本书的大部分例子也是使用 JSP。

1.3. 框架，框架，还是框架

先前我们说过， JavaServer Faces 是一个在 Java 中开发 Web UI 的框架。这些日子好象框架非常流行，有一个很好的原因：那就是它们使得 Web 开发更加容易。象大部分 Java web 框架一样，JSF 遵循业务逻辑和显示的分离。并且，它更偏重于问题的 UI 那一侧，并且可以和其它框架进行集成，比如 Struts。

1.3.1. 我们为什么要使用框架 ?

随着人们开发越来越多的 web 应用，很明显，虽然 servlets 和 JSP 非常有用，但它们不能在不用那些讨厌的代码的情况下处理很多通用的任务。框架的目的就是简化这种任务。

最基本的任务是表单处理。HTML 页面中有表单，它们是用户输入的控件集，比如文本框，查找列表，复选框等等。当用户提交一个表单时，输入字段中的所有数据都发送给服务器。HTML 中的一个文本字段可能象这样：

---

6 将 Java 代码嵌入到 JSP 被很多人视为是一种不好的设计(也是一种不好的做法) ，并且引起了一场巨大的口水战。主要的争论是他们没有遵循显示和业务逻辑的分离。这种“特性”是为什么有很多不同的显示技术的原因。在 JSP 2.0 中，你可以关闭这个特征。

7 从技术上说，定制标签也称“定制动作,”但是我们在本书使用更通用的词汇“定制标签（custom tags）”。

```
<input maxLength=256 size=55 name="userName" value="">
```

在一个标准的 servlet 应用中，开发人员必须直接从 HTTP 请求中检索出这些值：

```
String userName = (String)request.getParameter("userName");
```

如果有大量的表单，这同样让人生厌。因为你要直接处理这些发自浏览器的值，而且 Java 代码必须保证这些数据的有效性。此外，这些值中每一个都必须手动的和应用的业务对象发生关联。

表单仅仅是这些 JSP 和 Servlet 不能完全解决的任务的一个例子。Web 应用必须管理大量的页面和图片，如果你没有一个中心化的管理方式来管理它们，在一个大型的应用中引用所有这些资源将是一个恶梦。

页面结构的管理是另外一个问题。虽然 JSP 提供了一个机制来创建动态页面，但它并没有提供将一个页面组合为更小的，可重用的部件的可扩展性支持。更有意思的是，servlet 的处理居然不支持国际化，类型转换和错误处理。

为了以一种简单的方式来解决这些所有问题，诞生了许多框架。有些还非常流行，比如 Struts [ASF, Struts] 和 WebWork [OpenSymphony, WebWork]。这些框架的开发都是旨在解决以上这些公共问题。

1.3.2. 她是模型 2

对大型 Web 应用来说，基本的组织和管理服务是必要的，但是它们也需要良好的结构。大部分 web 框架，包括 JSF，都遵循某种模型-视图-控制器 (MVC) 设计模式的变体。为了理解 MVC 到底是什么，我们来看看驾驶的问题。

当你在高速公开路上向着一个方向驾驶，通常在你和向着你的反方向行驶的车辆之间有一个隔离带或者中心线。这里设置隔离带有一个很好的原因，即反方向快速移动的车辆不能混合到你行驶方向的车流中来，否则，没有隔离带，轻微的事故也可能造成大的后果。

应用也有相似的问题：业务逻辑代码不能混合到 UI 代码之中。当这两者混起来的时候，应用将难以维护，仅具有很小的伸缩性，并且通常更加脆弱。此外，你不能有一个团队工作于 UI 表现代码，而另一个团队则负责业务逻辑。

MVC 是这种问题的标准解决方案。当你观看新闻的时候，你观看的是现实的一个版本。如果有某个事件发生，新闻频道一般都会负责解释该事件并广播该解释。虽然你在电视上看到了节目，但是在实际发生的事情，做报道的记者如何理解该事件，以及你正在电视上看到的节目内容之间，仍然可能存在截然的不同。新闻频道*控制*着电视节目—*视图*—与实际的事件—*模型*之间的交互。即使你可能正在电视上观看新闻，但同样的频道可能正通过 Internet 广播或者产生书面的出版物。有多种可选的视图。如果这些生产部件不是分离的，就不可能这样。

在软件的角度看来，视图是表现层，它负责与用户进行交互。模型是业务逻辑和数据，而控制器则是响应用户事件和集成模型和视图的应用代码。这种架构确保了应用是松散耦合的，因而减小了层间的依赖性。

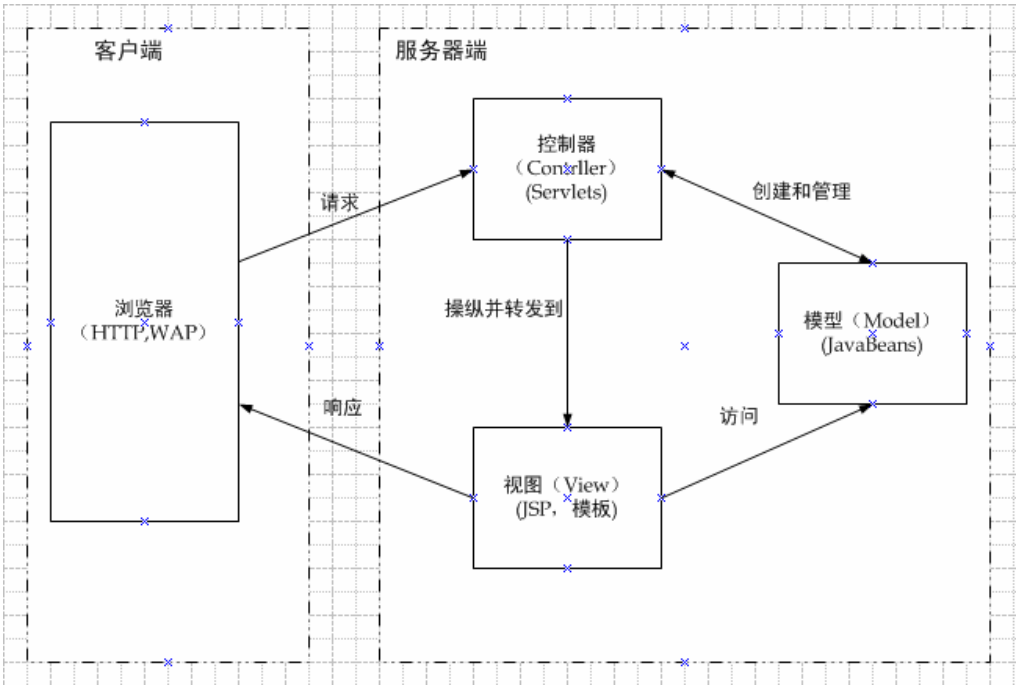


图 1.5 大多数 web 框架都使用 Model 2 设计模式的一些变体。

模型 2 (如图 1.5) 是一个 MVC 的 Web 应用变体。其基本点是：

- 模型可以由普通旧式 Java 对象(POJOs), EJB, 或者其它东西组成。
- 视图可以是 JSP 或者其它表现技术。
- 控制器总是实现为 servlet。

所以，如果 JSP 页面包含一个错误，它不会影响到应用代码或者模型。如果在模型中有一个错误，它也不会影响到应用代码和 JSP 页面。这种分离允许在每一个层进行单元测试，并且也使得工作于不同层的开发小组彼此间能够独立工作。例如，前端开发人员可以在业务对象和应用代码完成之前构建 JSP 页面。其它部分的层也可以在这 3 个层全部完成之前集成进来。

这些优点实际上就是为什么大多数框架，包括 JSF，都支持 MVC 设计模式的某种变体的原因。

1.3.3. JSF，Struts，和其他框架

我们目前实际面临这一情况：有大量的 Java web 框架可用。它们中有些，比如 Struts [ASF, Struts] 和 WebWork [OpenSymphony, WebWork]，有助于解决表单处理和其它问题，比如遵循 Model 2，集成数据源，以及通过 XML 配置文件集中控制引用的所有应用资源。这些基本框架都提供了广泛的基础设施，但是都还没有屏蔽基本的 HTTP 请求响应处理的复杂性。

其他框架，象 Tapestry [ASF, Tapestry]，Oracle 的应用开发框架 (ADF) UIX [Oracle, ADF UIX]，以及 SOFIA [Salmon, SOFIA]，都提供一个 UI 组件模型和某些事件处理机制。这些 UI 框架，包括 JSF，目的是为了简化整个编程模型。经常，基础架构和 UI 框架具有重叠的功能。

为了理解这种重叠，你可以将 web 应用架构想象成一个服务栈。靠近栈底部的服务没能抽象基础协议的许多细节；它们更像粗加工品。栈中靠近顶部的服务则隐藏了更多讨厌的细节，提供更高级别的抽象。最低层的服务由 web servers，Servlet API，和 JSP 处理。 大部分框架都提供一些附加服务的子集。图 1.6 显示了这个栈，以及与 JSF，Struts，servlets，JSP，和典型的 web server 的关系。

你可以从图中看到 JSF 支持了很多的服务，这也使得它自己成为一个强大的框架。在大多数情况下，这就是你需要的东西。后续发布的 Faces 极有可能也会包括一些更多的服务。

然而，即使 Faces 与 Struts 这样的框架有些重叠，也并不是必须替代它们。 (事实上，如 Struts 的领导, Craig McClanahan, 也是 JavaServer Faces 的开发领导) 如果你将它们集成起来，你就可以访问栈中的所有服务(第 14 章将包含 Struts 集成)。你也可以和其他框架一起使用 JSF，比如 Spring [Spring-Faces]。

对于面向 UI 的框架，JSF 和他们很多功能都有重叠。这些项目的某些声称将在其未来版本中支持 JSF。Faces 的独特之处在于有通过 JCP 的工业巨头参与的开发联盟，以及将成为 J2EE 的一部分。作为结果，它将分享受强有力的工具支持，并将随很多 J2EE server 一起交付。

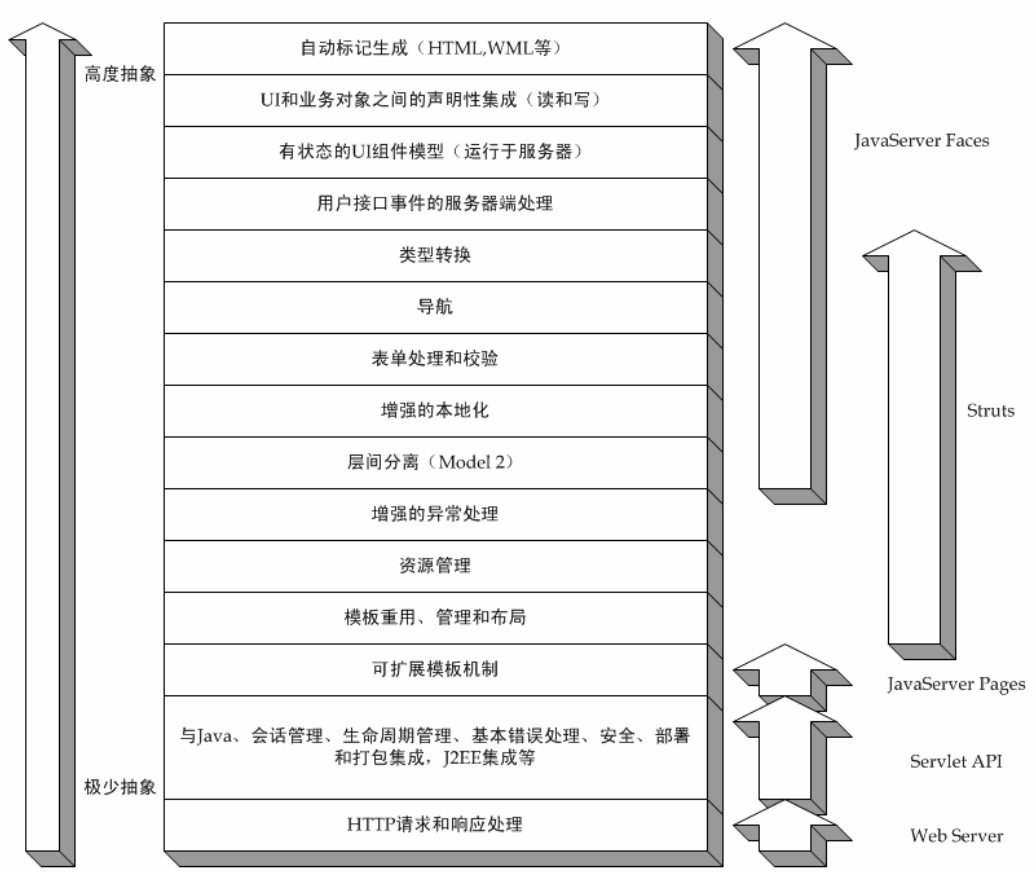


图 1.6 Web 应用基础架构可以视为一个服务栈。底层服务提供的基础支撑稍微有一点的抽象，而栈的顶层的服务则提供更多抽象。将所有服务整合在一起将极端强大。

1.4. 组件无处不在

令人悲哀的是，“组件”一词的过度滥用在今天已经到处蔓延了。操作系统是一个组件，应用程序是一个组件，EJB 是组件，库是组件，甚至厨房的水槽也是。有大量的书籍论及组件，而那些优秀的好书都指出组件有好多定义存在。

如果你知道它的确实意义，对这个词的滥用，你就不会感到陌生。如果你在词典中查找“组件（component）”这个词，你就会看到它是要素（*constituent*）—整体的一个部分，的一个同义词。因此，如果你使用这个词的字面意思，那么在一个分布式应用的上下文中，操作系统确实是一个组件。

更有趣的是，从概念上讲，厨房水槽与操作系统相比对 Faces 组件来说更有共通之处。你不用自己从头制造它—你只需要购买一个符合你需要的水槽：尺寸，颜色，材料，容器数，等等。对其它厨房用品也是如此，比如橱柜和工作台面。所有这些组件都有特定的接口可以使它们能够和其它东西进行集成，但是依赖于特定的环境服务。(例如，接口管)。最终的结果可能是独特的，但整体是由独立可重用的部件组成。

如果我们采用厨房组件的概念，并应用到软件上，我们会得出这个定义：

定义 一个软件组件是一个具有特定的契约接口并且仅具有显式上下文（环境）依赖性的成分单元。一个软件组件可以独立部署，并且服从第 3 方的组成。[Szyperski].

厨房的*“上下文依赖性（context dependencies）”*就是指对诸如房间本身，配管，电路等等因素的考虑。本质上，上下文是所有组件的容器。而*容器*则是拥有组件，并且提供一系列允许进行组件操作的服务的系统。有时，这种操作在 IDE (*设计时*)中进行，有时则在部署环境中运行，比如 J2EE server 之中 (*运行时*)。

短语“独立部署” 意味着一个组件是一个自包含的单元，可以被安装到一个容器中。厨房水槽是一个独立的，自包含的组件，可以安装在工作台中。

当你改造你的厨房时，你雇用一个承包商，由他来将你所选择的组件 (橱柜，抽屉，水槽等等) 组装成为一个完整的厨房。我们使用组件构建软件时，我们也是将各种组件组装起来，创建能够运行的软件系统。

JSF 组件，Swing 组件，servlet, EJB, JavaBean, ActiveX 控件，以及 Delphi 可视组件库 (VCL) 组件都符合这个定义。但这些组件却各自关注不同的事情。JSF 和 Swing 组件单独针对 UI 开发，而 ActiveX 和 VCL 控件可以影响，也可以不影响 UI。Servlets 和 EJBs 则更加粗粒度一些— 它们在应用和业务逻辑领域提供大量的功能。

因为 JSF 着眼于 UI 组件，我们来相应窄化我们的组件定义。

定义 一个 UI 组件，或者控件（control），是提供与最终用户交互的特定功能的组件。经典的例子包括工具条，按钮，面板，和日历等。

如果你是在开发传统的 GUI 应用，那么 UI 组件的概念应该对你非常熟悉了。JavaServer Faces 的精彩之处在于将标准的 UI 组件模型引入到 Web 世界。它为桌面开发人员所认定的东西：即通过扩展工具支持提供对封装的 UI 功能的宽范围选择，搭建了一个舞台。它也为创建处理针对特定业务领域任务的定制组件---比如提供报表浏览器或者利息计算器，打开了一扇大门。

1.5. Hello, world!

既然你已经有一个对 JavaServer Faces 能够解决的问题的初步理解，我们就开始来做一个

简单的应用吧。本节假定你熟悉 Java web 应用和 JSP。(关于这些技术的信息，请参考第 1.2 节获得一个概貌。) 我们将解剖一个简单的基于 HTML 的 web 应用，该应用有两个页面： hello.jsp 和 goodbye.jsp。

hello.jsp 页面做以下事情：

- 显示文本“Welcome to JavaServer Faces!”
- 有一个表单，其中有一个文本框，要求输入 1 到 500 的整数
- 在名为 `numControls` 的 JavaBean 属性中存储提交的文本值
- 文本框下面有一个表格
- 有一个标注为“Redisplay”的按钮，当点击它时，将添加一个 `numControls` 输出 UI 组件到表格中(如果先前已经有了 UI 组件先清除它们)
- 有一个标注为 “Goodbye”的按钮，点击它显示 goodbye.jsp 页面

goodbye.jsp 页面做以下事情：

- 显示文本“Goodbye!”
- 显示 JavaBean 属性 `numControls` 的值

JSF 执行了我们的 Hello, world! 应用的大部分工作，但是除了 JSP 页面之外，还有一点其他要求：

- `HelloBean` backing bean 类
- Faces 配置文件
- 正确配置的部署描述符

某些工具可以简化一些或者全部需求的创建工作，但在本节，我们将详细讨论这些文件的原始内容。

我们涉足细节之前，先来看看 Hello, world! 到底在 web 浏览器中看起来是什么样。应用开始于 hello.jsp，如图 1.7 所示。页面中的文本框与 `HelloBean` 类中的一个 JavaBean 属性相关联；当用户输入一个值到此文本框中，属性将自动更新 (如果值是有效的话)。

图 1.7 在数据提交之前的 Hello, world! 应用。

如果在文本框中输入数字“64”，点击 Redisplay 按钮，页面将重新显示，如图 1.8—总共 64 个 UI 输出组件显示在表格中。如果你清除文本框，然后点击 Redisplay 按钮，你将会得到一个校验错误，如图 1.9 所示。如果你在其中输入数字 “99999” ，然后点击 Redisplay 按钮，也会得到校验错误，如图 1.10。

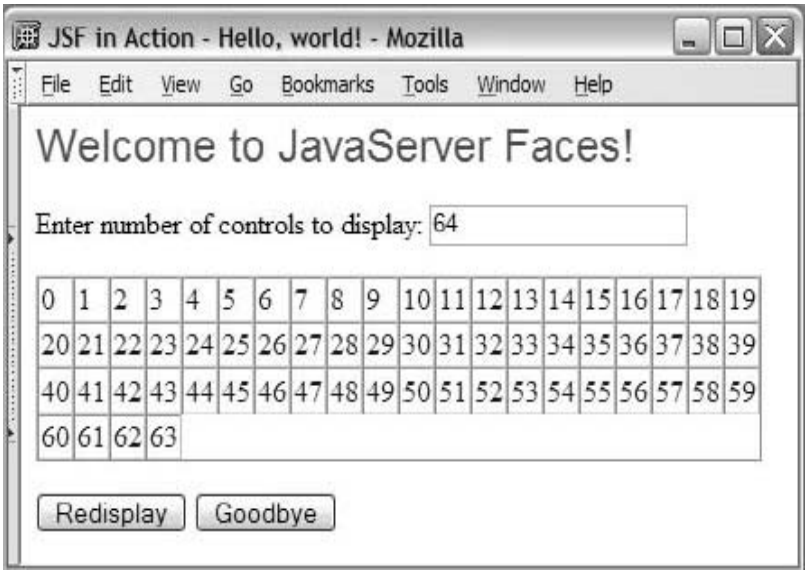


图 1.8 输入“64”并且点击 Redisplay 按钮后的 Hello, world! 应用。表格内组装了 64 个 UI 输出组件。

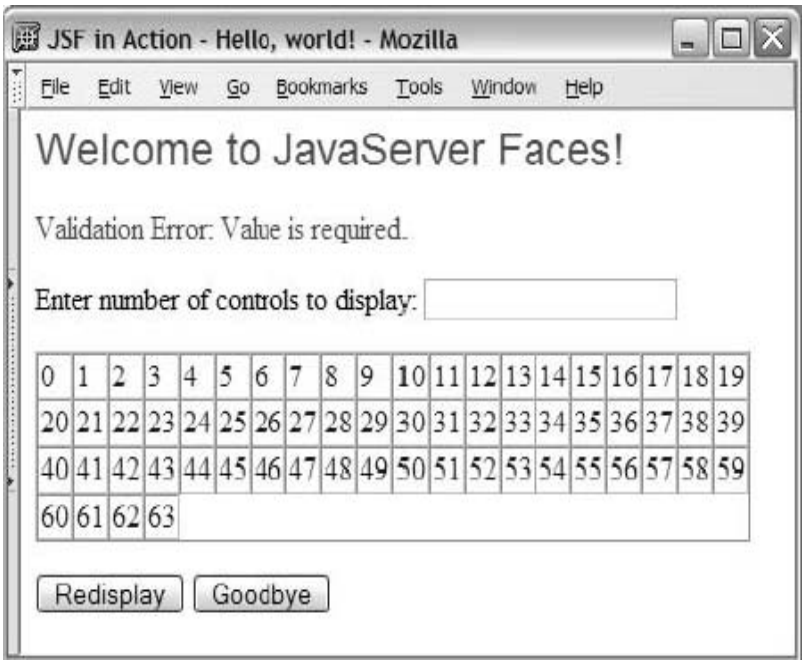


图 1.9 对必须填写的字段提交了空值，并且点击了 Redisplay 按钮之后的 Hello, world! 应用。因为发生了校验错误，JavaBean 属性的相关值不会被修改。

不要担心错误信息，在你的应用中，你完全可以定制它们。重要之处在于，在这两种情况下，当表单提交时，相关的 JavaBean 属性没有被修改。

如果你点击 Goodbye 按钮，你将看到 goodbye.jsp 页面，如图 1.11。虽然这是一个完全不同的页面，JavaBean 属性的值也会显示。JSF 组件可以引用生长在应用范围上下文中的 JavaBean。

我们的 Hello, world! 示例是一个标准的 Java web 应用，因为使用标准的 Servlet API (虽然，它要求标准的 Faces 库)。这 5 个图都是由两个 JSP 产生的。我们来详细讨论它们。

1.5.1. 解剖 hello.jsp

我们的主页面，hello.jsp，提供了图 1.7 到 1.10 的接口。JSF 通过使用定制标签库和 JSP 集成。JSF 定制标签允许 JSP 中使用 Faces UI 组件。某些工具将使得你可以通过从面板中拖放 JSF 组件来设计 JSP 页面。事实上，图 1.1 到 1.3 就是在不同的 IDE 中设计 hello.jsp 的屏幕截图。这些 IDE 最终都产生如清单 1.1 类似的代码 (当然，你也可以手工编写 JSF 页面)。

图 1.10　在文本框内输入"99999" 然后点击 Redisplay 之后的 Hello, world! 应用。字段仅接受 1 到 500 之间的数值，所以产生如图所示的校验错误。因为发生了校验错误，相关的 JavaBean 属性的值不会被修改。

图 1.11　点击 Goodbye 按钮之后的 Hello, world! 应用。注意 JavaBean 属性，它与第一个页面中的文本字段保持同步，也被显示在此页面中。

清单 1.1 Hello.jsp: Hello,world! 应用的起始页面（浏览器输出示于图 1.7-1.10）

Sample

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
 <html>
  <head>
   <title>
     JSF in Action - Hello, world!
   </title>
  </head>
  <body>
   <h:form id="welcomeForm">
    <h:outputText id="welcomeOutput"
                  value="Welcome to JavaServer Faces!"
                  style="font-family: Arial, sans-serif; font-size: 24;
                         color: green;"/>
    <p>
      <h:message id="errors" for="helloInput" style="color: red"/>
    </p>
    <p>
      <h:outputLabel for="helloInput">
        <h:outputText id="helloInputLabel"
                      value="Enter number of controls to display:"/>
      </h:outputLabel>
      <h:inputText id="helloInput" value="#{helloBean.numControls}"
                   required="true">
          <f:validateLongRange minimum="1" maximum="500"/>
      </h:inputText>
    </p>
    <p>
      <h:panelGrid id="controlPanel"
                   binding="#{helloBean.controlPanel}"
                   columns="20" border="1" cellspacing="0"/>
    </p>
    <h:commandButton id="redisplayCommand" type="submit"
                     value="Redisplay"
                     actionListener="#{helloBean.addControls}"/>
     <h:commandButton id="goodbyeCommand"
                      type="submit" value="Goodbye"
                      action="#{helloBean.goodbye}"
```

① JSP 标签库

②包围所有 JSF 标签的标签

③ HtmlForm 组件

④ HtmlOutputText 组件

⑤ HtmlMessage 组件

⑥HtmlOutputLabel 及子组件HtmlOutputText

⑦ HtmlInputText 组件

⑧ HtmlPanelGrid 组件

⑨ HtmlCommandButton 组件

```
                        immediate="true"/>
     </h:form>
    </body>
  </html>
</f:view>
```

①首先，我们导入核心 JavaServer Faces 标签库。该库提供象校验和事件处理之类的基本任务的定制标签。接着，我们导入了基本 HTML 标签库，它提供诸如文本框，输出标签和表单之类的 UI 组件。(前缀“f ”和 “h” 是建议的，非必要。)

②<f:view> 标签必须将所有其他 Faces 相关的标签扩起来,（包括来自于核心标签和基本 HTML 标签库中的标签)。

③<h:form> 标签表示一个 HtmlForm 组件，它是一个其它组件的容器，并用于将信息提交给服务器。在一个页面内可以有多个 HtmlForm，但输入组件必须嵌入<h:form> 标签中。

④<h:outputText> 标签创建一个 HtmlOutputText 组件，该组件只是在屏幕上显示只读数据。此标签有一个 id 属性和一个 value 属性。id 属性对每个组件是可选的；它不是必须属性,除非你想要在其它地方引用该组件。(组件可以使用客户端技术如 JavaScript 引用或者在 Java 代码中引用)。Value 属性的值是需要显示的文本。

⑤<h:message> 标签表示一个 HtmlMessage 组件，它显示特定组件的校验和转换错误信息。for 属性告诉它需要显示标识符为 helloInput 的错误信息。helloInput 是页面中④处的文本框的标识符。如果没有错误则什么都不显示。

⑥<h:outputLabel> 标签创建一个新的 HtmlOutputLabel 组件，它用于输入控件的标注。For 属性将它和该输入控件相联系，即 helloInput 文本框⑦。HtmlOutputLabels 并不显示什么，所以我们还需要一个下属 HtmlOutputText (由嵌套的<h:outputText> 标签创建)来显示标注的文本。

⑦<h:inputText>标签用于创建一个 HtmlInputText 组件，该组件接收用户的文本输入。注意，该组件的 value 属性是"#{helloBean.numControls}"，这是一个 JSF 表达式语言(EL)写成的表达式，引用了 Backing Bean 中的 numControls 属性，该 Bean 名为 HelloBean。(JSF EL 基于 JSP 2.0 引入的表达式语言)。Faces 将自动在不同的上下文（请求，会话，应用）中查询特定的 backing bean。 这样，它将在应用会话中找到在关键字 helloBean 下存储的 Backing Bean。组件的 value 属性和 helloBean 的 numControls 属性是同步的，如果一个变更了，另一个也要修改(除非 HtmlInputText 组件中的文本无效)。

输入组件有一个 required 属性,它决定该字段是否必须有一个值。这样,如果 required 属性被设置为 true，则组件将只能接受非空输入。如果用户输入了一个空值，页面将重新显示，而 HtmlMessage ⑤将显示一个错误信息，如图 1.9。

JSF 也支持校验器，它负责确保用户的输入是可接受的值。每个输入控件都可以和一个或者多个校验器相关联。<f:validateLongRange> 标签注册了一个 LongRange 校验器到 HtmlInputText 组件。校验器检查以确保用户输入是在预设的 0 到 500 之间的一个数

（包含端点）。如果用户输入一个超出此范围的值，校验器将拒绝该输入，页面重新显示，并且 HtmlMessage ⑤组件显示一个错误信息，如图 1.10。

当用户输入被拒绝时，HtmlInputText 组件引用的对象的 value 属性将不会更新。

⑧HtmlPanelGrid 组件用<h:panelGrid> 标签来表示。HtmlPanelGrid 表现为一个其它组件的可配置容器，将显示为一个 HTML 表格。

许多 JSF 组件都可以通过其 JSP 标签的 binding 属性直接和 Backing Bean 相联系。 (某些工具能够自动对页面上的所有组件做这件事。) 此标签的 binding 属性被设置为 "#{helloBean.controlPanel}"。这也是一个 JSF EL 表达式，它引用了 helloBean 的 controlPanel 属性,这是一个 HtmlPanelGrid 类型的属性。这样确保 helloBean 总是能够访问页面上的 HtmlPanelGrid 组件。

⑨<h:commandButton> 指示一个 HtmlCommandButton 组件，它显示为一个 HTML 表单按钮。HtmlCommandButton 在被用户点击后会发送活动事件到服务器中的应用。可以通过 actionListener 属性直接引用事件监听器(一个执行事件响应的方法)。第一个 HtmlCommandButton 的 actionListener 属性设置为 "#{hello-Bean.addControls}"，这仍是一个 JSF EL 表达式，告诉 JSF 去查找 helloBean 对象，然后调用其 addControls 方法来处理事件。一旦方法执行完毕，页面将重新显示。

第 2 个 HtmlCommandButton 按钮设置 action 属性而不是 actionListener 属性。该书性的值为"#{helloBean.goodbye}"，这个表达式引用到一个特定的处理导航的事件监听器。这也是为什么点击这个按钮将显示 goodbye.jsp 页面而不是重新显示 hello.jsp 页面。这个按钮还有个 immediate 属性设置为 true，这告诉 JSF 在校验和更新发生之前就执行相关的监听器。这样，即使输入不正确，点击此按钮也可以工作。这就是整个 hello.jsp。

清单 1.2 列出了校验出错之后的 HTML 输出。 (浏览器的视图显示于图 1.10).

清单 1.2 hello.jsp 的 HTML 输出(此代码是图 1.10 的源代码)

```
<html>
 <head>
  <title>
   JSF in Action - Hello, world!
  </title>
  </head>
  <body>
   <form id="welcomeForm" method="post"
         action="/jia-hello-world/faces/hello.jsp"
         enctype="application/x-www-form-urlencoded">

     <span id="welcomeForm:welcomeOutput"
           style="font-family: Arial, sans-serif; font-size: 24
           color: green;">Welcome to JavaServer Faces!
```

①HtmlForm组件

HtmlOutputText组件

```
 </span>
  <p>
    <span id="welcomeForm:errors" style="color: red">
      Validation Error: Specified attribute is not
      between the expected values
      of 1 and 500.</span>
 </p>
 <p>
   <label for="welcomeForm:helloInput">
      <span id="welcomeForm:helloInputLabel">
          Enter number of controls to display:</span>
  </label>
  <input id="welcomeForm:helloInput" type="text"
        name="welcomeForm:helloInput" value="99999"/>
</p>
<p>
<table id="welcomeForm:controlPanel" border="1" cellspacing="0">
  <tbody>
   <tr>
     <td><span style="color: blue"> 0 </span></td>
        ...
     <td><span style="color: blue"> 19 </span></td>
   </tr>
   <tr>
     <td><span style="color: blue"> 20 </span></td>
      ...
     <td><span style="color: blue"> 39 </span></td>
   </tr>
   <tr>
     <td><span style="color: blue"> 40 </span></td>
       ...
      <td><span style="color: blue"> 59 </span></td>
  </tr>
  <tr>
    <td><span style="color: blue"> 60 </span></td>
       ...
    <td><span style="color: blue"> 63 </span></td>
  </tr>
 </tbody>
</table>
```

② HtmlMessage组件

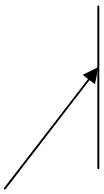
HtmlOutputLabel组件
和
HtmlOutputText组件

HtmlInputText组件

③HtmlPanelGrid
组件

Sample

```
  </p>
   <input id="welcomeForm:redisplayCommand" type="submit"
      name="welcomeForm:redisplayCommand" value="Redisplay" />
   <input id="welcomeForm:goodbyeCommand" type="submit"
      name="welcomeForm:goodbyeCommand" value="Goodbye" />
     ...
  </form>
</body>
</html>
```



④HtmlCommandButton组件

你可以看到在 JSP 中定义的每一个组件都在 HTML 页面中有一个对应的表现。注意，`<h:form>` 标签①，它表示一个 `HtmlForm` 组件，具有一个 `action` 属性，它指向调用此页面的 JSP，但是具有一个前缀“faces”。这是 Faces servlet 的一个别名，是在应用的部署描述附中定义的。重新显示调用页面是默认行为，但 Faces 应用也可以被导航到其它页面 (这就是用户点击 Goodbye 按钮后发生的事情)。

`HtmlMessage` ②组件的输出是文本 “Validation Error: Specified attribute is not between the expected values of 1 and 500。” 你可以想到，此消息正是由我们注册到页面中的 `LongRange` 校验器产生的。当校验器拒绝了提交不正确输入值的尝试时，它也产生一个错误信息，而框架则避免更新相关的 JavaBean 的属性值。

每个对应于 JSF 组件的 HTML 元素都有一个继承自 JSP 中指定的 `id` 值的 `id` 属性。 (如果没有指定，将自动创建一个)。这就是客户端标识符，并且它是 JSF 用来将输入值映射到服务器组件的标识。某些组件也使用名称来作为客户端标识符。

`HtmlPanelGrid` 组件③的输出是一个 HTML 表格。注意， JSP 中的 `border` 和 `cellspacing` 属性直接传递到 HTML。(大多数标准的 HTML 组件都会暴露一些将直接传递到浏览器的 HTML 特定的属性。) 表格中的每个单元格都输出一个 `HtmlOutputText` 组件，它是通过 Java 代码在响应用户点击 Redisplay 按钮之后添加到其中的。 (在实际的 HTML 中，有 64 个单元格，因为这是用户在文本框中输入的数字。我们在其中省略了好些，因为它会浪费大量的篇幅!)

我们将会很快研究 Java 代码，但现在看看 goodbye.jsp 先。

1.5.2. 解剖 goodbye.jsp

goodbye.jsp 页面的效果示于图 1.11，当用户点击 Goodbye 按钮时显示。页面代码 (清单 1.3) 包含了一些和 hello.jsp 页面中项同的组件：到入 JSF 标签库，一个 HtmlForm 组件，一个 HtmlOutputText 组件。一个 HtmlOutputText 组件引用到河前一个页面中相同的 helloBean 对象。这运行的很好，因为对象在 application 的会话中，并且因为可以在页面请求间存活。

清单 1.3 goodbye.jsp: Hello, world! 应用的结束页面(浏览器输出示于图 1.11)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
  <html>
    <head>
      <title>
        JSF in Action - Hello, world!
      </title>
    </head>
    <body>
      <h:form id="goodbyeForm">
      <p>
        <h:outputText id="welcomeOutput" value="Goodbye!"
              style="font-family: Arial, sans-serif; font-size: 24;
                     font-style: bold; color: green;"/>
      </p>
      <p>
         <h:outputText id="helloBeanOutputLabel"
                       value="Number of controls displayed:"/>
         <h:outputText id="helloBeanOutput"
                       value="#{helloBean.numControls}"/>
      </p>
    </h:form>
    </body>
  </html>
</f:view>
```

和 **hello.jsp** 中同一个 **backing Bean**

此页面产生的 HTML 和前一节相比没有什么特别需要说明的地方，所以我们就不浪费时间。重要的是我们可以创建一个有功能的应用，包括校验和导航，尽管仅有两个页面。(如果我们不打算展示导航，第一个页面就足够了。)

现在，我们可以看看这些页面背后的代码了。

1.5.3. 检视 HelloBean 类

页面 hello.jsp 和 goodbye.jsp 都包含了通过 JSF EL 表达式引用一个名为 helloBean 的 Backing Bean 的 JSF 组件。这个 JavaBean 包含了这个应用所需的所有东西：两个属性和两个方法。其代码示于 listing 1.4。

清单 1.4 HelloBean.java: 我们的 Hello, world! 应用的一个简单的 backing bean

```
package org.jia.hello;
import javax.faces.application.Application;
import javax.faces.component.html.HtmlOutputText;
import javax.faces.component.html.HtmlPanelGrid;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;
import java.util.List;
public class HelloBean
{
  private int numControls;
  private HtmlPanelGrid controlPanel;

  public int getNumControls()
  {
    return numControls;
  }

  public void setNumControls(int numControls)
  {
    this.numControls = numControls;
  }

  public HtmlPanelGrid getControlPanel()
  {
    return controlPanel;
  }

  public void setControlPanel(HtmlPanelGrid controlPanel)
  {
    this.controlPanel = controlPanel;
  }

  public void addControls(ActionEvent actionEvent)
  {
    Application application =
```

① 没有必需的超类

② 两个JSP中都引用了的属性

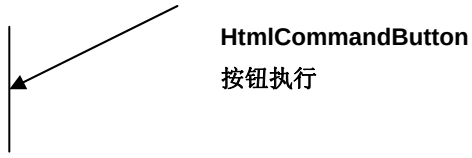
③ 绑定到HtmlPanelGrid的属性

④ 被RedisplayHtmlCommand按钮执行

```
        FacesContext.getCurrentInstance().getApplication();
    List children = controlPanel.getChildren();
    children.clear();
    for (int count = 0; count < numControls; count++)
    {
      HtmlOutputText output = (HtmlOutputText)application.
            createComponent(HtmlOutputText.COMPONENT_TYPE);
      output.setValue(" " + count + " ");
      output.setStyle("color: blue");
      children.add(output);
    }
  }

  public String goodbye()
  {
    return "success";
  }

}
```



①和其他很多框架不同，JSF backing bean 并不是非要继承自什么特定的类。它们只需要简单地使用普通的 JavaBean 约定和在它们的事件处理方法中使用特定的方法签名来暴露其属性和方法即可。

②`numControls` 属性被 hello.jsp 的 `HtmlInputText` 组件和 goodbye.jsp 中的 `HtmlOutputText` 组件所引用。无论何时用户改变 `HtmlInputText` 组件的值，此属性的值也都要相应被改变(如果输入有效的话)。

③ `controlPanel` 属性的类型是 `HtmlPanelGrid`，后者是在 hello.jsp 中使用 `<h:panelGrid>` 标签创建的一个实际的 Java 类。该标签的 `binding` 属性和标签的 `controlPanel` 属性创建的组件相关联。这样可以使 `HelloBean` 能够操作实际的代码—即它在④处非常愉快地执行的任务。

④ `addControls` 是一个用来处理 action 事件的方法(即一个*事件监听器（action listener）方法*)；你可以告诉它应该做些什么，因为它接受 ActionEvent 作为其唯一的参数。在 hello.jsp 中，`HtmlCommandButton` 类型的“Redisplay”按钮通过其 `actionListener` 属性引用这个方法。这就告诉了 JSF，当用户点击 Redisplay 按钮时，在处理当用户点击 Redisplay 按钮时产生的动作事件时调用这个方法。(如果你习惯于使用 Swing 之类的框架，将一个组件和一个事件监听器相联系似乎会显得有些陌生，因为它们通常需要一个单独的事件 listener *接口*。JSF 也支持接口风格的 listeners，但是最好的方法还是使用 listener 方法，因为它减轻了 Backing Bean 中对适配器类的需求。)

当此方法被执行时，它添加一个新的 `HtmlOutputText` 组件到 `controlPanel` 中，并执行 `numControls` 次(首先清除它)。所以，如果 `numControls` 的值是 64，如例

子中所述，代码将创建并添加 64 个 HtmlOutputText 实例到 controlPanel 中。每个实例的值被设置为其顺序号，从 0 开始，直到 64。最后，每个实例的 style 属性都设置为"color: blue"。

controlPanel 是一个 HtmlPanelGrid 实例，它将在一个 HTML 表格中显示其所有的子控件；每个 HtmlOutputText 组件将在一个单独的表格单元中显示。图 1.8 显示了方法执行后 controlPanel 看起来的样子。

⑤ 和 addControls 方法类似，goodbye 方法也是一个事件监听器。然而，它和 JSF 的导航系统相关联，所以其职责是返回一个字符串，或者说是一个逻辑结果，这样导航系统可以使用这个结果来决定下一步将装入哪一个页面。这种类型的方法称为是***动作方法(action method)***。

goodbye 方法通过 hello.jsp 中的“Goodbye” HtmlCommandButton 按钮的 action 属性和该按钮相关联。所以当用户点击 Goodbye 按钮时， goodbye 访法将被执行。在这里，goodbye 并不执行什么工作来决定逻辑结果；它只是返回字符串"success"。这一结果在 Faces 配置文件中和一个特定的页面相关联，这我们下面会提及。

因为 goodbye 方法并不执行任何处理 (在实际应用中也可能如此)，我们可以可以在按钮的 action 属性中用硬编码“success”来达到同样的效果。这是因为导航系统可以使用 HtmlCommandButton 的 action 属性，也可以使用一个 action 访法的结果 (如果该属性引用了一个动作方法的话)。

1.5.4. 通过 faces-config.xml 进行配置

像大部分框架一样， Faces 也有一个配置文件；不管你信不信，它名为 faces-config.xml。(技术上说， JSF 支持多个配置文件，但是我们现在我们先保持简化) 这一配置文件允许你可以定义导航规则，初始化 JavaBeans，注册你自己的 JSF 组件和 validator，以及配置 JSF 应用的其他部分。我们这里的简单应用只需要配置 bean 初始化和导航；具体的配置文件为示于 listing 1.5。

清单 1.5 faces-config.xml : Hello World 应用的 Faces 配置

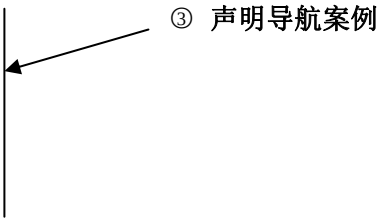
```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
"http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
<faces-config>
  <managed-bean>
    <description>The one and only HelloBean.</description>
    <managed-bean-name>helloBean</managed-bean-name>
    <managed-bean-class>org.jia.hello.HelloBean
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
```

① 包含所有配置元素的根元素

② 将HelloBean声明在session中

Sample

```
  <navigation-rule>
    <description>Navigation from the hello page.</description>
    <from-view-id>/hello.jsp</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/goodbye.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```



首先， JSF 配置文件是一个 XML 文档，根节点是`<faces-config>` (①)。

在这个文件中，你可以声明一个多者多个在你的应用中使用的 JavaBeans。你可以给它们每个都取一个名字(该名称可以通过 JSF EL 表达式引用)，一个说明，一个范围，甚至可以初始化它们的属性。在配置文件中声明的对象称为是***受管 Bean(managed bean)***。在代码清单中，我们声明了在整个 Hello, world! 应用中使用的 `helloBean` 对象 (②)。请注意对象的名称是“`helloBean`”，这和在两个 JSP 中的 JSF EL 表达始终使用的名称是一样的。其类是 `org.jia.hello.HelloBean`，这是我们在前一节所讲的 backing bean 的类名称。受管 bean 的名称和对象的类名称不一定要一样。

声明导航和声明受管 Bean 一样简单。每个 JSF 应用可以有一个或者多个导航规则。一个导航规则定义了从一个特定页面出发的可能路由路径。被一种路由都称为是一个***导航案例(navigation case)***。配置文件清单显示了 Hello, world!应用的 hello.jsp 页面的导航规则(③)。hello.jsp 有一个 Goodbye 按钮，该按钮要载入另一个页面，所以这里是一个单一导航案例：如果结果是"`success`"，则显示 goodbye.jsp。而该结果是由 `helloBean` 的 `goodbye` 方法返回的，该方法则在用户单击 Goodbye 按钮时被执行。

值得指出的是 JSF 配置的某些方面，特别是导航，可以使用工具来进行可视化处理。

现在，我们来看看我们的应用在 Web 应用的层面是如何进行配置的。

1.5.5. 配置 web.xml

所有的 J2EE web 应用都通过 web.xml 部署描述符来进行配置；Faces 应用也不例外。然而，JSF 应用要求你必须指定 FacesServlet，而这通常是 Face 应用的主 servlet。另外，请求必须被映射到这个 servlet。我们的 Hello, world! 应用的部署描述符示于 listing 1.6。你可以使用一些工具来为你产生 Faces 应用所需要的一些元素。

清单 1.6 Web.XML：Hello World!应用的部署描述符

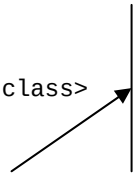
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3/
/EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Hello, World!</display-name>
  <description>Welcome to JavaServer Faces</description>
```

Sample

```
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>

</web-app>
```



标准 JSF 映射

就是这样—我们解剖完了 Hello world! 。

你可以看到 JSF 为你做了大部分的工作—校验，事件处理，导航， UI 组件管理等等。随着我们更加深入地探讨 JSF 的各个方面，你将获得对其所提供的各种服务的更深的理解，以便你能够集中于将它应用在你的应用中而避免一些令人抱怨的繁琐工作。

1.6. 小结

JavaServer Faces (JSF, 或者 “Faces”)是一个构建 Java Web 应用的 UI 框架；它基于 Java 社区流程 (JCP) 开发，并且将成为 J2EE 的一个组成部分。Faces 的一个主要目标是为 Java Web 应用开发领域带来 RAD 风格的应用开发，就像 Microsoft Visual Basic 和 Borland Delphi 一样。

JSF 提供一系列标准部件 (按钮、超链接、复选框等等)，一个创建定制组件的模型，以及在服务器端处理客户端产生事件的方式，并且有良好的工具支持。你可以同步 UI 组件和相关对象的值，而不必编辑太多令人生厌的代码。

所有的 JSF 应用都构建于 Servlet API 之上，使用 HTTP 进行通信，并且使用 JSP.之类的显示技术。虽然 JavaServer Faces 应用并不是一样需要 JSP。它也可以使用象 XML/XSLT，其它模版引擎，或者普通 Java 代码作为显示技术。然而，因为 Faces 实现提供了对 JSP 的集成基础，所以本书的绝大部分例子都使用 JSP。

Faces 的组件架构使用 JavaBeans 来处理属性，并提供基本的工具支持，以及事件模型和其他一些东西。JSF 被视为是一个 web 应用框架是因为它它执行了许多通用的开发任务，所以开发人员可以更加集中精力于更有趣的事情，比如业务逻辑之上。它的一个关键的特征是支持 Model 2 设计模式，这样可以执行表现和业务逻辑代码之间的分离。但是，Faces 主要还是集中于 UI 组件和事件。因此，它可以很好的和其它框架进行集成，比如 Struts，并且在高层次框架上和它们的功能一些重合之处。

Hello, world! 示例展示了 JavaServer Faces 应用的基本方面。它也展示了如何可以很容易地定义文本框、标签和按钮等 UI 组件。也展示了 Faces 如何自动的处理输入校验并且根据文本框的值自动更新 JavaBean 的属性。

在下一章中，我们将看到核心的 JSF 概念，并且研究框架如何隐藏 HTTP 请求和响应。