# <<NetBeans富客户端编程权威教 >

## 图书基本信息

书名：<<NetBeans富客户端编程权威教程>>

13位ISBN编号：9787302182030

10位ISBN编号：7302182035

出版时间：2008-9

出版时间：清华大学出版社

作者：（美）波德鲁，（美）图拉赫，（美）威尔兰格 著，叶亮 译

页数：445

译者：叶亮

## 前言

欢迎进入NetBeans 平台的富客户端开发世界。
虽然因特网的蓬勃发展使得人们更热衷于服务器端的开发，但对高质量桌面软件的需求却从未减少，反而在一定程度上有所增加。
其中部分原因包括： Web页面—— 通常作为服务器驱动的应用程序的界面，经常无法充分满足最终用户的需求。
并不是所有的应用都要求持续的网络连接，很多应用需要脱机运行。
本书将重点介绍如何使用NetBeans平台作为框架，开发“ 一次编写，处处运行 ”的富客户端应用程序。
NetBeans平台是NetBeans IDE(集成开发环境)的基础。
数以万计的开发者利用NetBeans IDE完成了不同规模和复杂度的应用程序。
毋庸置疑，NetBeans平台十分强大而健壮，完全可以作为大多数应用程序的基础，无论它们是商业的还是内部的解决方案。
除此之外，本书还会介绍如何编写应用于NetBeans IDE本身的插件模块。
富客户端应用程序 “ 富客户端应用程序 ”到底是指什么呢？
富客户端应用程序是指绝大多数功能(可以不是全部)运行在用户本地系统上的应用程序。
与之相反的是Web应用程序，它的所有功能完全依赖于运行在远程服务器上的代码，用户通常使用Web浏览器访问Web应用程序。
“ 富客户端 ”这个词可以被当作是“ 桌面应用程序 ”的一个时髦的绰号。
举例来说，NetBeans IDE本身就是一个典型的富客户端应用程序。
什么是NetBeans NetBeans是一个流行的、屡获殊荣的、用于Java开发的集成开发环境(IDE)。
它的核心是NetBeans平台—— 一个模块化的、可扩展的应用程序框架。
所以，NetBeans IDE本身是NetBeans 平台以及一系列模块的完美集合。
在早期发展阶段，为了使NetBeans IDE的开发更加灵活，它的体系结构被设计得高度模块化。
NetBean IDE模块化的体系结构带来了如下优势： 创建新特性十分简单。
用户可以轻松地增加或删除功能。
安装后，用户可以方便地更新某个特性，而不会影响其他功能。
NetBeans平台高度模块化的特点吸引了全世界的众多开发者。
数量众多的NetBeans平台应用应运而生。
NetBeans IDE就是其中最著名的一个。
NetBeans平台被用于各行各业的应用程序的开发，从语音处理到地质填图再到股票交易。
为什么使用NetBeans 基于NetBeans平台的应用程序有很多优势，其中首屈一指的当然是其真正的跨平台特性。
实际上，开发跨平台的富客户端应用程序有很多种方法。
例如，可以使用Swing组件自行编写桌面应用程序的所有模块。
但是，如果使用NetBeans平台的话，可以直接使用应用程序所需的众多强大的构造模块和底层构架，不用再像以前那样从头开始编码了。
显然，这将节省大量的时间。
开发者可以根据应用程序的逻辑编写新的Swing组件，并将其加入到NetBeans平台中，也可以在NetBeans平台中直接使用诸如JGraph、JFreeChart等第三方库。
总而言之，利用NetBeans平台可以很容易地开发出健壮且灵活的应用程序。
开发者只需要关注应用程序的业务逻辑。
就像JSF(JavaServer Faces)技术和Struts是为了Web开发而生，NetBeans平台是为了Swing开发而生。
以下是NetBeans平台最突出的几个优势： NetBeans是免费的，任何人都可以免费使用其代码，无论将它用于开发商业软件还是非商业软件都是如此。
NetBeans是一个成熟的、功能丰富的应用程序框架。

NetBeans平台由众多的组件构成，它们原本是为NetBeans IDE服务的——不计其数的软件工程师使用过NetBeans IDE。
但实际上，NetBeans平台可以用来开发任何类型的桌面应用程序，不仅仅是某种集成开发环境或类似的应用。
NetBeans平台为应用程序提供了高质量的基础。
NetBeans是一个真正的“一次编写，处处运行”的平台。
NetBeans基于Swing——一个纯Java的可视化工具集，它是所有桌面Java安装版的一部分。
NetBeans技术是标准化的、开源的。
这意味着，任何人都不用担心在NetBeans平台基础上进行开发会产生专利锁定的问题。
NetBeans IDE的插件拥有大量的潜在用户。
假如某人希望向公众展示一些新技术，他只需要开发一个相应的NetBeans IDE插件模块，将其发布在NetBeans 插件的门户网站上(http：//plugins.netbeans.org/PluginPortal)，立刻就可以轻松地吸引众多用户。
NetBeans拥有一个活跃的社区。
NetBeans平台的成长正是得益于这个强大社区中的众多开发者，他们热切地希望与大家共享他们的经验。
如果你正好有一个关于NetBeans的问题，那么请去社区中寻找答案。
因为很可能某人曾经遭遇过和你一样的问题，他可以帮助你。
NetBeans平台为开发者提供哪些资源　　NetBeans平台为开发者提供的资源包括：一系列丰富的基础功能，可用于创建新特性的扩展API，以及一组用来协助开发的强大工具。
开发基于NetBeans平台的应用程序时，开发者可以利用的资源包括：　　窗口系统(Window System)，它极大地简化了在一个frame中对众多组件的操作。
操作系统(actions system)，利用它可以简单地以声明的方式安装或卸载菜单项、工具栏、键盘快捷键等资源。
自动更新机制，为应用程序提供了一种动态更新的方式。
NetBeans IDE本身提供的所有特性都可以用来简化应用程序的开发。
例如，代码补全和高效的GUI编辑器(之前的代号叫做Matisse)——通过简单地拖拽和重新排列组件，能够可视化地创建用户界面。
除此之外，NetBeans IDE还提供了针对模块开发的特殊功能。
例如模块模板和方便的模块测试功能——随时可以把模块安装或者重新载入到当前运行的IDE中。
得益于NetBeans平台所鼓励使用的模块化编码技术，基于NetBeans平台的应用程序会更加健壮。
为什么阅读本书　　NetBean IDE越来越流行，同时，越来越多的人已认识到利用NetBeans平台可以很方便地创建任何类型的桌面应用程序。
显然，大家急切地需要一本指导书。
实际上，有很多NetBeans平台的信息都散布在不同地方，但没有一本完整的教材来演示如何使用整个NetBeans平台。
本书集作者多年积累的经验、最佳实践和实用信息于一身。
本书将帮助读者快速掌握模块的开发，并且指导读者学习绝大多数重要的API。
在这个过程中，读者也将学习到一些使NetBeans可靠而又灵活的开发实践。
如何使用本书　　本书由22个章节、2个用例及3个附录组成。
第1章，做一些准备工作，演示创建一个模块的基本过程。
第2章和第3章，讨论模块化开发的优势并提供NetBeans平台模块化构架的概况。
第4章和第5章，解释NetBeans模块协同工作的基本概念，展示NetBeans平台使得模块化应用高耦合的机制。
第6章，介绍文件系统(Filesystems)API，它是NetBeans平台处理用户数据和系统配置数据的基础构架。
第7章，巩固第6章的内容，并且展示如何基于NetBeans平台创建一个简单的导航(Navigator)组件。

第8章，解释并演示在NetBeans中创建一个成熟的多窗口应用程序所需的组件和特性。
第9章，介绍Node和Explorer API，利用它们可以以多种方式向用户展示数据结构。
第10章，重点介绍数据系统(Datasystems)API，利用它可以以多种方式、简单地在程序中操作特定类型的文件。
第11章，介绍NetBeans IDE的GUI构建器，演示如何利用它简化基于NetBeans平台应用程序的用户界面开发。
第12章，在第11章的基础上，展示如何为单个文件的内容提供多种表现形式。
第13章和第14章，展示为某种文件类型加入编辑特性的方法。
第15章，解释如何创建一个对象调色板，并通过一个实例演示如何实现从调色板中拖拽代码片段到文本编辑器中。
第16章和第17章，介绍如何开发更多的编辑特性。
第18章，演示如何为应用程序加入用户定制选项。
第19章，以Wicket为例，介绍如何为IDE加入对Web应用框架的支持。
第20章，介绍如何让NetBeans平台应用程序支持使用Web服务。
第21章，演示如何在应用程序中集成帮助文档。
第22章，展示如何将更新后的模块(和新模块)加入到用户的动态更新中心。
最后两章是NetBeans平台的用户案例。
第23章介绍的案例演示了如何在NetBeans IDE中集成现有开发工具。
第24章的案例展示了如何创建一个创建音频的编辑器。
附录中的信息有助于学习如何提高代码的健壮性、可读性、灵活性及性能。
本书并没有涉及到很多NetBeans IDE本身的一些特性，而是更关注实现这些特性的API。
如果读者对如何使用NetBeans IDE开发标准Swing、Web、企业、移动或者其他Java应用程序感兴趣，可以找到很多优秀的参考资源。
例如NetBeans IDE Field Guide一书(同样由Prentice Hall出版)，NetBeans IDE自带的文档，以及www.netbeans.org和http：//wiki.netbeans.org 网站上的资源。
没有一本参考书可以代替所有参考文档，本书也不例外。
读者可以在NetBeans平台的网站(http：//platform.netbeans.org) 上找到完整的NetBeans API参考文档。
或者直接通过NetBeans IDE的“ 更新中心 ”下载它。
开发者问题集(Developer FAQ，http：//wiki.netbeans.org/wiki/view/NetBeansDeveloperFAQ)也是一个很好的资源，读者可以在其中查到不断重复的问题的答案。
NetBeans邮件列表—— 特别是 dev@openide.netbeans.org邮件列表 —— 对于寻求罕见问题的答案特别有用。
如果读者正计划开发NetBeans模块，我们强烈建议注册参与这个邮件列表。
通过它，读者可以与整个NetBeans开发团队和社区交流。
当然，其中也包括本书的作者。
在开始开发NetBeans模块之前，最好检查一下更新中心(在“ 工具 ”菜单中)，确保拥有最新版本的NetBeans模块开发工具。
这些工具包含很多模板以及其他简化模块开发的特殊支持。
如何更新 访问本书的主页(www.netbeans.org/books/rcp.html)，下载更新和其他额外的内容。
本书的内容主要针对NetBeans平台的5.5版本，它是目前最新的正式发布版。
新版本会持续地跟进。
尽管本书的内容对NetBeans 6.0版本也有价值，同时也适用于未来的其他版本。
但建议读者最好访问“ 升级指南 ”(www.netbeans.org/download/dev/javadoc/apichanges.html)，以便能在学习5.5版本之后明白有哪些API改变了、增强了或是抛弃了。
本书会对6.0版本改变的地方加以特别注明。

## 内容概要

这是一本有关NetBeans富客户端应用程序开发的权威指南，内容涵盖了NetBeans 5.5和6.x版本，重点介绍了如何使用NetBeans平台作为框架，开发“一次编写，处处运行”的富客户端应用程序。旨在帮助读者掌握NetBeans模块的开发，精通NetBeans的主要API，以及学会一些构建可靠桌面软件的技术与技巧。

本书由三位顶级NetBeans专家联合编写，并由Sun中国的专家团队叶亮等人翻译和审校，是中国读者不可多得的NetBeans学习用书，也是Sun中国技术社区推荐的NetBeans技术用书。

## 作者简介

Tim Boudreau是NetBeansTM：The Definitive Guide（由O ’ Reilly出版）一书的合著者，开源NetBeans核心团队的成员，一直致力于NetBeans项目的开发。

# <<NetBeans富客户端编程权威教 >

## 书籍目录

## 章节摘录

24.6 将WAV编辑器转变为多视图编辑器 本小节将介绍如何将WavComponent改为多视图组件。
关于多视图的详细介绍，请参考第12章。
WavOpenSupport需要返回的是一个多视图组件，而不是现在比较精简的WavComponent。
protected CloneableTopComponent createCloneableTopComponent() { // Create an array of MV descriptors with only one view of the // data (the one weve already created - the waveform view) WavPanelMultiViewDescriptor main = new WavPanelMultiViewDescriptor(); MultiViewDescription[] descArry = { main }; // Initialize the view with data WavDataObject dobj = (WavDataObject)entry.getDataObject(); main.setWavDataObject(dobj); // Create the multiview CloneableTopComponent tc = MultiViewFactory. createCloneableMultiView(descArry, main, new CloseHandler()); tc.setDisplayName(dobj.getName()); return tc; } 这段代码引入了两个新类：CloseHandler 和WavPanelMultiViewDescriptor。
CloseHandler的名称表明了自己的作用。
它负责处理多视图组件的关闭。
下面的实现会询问用户文件是否应该在关闭前保存。
根据用户对此的选择，对每个元素调用ProceedAction或者DiscardAction。
在没有看到其他不同的实现之前，这个实现应该是默认设置： private static class CloseHandler implements CloseOperationHandler, Serializable { private static final long serialVersionUID = 1L; public boolean resolveCloseOperation( CloseOperationState[] elements) { NotifyDescriptor nd = new NotifyDescriptor.Confirmation( "Save before closing?"); DialogDisplayer.getDefault().notify(nd); if (nd.getValue().equals(NotifyDescriptor.YES_OPTION)) { for (CloseOperationState element : elements) { element.getProceedAction().actionPerformed( new ActionEvent(this, ActionEvent.ACTION_PERFORMED, "xxx")); } return true; } else if (nd.getValue().equals(NotifyDescriptor.NO_OPTION)) { for (CloseOperationState element : elements) { element.getDiscardAction().actionPerformed( new ActionEvent(this, ActionEvent.ACTION_PERFORMED, "xxx")); } return true; } else { // Cancel return false; } } } WavPanelMultiViewDescriptor除了作为MultiViewElement的工厂类之外，还为每个视图提供一些描述(名称、图标等)： public class WavPanelMultiViewDescriptor implements MultiViewDescription, Serializable { private static final long serialVersionUID = 1L; public static Image ICON = Utilities.loadImage("org/foo/wavutils/sampleGraph.gif"); private WavDataObject dobj; public int getPersistenceType() { return TopComponent.PERSISTENCE_ALWAYS; } public String getDisplayName() { return "Waveform"; } public Image getIcon() { return ICON; } public HelpCtx getHelpCtx() { return null; } public String preferredID() { return "wavEditor"; } public MultiViewElement createElement() { return new WavComponent(dobj); } public void setWavDataObject(WavDataObject wav) { dobj = wav; } private void writeObject(ObjectOutputStream out) throws IOException { out.defaultWriteObject(); } }
现在，把老的WavComponent转变为一个MultiViewElement。
元素本身再也不必是一个TopComponent的实例(不过，即使是的话也无妨)。
public class WavComponent implements MultiViewElement { private static final CloseOperationState CLOSE_OPERATION_STATE = createCloseOperationState(); private transient WavPanel wavPanel; public WavComponent(DataObject dobj) { super(); wavPanel = new WavPanel(dobj); } public Action[] getActions() { return new Action[0]; } public Lookup getLookup() { return wavPanel.getWavDataObject().getNodeDelegate(). getLookup(); } public UndoRedo getUndoRedo() { return new UndoRedo.Manager(); } public

JComponent getVisualRepresentation() { return wavPanel; } public JComponent getToolbarRepresentation() { // We dont need any widgets on the toolbar return new JPanel(); } public CloseOperationState canCloseElement() { if (wavPanel.getWavDataObject().isModified()) { return CLOSE_OPERATION_STATE; } else { return CloseOperationState.STATE_OK; } } public void setMultiViewCallback(MultiViewElementCallback multiViewElementCallback) { // Dont need this } // Semantics similar to the equivalent methods in TopComponent public void componentDeactivated() {} public void componentActivated() {} public void componentHidden() {} public void componentShowing() {} public void componentClosed() {} public void componentOpened() {} public Object writeReplace() { return null; } private static CloseOperationState createCloseOperationState() { return MultiViewFactory.createUnsafeCloseState( "xxx", new ProceedAction(), new DiscardAction()); } private static class ProceedAction extends NodeAction { protected void performAction(Node[] node) { try { if (node != null && node.length > 0) { SaveCookie sc = (SaveCookie)node[0].getCookie(SaveCookie.class); sc.save(); } } catch(IOException ex) { ErrorManager.getDefault().notify(ex); } } protected boolean enable(Node[] node) { return true; } public String getName() { return "Save"; } public HelpCtx getHelpCtx() { return null; } } private static class DiscardAction extends NodeAction { protected void performAction(Node[] node) { if (node != null && node.length > 0) { DataObject dobj = (DataObject)node[0].getCookie(DataObject.class); try { // Throw away whats in memory. // The DataObject will be recreated from disk. dobj.setValid(false); } catch (PropertyVetoException ex) { ErrorManager.getDefault().notify(ex); } } } protected boolean enable(Node[] node) { return true; } public String getName() { return "Discard"; } public HelpCtx getHelpCtx() { return null; } } }

此时，编辑器应该和前一小节中的完全一样，但它是在多视图窗口中，如图24-10所示。

图24-10 只有一个视图的“多视图”WAV编辑器

接下来定义一个扩展点，让其他模块可以在多视图窗口中插入新的视图。

## 24.7 创建插入额外视图的API

在NetBeans平台上创建API的第一步是新建一个独立的包(例如，org.foo.wavsupport.api))。

依赖于wavsupport的模块应该只能访问这个包中的类。

为了确保这一点，请在wavsupport的“项目属性”窗口中，把这个包指定为“公共包”。

如图24-11所示。

图24-11 一个公共的API包

这个API的目的是让其他的模块可以提供它们自己的MultiViewDescriptions。

这意味着至少能够从其他模块中收集MultiViewDescription的实例，然后把它们插入多视图窗口中。

但是，为了让这些其他模块获得足够的信息以创建有意义的界面，它们需要访问WavDataObject。

所以，在API包中创建一个子接口：

```
public interface WavViewDescriptor extends MultiViewDescription, Serializable { void setWavDataObject(DataObject dobj); }
```

请注意，setWavDataObject()方法接受一个通用的DataObject类型对象作为参数，而不是更加特殊的WavDataObject。

这是因为WavDataObject并不在API包中。

最好能够让客户模块需要的任何数据都能够在DataObject的cookie集找到。

在API包中创建一个新的子接口，命名为WavCookie，将WavDataObject中所有公共常量都移到这个接口中。

同时，再为所有希望WavDataObject对外公开的方法声明公共API方法：

```
public interface WavCookie extends Node.Cookie { public static final String PROP_WAVEFORM = "waveform"; AudioFormat getAudioFormat(); WrappedAudioInputStream getAudioInputStream(); void
```

setAudioInputStream( WrappedAudioInputStream is ); void addPropertyChangeListener(PropertyChangeListener l); void removePropertyChangeListener(PropertyChangeListener l); } 然后，让WavDataObject实现这个接口： public class WavDataObject extends MultiDataObject implements WavCookie { ... } 请注意，不需要像在执行打开和保存cookie操作时一样把WavDataObject显式式添加到它的cookie集中。
这是一个特例。
所有DataObject都会自动地加入到它们自己的cookie集中。
目前WavOpenSupport类仅仅实例化WavPanelMultiViewDescriptor，然后把它放在一个单元素的数组中传递给MultiViewFactory。
现在，将描述符放进一个未知大小的列表中，作为第一个元素，使用Lookup(有关Lookup的详细信息，请参阅第4章的4.3节)填充列表中的其他元素： WavViewDescriptor main = new WavPanelMultiViewDescriptor(); List all = new ArrayList(); all.add(main); Lookup.Template template = new Lookup.Template(WavViewDescriptor.class); Lookup.Result result = Lookup.getDefault().lookup(template); for (Object wvd : result.allInstances()) { all.add((WavViewDescriptor)wvd); } 然后把数据对象的引用传递给所有描述符： for (WavViewDescriptor wvd : all) { wvd.setWavDataObject(dobj); } 最后，把列表转换成数组，传递给MultiViewFactory： WavViewDescriptor[] allArray = new WavViewDescriptor[all.size()]; all.toArray(allArray); CloneableTopComponent tc = MultiViewFactory.createCloneableMultiView(allArray, main, new CloseHandler()); 至此，客户模块就可以实现公开的WavViewDescriptor接口，并使用WavCookie提供的信息在多视图窗口中提供一个新的标签页——这些都完全不需要编辑wavsupport模块的源代码。

24.8 实现API，提供新视图

现在是时候成为我们自己API的客户，创建一个新模块，为WAV文件提供一个不同的可视化视图。
为了方便读者，wavutils模块包含了组件FFTGraph，它基于一个在网上找到的公开域FFT库(感谢来自宾夕法尼亚大学的Tsan-Kuang Lee！
)，可以绘制频率域视图。

首先在模块套件中创建一个新模块，命名为fftview。
别忘记将wavsupport 和wavutils加入到它依赖的模块集中。

接着，创建API接口 WavViewDescriptor的实现： public class FFTViewDescriptor implements WavViewDescriptor { private static final long serialVersionUID = 1L; public static Image ICON = Utilities.loadImage("org/foo/wavutils/sampleGraph.gif"); private DataObject dobj; public int getPersistenceType() { return TopComponent.PERSISTENCE_ALWAYS; } public String getDisplayName() { return "Frequency Domain"; } public Image getIcon() { return ICON; } public HelpCtx getHelpCtx() { return null; } public String preferredID() { return "FFT"; } public MultiViewElement createElement() { return new FFTComponent(dobj); } public void setWavDataObject(DataObject wav) { dobj = wav; } private void writeObject(ObjectOutputStream out) throws IOException { out.defaultWriteObject(); } } 然后，还要再创建一个类，提供真正的组件： public class FFTComponent implements MultiViewElement { private DataObject dobj; private final FFTGraph graph = new FFTGraph(); public FFTComponent(DataObject dobj) { super(); this.dobj = dobj; final WavCookie c = (WavCookie)dobj.getCookie(WavCookie.class); assert(c != null); graph.createGraph(c.getAudioInputStream()); c.addPropertyChangeListener(new PropertyChangeListener() { public void propertyChange(PropertyChangeEvent evt) { if (evt.getPropertyName(). equals(WavCookie.PROP_WAVEFORM)) { WrappedAudioInputStream wais = c.getAudioInputStream(); if (wais == null) graph.clearGraph(); else graph.createGraph(wais);

} } }); } public JComponent getVisualRepresentation() { return graph; } public JComponent getToolbarRepresentation() { return new JPanel(); } public void setMultiViewCallback( MultiViewElementCallback multiViewElementCallback) { // Do nothing (we dont need the callback) } public CloseOperationState canCloseElement() { // The main wav component handles asking the user to save. // _This_ component is OK, whatever the outcome. // If the main component needed to provide any visual // feedback before saving/closing, this component could // have its own Proceed/Discard actions return CloseOperationState.STATE_OK; } public void componentDeactivated() {} public void componentActivated() {} public void componentHidden() {} public void componentShowing() {} public void componentClosed() {} public void componentOpened() {} public Object writeReplace() { return null; } public Action[] getActions() { return new Action[0]; } public Lookup getLookup() { return dobj.getNodeDelegate().getLookup(); } public UndoRedo getUndoRedo() { return new UndoRedo.Manager(); } } 剩下的最后一步是发布这个实现，让WavOpenSupport中的Lookup代码知道在哪里能找到它。

最简单的方法是在src/META-INF/services/org/foo/wavsupport/api/WavViewDescriptor中新建一个文件，其中包含下列内容： org.foo.fftview.FFTViewDescriptor 现在运行应用程序，应该能够看到两个视图，如图24-12所示。

图24-12 一个WAV文件的频率域视图 在全局Lookup中查询作为API的接口的实例是非常有用的技术。

特别是对于一些特殊情况就更加有帮助——例如，必须集成API经常改变的产品或库，或者集成相互竞争的厂家的多个产品。

开发者定义的接口会成为一个稳定的“桥梁”，每一个客户模块都是一个接口的实现，它们内部可以使用不同特定厂家的API。

例如，假设正在构建一个填写税务表格的软件，开发者可能拥有一个接口，用来向客户模块查询税率、法律和表格要求的格式。

然后为不同的权限提供不同的模块。

到第二年，只需要更新这些模块即可，而应用程序的核心照常工作，无需改变。

## 媒体关注与评论

我访问过一次中国，只去了北京。
此时，当我提笔写下这段文字时正是在2008年8月8日——奥林匹克运动会开幕的日了——这不禁让我想起那次印象深刻的北京之旅。
在北京，我们见到了众多的Java开发者，他们非常热情，积极地与我们聊天和讨论。
感谢叶亮让中国的开发者也能够看到这本书。
对于各位读者，我想说：“这本书不仅关乎一个技术主题，它还包含一种我们努力创造的激情。”您在工作中很可能也会有这样的激情。
如果这本书能够帮您点燃这种激情，那我们就会很满足。
因为这意味着我们的工作非常有价值。
——Tim Boudreau 在不久的将来，模块化应用程序会变得越来越重要。
在本书中，我们尽力介绍了模块化所需的基础知识，并且洋细描述了如何存NetBeans中实现它。
我衷心期望尊敬的读者们都能轻松地读完这本书，并且学到对今后的职业生涯都非常有用的知识。
——Jaroslav Tulach 随着应用程序的规模和复杂程度不断增加，您会慢慢发现NetBeans平台所提供的服务极其有用。
我希望这本书能够提供您所需的绝大多数知识，以及观察应用程序的新视角。
最重要的是，我希望本书给您带来的不仅是有用的知识，还有快乐。
——Geertjan Wielenga

## 编辑推荐

开源NetBeans平台是一个功能极其强大的架构，主要用于构建“只编写一次，就能到处运行”的富客户端应用程序。

对于Java开发人员和架构师来说，现有的基本Swing组件已不再满足他们的开发需求。为了解决这一问题，这本有关NetBeans平台上富客户端程序开发的权威指南便应运而生，旨在帮助读者掌握NetBeans模块的开发，精通NetBeans的主要API，以及学会一些构建可靠桌面软件的技术与技巧。

书中的每一章都给出了实际的例子，并按步骤详细说明了如何在NetBeans平台上创建功能完善的富客户端应用程序和NetBeans IDE的插件。

本书主要内容：·模块开发对于小、中和大型应用程序的重要意义 ·使用NetBeans加快开发速度和提高效率 ·利用NetBeans的各项生产率特性（从组件面板到代码完成） ·在自己开发的应用程序中利用NetBeans的模块体系结构 ·实现松散耦合的交互，以提高代码的可维护性和健壮性 ·管理用户配置和系统配置数据 ·使用牢固的线程模块构建可重载的组件 ·构造复杂的多窗口应用程序，并将富客户端数据结构显示给用户 ·添加用户可配置的选项 ·整合Web服务与NetBeans桌面应用程序 ·自动化模块更新和为用户提供帮助

## 版权说明