8 二次开发指南

8.1 二次开发准备

8.1.1 前期准备与 API 程序创建

2BizBox API 基于 Java 开发。所以您需要准备以下内容：

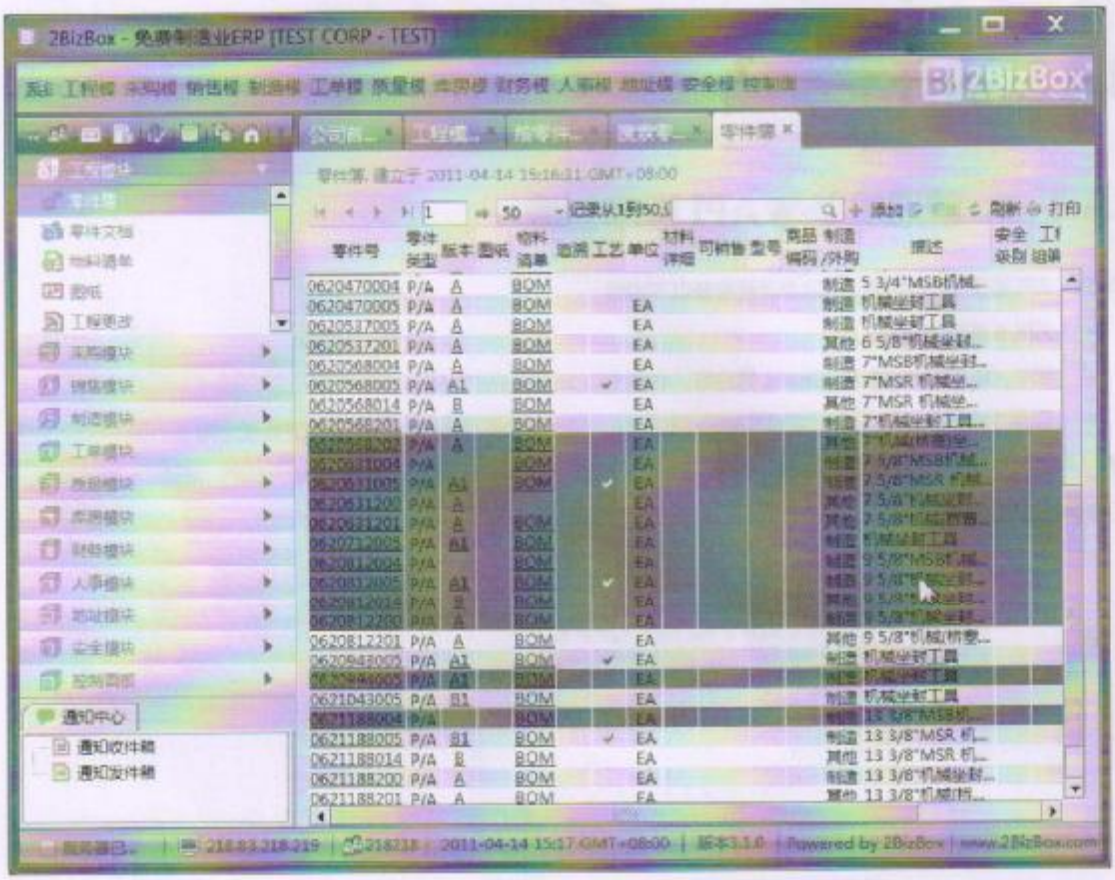
- JDK 6 或以上版本；
- IDE，如 Eclipse 或 NetBeans 等均可。本指南以 NetBeans 为例；
- 安装 2BizBox ERP，并保证其能够运行和登录。创建一些样例数据，以便于开发测试之用；

当然，您需要有 Java 编程的基本知识。例如，写一个 class，添加一个 main 函数，编译并运行，debug 调试等等。有了这些，就可以开始。

8.1.2 搭建 2BizBox 运行环境

首先从 2BizBox 官方网站或各大下载网站下载 2BizBox 免费 ERP 软件（注意要下载 All-in-One 版本，它包括了软件的客户端和服务器程序）。双击安装文件后，按照提示进行安装。安装结束后，可以自动运行 2BizBox ERP 服务器和客户端。注意服务器启动要大约半分钟到一分钟所有。结束后，可以点击客户端的“登录”按钮，登录到系统的“测试”公司中。

在系统中，可以首先创建几个零件。可以观看在线视频学习如何添加一个零件。添加的零件，可以用于下面 API 程序的测试。这个截图是作者所在系统的零件数据：



8.1.3 搭建开发环境

要使用 2BizBox API，需要以下 jar 包：

第三方包：

commons-lang-2.4.jar，Apache 的 Commons Lang 包，一个标准的 Java 增强工具包；

2BizBox 运行包：

common.jar，2BizBox 的数据结构定义包，包含 2BizBox 所有涉及到的业务数据的 POJO 定义；

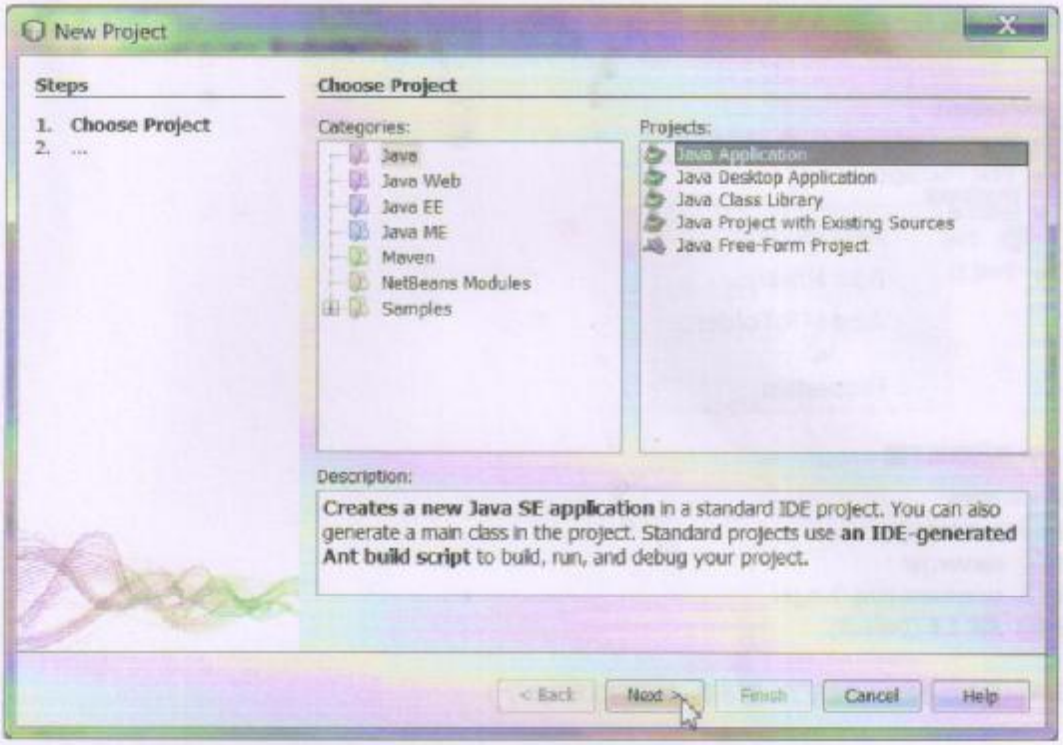
server.jar，2BizBox 后台业务模块 API 函数封装包，封装了全部 2BizBox 业务 API；

以上所有运行包，都可以在站点 http://www.2bizbox.cn/product/download 下载。

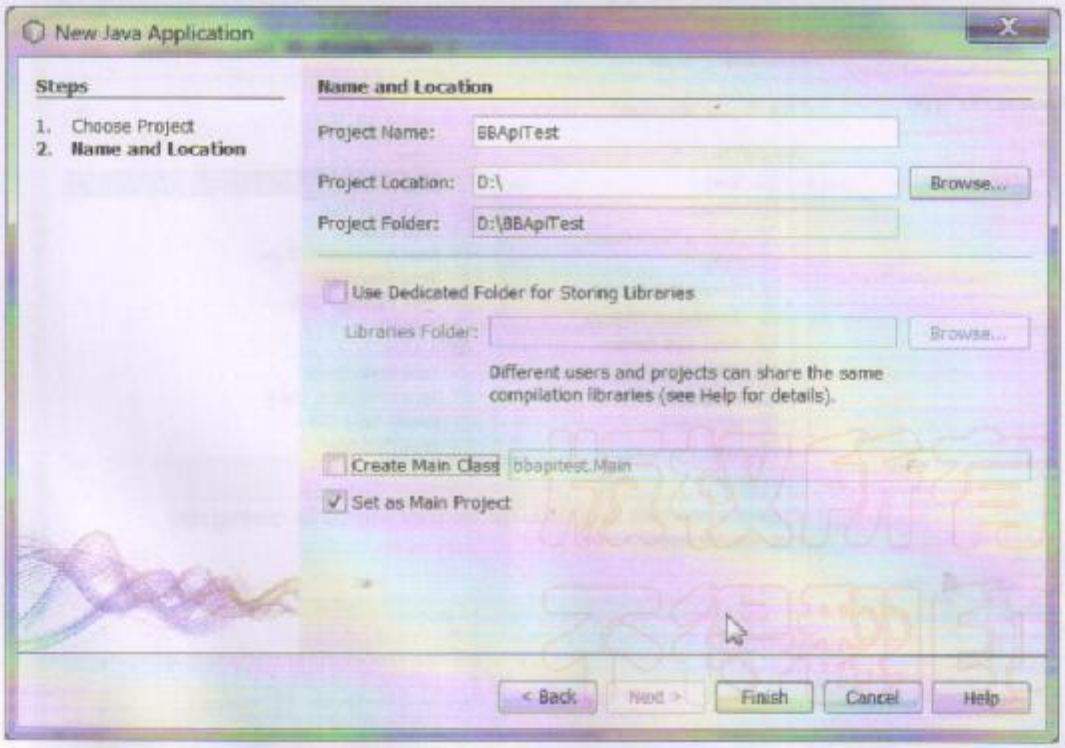
8.1.4 创建工程

本教程使用 NetBeans 作为 IDE 进行开发介绍。对于熟悉 Eclipse 工具的读者，操作方法类似，应当没有太大困难。

第一步：创建工程 在 NetBeans 中创建一个全新的 Java 工程，如下图：

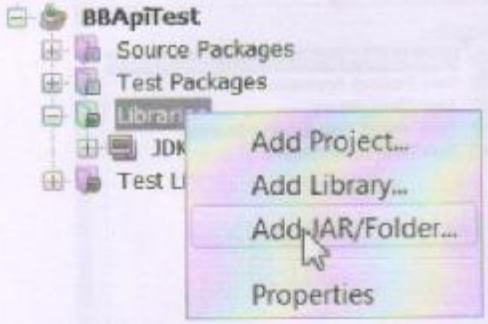


给新工程命名 BBApiTest，并设定工程目录。

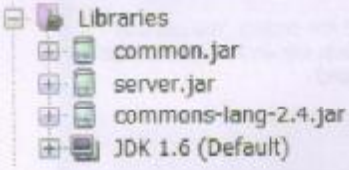


这样，我们就创建了一个全新的、空的 Java 工程。

第二步：添加 Jar 包 如上所述，工程需要用到 3 个 jar 包。右键点击工程的 Libraries 目录，添加准备好的 3 个 jar 文件包。

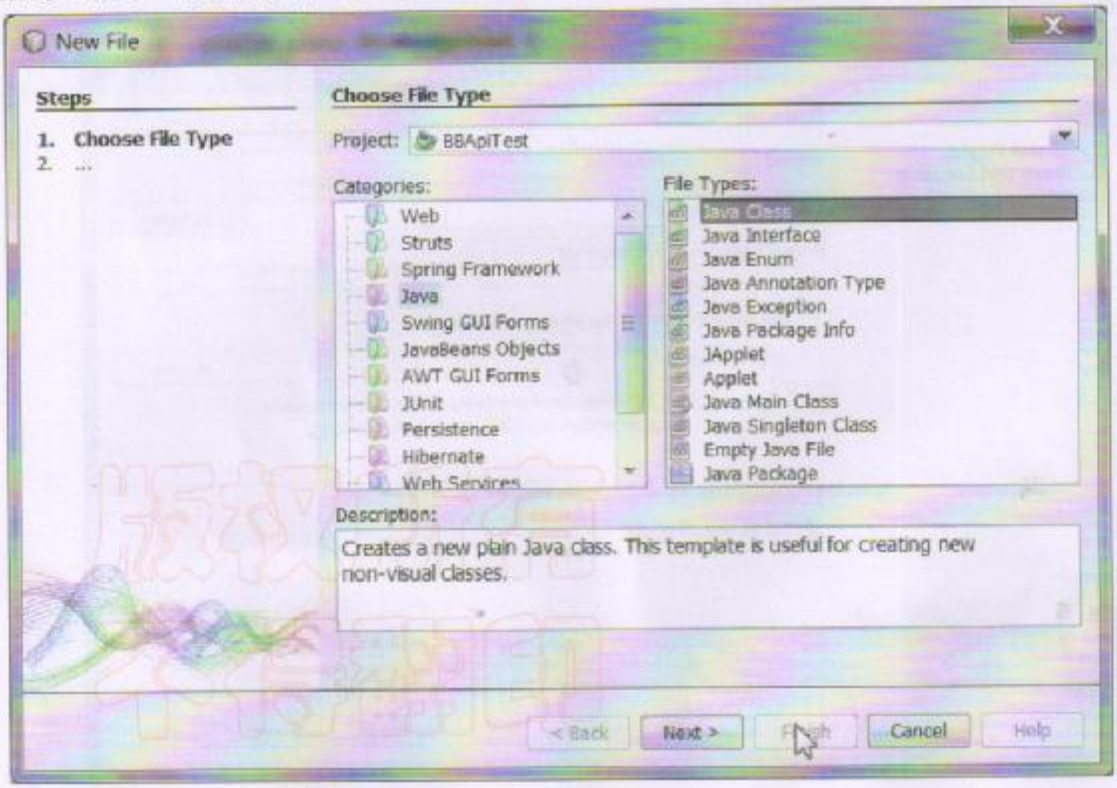


添加后，jar 包列表如下图：

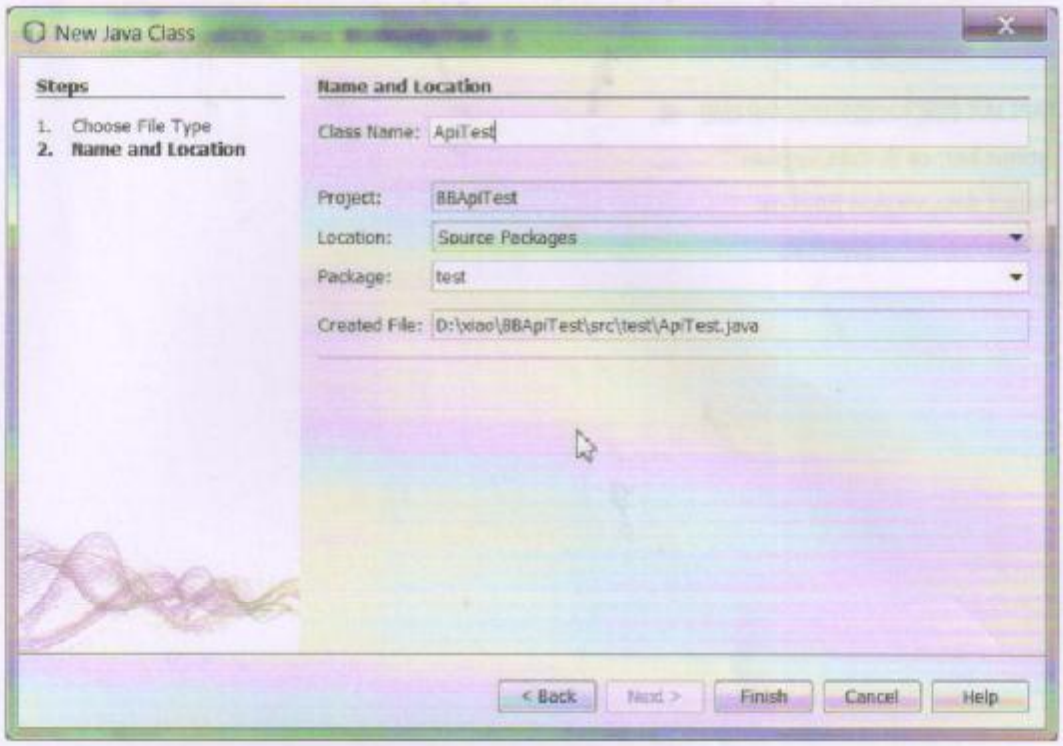


至此，工程以及工程环境就搭建好了。接下来，我们可以写代码了！

第三步：创建 Java 类 点击菜单，生成一个普通的 Java 类。



给这个 Java 类命名 ApiTest，并设置 package 为 test。



第四步：敲入代码

在新建的 Java 类中敲入如下代码：

```
package test;

import com.serva.bb2.common.BizBoxServer;
import com.serva.bb2.gui.server.EngServerActionManager;
import com.serva.bb2.gui.server.ServerActionUtil;

public class ApiTest {

    public static void main(String[] args) throws Exception {
        BizBoxServer server = new BizBoxServer("localhost", "80", "test", "username", "password", "zh");
        ServerActionUtil.setTargetServer(server);

        EngServerActionManager eng = EngServerActionManager.getInstance();
        int count = eng.getAllPartsCount();
        System.out.println("2BizBox ERP系统中物料总计: " + count + "个");
    }
}
```

这是一段极其简单的 Java 代码。一个普通的 Java 类，一个 main 函数。在函数中，创建了一个 2BizBox 服务器目标，并设置 ip 地址、端口、公司、用户名、密码、连接语言。随后，获得工程模块的 API 管理器，调用函数 getAllPartsCount 获得连接 2BizBox 系统中所有物料总数。

第五步：执行工程 按 F6 键，执行该文件。在作者所在的及其环境下，输出结果如下：

```
run:
2BizBox ERP系统中物料总计: 87042个
BUILD SUCCESSFUL (total time: 1 second)
```

至此，我们已经成功创建了第一个 2BizBox API 程序。

注意：

- API JAR 的版本和服务端版本必须要一致。
- status bar; co 表 data_version
- select data_version from co;
- 尽量在最新版本上面做开发。

8.2 API 基础

8.2.1 什么是 2BizBox API

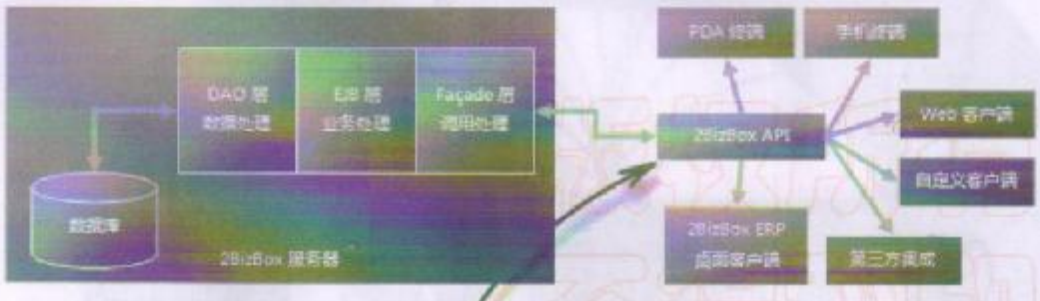
2BizBox 是免费的 ERP 软件，也是一个开放的 ERP 平台。2BizBox 面向开发者提供了完整的 API 二次开发接口，用于对 2BizBox 进 行集成和二次开发。2BizBox API 涵盖了 2BizBox 的全部功能。掌握了 2BizBox API，可以帮助开发者、系统集成商、实施服务商等深入了解 2BizBox 的功能和机制，并与其他软件系统进行无缝集成，满足企业客户的各种实际需求。例 如，和 OA、CRM、PDF、其他厂商 ERP 系统等的系统集成和数据交换，都是典型的应用场景。

举一个简单的例子：一家企业已经成功实施了**厂商的 CRM 系统和 2BizBox ERP 系统。一般来说，CRM 中并没有产品的实时成本信息和 BOM 信息。在报价时候，企业希望能够在 CRM 中直接从 2BizBox 中获取报价商品的当前库 存、成本、以及 BOM 信息，方便报价。此外，达成销售协议后，企业还希望通过鼠标点击，直接在 2BizBox 生成销售单。该如何进行呢？此时，如果您掌握 了 2BizBox API 的开发技巧，这一切都将变得易如反掌。

8.2.2 2BizBox API 的位置

2BizBox API 是 2BizBox 整个系统的后台功能定义和数据结构定义，用于各种外部程序的调用。这些外部程序包括 2BizBox 系统本身的 GUI 桌面终端、基于 Web 的各种模块终端、PDA 掌上终端、手机终端等等。同样，如果您掌握了 2BizBox API 的开发方法，也可以自己创建基于 2BizBox 的各种终端应用，或与 2BizBox 进行系统集成和数据联动。

下图展示了 2BizBox API 在整个 2BizBox 系统中的位置和作用。



2BizBox API 在系统中的位置

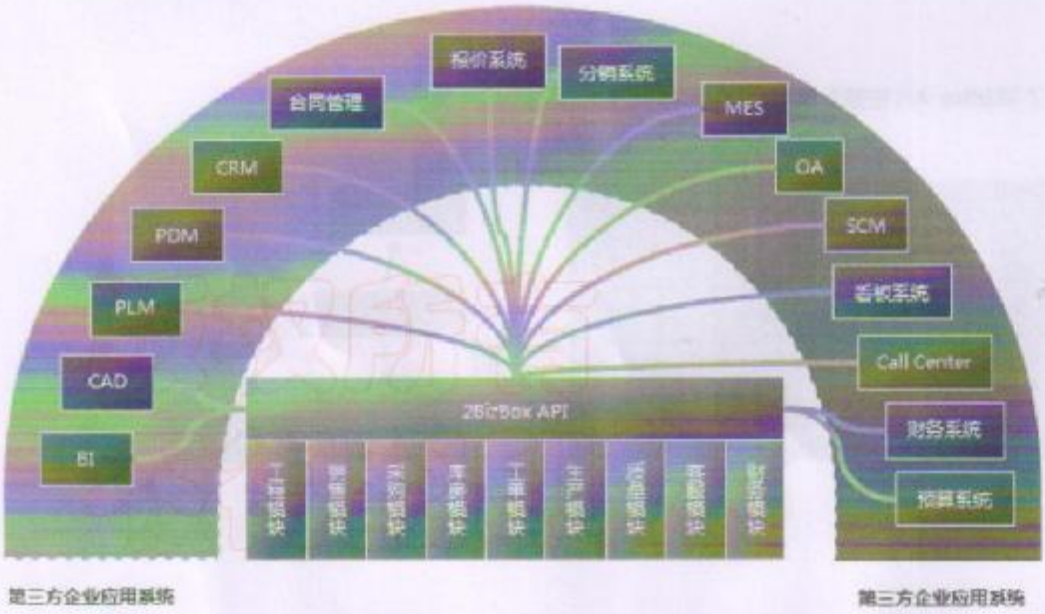
8.2.3 2BizBox API 可以做什么

2BizBox API 包含了 2BizBox 系统本身的全部功能和数据结构。所以，理论上来说，通过 2BizBox API，您可以操控系统，让它做任何它可以做的事。例如，创建单据、查询数据、修改订单等等。您甚至可以通过 2BizBox API 开发一个全新的客户端 GUI 程序，当然这一工作量将巨大无比。更多时候，我们是通过 2BizBox API 对一部分特定功能进行调用，以完成系统集成、数据查询、二次开发等工作。

2BizBox API 目前提供的是 Java 版本。也就是说，您可以通过 Java 语言来使用 2BizBox API，并集成在您的 Java 系统中。2BizBox 可以应用在桌面 Java 程序中（例如 Swing、SWT 等），也可以应用在基于 Web 的 Java 环境中（例如 Servlet、JSP 等）。

8.2.4 为什么是 API 而不是源代码

2BizBox 是免费软件，但并不是开源软件。2BizBox 软件的源代码并不对公众开放。所以，提供完整的 API 就是对 2BizBox 进行二次开发和系统集成的最佳方式。此外，相比源代码，API 有更多的优势。2BizBox 有超过 2 百万行源代码，任何个人甚至企业要想消化理解这些源代码，都并非易事，而对其进行修改和定制，就更加的困难。此外，直接修改源代码，也会带来各种各样的 Bug 和安全风险，导致整个 2BizBox 系统的不稳定。当更多的开发者对源代码进行各种各样的修改的时候，系统的集成和发展将变得更加困难，甚至会产生分裂的危险，对最终用户和 2BizBox 的长远发展都将是不利的。而通过统一、一致、简单、开放的 API 接口，开发者就不用担心 2BizBox 的内部运作机制，也不必去理解浩如烟海的源代码。只要 API 接口一致且保持兼容，2BizBox 就会携广大开发者不断地向前发展，齐心协力让 2BizBox 系统变得更加强大。



所以，我们认为 API 优于源代码。2BizBox 会不断丰富和开放高质量的 API，聚集广大开发者一起参与到 2BizBox 的发展中来。

本教程面向 2BizBox API 开发者、2BizBox 开发合作伙伴、2BizBox 系统集成商。如果您是软件厂商，如果您提供各种 BI、CAD、PLM、PDM、CRM、MES、OA、SCM、Call Center、财务系统、库房系统等等，都可以通过本文掌握 2BizBox API 的开发方法，和 2BizBox 进行集成，共同建立一个企业应用的生态圈。

对于 2BizBox 最终用户而言，无需了解 2BizBox API 和 2BizBox 二次开发相关知识。

此外，目前 2BizBox API 仅提供 Java 版本，所以您需要掌握基本的 Java 开发技术。

8.2.5 API 通用规则

查看某条数据： getXXX or findXXX
查找（搜索）多条数据：findXXX
获得符合条件的数据的数量：getXXXCount
更新数据：updateXXX
添加数据：addXXX
删除数据：deleteXXX（不是所有数据都有删除功能）
ByYYY YYY 只条件。

在开始开发之前，您需要熟悉以下基础知识：2BizBox 中每个模块的 API 函数都定义在相应的一个类里面统一进行管理。它们分别是：

- SalesServerActionManager（销售）
- AccountingServerActionManager（财务）
- AddressServerActionManager（地址）
- CompanySettingServerActionManager（公司设置）
- CSServerActionManager（客户服务）
- DocumentServerActionManager（文档中心）
- EngServerActionManager（工程）
- HRServerActionManager（人事）
- LogServerActionManager（日志）
- ManageServerActionManager（管理）
- ManServerActionManager（生产）
- PlanningServerActionManager（计划）
- PurchasingServerActionManager（采购）

- QualityServerActionManager（质量）
- SecurityServerActionManager（安全）
- WarehouseServerActionManager（库房）
- WorkOrderServerActionManager（工单）

更多信息可以参考 java doc 文档：

http://2bizbox.cn/download/api/javadoc-4.0/

8.2.6 创建数据库表与字段

在 CompanySettingServerActionManager 这个接口类下面有一些可以创建数据表和 sql 访问数据表的方法

- createAppTable
- createAppColumn
- dropAppTable
- dropAppColumn，
- findTableColumnsByName
- findTablesByName
- isAppTableExist
- isAppTableColumnExist
- updateAppData
- findAppDataBySql
- getAppDataCountBySql
- getAppProperty
- setAppProperty
- removeAppProperty

以上函数顾名思义，非常简单，就不做详细解释了。主要用来创建表、添加列、检查表或列是否存在、执行 SQL 语句进行查询或修改数据库结构、设置 App 自定义属性等。

此外，我们需要经常到数据库中查看数据。可以通过下面的方法登录数据库控制台：

```
mysql –uroot –ppassword –Pport bb2_test
```

将其中的 password 和 port 换成实际的密码和端口号即可。

8.2.7 ClassUtil 用法

获得某个包名下的所有类

```
Set<Class<?>> classes = ClassUtil.getClasses( "bb.common" , false); 得到所有 bb.common 下面的所有类。
Iterator<Class<?>> clt = classes.iterator();
while (clt.hasNext()) {
Class clazz = clt.next();
BeanInfo info = Introspector.getBeanInfo(clazz);
PropertyDescriptor[] propDesrs = info.getPropertyDescriptors();
for (int i = 0; i < propDesrs.length; i++) {
String field = propDesrs[i].getName();
if (field.equalsIgnoreCase("class")) {
continue;
}
System.out.println(clazz.getSimpleName() + "." + field);
}
}
```

1 如果 VO 下面有子的 VO 或者 PK，需要进一步分析。

2. clientProperties 不需要处理。

3. customProperties 对应 Custom Properties 功能。

8.3 客户端 App 开发基础

8.3.1 2BizBox 客户端构建介绍

8.3.1.1 客户端构建介绍

- MainUI（APPMainUI）JFrame
- MenuBar
- Main Tree
- ClientUI
- MainTabbedPane
- Homepage （ HomePageTabPane ， HomePageGroupPane ）
- Homepage Search Pane （ HomePageSingleRowSearchPane，HomePageSingleTextFieldSearchPane，HomePageDoubleRowSearchPane，HomePageMultiRowSearchPane，HomePageAddPane ）
- ListUI（AbstractListUI）

8.3.1.2 通用组件

- Table （StudentListUI）
- Tree （MyAppMainUI）

8.3.2 开发类 2BizBox 的独立客户端

- 构建一个客户端。
  构建客户端很简单，只需要写一个 Class (MyAppMainUI)继承 AppMainUI 这个 Class，然后用 bb.gui.Main.launchBizBox(MyAppMainUI.class.getName());这个方法启动客户端即可。
- 创建客户端菜单。在 AppMainUI 里面有一个获得菜单：FreeMenuBar menubar = getFreeMenuBar(); 然后即可通过编程在 menubar 上面添加菜单
- 创建客户端模块树。重写方法：isDisplayTree，返回 true ； 重写 getMainTreeXML，返回一个构建 tree 的 xml，或者通过 getMainTreePane 得到 TreePane，然后自己手动添加一个 tree，推荐第一种方法。

8.3.2.1 主菜单

- FreeMenuBar
- FreeRootMenu
- FreeMenu
- FreeMenuItem

8.3.2.2 模块树

- API（Tree 的配置。（一般用 TTree，不用 JTree））
- XML 配置文件，默认（gui.jar）下面/bb/gui/conf/MainTree.xml

8.3.3 开发 2BizBox 客户端的集成插件

8.3.3.1 三种集成方式，

集成的 jar 包，需要通过 app.xml 文件来配置。

1. 菜单集成。

```
<menuitem text-i18n-key="/app/test/i18n:AppPlugin.TestMenuText"
          tooltip-i18n-key="/app/test/i18n:AppPlugin.TestMenuTextTooltip"
          class-name= "app.test.TestPluginUI"      /> 类必须继承于 ClientUI
```

2.Homepage 集成。

```
<homepage node-name = "Physical_Inventory" class-name =
"bb.app.cyclecount.CycleCountHomePageAddCycleCountSheetPane"/>
```

类必须继承于 HomePageBaseSearchPane 及其子类。

添加，查看，搜索 的功能。

3. 快捷方式集成。

```
<shortcut value = "Ctrl+Shift+L" class-name = "bb.app.screenlock.ScreenLockPane" /> 类在构造函数
```

中自己控制需要打开的界面。

8.3.3.2 配置文件示例

```
<?xml version="1.0" encoding="UTF-8"?>
<apps>
        <menuitem text-i18n-key="/app/test/i18n:AppPlugin.TestMenuText"
                  tooltip-i18n-key="/app/test/i18n:AppPlugin.TestMenuTextTooltip"
                  class-name="app.test.TestPluginUI" />
      <homepage node-name = "Physical_Inventory" class-name =
"bb.app.cyclecount.CycleCountHomePageUpdatePartCycleCountPane"/>
```

```
    <homepage node-name = "Physical_Inventory" class-name =
"bb.app.cyclecount.CycleCountHomePageAddCycleCountSheetPane"/>
        <shortcut value = "Ctrl+Shift+L" class-name = "bb.app.screenlock.ScreenLockPane"/>
</apps>
```

8.3.4 创建一个简单的 App

如果通过 webstart 启动客户端，把配置文件 app.xml 打包到插件的 jar 包的更目录下，然后把 jar 包签名，放到服务器的<2BizBox 安装目录>/server/jboss/server/default/deploy/webstart.war 下面。

如果直接启动客户端，需要把 jar 和 app.xml 文件压缩到一个 zip 文件下,结构如下：

-app.xml
-libs（文件夹）
 --jar 包

然后把 zip 文件放到<2BizBox 安装目录>/client/apps 下面，如果没 apps 这个目录，手动创建即可。

参考示例：

http://www.2bizbox.cn/app/lock-screen

8.3.5 App 与主系统的集成

8.3.5.1 基于 Swing 程序的 App 集成

上面的例子简单的演示了如何通过 2BizBox API 来连接 2BizBox 后台服务器并获取数据。本章节继续用复杂一点的例子来演示如何在一个桌面程序或 Web 程序中通过 2BizBox API 与已有 2BizBox 系统进行集成。

Swing 程序集成示例

我们假设您已经有了一个基于 Java Swing 的复杂的桌面应用系统，需要和 2BizBox 进行数据交换。或者，您已经有了一套基于桌面的 OA 系统，可以处理各种单据的审批，并在审批结束后，自动到 2BizBox ERP 中进行单据创建（例如采购单）。此时，2BizBox API 就可以大有所为。

我们这里创建并模拟了一个非常简单的基于 Java Swing 的桌面应用程序，并通过 2BizBox API 与 2BizBox 服务器获取数据。这里我们以“模糊查询零件名字并列出”为例子，介绍如何在一个 Java Swing 程序中对后台的 2BizBox 服务器进行操作和通讯。

本例子假设您对 Java 和 Swing 技术熟悉。

重新创建一个新的 Java 类，命名 ApiTestSearchPartUI，并从 JFrame 继承。在这个窗口中，上部放置几个输入框，分别用于输入要连接的服务器的 IP 地址、公司、用户名、密码，以及要查询零件的关键字。窗口中间，放一个大的文本框，用于显示返回结果。

下面几行代码用于对窗口初始化。包括大小、关闭动作、标题等：

```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setTitle("2BizBox API测试");
this.setSize(750, 500);
this.setLocation(300, 300);
```

接下来，把创建好的 Swing 组件放在一个 Panel 中，并放置在窗口顶端：

```
JPanel settingPane = new JPanel(new FlowLayout(FlowLayout.LEADING, 5, 0));
settingPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
settingPane.add(new JLabel("Server:"));
settingPane.add(this.txtServer);
settingPane.add(new JLabel("Company:"));
settingPane.add(this.txtCompany);
settingPane.add(new JLabel("User:"));
settingPane.add(this.txtUser);
settingPane.add(new JLabel("Password:"));
settingPane.add(this.txtPassword);
settingPane.add(new JLabel("Search:"));
settingPane.add(this.txtSearch);
settingPane.add(btnConnect);
```

在后面，放置一个按钮。当点击按钮时，我们根据用户输入的信息，对服务器进行连接，并根据搜索条件，从后台搜索符合条件的零件：

```
private void initServer() {
    String server = this.txtServer.getText().trim();
    String port = "80";
    String company = this.txtCompany.getText().trim();
    String user = this.txtUser.getText().trim();
    String password = new String(this.txtPassword.getPassword());
    String language = "en";
    BizBoxServer targetServer = new BizBoxServer(server, port, company, user, password, language);
    ServerActionUtil.setTargetServer(targetServer);
}
```

代码中，后台调用可能发生异常。所以，在 catch 异常后，我们要进行处理。这里，使用 2BizBox API 中提供的 ExceptionWorker 对异常进行分析和处理，并将结果翻译、解析，通过 getMessage()函数返回。我们直接把异常的详细描述通过文本框显示出来即可。

在调用后台时，先通过如下代码初始化服务器。其中，端口默认是 80，登录语言默认是英文。

```
private void initServer() {
    String server = this.txtServer.getText().trim();
    String port = "80";
    String company = this.txtCompany.getText().trim();
    String user = this.txtUser.getText().trim();
    String password = new String(this.txtPassword.getPassword());
    String language = "en";
    BizBoxServer targetServer = new BizBoxServer(server, port, company, user, password, language);
    ServerActionUtil.setTargetServer(targetServer);
}
```

接下来是真正的调用代码。首先获得工程模块的接口 EngineeringBox，然后调用其中的 findPartsByDescriptionInside 函数，来模糊搜索符合条件的零件。其中几个参数解释如下：

- 第一个参数：零件号。如果不限定零件号，直接输入空字符串""即可；
- 第二个参数：零件类型。如果不限定零件类型，直接输入空字符串""即可；
- 第三个参数：零件描述。这是我们要使用的零件描述过滤条件，传入用户在界面上的录入；
- 第四个参数：返回集合的开始游标。也就是从符合条件的结果集的第几个记录开始；
- 第五个参数：返回集合的长度。也就是最多返回多少个符合条件的记录，注意不要太大，导致服务器附在过重。这里设置 100 条；

```
private void findPartsByDescriptionInside() throws Exception {
    EngineeringBox eng = EngServerActionManager.getInstance();
    String search = this.txtSearch.getText().trim();
    Collection parts = eng.findPartsByDescriptionInside("", "", search, 0, 100);
    String result = "";
    if (parts != null) {
        Iterator it = parts.iterator();
        while (it.hasNext()) {
            PartBasicInfo part = it.next();
            result += part.getPartPk().getPartNumber();
            result += "\t";
            result += part.getDescription();
            result += "\n";
        }
    }
    this.txtResult.setText(result);
}
```

返回结果集是 PartBasicInfo 对象的集合。PartBasicInfo 对象包含了零件对象的简要信息。对返回结果集进行遍历，将零件号和零件描述输出到文本框中进行显示。

完整程序如下：

```
package test;

import com.serva.bb2.common.BizBoxServer;
import com.serva.bb2.common.PartBasicInfo;
import com.serva.bb2.common.box.EngineeringBox;
import com.serva.bb2.gui.server.EngServerActionManager;
import com.serva.bb2.gui.server.ServerActionUtil;
import com.serva.bb2.gui.swing.ExceptionWorker;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Collection;
import java.util.Iterator;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class ApiTestSearchPartUI extends JFrame {

    private JTextField txtServer = new JTextField("192.168.1.121", 8);
    private JTextField txtCompany = new JTextField("test", 4);
    private JTextField txtUser = new JTextField("admin", 4);
    private JPasswordField txtPassword = new JPasswordField("password", 4);
    private JTextField txtSearch = new JTextField(8);
    private JTextArea txtResult = new JTextArea();
    private JButton btnConnect = new JButton("Connect");

    public ApiTestSearchPartUI() {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("2BizBox API测试");
        this.setSize(750, 500);
        this.setLocation(300, 300);

        JPanel settingPane = new JPanel(new FlowLayout(FlowLayout.LEADING, 5, 0));
        settingPane.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        settingPane.add(new JLabel("Server:"));
        settingPane.add(this.txtServer);
        settingPane.add(new JLabel("Company:"));
        settingPane.add(this.txtCompany);
        settingPane.add(new JLabel("User:"));
        settingPane.add(this.txtUser);
        settingPane.add(new JLabel("Password:"));
        settingPane.add(this.txtPassword);
        settingPane.add(new JLabel("Search:"));
        settingPane.add(this.txtSearch);
        settingPane.add(btnConnect);
```

```
        this.add(settingPane, BorderLayout.NORTH);
        JPanel resultPane = new JPanel(new BorderLayout());
        resultPane.setBorder(BorderFactory.createEmptyBorder(0, 10, 10, 10));
        resultPane.add(new JScrollPane(txtResult), BorderLayout.CENTER);
        this.add(resultPane, BorderLayout.CENTER);

        this.btnConnect.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                try {
                    initServer();
                    findPartsByDescriptionInside();
                } catch (Exception ex) {
                    ExceptionWorker worker = new ExceptionWorker(ex);
                    String message = worker.getMessage();
                    JOptionPane.showMessageDialog(ApiTestSearchPartUI.this, message);
                }
            }
        });
    }

    private void initServer() {
        String server = this.txtServer.getText().trim();
        String port = "80";
        String company = this.txtCompany.getText().trim();
        String user = this.txtUser.getText().trim();
        String password = new String(this.txtPassword.getPassword());
        String language = "en";
        BizBoxServer targetServer = new BizBoxServer(server, port, company, user, password, language);
        ServerActionUtil.setTargetServer(targetServer);
    }

    private void findPartsByDescriptionInside() throws Exception {
        EngineeringBox eng = EngServerActionManager.getInstance();
        String search = this.txtSearch.getText().trim();
        Collection parts = eng.findPartsByDescriptionInside("", "", search, 0, 100);
        String result = "";
        if (parts != null) {
            Iterator it = parts.iterator();
            while (it.hasNext()) {
                PartBasicInfo part = it.next();
                result += part.getPartPk().getPartNumber();
                result += "\t";
                result += part.getDescription();
                result += "\n";
            }
        }
        this.txtResult.setText(result);
    }

    public static void main(String[] args) throws Exception {
        ApiTestSearchPartUI ui = new ApiTestSearchPartUI();
        ui.setVisible(true);
    }
}
```

运行效果如下图：

在这个例子中，我们输入了正确的登录信息，并收入汉字“机”作为零件描述的关键字，在要连接的 2BizBox 服务器上进行数据查询，搜索出了大量符合条件的记录，并显示在了下面的文本框中。读者可以自行修改查询条件，在您的 2BizBox 系统中自由查询，体现“自由操控 2BizBox”的乐趣！

以上例子是一个非常简单、典型的 Swing 程序，但它已经可以成功的连接任意 2BizBox 服务器，只要您有正确的用户登录信息即可，并且可以查询和搜索零件信息。只要稍加改造，就可以把上面的代码嵌入在您自己的 Swing 程序中，和 2BizBox ERP 进行联动或数据集成了。

8.3.5.2 Web 程序集成示例

在 Web 开发环境中集成 2BizBox API 也是非常简单的事情。本例子用一个非常简单的 JSP 来说明如何在 JSP 中通过 2BizBox API 来调用远程 2BizBox 服务器并获得数据、在浏览器中输出。在实际 Web 开发中，您可以通过这种方式来和 2BizBox 服务器进行通讯和数据交换，并与您现有的 Web 系统进行集成和整合。

首先准备 tomcat。可以到 tomcat 官方网站下载免安装版本的 zip 格式文件。下载后，解压 zip 文件。进入其中的 bin 目录，双击 startup.bat 文件，确保能够正常启动 tomcat。如果不能启动，最可能的问题是没有安装 Java 环境，或没有正确设置 JAVA_HOME 环境变量。

接下来，我们手工创建一个非常简单的 JSP。在 webapps 目录中，我们创建一个 2bizbox 目录，作为测试工程。在其中，创建 WEB-INF 子目录，并在 WEB-INF 中创建一个 web.xml 文件，用来定义这个 Web 工程的信息。我

们在其中放入最简单的一个配置文件即可，内容如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

</web-app>
```

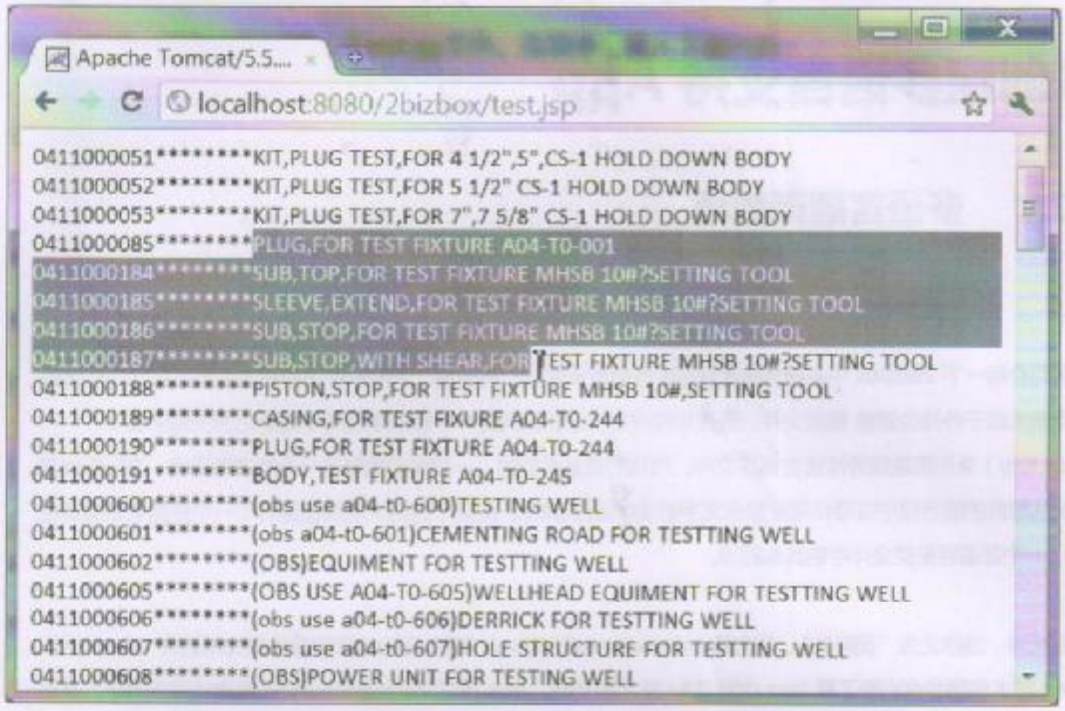
然后，在 WEB-INF 中创建 lib 文件夹，将 2BizBox API 需要的几个运行 jar 包放进去，分别是 common.jar、commons-lang-2.4.jar、server.jar 几个包。好了，这样，一个空的 Web 环境就搭建成功了。

下面，我们创建 JSP 测试文件。在 2bizbox 目录中，用写字板创建一个 test.jsp 文件。在其中，输入下面代码：

```
<%@ page import="java.util.*, com.serva.bb2.common.*, com.serva.bb2.common.box.*, com.serva.bb2.gui.ser

<%
BizBoxServer server = new BizBoxServer("localhost", "80", "test", "admin", "password", "en");
ServerActionUtil.setTargetServer(server);

EngineeringBox eng = EngServerActionManager.getInstance();
String search = "TEST";
Collection<PartBasicInfo> parts = eng.findPartsByDescriptionInside("", "", search, 0, 100);
String result = "";
if (parts != null) {
    Iterator<PartBasicInfo> it = parts.iterator();
    while (it.hasNext()) {
        PartBasicInfo part = it.next();
        result += part.getPartPk().getPartNumber();
        result += "********";
        result += part.getDescription();
        result += "<br>";
    }
}
out.write(result);

%>
```

这段代码很简单，首先创建 2BizBox 目标服务器的地址和登录信息，并进行连接。然后，从其中的工程模块调用 findPartsByDescriptionInside 方法查询最多 100 条描述中包含" TEST" 字符串的零件。最后，把返回的结果集通过字符串把零件号和零件描述输出在网页中。

在作者的环境下，输出结果如下图：



如果进一步简单改造，可以在这个页面上增加表单，自由输入服务器地址、用户名、密码、查询条件等，并提交本页面进行动态查询，就可以实现真正的零件查询了。

8.4 创建多语言支持 App

8.4.1 多语言翻译教程

8.4.1.1 基础知识

首先，我们介绍一下 2BizBox 中多语言技术架构的大概实现方法。简单说，2BizBox 中所有文字都不是写死在代码中的，而是来自于外挂的语言资源文件。而应用程序中，仅仅是通过一个标识字符串（也称之为资源键值，Resource Key）来引用这些外挂语言资源文件。当我们登录系统时，一旦我们选择了一个特定的语言，2BizBox 就会动态的从相应的语言包中去寻找需要显示文字的具体语言翻译。所以，有两个东西很重要：一个是语言资源键值（Key），一个是语言资源文件中的具体翻译。

多语言的支持，也称之为“国际化”，英文是 Internationalization，简称 i18n（原因是该英文单词有 18 个字母）。熟悉 Java 技术的读者会比较了解 Java 中对 i18n 技术的支持。Java 通过 ResourceBundle 来定义语言资源。这些资源可以来自 Java 类（ListResourceBundle），也可以来自资源文件（PropertyResourceBundle）。而 2BizBox 就是使用了 Java 中的 PropertyResourceBundle 技术，并做了进一步的改进和封装。
默认情况下，2BizBox 的语言资源文件都以标准的 Java 支持的 PropertyResourceBundle 文件存在。例如，财务模块的所有美国英文文字翻译，都存储在文件 accounting_en_US.properties 中，而所有工程模块的大陆简体中文文字翻译，都存储在 engineering_zh_CN.properties 中，等等。其中，文件名以模块名+语言名+后缀进行定义。

每一个具体的资源文件，都是一个简单的文本文件。每一行定义了一个资源字符串的翻译，前半部分是 Key 键值，后半部分是具体翻译，中间用等号“=”隔开。下面是一段典型的资源文件片段：

```
SalesOrderGroupByPartListUI.Detail                    =Detail
SalesOrderGroupByPartListUI.SOShippedQty              =SO Shipped Qty
SalesOrderGroupByPartListUI.SONotShippedQty           =SO Not Shipped Qty
SalesOrderGroupByPartListUI.InSIPInSO                 =InSIP in SO
SalesOrderGroupByPartListUI.OnDemand                  =On Demand
SalesOrderGroupByPartListUI.OnOrder                   =On Order
SalesOrderGroupByPartListUI.InSIP                     =InSIP
```

理论上来说，我们可以直接打开 2BizBox 的这些资源文件，对其进行修改和编辑，就可以改变默认的文字翻译。当然，这样做太繁琐，所以我们才会介绍下面的软件工具，让这一过程变得简单。

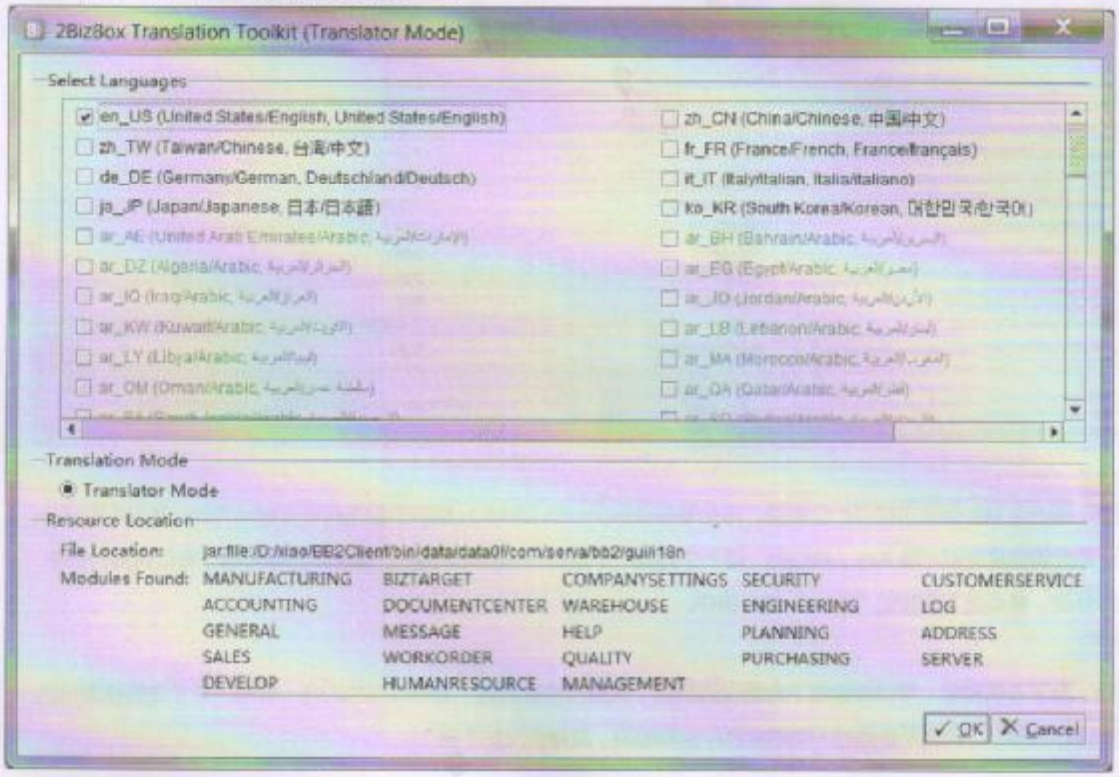
此外，我们还需要在语言定义的基础知识方面有所了解。2BizBox 完全遵守 Java 定义的语言机制。在 Java 中，一个我们通常所说的“语言”，定义为一个 Locale（地区），实际上它有两部分组成：国家和语言。也就是说，国家+语言=Locale，也就是我们通常所说的一个语种。举例来说：en_US 代表美国英语，zh_CN 代表了中国大陆简体中文，等等。对于语言代码，是通过两个小写的英文字母组成，由 ISO-639 进行定义，具体链接如下：[1] 对于

国家代码，是通过两个大写的英文字母组成，由 ISO-3166 进行定义，具体链接如下：[2]。。有了这些知识，再看到文件 purchasing_en_US.properties 就应当大概知道是什么内容了。

在目前 2BizBox 软件中，已经提供了英文（en_US）、简体中文（zh_CN）两个内置语言。如果要创建其他语言，可以通过本文介绍的软件工具来完成。

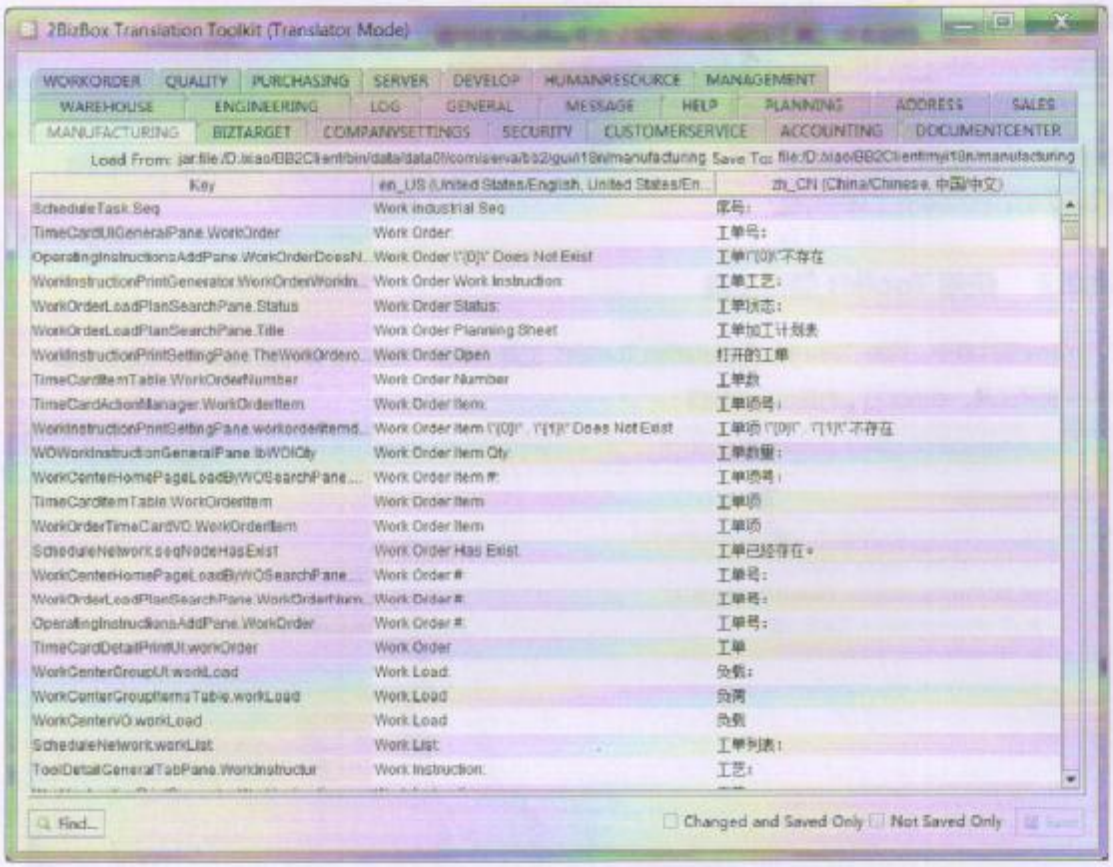
8.4.1.2 使用 Toolkit 翻译工具

在 2BizBox 插件库中，找到“2BizBox Translation Toolkit”工具，点击“安装”，即可在 2BizBox 平台上安装 Toolkit 翻译工具。点击运行，将显示以下窗口：



在该界面上，我们首先需要选择要翻译的语言。语言列表中，顶部黑色的语言是 2BizBox 已经内置的语言，下面灰色的语言是 2BizBox 尚不存在 的语言。选择黑色语言，可以对现有文字翻译进行校正、修改；选择灰色语言，可以用来创建一个全新的语言翻译。另外，第一个是英文，并且已经默认勾选。请确 保至少选择了一种语言，才可以继续。也可以同时勾选多种语言，用于多语种的对比、参考等。

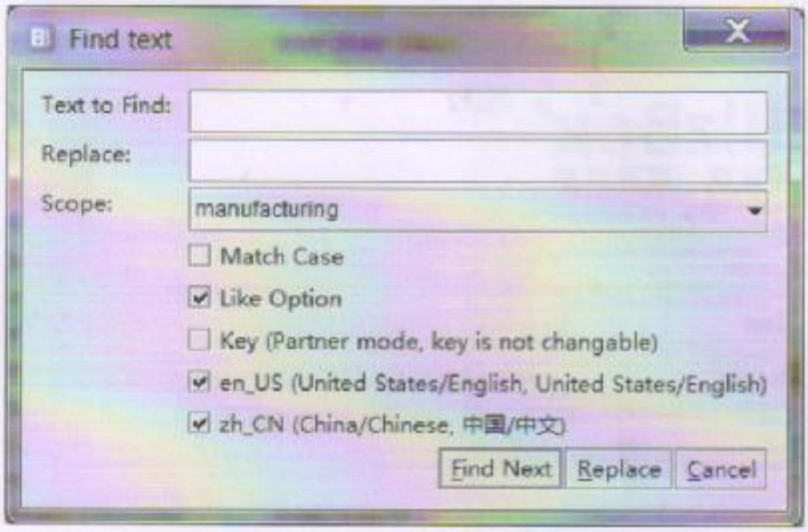
已选择“英文”和“中文”为例。勾选后，点击 OK 按钮，进入工具，显示下图。



图中，每一个 tab 标签页都有一个表格。每个表格显示了一个 2BizBox 模块中的所有语言翻译内容。表格中有多个列，最左边的是资源主键 Key，其他列，每一个列是一个语言对应的具体翻译。鼠标点击单元格，可以直接对翻译进行修改，并点击下方按钮“Save”进行保存。

此外，为了方便查看，可以勾选右下角的过滤选项，只显示修改过的，或未保存的内容，等等。对于曾经修改过的、尚未保存的翻译，界面都会通过不同的颜色来高亮标识，具体含义如下图：

点击左下方 Find 按钮，可以对界面进行搜索、批量替换等工作。



当对任何内容进行修改，标签页文字会发生变化，提示有内容被修改且尚未存盘。同时，Save 按钮也会生效，可以点击进行保存。此外，如果点击右上角关闭按钮来关闭程序，会提示用户是否要对修改内容进行保存，放置误操作将工作丢失。

点击 Save 按钮保存后，所有修改将被保存到磁盘中，并会在 2BizBox 中立刻生效。这可以通过立刻登录 2BizBox 软件来观察其变化。所有您修改过的内容，都会被单独保存在 2BizBox 安装目录 client 子目录中的 myi18n 文件夹中。为了保存翻译成果，可以定期将该目录进行备份。尤其要注意，重新安装 2BizBox 将会导致该文件夹丢失。如果有翻译成果，请确保重装 2BizBox 之前对该目录进行完整备份。

8.4.1.3 启用 2BizBox 翻译调试模式

通过上面介绍的翻译 Toolkit 工具，可以对整个 2BizBox 进行全面、系统的新语言创建工作。但是，该工具也有一个很大的缺点：缺乏上下文，很难搞清楚翻译的具体应用场景，从而很难精确的确定翻译的准确性。一些翻译，势必需要更好的方法进行更精细化的调整。

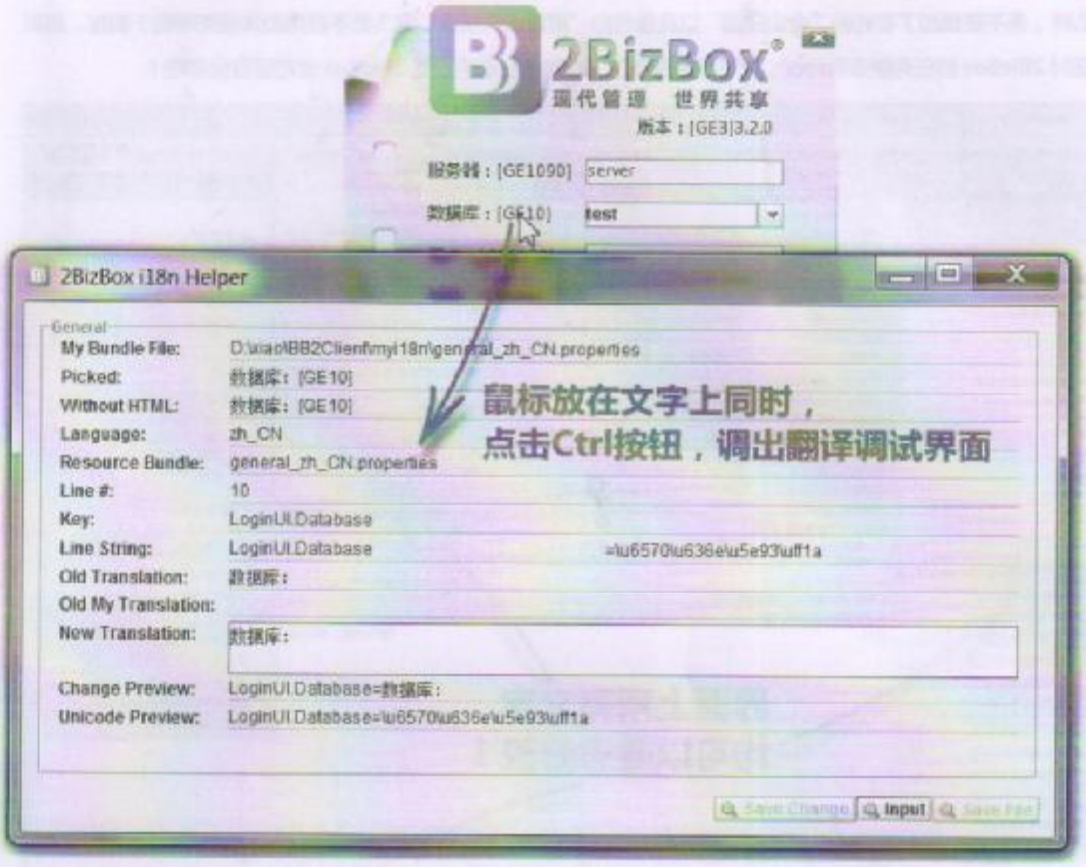
为此，2BizBox 提供了另外一个方法，可以对界面上的显示内容进行更加精细化的调试、调整。这就是 2BizBox 的翻译调试模式。也就四说，2BizBox 可以在正常模式下运行，也可以在翻译调试模式下进行。在翻译调试模式下，2BizBox 会显示一些很特别的信息。

在 2BizBox 的在线程序商店中，您很快就会发现一个“2BizBox 翻译优化工具”的应用程序。安装后，可以将 2BizBox 启动到翻译调试模式。此时观察登录界面，已经发生了很多变化：

此时的 2BizBox 界面和功能与正常模式完全相同，唯一的变化是，所有的文字都有一个奇怪的后缀，例如“[GE10]”、“[GE764]”，等等。甚至鼠标提示文字 tooltip 都有这些奇怪的编码。那么这些奇怪的代码作用是什么呢？它就是用来标识界面中所有文字的翻译来源的代码，也就是标记了它对应的模块、资源文件、语言、key 键值、行号等信息。通过这些代码，我们可以快速对界面上不合适的翻译进行调整和修改。那么，如何进行呢？

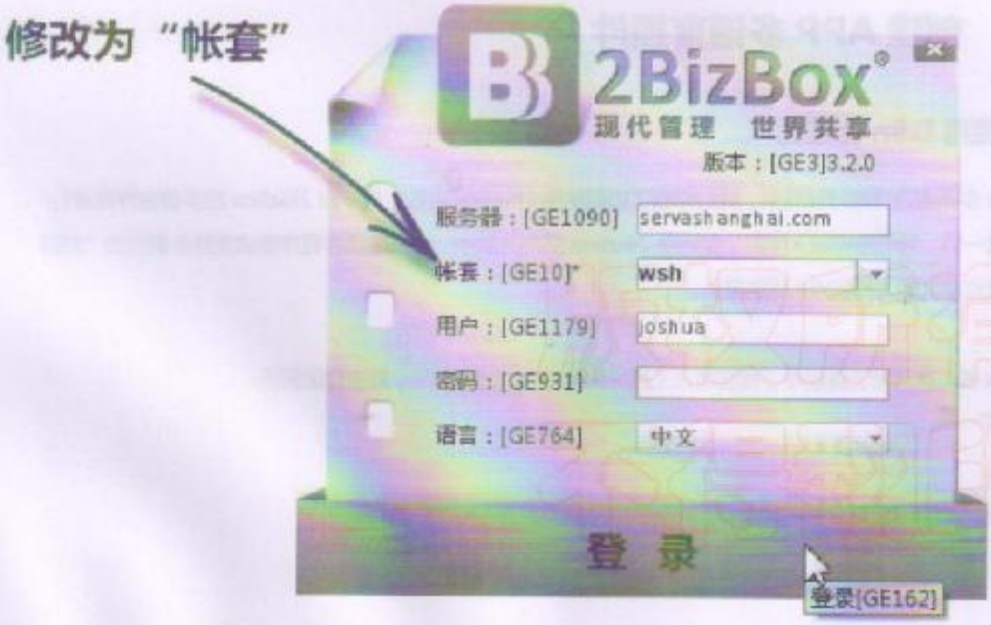
以登录界面的“数据库”为例。很多朋友不喜欢“数据库”这一中文翻译，他们更加习惯财务软件中“帐套”的说法，希望修改这一翻译。做这一点非常简单：首先将鼠标移动到“数据库”文字上方，同时按下键盘上的 Ctrl 键。大约半秒钟，软件就会显示一个神秘界面，将“数据库”这一文字翻译的来龙去脉交代的清清楚楚。

在这一界面中，显示了“数据库”这一单词的来源、原始翻译、资源文件位置、行号，等等信息。更重要的是，我们可以在该页面中直接修改原来的翻译，并点击按钮保存。



保存后，被修改的资源同样会被保存在 2BizBox 安装目录的 myi18n 文件夹中。重启客户端，会立刻看到修改效果。

下图就是把“数据库”修改为“帐套”，保存、重启客户端，显示效果如下图：



怎么样，是不是想起了著名的“金山词霸”以及著名的“屏幕取词技术”呢？是不是用起来很简单呢？现在，如果您还对 2BizBox 的任何翻译有异议，就不用再抱怨了，立刻自己动手，把 2BizBox 改得面目全非吧！

8.4.2 创建 APP 多语言插件

8.4.2.1 使用 i18n 化工具

在文章 2BizBox 多语言翻译教程章节中，我们介绍了如何使用 2BizBox 内置工具，对 2BizBox 的多语言界面进行定制和修改。这一讲，我们继续深入学习，如何在 2BizBox 插件开发中，将您的插件程序做成支持多语言的“跨国插件”，以便让全球各国人民都用上您的插件。

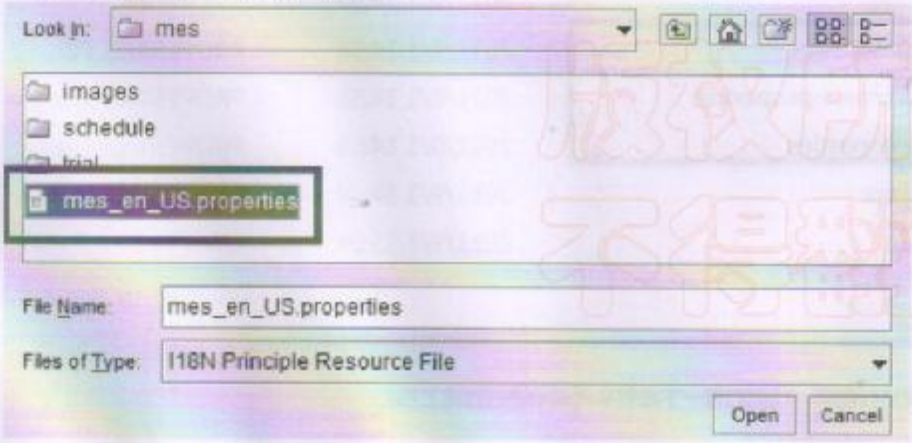
首先在 2BizBox 的二次开发工具包中，找到翻译工具“i18n_for_app.bat”，双击启动程序:

i18n 翻译工具首先让您选择要翻译的语言种类。其中英语为整个 2BizBox 软件的系统语言和主语言，必须选择。接下来，您可以勾上您的插件想要支持的语言，例如英语、繁体中文、日文等等。

接下来，在下方的文本框中，输入您的 i18n 资源文件欲存放的位置。例如，您的当前的 Eclipse 的工程是在 d:\workspace \MyProject 下，源代码在\src 下，您的插件代码在\src\com\myapp 下，您的资源文件想放在和代码同一目录下，文件名字是 app， 可以直接输入一下代码即可：

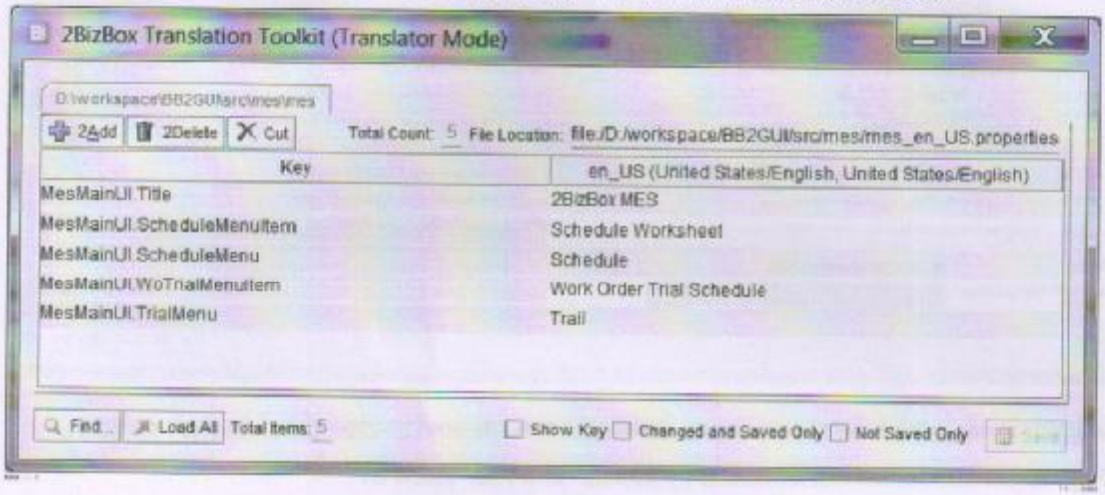
```
1 | D:\workspace\MyProject\src\com\myapp\app
```

如果资源文件是首次使用、尚未存在，点击 OK 按钮，程序会提示您是否要创建它。如果文件已经存在，也可以点击右侧小箭头按钮，在代码目录中进行选择：

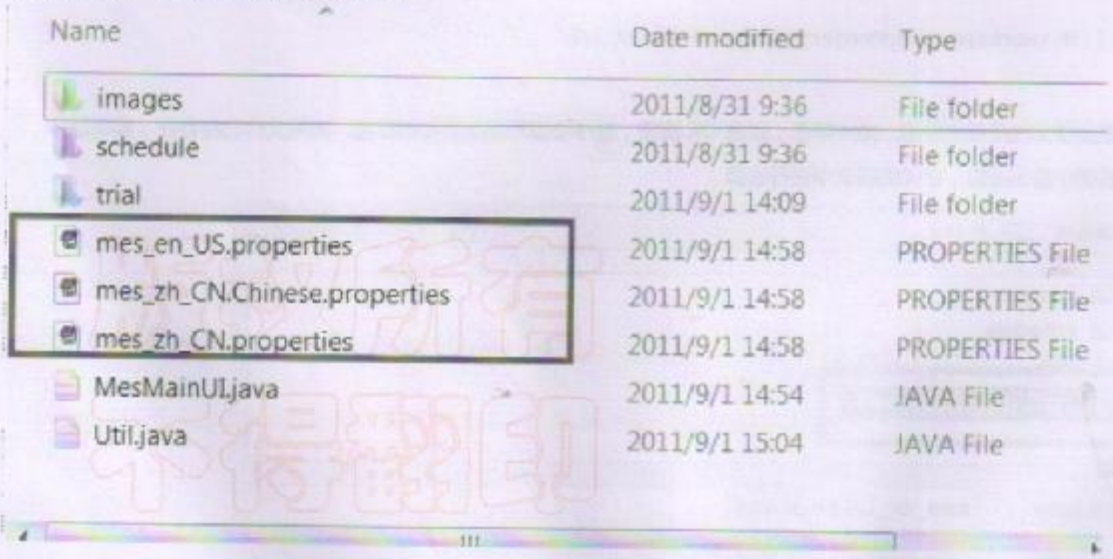


选择后，直接点击 OK 按钮，进入编辑状态。请注意：在浏览目录时，程序只认以 "****_en_US.properties" 结尾的文件名。这是因为，所有的 i18n 资源文件都是以" 文件名_语言（小写）_国家（大写）.properties" 的格式存在的，而 en_US 这是 2BizBox 的主语言，是任何一个 i18n 资源文件必然存在的文件。所以，它用来唯一的标识一个资源文件。当您勾选简体中文时，程序还会自动生成***_zh_CN.properties 等文件，以此类推。

进入翻译工具主界面，我们就可以进行多语言工作了。主界面和前文介绍的翻译工具基本相同，唯一不同的是，它不再是翻译 2BizBox 默认的内部资源文件，而是在翻译我们正在进行中的插件代码中的字符串资源。



上图中，如果新建资源文件，则表格是空；如果已经存在，则显示目前所有的字符串资源。每一列是对应一种语言的具体翻译，这取决于你在上一个界面上都勾选了什么语言。每一个列的语言翻译都会被分别存放在不同的资源文件中（如下图所示）。点击 "Add"、"Delete"、"Cut" 等按钮进行增加、删除、剪切等动作。修改后，在后下方点击按钮 "Save" 保存所有翻译工作。



掌握了如何使用这个工具进行翻译，我们再看一下如何从代码中进行翻译工作。

8.4.2.2 代码的翻译

要对插件程序进行翻译，首先要掌握一个 2BizBox 提供的 i18n 翻译字符串获取类：AppI18nHelper。

AppI18nHelper

首先，在你的插件代码工程中的某个类内，创建一个 AppI18nHelper 实例。这个类是用来获得 i18n 资源文件内容的工具类。在构造函数中，给出你的资源文件所处的目录/包位置。例如，我们的资源文件主文件位于 /src/com/mycompany/myapp /app_en_US.properties，则我们直接构造函数输入 "/com/mycompany/myapp/app"，注意不必带顶层包以上的目 录以及文件后缀。下面是一个具体例子：

```
private static final AppI18nHelper i18n = new AppI18nHelper("/mes/mes");
```

上面的 i18n 这个实例，可以用来访问 src/mes/mes_**_**.properties 资源文件。

在实际开发中，为了避免反复 new AppI18nHelper 实例，我们还可以进一步封装，将这一实例放置在一个 Util 的静态工具类中，方便使用。例如，在本文的 mes 这个插件中，我们定义一个/mes/Util.java 类，它是一个静态的工具类，其中二次封装了 AppI18nHelper 提供的一些基本方法：

```
package mes;

//***

public class Util {
    private static final AppI18nHelper i18n = new AppI18nHelper("/mes/mes");

    //***

    public static String getString(String key) {
        return i18n.getString(key);
    }

    public static String getString(String key, Locale locale) {
        return i18n.getString(key, locale);
    }

    public static String getString(String key, String arg) {
        return i18n.getString(key, arg);
    }

    public static String getString(String key, Object[] args) {
        return i18n.getString(key, args);
    }

    public static String getString(String key, Object[] args, Locale locale) {
        return i18n.getString(key, args, locale);
    }
}
```

这样，原来在 i18n 这个 AppI18nHelper 实例中的几个主要方法，就被静态的封装到 Util 中。以后再在整个 mes 工程中任何代码中，要获得一个多语言字符串，就可以直接这样：

```
String text = Util.getString("SomeUI.SomeTextKey");
```

通过这样二次封装，使用起来就更加方便了。

下面，介绍一下 AppI18nHelper 类中的几个主要函数它们都在上面代码中进行了二次封装。

- public static String getString(String key) 获得 key 对应的当前登录语言的翻译字符串。
- public static String getString(String key, Locale locale)获得 key 对应的给定的语言的翻译字符串。其中 Locale 类详细解释，请见 JDK。
- public static String getString(String key, String arg)获得 key 对应的当前登录语言的翻译字符串，且插入指定的一个元参数。举例：如果我们要显示这样一个字符串：您输入的 abc 是一个非法数字。而其中的 abc 是用户动态输入的字符串，我们在翻译时候是不知道的，那么怎么办呢？此时可以使用元参数。我们可以这样定义这个翻译：TestUI.Message=Your input {0} is not valid number（英文）
  TestUI.Message=您输入的{0}不是一个合法数字（中文）。您可以注意到，在翻译字符串中，我们置入了一个{0}这样一个占位符。它表示这个位置会放入一个动态参数，这个参数就是函数中的 arg 字符串。当运时时刻调用这个函数，2BizBox 就会用 arg 代替{0}然后把翻译结果返回。
- public static String getString(String key, Object[] args)获得 key 对应的当前登录语言的翻译字符串，且插入指定的一系列元参数。和上一个函数类似，只是有更多的元参数。每一个元参数用一个对应数字表示，从 0 开始。举例：TestUI.Message=Your input {0} and {1} is not valid number。args 是一个数组，一般是 String 类型，否则调用.toString()函数获得元参数的字符串，并替换对应的占位符。
- public static String getString(String key, Object[] args, Locale locale)同上，只是用函数指定的语言。

一般来说，有这几个函数，已经足够满足我们一般的 i18n 翻译工作了。

代码中使用

有了这个 Util 二次封装类，我们以一个实际代码为例，看一下在实际代码开发中如何使用它。

```
1  package mes;
2
3  import com.serva.bb2.gui.Main;
4  import free.AppMainUI;
5  import free.FreeMenuBar;
6  import free.FreeRootMenu;
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ActionListener;
9  import javax.swing.JMenuItem;
10 import mes.schedule.ScheduleUI;
11 import mes.trial.TrialUI;
12
13 public class MesMainUI extends AppMainUI {
14
15     public MesMainUI() {
16         init();
17     }
18
19     private void init() {
20         initMenu();
21     }
22
```

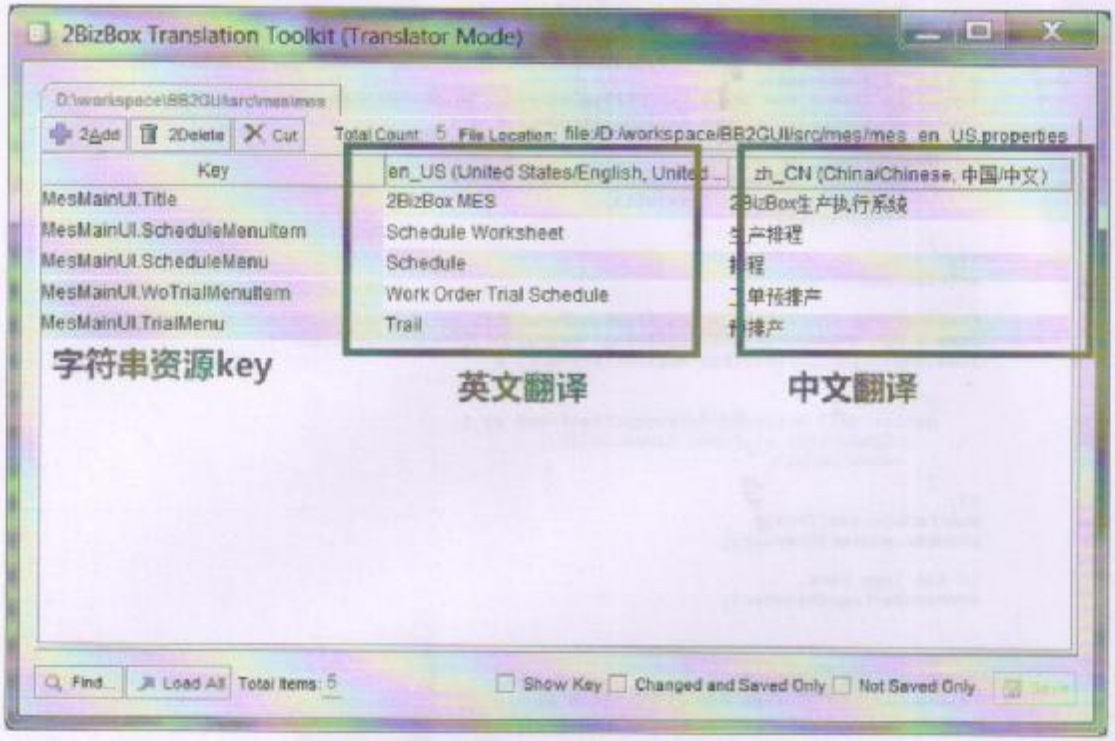
```
    private void initMenu() {
        FreeMenuBar menubar = getFreeMenuBar();
        FreeRootMenu mnTrial = new FreeRootMenu(Util.getString("MesMainUI.TrialMenu"));
        menubar.add(mnTrial);
        JMenuItem item = new JMenuItem(Util.getString("MesMainUI.WoTrialMenuItem"));
        item.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                TrialUI ui = new TrialUI();
                showTab(ui);
            }
        });
        mnTrial.add(item);

        FreeRootMenu mnSchedule = new FreeRootMenu(Util.getString("MesMainUI.ScheduleMenu"));
        item = new JMenuItem(Util.getString("MesMainUI.ScheduleMenuItem"));
        item.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                ScheduleUI ui = new ScheduleUI();
                showTab(ui);
            }
        });
        mnSchedule.add(item);
        menubar.add(mnSchedule);

        // add logo here.
        addMenuBarLogo(menubar);
    }

    @Override
    public String getTitle() {
        return Util.getString("MesMainUI.Title");
    }

    public static void main(String[] args) {
        Main.launchBizBox(MesMainUI.class.getName());
    }
}
```

这是一段典型的外挂插件程序。它继承了 AppMainUI 类，创建了一个外挂程序主界面。然后，在主菜单中加入了两个主菜单和两个菜单项。菜单的文 字都不是某个语言的硬编码，而是使用类似

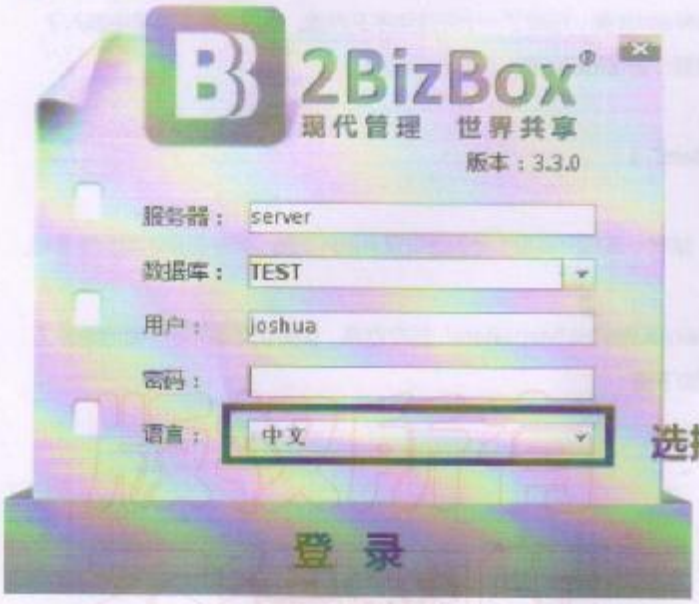
Util.getString("MesMainUI.WoTrialMenuItem")

的方法来从资源文件中 获得不同语言的翻译。这样，当用户使用不同的语言登录插件系统，就会显示不同的翻译项。

当然，还要记得所有的代码中的类似" MesMainUI.WoTrialMenuItem" 的字符串，都要在文章开始介绍的翻译工具中进行具体翻译，并保存在资源文件中。例如下图：



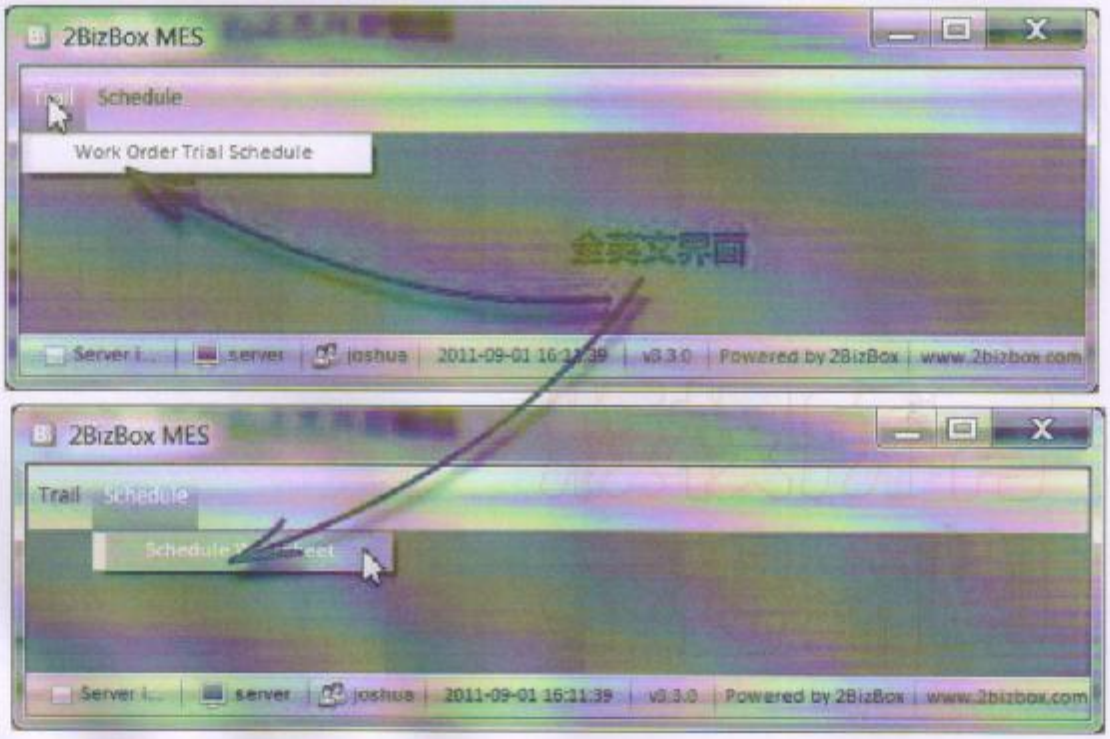
这样，就可以看到多语言效果了。下面我们运行这个插件，看看是什么效果。执行插件，进入正常的 2BizBox 登录界面，选择登录语言为中文：



选择插件登录语言

点击登录，进入插件主界面：

可以看到，界面语言是全中文的；退出后重新登录，选择语言为英文后，再进入主界面，界面语言就变成了全英文：



这样，我们就完成了一个简单而完整的多语言插件程序。

8.4.2.3 更多实践

当然，在实际开发中，我们的源代码可能非常多，而每个类中都有大量的翻译字符串。我们知道，字符串的 key 在一个资源文件中是不可以重复的。那么，该如何小心定义这些 key，避免冲突和重复呢？例如我们有 3 个类，都有标题 Title 这个 key，如果都把 key 定义为“Title”，则会造成冲突，Util.getString(“Title”)的时候，也就无法知道到底取哪一个了。所以，一个重要经验是：使用分段编码的方法来定义 key，用 ClassName.Key 的方式来定义 key，避免冲突。例如，TestA.java 中的标题，我们定义为“TestA.Title”，TestB.java 中的标题，我们定义为“TestB.Title”，以此类推。这样就避免了冲突和重复的情况。

不过，有时候我们也可以采用复用的字符串。例如“确定”这个字符串非常常用，在很多类里面都可能用到。为了避免重复定义，我们可以直接定义一个“Mes.OK”这个资源字符串 key，在 Mes 项目的其他任何地方，都用 Util.getString(“Mes.OK”)来获得这个字符串，避免了重复工作。

8.5 2BizBox 平台与插件开发概述

软件的平台化，是 2BizBox ERP 的重要发展方向之一。通过不断的优化 2BizBox ERP 软件体系架构，并对开发者开放完整的 API 开发接口，让 2BizBox 形成一个 ERP 应用平台和二次开发平台，让更多的合作伙伴和开发者投入到 2BizBox ERP 发展之中。有了完整的 2BizBox 开发平台，才能将厂商、合作伙伴、开发者、最终用户紧密的在同一个平台上共同协作，实现各自的价值。

8.5.1 概述

2BizBox ERP 已经开始了平台化之路。之前，2BizBox 已经提供了完整的 API 二次开发接口。通过 API，开发者可以将 2BizBox ERP 后台和其他第三方应用程序进行无缝的系统集成。接下来，2BizBox 还会开放整个客户端的二次开发平台，让开发者基于现有 2BizBox ERP 客户端进行快速开发和定制，创建自己的应用。再结合 API，开发者与合作伙伴完全可以基于 2BizBox 前后台来创建完全属于自己的企业应用，同时充分利用 2BizBox 软件平台所提供的一切基础设施。

通过 API 和 2BizBox 客户端开发平台，开发者可以创建一系列自己的应用，或创建“插件”或“自定义模块”，插入到现有的 2BizBox 平台上，完善、修改和定制 2BizBox ERP 固有的软件流程、功能界面等等。通过下图，可以更加清晰的了解整个 2BizBox 软件开发平台的结构：

8.5.2 功能

那么，2BizBox 平台都提供了什么样的功能，通过 2BizBox 平台又可以做什么呢？以下罗列了一些典型的应用方法：

2BizBox 客户端平台

- 创建一个全新的 2BizBox 客户端框架；
- 修改 2BizBox 客户端功能入口，包括菜单、模块等；
- 添加自定义模块；

- 整合第三方功能模块；
- 修改默认图形界面风格及内容；

2BizBox 服务器端平台

- 创建自定义数据库表；
- 创建各种自定义数据库查询；
- 创建自定义数据结构；
- 添加自定义数据字段；
- 创建各类自定义数据报表；

有了这些激动人心的功能和接口，我们就可以充分利用 2BizBox 软件平台，“随心所欲”的创建自己的应用了。

8.5.3 客户端平台二次开发

下面介绍如何基于 2BizBox 客户端平台进行二次开发。我们不禁要问：既然有了 API，它可以做任何事，那么为什么还需要 2BizBox 客户端平台呢？

大家知道，2BizBox 客户端不仅提供整个 2BizBox ERP 所有的操作图形界面，还处理了大量的基础业务。例如：登录、安全控制、数据的调用与传输、异常的处理、程序界面的组织和显示，等等。如果我们想要开发一个全新的模块应用，如果仅仅基于 API 对后台进行交互，势必要自己写代码，来重新编写全部刚才提到的基础功能。这将是一个无法想象的艰巨任务，任何开发者都不可能快速、轻易完成。

而 2BizBox 客户端平台，就是将现有的 2BizBox 客户端程序进行平台化改造处理，使之成为一个功能完整的 2BizBox 客户端，同时有为开发者提供了丰富的开发接口，我们不必再重写登录、安全控制、数据交换、程序显示框架等等基础任务，而直接开发自己的图形界面和模块即可。从而，可以达到快速开发“自定义模块”和“插件”的目的。

下面我们就以实际代码来做一个例子。通过这个例子，大家就能够体会到，通过 2BizBox 客户端平台进行开发是多么容易的事情。

我们首先介绍如何创建和定制一个完全属于自己的 2BizBox 客户端框架。

在此之前，我们再次熟悉一下 2BizBox 客户端的结构。打开 2BizBox 客户端，我们可以看到，整个图形界面只有一个主窗口，我们称之为 MainUI。在主窗口内部，有顶部的菜单栏、底部的状态栏、左边的模块栏，以及中间的内容栏。

对于开发者来说，我们首先想到的，就是想创建一个干净的、纯净的、自定义的主窗口，我们可以定制要显示什么、不要显示什么，并任意添加自己的内容。要做到这一点非常容易：我们可以继承 2BizBox 客户端平台提供的主窗口类：AppMainUI。AppMainUI 是一个干净的、具有完整功能的、没有任何内容的主窗口。它具有完整的登录、通讯、异常处理、显示框架、显示风格等内容，但是界面上却是一个完全空空的主窗口，我们可以在上面添加任何自定义功能。这正是我们想要的。

下面我们就亲自动手，通过简单的代码来尝试如何使用 AppMainUI。不要被“写代码”吓到，即使您不是一个程序员，不动写代码，您也完全可以编写和运行本文中提到的例子，因为它非常简单。

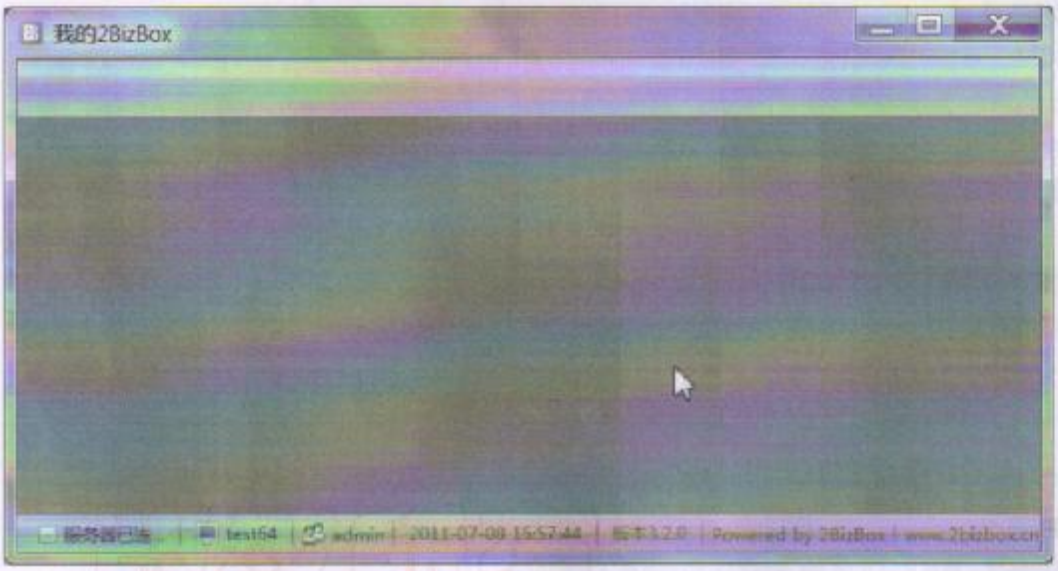
8.5.3.1 创建窗口

启动 Netbeans 或 Eclipse。如对此不熟悉，请先阅读 2BizBox API 开发教程。在 Netbeans 中创建一个工程，并引入 2BizBox 平台所需的所有运行库文件。所需所有库文件都在这里可以下载：点击下载。将接口库文件、环境库文件下载后解压，将其中的所有 jar 文件引入您的 IDE 工程即可。
然后，创建一个 Java 类，内容如

```
package test;

import com.serva.bb2.gui.Main;
import free.AppMainUI;

public class MyMainUI extends AppMainUI {

    @Override
    public String getTitle() {
        return "Hello 2BizBox";
    }

    public static void main(String[] args) {
        Main.launchBizBox(MyMainUI.class.getName());
    }
}
```

是的，就这么简单，一个简单的 2BizBox 程序就创建完成了。其中，我们创建了新的 Java 类 MyMainUI ，它继承于 2BizBox 客户端平台提供的 AppMainUI 主窗口类。我们重载了 getTitle 方法，返回了一个新的窗口标题。最后，编写启动主函数 main，其中用 Main.launchBizBox 方法来启动 2BizBox 主窗口。

编译，运行这个 Java 类。怎么样？奇迹发生了。首先，程序会显示一个完整的 2BizBox 登陆界面。此时，您可以输入连接信息，连接一台正在运行的 2BizBox 服务器。点击登录按钮后，一个我们自定义的全新主界面呈现在眼前。正如我们预期的那样：它具有完整的登录和显示功能，但内容空空。如下图：



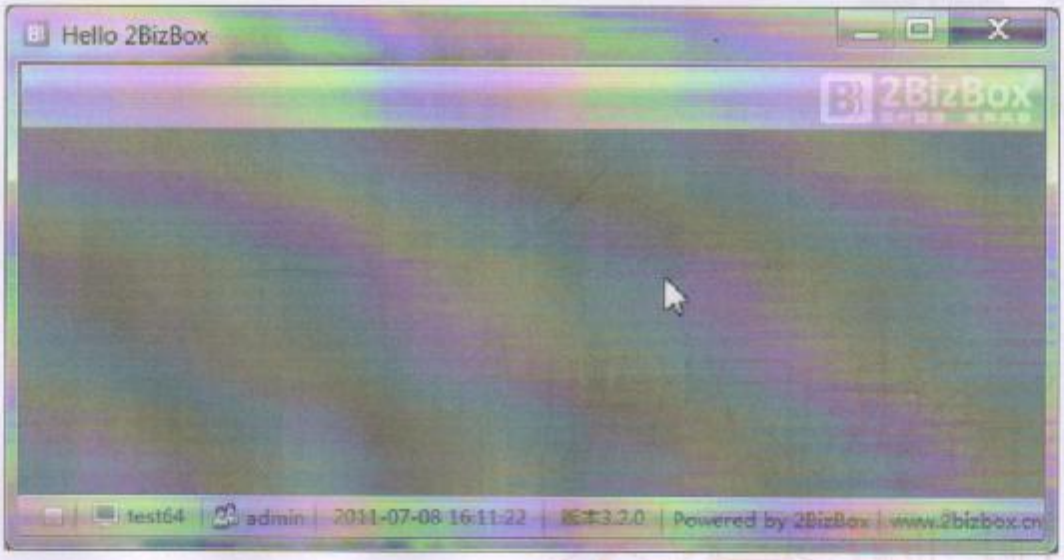
这是一个干净的主窗体。但是，它具有完整的、基本的 2BizBox 客户端功能。它已经实现了登录、安全控制；它已经与服务器建立的通讯，并进行心跳连接；它已经获取了服务器和当前数据库的各种信息。只是，它没有显示任何内容而已。

8.5.3.2 设置 logo

细心的读者可能已经注意到，这个空窗口虽然有完整的状态条，但是右上角并没有显示系统 logo。下面我们稍微修改代码，加上 logo：

```
package test;

import com.serva.bb2.gui.Main;
import free.AppMainUI;
import free.FreeMenuBar;
public class MyMainUI extends AppMainUI {

    public MyMainUI() {
        FreeMenuBar menubar = getFreeMenuBar();
        //add logo here.
        addMenuBarLogo(menubar);
    }

    @Override
    public String getTitle() {
        return "Hello 2BizBox";
    }

    public static void main(String[] args) {
        Main.launchBizBox(MyMainUI.class.getName());
    }
}
```

上面的修改，仅仅是在构造函数中，加入了两句话。第一句话 FreeMenuBar menubar = getFreeMenuBar()用来获取当前主窗口的主菜单，第二句话 addMenuBarLogo(menubar);用来在主菜单上加入 logo。再次运行代码，效果如下图：



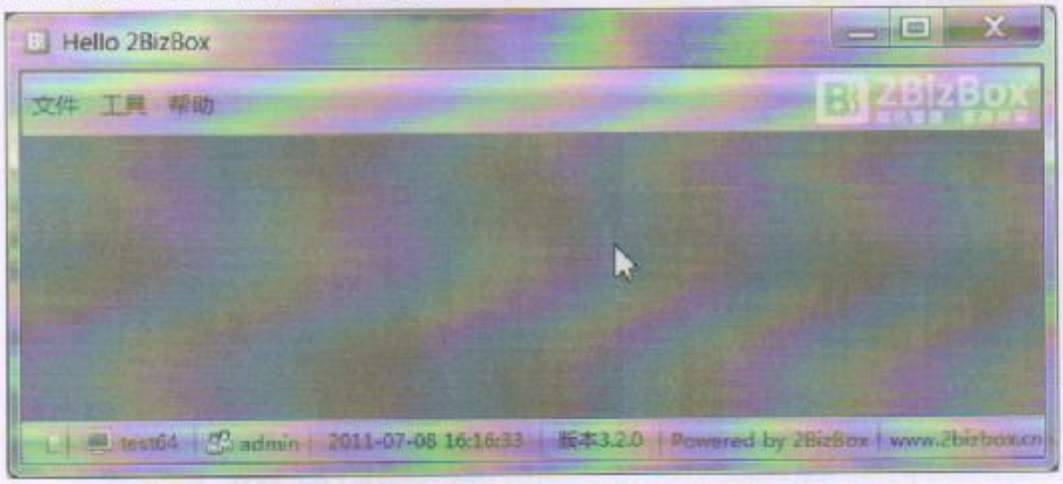
怎么样，右上角熟悉的“现代管理、世界共享”的 2BizBox 口号 logo 出现了。

8.5.3.3 添加主菜单

接下来，我们在主菜单上添加两个我们自己定义的菜单。在创建菜单、添加 logo 前面，添加以下几句话：

```
FreeMenuBar menubar = getFreeMenuBar();
menubar.add(new FreeRootMenu("文件"));
menubar.add(new FreeRootMenu("工具"));
menubar.add(new FreeRootMenu("帮助"));
//add logo here.
addMenuBarLogo(menubar);
```

好了。重新运行程序，我们的主菜单有内容了：

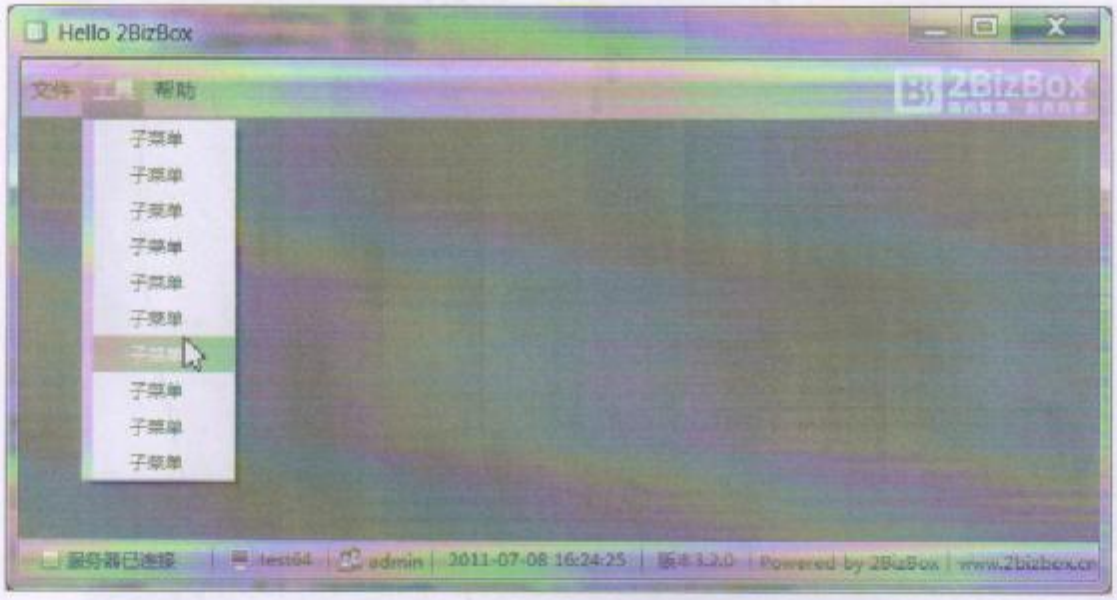


8.5.3.4 添加子菜单

熟悉 Java Swing 编程的朋友，添加一个子菜单是很简单的。下面我们继续写代码：

```
public MyMainUI() {
    FreeMenuBar menubar = getFreeMenuBar();
    menubar.add(createMenu("文件"));
    menubar.add(createMenu("工具"));
    menubar.add(createMenu("帮助"));
    //add logo here.
    addMenuBarLogo(menubar);
}

private FreeRootMenu createMenu(String text) {
    FreeRootMenu menu = new FreeRootMenu(text);
    for (int i = 0; i < 10; i++) {
        JMenuItem item = new JMenuItem("子菜单");
        menu.add(item);
    }
    return menu;
}
```

以上代码稍作修改，每个主菜单通过一个 createMenu 函数生成，里面添加了 10 个子菜单项。重新运行程序，我们的主菜单有子菜单了：



8.5.4 创建页面

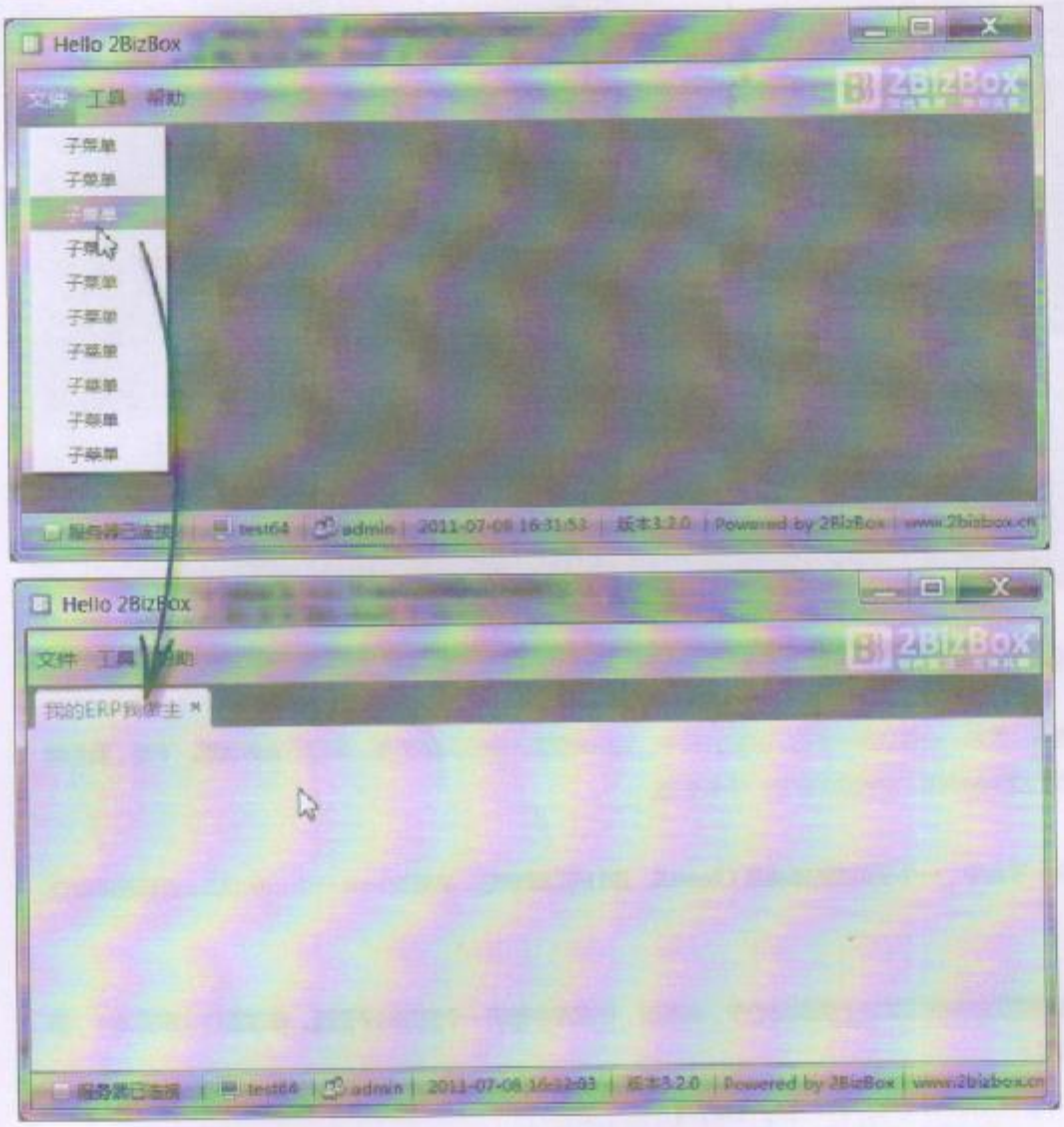
有了主窗口、菜单，还是远远不够的。实际开发中，我们需要显示更多具体页面，来显示业务数据。下面，我们就在菜单项上面添加动作，来创建和显示一个子页面。

在 2BizBox 平台中，一个子页面的基类是 ClientUI。我们可以继承它，或直接 new 一个 ClientUI，显示在主窗口中。

下面，我们就在前面的菜单项上面添加动作，点击后，在程序中显示一个空白的子页面。修改创建菜单的循环，添加代码如下：

```
for (int i = 0; i < 10; i++) {
    JMenuItem item = new JMenuItem("子菜单");
    item.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            ClientUI ui = new ClientUI();
            ui.setTitle("我的ERP我做主");
            showTab(ui);
        }
    });
    menu.add(item);
}
```

以上代码，给每一个菜单项添加一个动作，创建一个空白的页面，设置标题，并显示出来。运行，点击菜单，效果如下图：



8.5.5 接下来

接下来，我们可以做什么呢？其实，可以做任何事了。我们可以自己创建 ClientUI 页面，自己添加按钮、通过 API 从后台获得数据，并进行显示、编辑、处理、报表、打印....无所不能。难道不是么？接下来，就看您的想象力了。

8.6 菜单集成式插件开发示例

8.6.1 前言

2BizBox 的应用程序二次开发，简单说，有三类典型的方式：集成式、外挂式、插件式。

- 集成式：类似系统集成，就是通过 2BizBox API，把 2BizBox ERP 和其他第三方应用进行接口和通讯，实现数据交换和信息整合，达到系统集成的目的。例如和 OA、财务软件等软件的集成。集成方式下，2BizBox 和第三方应用都独立运行，功能不便，只是在特定事件下进行数据通讯。
- 外挂式：通过前面的教程可以看到，2BizBox 提供了一个二次开发平台，可以在保留整个 ERP 功能的“裸”平台上，直接添加属于自己的图形界面和新功能，同时有可以利用 2BizBox 原有的一切功能。它看起来是一个独立的程序，外观和 2BizBox 客户端程序一致，却可以添加自己的功能界面。这类程序类似一个“外挂”插件：独立运行而又依赖 2BizBox 的后端服务器和前端平台。
- 插件式：插件式更加直接，就是把做好的一些新功能、新模块、新报表等程序，直接以菜单项的方式集成在当前 2BizBox 客户端之上，可以进入 2BizBox 界面后直接点击插件菜单调出插件程序。这种方式有最佳的集成性和易用性。本文就介绍如何进行插件式应用程序的二次开发。

8.6.2 创建插件程序界面

首先，我们创建一个模拟的、非常简单的插件程序界面。它就是一个类似 Hello World 一样的标准的 2BizBox 界面，上面显示两个按钮，仅此而已，没有任何功能。这个界面只是模仿了一个真实的、2BizBox 开发者创建的一个功能插件界面而已。

2BizBox 中，所有的界面都是在主界面的主 Tab 中，以一个 Tab 标签页的形式存在的，如下图：

所有这些页面，都是继承于 com.serva.bb2.gui.base.ClientUI 这个类。ClientUI 其实本质上是一个 Swing 的 JPanel 组件，只是被封装了很多方法而已。只有 ClientUI 类型的组件才能显示在 2BizBox 的主窗口之中。

所以，我们就生成一个非常简单的类，从 ClientUI 继承：

```
package app.test;

import java.awt.BorderLayout;

import javax.swing.BorderFactory;
import javax.swing.JButton;

import com.serva.bb2.gui.base.ClientUI;

public class TestPluginUI extends ClientUI {
    public TestPluginUI() {
        this.setTitle("我的第一个华丽插件");
        this.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        this.add(new JButton("这是一个华丽的按钮，在窗口上部"), BorderLayout.NORTH);
        this.add(new JButton("这是一个华丽的按钮，在窗口的中部"), BorderLayout.CENTER);
    }
}
```

只是一个不能再简单的界面。设置一下四周边距为 10 后，直接在界面的上部和中部各放置一个按钮。注意在写代码时，先引入 2BizBox 开发平台的 jar 包，并 import 进 com.serva.bb2.gui.base.ClientUI 这个类。

编译通过后，可以把它打成一个 jar 包。通过 Eclipse 或 NetBeans 等工具都可以方便的进行打包。或者，更简单的方法是，直接用压缩工具创建一个 test.zip 文件，在里面创建 app/test 目录，再把编译好的 TestPluginUI.class 放在这个目录下，最后把 test.zip 修改为 test.jar，也可以。

获得 test.jar 后，我们就可以进行下一步的配置工作了，其目的是把这个界面集成在 2BizBox 软件客户端中。

8.6.3 配置插件

为了将插件配置在 2BizBox 客户端中，可以进入 2BizBox 安装目录，例如 c:\Program Files\2BizBox ERP 3\。在 client\apps 子目录，可以手工创建一个 test 目录，将 test.jar 放入 test 目录。apps 目录是放置所有 2BizBox 扩展程序的目录，每个插件都有自己独立的目录，便于管理。然后，在上层目录找到 apps.xml 配置文件，并将其打开。

```
<?xml version="1.0" encoding="UTF-8"?>
<apps>
</apps>
```

这是一个空空的配置文件，只有一个标签。下面，我们在标签的中间，插入我们刚刚定义的插件信息，如下：

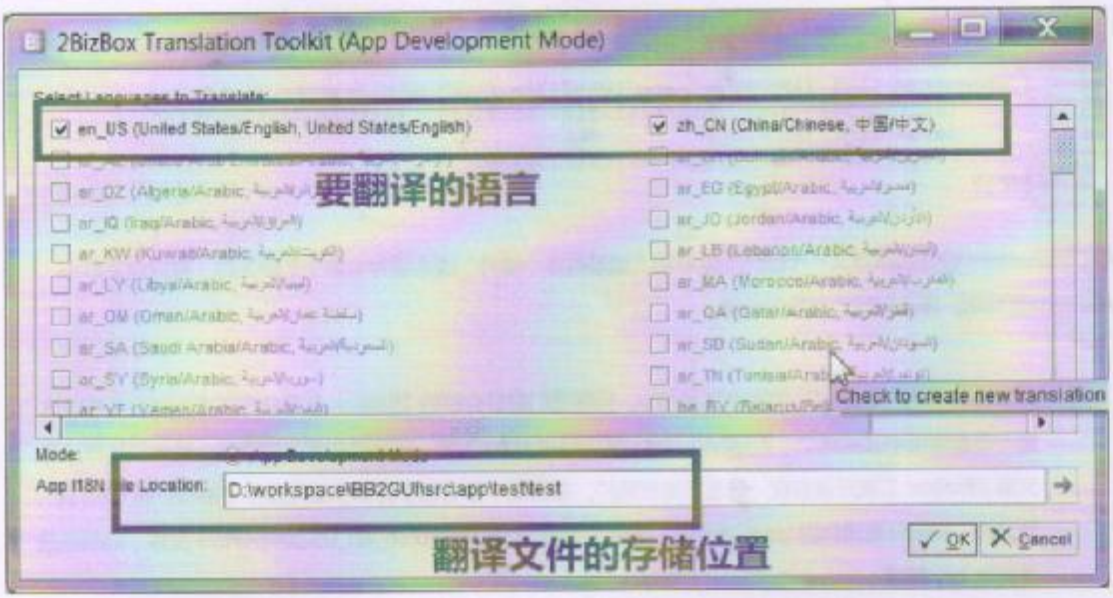
```
<?xml version="1.0" encoding="UTF-8"?>
<apps>
    <menuitem text-i18n-key="/app/test/i18n:AppPlugin.TestMenuText"
        tooltip-i18n-key="/app/test/i18n:AppPlugin.TestMenuTextTooltip"
        jars="test.jar"
        client-ui-class="app.test.TestPluginUI"      />
</apps>
```

其中，menuitem 标签表示定义一个插件的菜单项。如果有多个插件，或者插件有多个功能入口，可以定义多个 menuitem 菜单项。每个菜单项有如下信息：

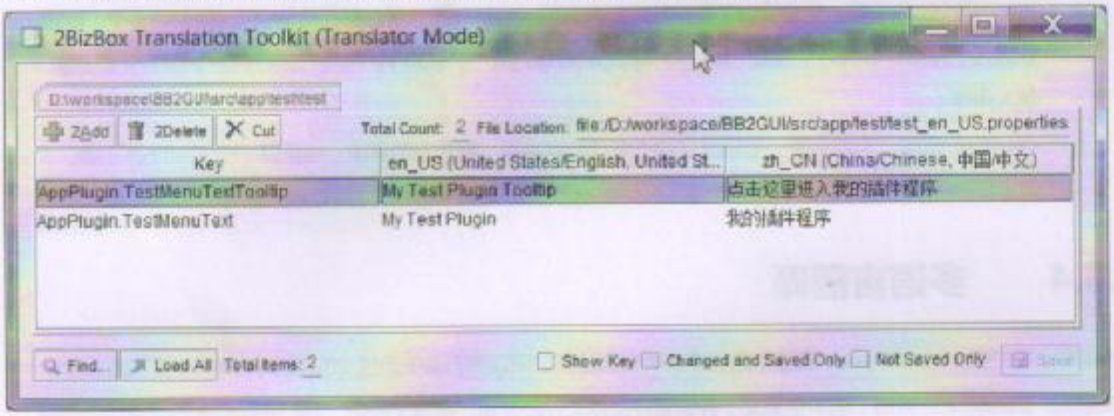
- text-i18n-key：菜单文字的翻译资源 key。记得我们曾经介绍的 2BizBox 是完全多语言环境吗？所以这里不要直接写中文或英文，而是写入 key，然后通过 i18n 工具，在插件中定义多语言。具体使用方法见文章 2BizBox 二次开发教程：开发多语言插件。在这里，冒号把字符串分成两部分：前部分是 i18n 资源文件，后部分是具体的 key。本例中，资源文件是/app/test/i18n_en_US.properties 文件，而后面这是具体 key 键值；
- tooltip-i18n-key：同上类似，是菜单的 tootlip 提示文字；
- jars：这个比较关键。它是给出了插件的运行库文件，也就是插件所有需要的 jar 包。如果有多个，则连续输入，并用分号隔开，例如“test.jar;app.jar;app2.jar”。这些 jar 包会在运行时被 2BizBox 动态加载入内存；
- client-ui-class：非常重要。它定义了这个菜单项点击后，要显示的界面，也就是一个 ClientUI 的类。在上面的代码中可以看到，我们的主界面类是 app.test.TestPluginUI，所以输入即可；

8.6.4 多语言翻译

上一步中，我们在 text-i18n-key 属性中定义了多语言的键值，所以我们需要通过 2BizBox 提供的翻译工具进行字符串定义。启动翻译工具后，选中中英文，并设置好资源文件的位置。在本工程中，我们放在和主界面相同的包路径：/app/test 目录下，名字为 test。



按照下图方式，添加两项字符串翻译信息。注意保持 key 和配置文件中的内容相同。



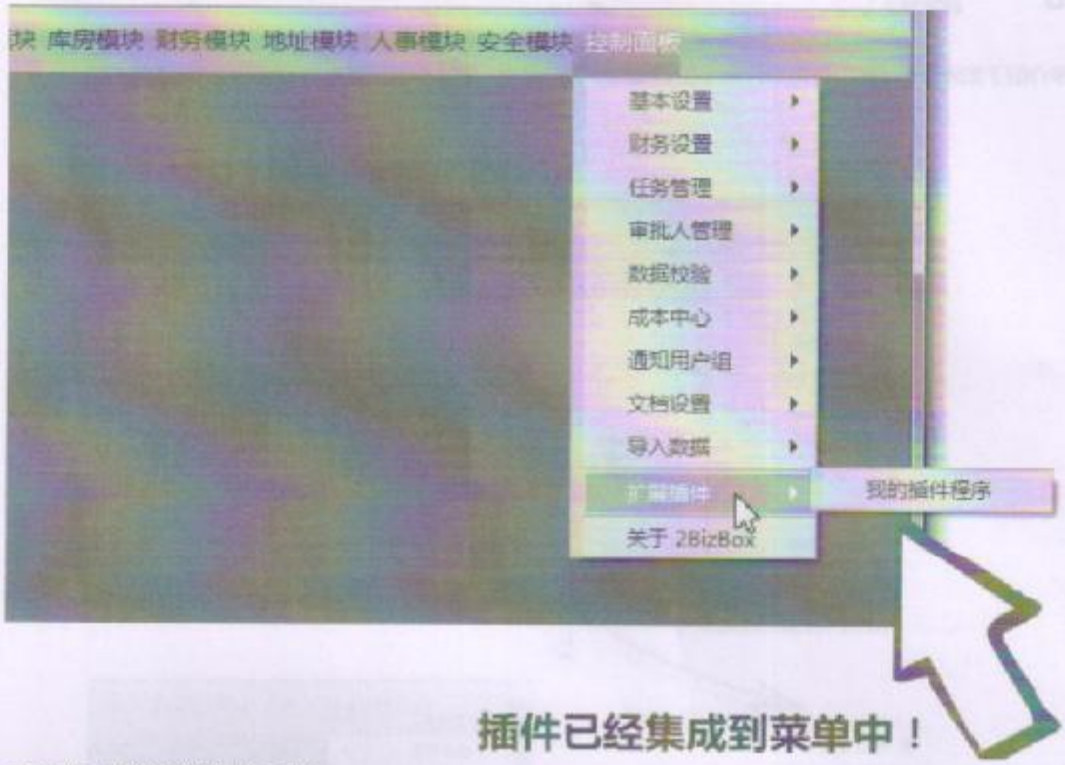
保存后，将生成的所有 properties 文件打包到 jar 文件中。最终，jar 包内容如下图：

名字	大小	包裹大小	类型
..			Folder
TestPluginUI.class	754	522	文件 class
i18n_en_US.properties	91	50	文件 properties
i18n_zh_CN.Chinese.properties	109	79	文件 properties
i18n_zh_CN.properties	163	99	文件 properties

8.6.5 运行

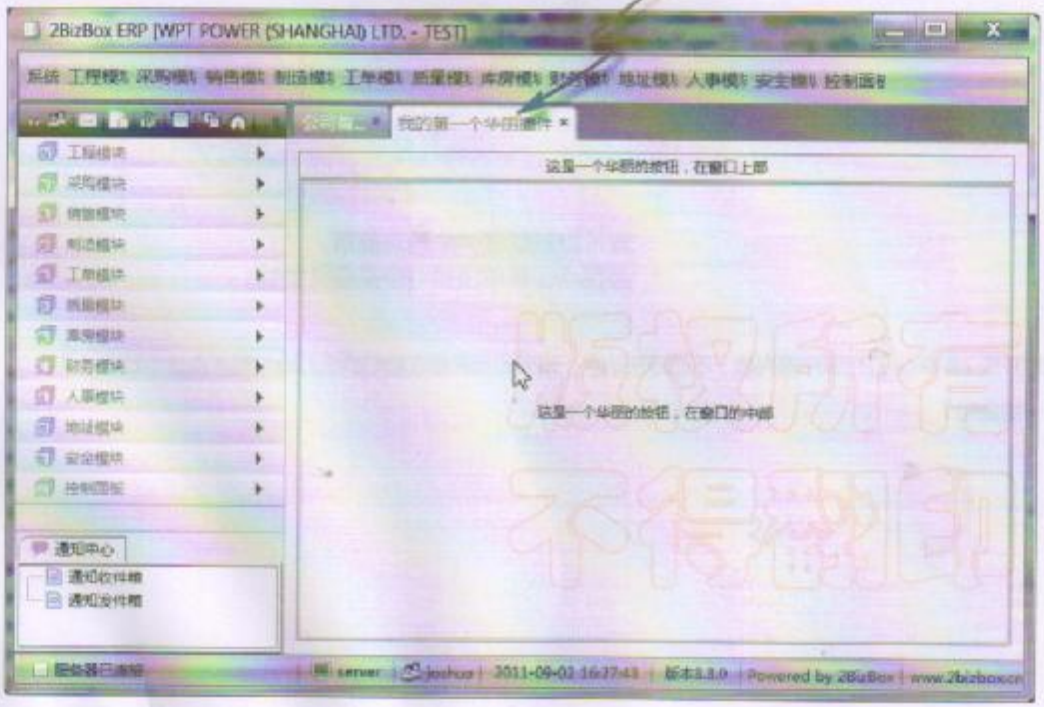
将 jar 包、配置文件放在安装目录中。启动 2BizBox 客户端程序。可以看到，在控制面板的主菜单中，已经看到插

件扩展菜单，如下图。



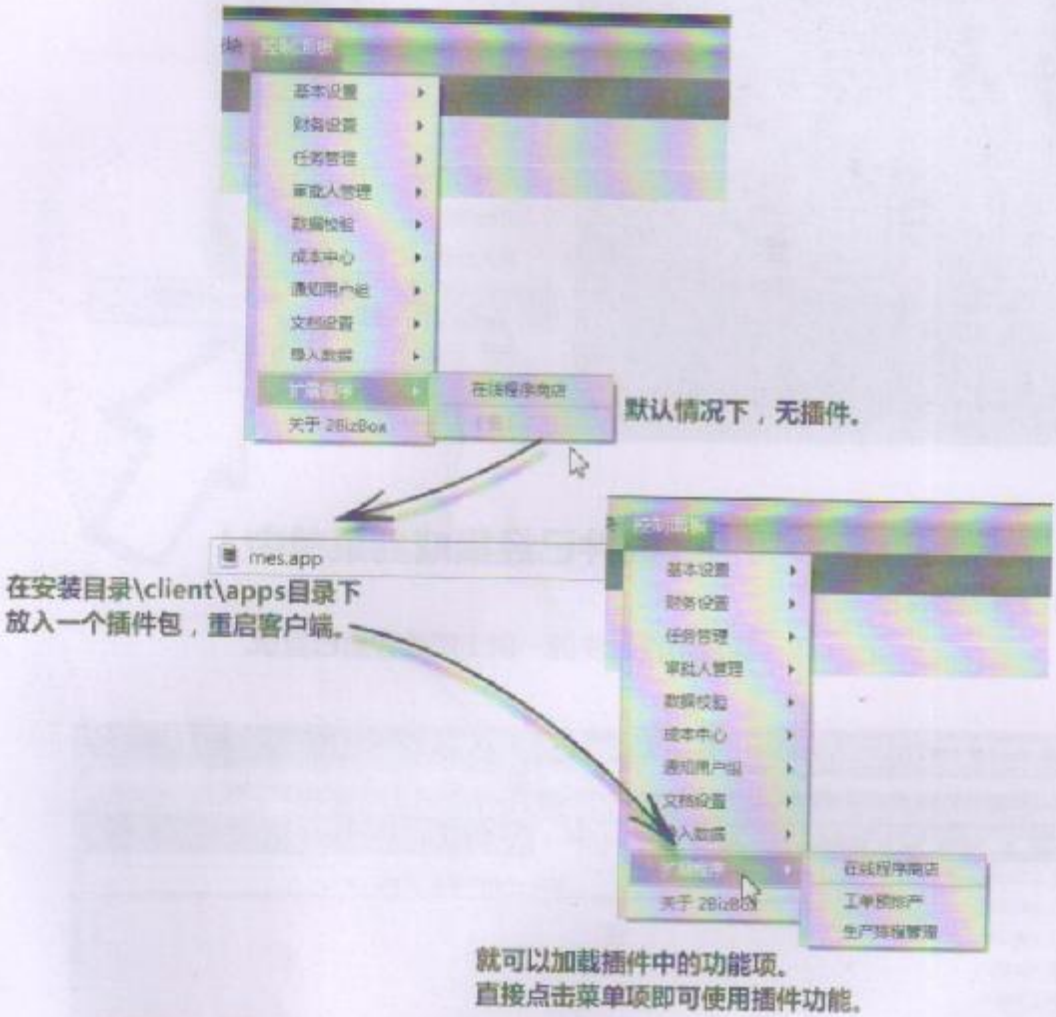
点击菜单，见证奇迹发生的一刻！

见证奇迹发生的一刻！插件界面已显示。



8.6.6 后记

本文简单介绍了如何开发插件，并将插件集成在 2BizBox 客户端中。下图展示了其更加清晰的结构和使用方法。



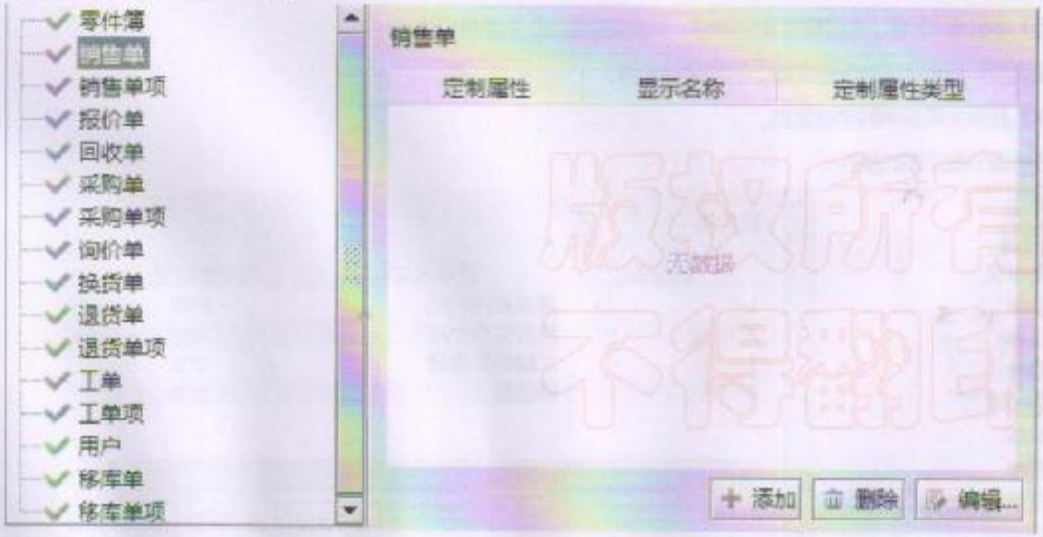
如果您对开发 2BizBox 应用程序感兴趣，不妨尽快动手，将您的插件放在我们的官方应用程序商店中展示，或许还能赚上一桶金呢！

8.7 定制属性与数据过滤

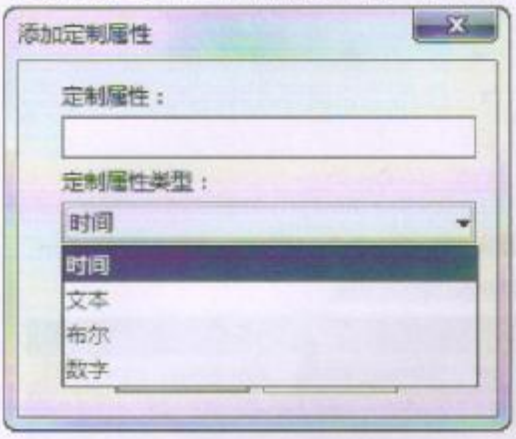
8.7.1 定制属性

定制属性，即自定义字段功能，有了它，大家就能按自己公司的要求，自己定义零件、销售单、采购单等单据的一些属性。"定制属性"的设置入口在控制面板下，点击进去就能看到定制属性的全貌。

目前可以定制属性功能可以涉及到零件簿、销售单、销售单项、报价单、回收单、采购单、采购单项、询价单、换货单、退货单、工单、工单单项。



可定制的属性类型有四种，点击右下角的“添加”按钮后可以进行查看。



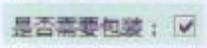
每种属性类型都可以添加显示名称，类型注解如下：

文本：表示添加一个文本输入对象

数字：添加一个数字输入对象

时间：表示添加一个日期选项框，可用于定制一个日期属性

布尔：表示添加一个复选框



下面我们就以销售单为例，介绍一下定制属性是如何操作的。

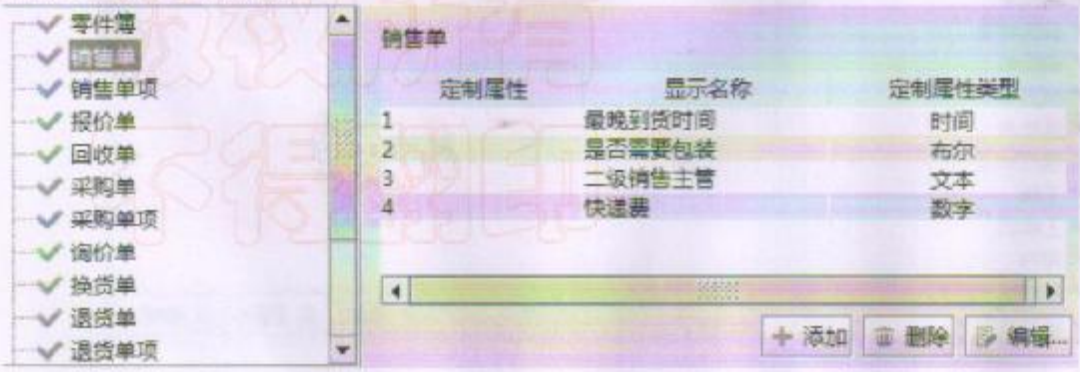
销售单可以在两个地方定制属性：销售单和销售单项。在左侧选择框内选择“销售单”，开始定制销售单的字段。点击“添加”，填写“定制属性”、“显示名称”和“定制属性类型”后，点击“确定”按钮，增加的字段就会显示在上方的定制属性一览表中。点击“删除”，可删除指定的定制属性；若要修改已经设定的属性，则可直接修改显示名称，再点击“编辑”即可。

【定制属性】字段的 ID 号，不可填写非法字符，长度限制在 50 个字符以内。

【定制属性类型】限定了该字段的类型，下拉菜单选择。

【显示名称】显示在单据中的字段名称。

例如，我们添加下图所示的属性：



添加完毕后，点击“确定”，耐心等待系统自动重启服务器。当定制属性界面消失后，就表明添加成功。

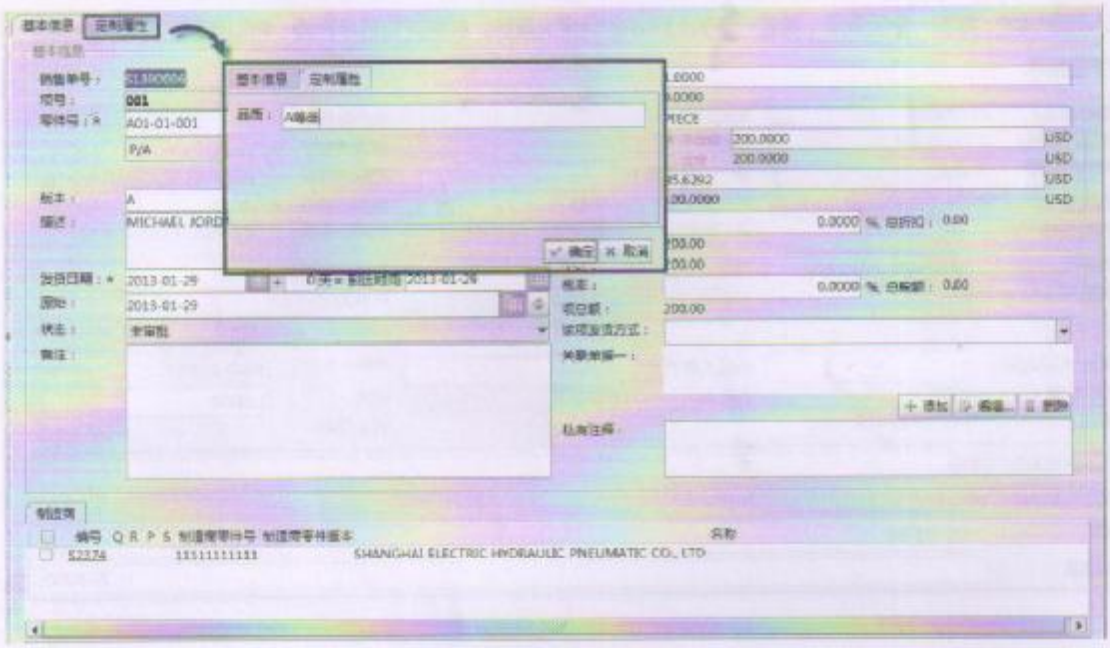
进入销售模块，添加一个销售单，验证一下我们添加的自定义属性。在更新销售单界面，可以看到所有添加的属性，如下图所示：

同时，在销售单属性界面，会出现“定制属性”标签：

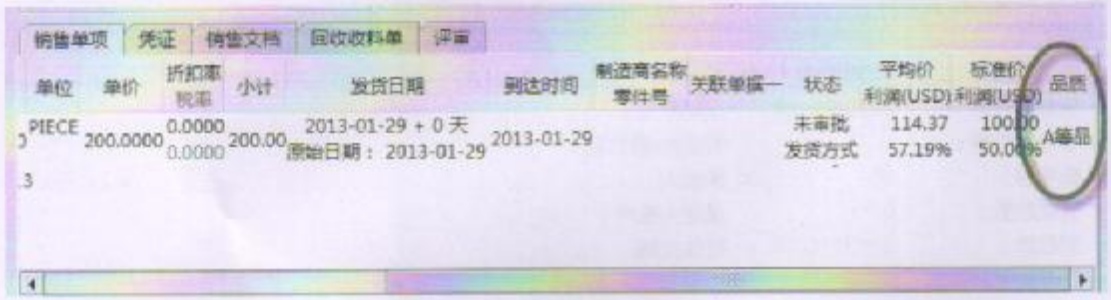
点击标签可以查看自定义属性详情：

我们再来看一下销售单项的定制属性。

同样地，为销售单项添加一个定制属性，在销售单项更新界面，也会出现“定制属性”标签。



销售单项添加后，定制属性在销售单属性界面显示如下：



ERP 不可能做到各个行业都兼顾，很多时候，为了增加一个小小的备注栏，都要进行二次开发。有了定制属性这一功能，企业可以方便地根据自己的行业特点添加和设置相关参数了。

8.7.2 数据过滤

经常有一些朋友提到，希望能够对销售单的权限控制进行细分，比如要求做到有的人能够看到所有人创建的销售单，而有的人只能看到自己创建的销售单。

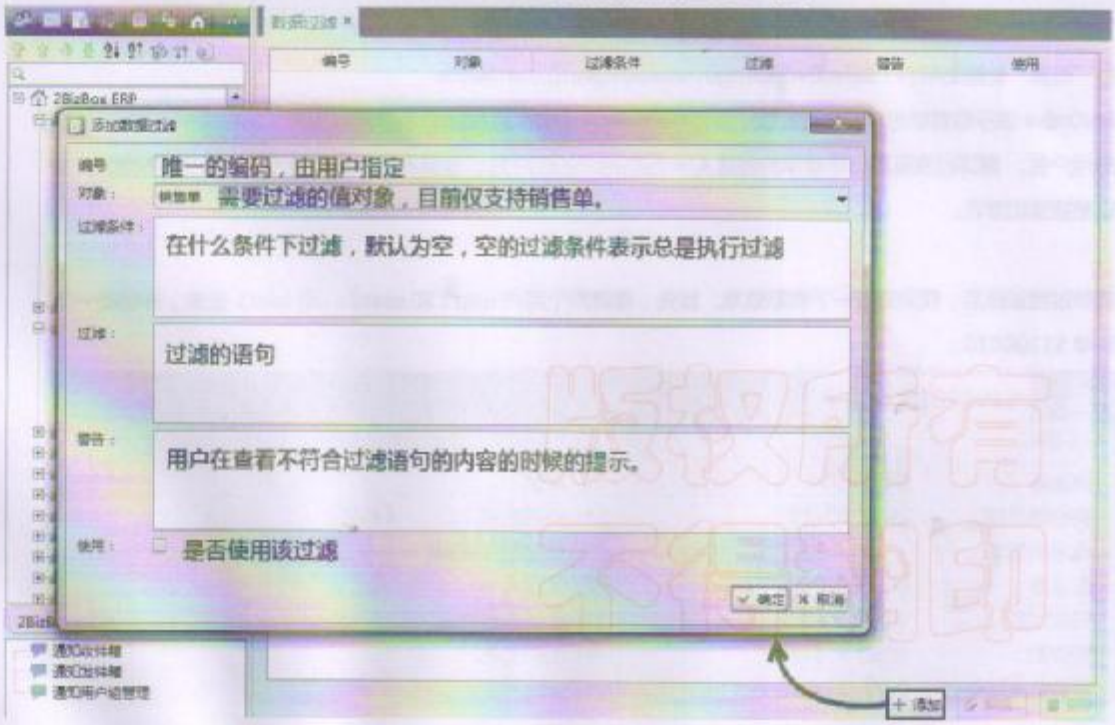
2BizBox 3.5.1 提供了一个数据过滤的功能，通过该功能正好可以实现上面的需求。所谓数据过滤，就是一个对于特定数据实现动态过滤的功能，对于特定的数据，通过数据过滤过滤掉不希望出现的内容，只剩下希望出现的内容。这个数据过滤就是在我们原来的程序的基础上，增加了一个过滤层。

下面就为大家讲述如何通过数据过滤来实现销售单权限细分。

8.7.2.1 数据过滤功能概览

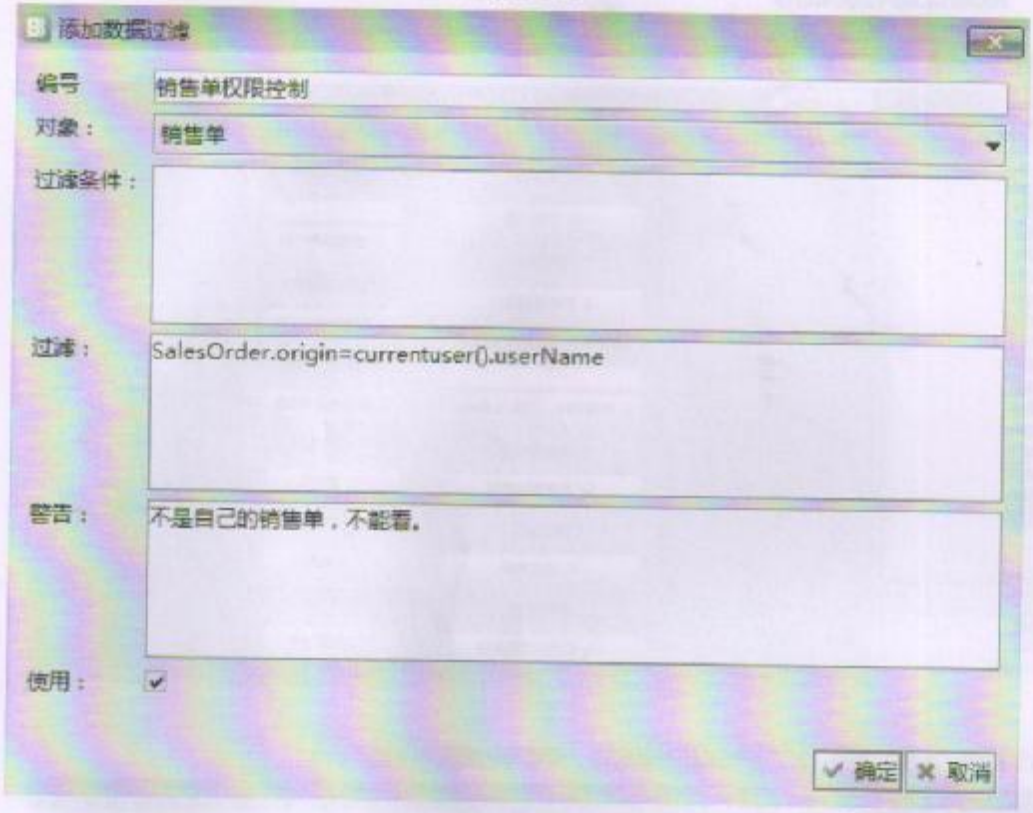
在控制面板界面，点击“数据过滤”按钮即可进入数据过滤的界面。

点击界面右下角的“添加”按钮，打开添加数据过滤窗口，要填写的属性如下图所示：



8.7.2.2 销售单权限控制

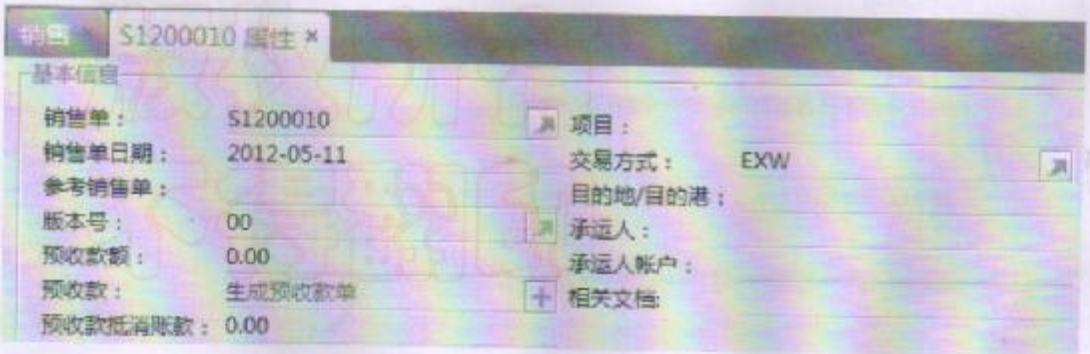
按上文所述，我们按下面的格式，来添加销售单的数据过滤：



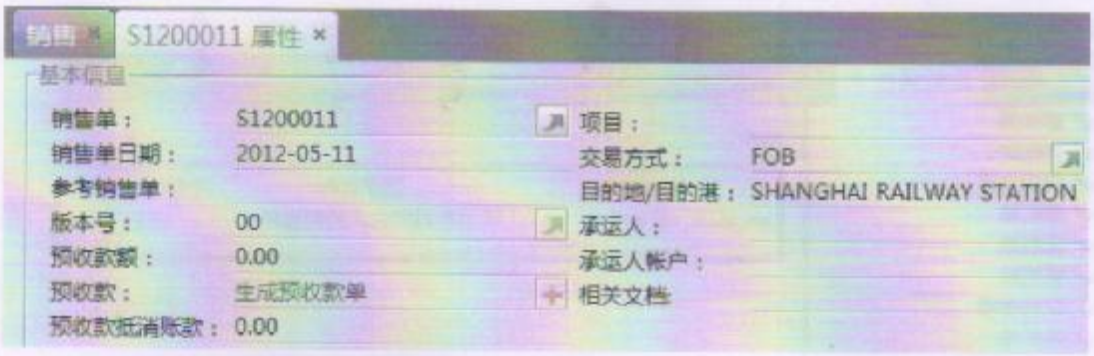
其中“过滤”处的语句为：SalesOrder.origin=currentuser().userName

SalesOrder 表示销售单对象，origin 表示销售单的创建人，currentuser() 表示当前用户，userName 表示当前用户的用户名。翻译过来就是：销售单的创建人 = 当前用户的用户名，也就是说，只有符合这个条件的销售单，才可以被搜索和查看。

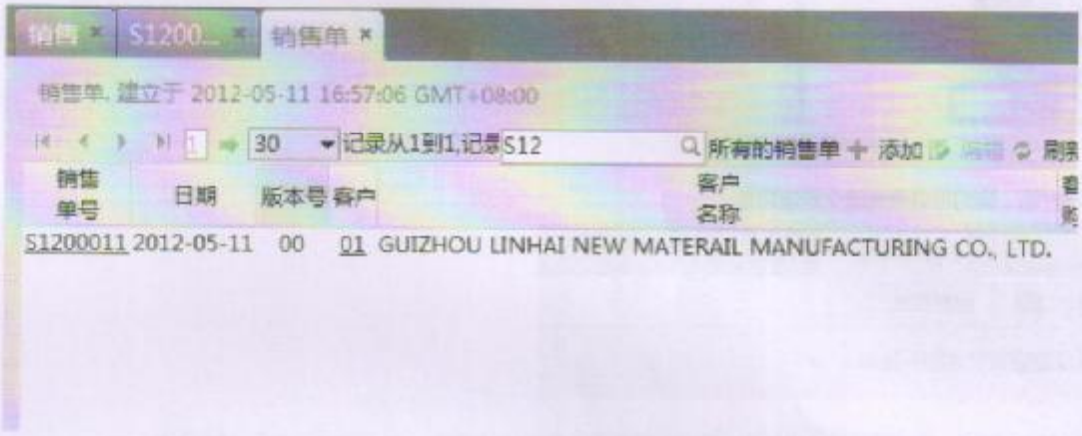
销售单过滤创建后，我们来看一下实际效果。首先，创建两个用户 user1 和 user2，用 user1 登录，并创建一个销售单 S1200010：



然后用 user2 登录，创建销售单 S1200011：



此时，用 user2 来搜索销售单，结果如下图所示，只有一条数据，那就是 S1200011：



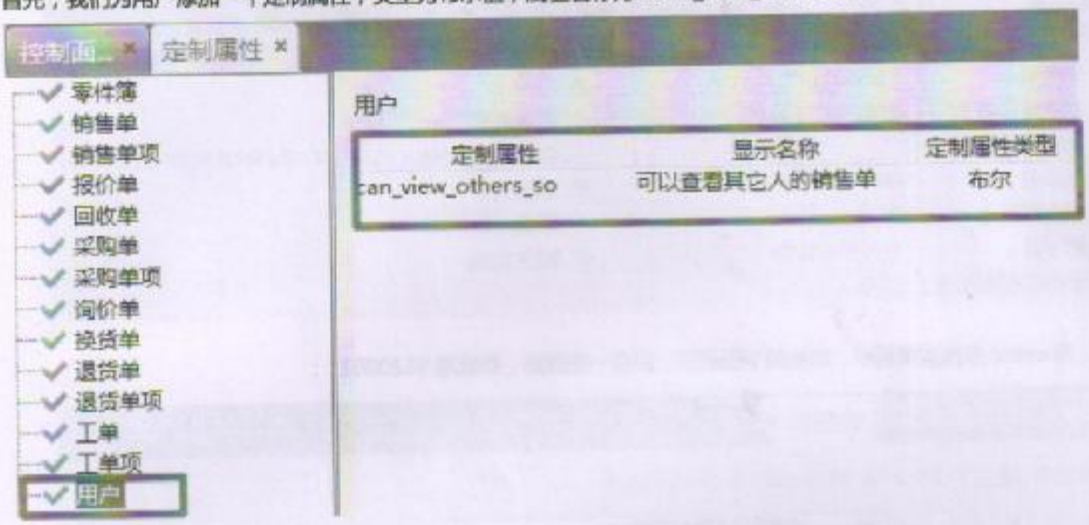
而用 user2 查看 user1 创建的销售单 S1200010，会有如下提示：

8.7.2.3 定制属性+数据过滤实现不同人员不同权限

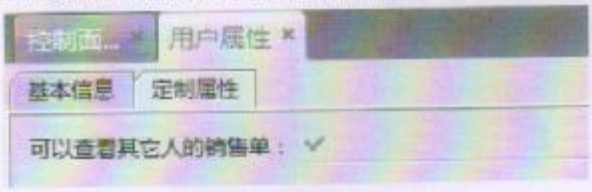
上文讲述的是如何通过数据过滤实现只能查看自己的销售单的功能，也就是限制所有人只能看自己的销售单，这在实际使用中是不合理的，因为老板和管理层人员更需要的是能够看到所有的销售单。下面，我们就来讲述如何通过定制属性的功能实现老板和管理人员可以查看所有销售单。

从本章节的标题就能看出，该功能需要用到“定制属性”，不熟悉该功能的用户可以先阅读 23.9.1 章节的内容。

首先，我们为用户添加一个定制属性，类型为布尔值，属性名称为：can_view_others_so：



在用户界面，我们可以看到这个定制属性：



对于 Admin 用户，我们把这属性设置为“真”，然后进入数据过滤界面，在过滤条件一栏，增加条件：

currentuser().customProperties.can_view_others_so = false：