

# 第15章 树 ( TreeViewer类 ) 和列表 ( ListView类 )

## 15.1 概述

前面介绍过了SWT中Tree组件的使用，和Table有JFace的扩展TableViewer一样，Tree也有一个JFace中的扩展，就是TreeViewer。

因为TreeViewer和TableViewer两者都是继承自同一个父类StructuredViewer（图15.1），所以TreeViewer的使用和TableViewer在很多地方都类似，比如，也用setInput方法输入数据，也有内容器、标签器以及排序器、过滤器。

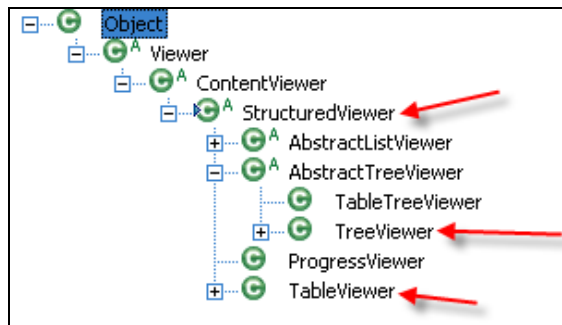


图15.1 TreeViewer的谱系图

## 15.2 前期准备：实例所用数据模型说明

### 15.2.1 建立国家实体和城市实体

为了编写TreeViewer的实例，要改造一下原来用在TableViewer的数据模型，在原有数据模型的基础上再增加两个实体类：国家、城市。国家包含城市，城市包含人。以下给出国家和城市的实体类代码，代码风格与原有的PeopleEntity类似。

```
/**
 * 国家的实体类
 */
public class CountryEntity {
    //-----三个字段-----
    private Long id; //唯一识别码，在数据库里常为自动递增的ID列
    private String name; //国家名
    private List cities; //此国家所包含的的城市的集合，集合元素为City对象
    //-----两个构造函数-----
    public CountryEntity() {}
    public CountryEntity(String name) { this.name = name; }
    //-----以下为字段相应的get/set方法-----
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public void setChildren(List children) {this.cities = children;}
    public List getChildren() {return cities;}
}
```

```

/**
 * 城市的实体类
 */
public class CityEntity {
    //-----三个字段-----
    private Long id; //唯一识别码，在数据库里常为自动递增的ID列
    private String name; //城市名
    private List peoples; //城市中的人，装在一个List集合中
    //-----两个构造函数-----
    public CityEntity() {}
    public CityEntity(String name) {this.name = name;}
    //-----以下为字段相应的get/set方法-----
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) {this.name = name;}
    public void setChildren(List children) {this.peoples = children;}
    public List getChildren() {return peoples;}
}

```

```

/**
 * 为了以后生成对象方便，我们也给PeopleEntity类多加一个构造函数
 */
public PeopleEntity() {}
public PeopleEntity(String name) { this.name = name; }

```

### 15.2.2 建立树结点的接口类

树结点有两个基本特征：名称和子结点。我们将这两个特征抽象出来写成一个接口(interface)，然后将要做树结点的实体类实现此接口。注意：这个接口类不是必须的，是为了今后的操作方便，以及规范化设计才建立此接口。

```

/**
 * 树结点的接口
 */
public interface ITreeEntry { //接口名称一般以大写的I开头，第二个字母也大写
    //设置与得到树结点的名称
    public String getName();
    public void setName(String name);
    //设置与得到子结点集合
    public void setChildren(List children);
    public List getChildren();
}

```

### 15.2.3 让国家、城市、人三个实体类实现此接口

它们的关系如图15.2所示

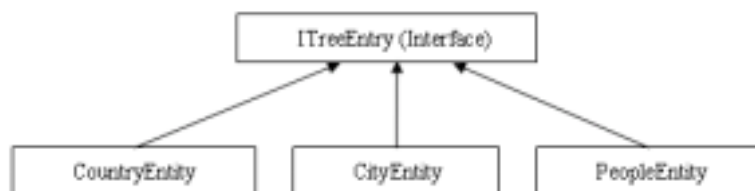


图15.2 实体类与接口的关系图

在实体类的起始处加“implements ITreeEntry”，以表明此类实现于接口ITreeEntry，代码如下：

```
public class CountryEntity implements ITreeEntry{
    .....
}
```

接着就要实现接口定义的setChildren、getChildren方法，不过原来在国家和城市的实体类已经实现了这两个方法（当初在设计这两个实体类时，笔者就考虑到了这一点）。人的实体类还没有这两个方法，但“人”在本例中是不会有子结点的，所以就让它的这两个方法空实现。

```
/**
 * 人实体。由于人不会有子结点，所以这两方法无用，让它空实现
 */
public void setChildren(List children) {}
public List getChildren() {return null;}
```

## 15.3 让数据在树中显示出来

树的效果如图15.3所示：

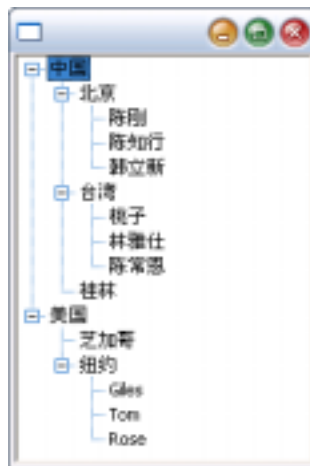


图15.3 树的效果图

### 15.3.1 树的数据结构的创建

在上节15.2，介绍了三个实体类的数据模型，接下来就要象TableViewer里的一样，用实体类来创建树的所有数据对象，并用集合List把国家、城市、人连结起来。和表格TableViewer这章一样，我们建立一个数据对象生成工厂来负责树的数据对象的创建。

```
//-----文件名：DataFactory.java-----
public class DataFactory {
    public static Object createTreeData() {
        /*
         * -----生成数据对象-----
         */
        //生成人的数据对象
        PeopleEntity p1 = new PeopleEntity("陈刚");
        PeopleEntity p2 = new PeopleEntity("陈知行");
        PeopleEntity p3 = new PeopleEntity("韩立新");
        PeopleEntity p4 = new PeopleEntity("桃子");
```

```

PeopleEntity p5 = new PeopleEntity("林雅仕");
PeopleEntity p6 = new PeopleEntity("陈常恩");
PeopleEntity p7 = new PeopleEntity("Giles");
PeopleEntity p8 = new PeopleEntity("Tom");
PeopleEntity p9 = new PeopleEntity("Rose");
//生成城市的数据对象
CityEntity city1 = new CityEntity("北京");
CityEntity city2 = new CityEntity("台湾");
CityEntity city3 = new CityEntity("桂林");
CityEntity city4 = new CityEntity("芝加哥");
CityEntity city5 = new CityEntity("纽约");
//生成国家的数据对象
CountryEntity c1 = new CountryEntity("中国");
CountryEntity c2 = new CountryEntity("美国");
/*
 * -----将数据对象连结起来-----
 */
//-----人和城市的关系-----
{ //陈刚、陈知行、韩立新是在北京的
    ArrayList list = new ArrayList();
    list.add(p1);
    list.add(p2);
    list.add(p3);
    city1.setChildren(list);
}
{ //桃子、林雅仕、陈常恩是在台湾的
    ArrayList list = new ArrayList();
    list.add(p4);
    list.add(p5);
    list.add(p6);
    city2.setChildren(list);
}
{ //Giles、Tom、Rose是在纽约的
    ArrayList list = new ArrayList();
    list.add(p7);
    list.add(p8);
    list.add(p9);
    city5.setChildren(list);
}
//-----城市和国家的关系-----
{ //北京、台湾、桂林属于中国
    ArrayList list = new ArrayList();
    list.add(city1);
    list.add(city2);
    list.add(city3);
    c1.setChildren(list);
}
{ //芝加哥、纽约属于美国
    ArrayList list = new ArrayList();
    list.add(city4);
    list.add(city5);
    c2.setChildren(list);
}
//-----将国家置于一个对象之下这个对象可以是List,也可以是数组-----
{
    ArrayList list = new ArrayList();
    list.add(c1);
    list.add(c2);
}

```

```

        return list;
    }
}

```

### 15.3.2 标签器和内容器的使用

TreeViewer和TableViewer一样,也是用内容器和标签器来控制记录对象的显示,并且使用内容器和标签器的语句也一样。这里先给出TreeViewer的主程序如下:

```

public class TreeViewer1 {
    public static void main(String[] args) {
        TreeViewer1 window = new TreeViewer1();
        window.open();
    }
    public void open() {
        final Display display = new Display();
        final Shell shell = new Shell();
        shell.setSize(200, 300);
        //-----
        shell.setLayout(new FillLayout());
        Composite c = new Composite(shell, SWT.NONE);
        c.setLayout(new FillLayout());
        TreeViewer tv = new TreeViewer(c, SWT.BORDER);
        tv.setContentProvider(new TreeViewerContentProvider());
        tv.setLabelProvider(new TreeViewerLabelProvider());
        //和TableViewer一样,数据的入口也是setInput方法
        Object inputObj = DataFactory.createTreeData();
        tv.setInput(inputObj);
        //-----
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
    }
}

```

### 15.3.3 标签器 (实现I Label Provider接口)

将标签器写成单独的外部类,以便于后面的实例共用,其代码如下:

```

//-----文件名:TreeViewerLabelProvider.java-----
/**
 * 标签提供者。控制记录在树中显示的文字和图像等
 */
public class TreeViewerLabelProvider implements ILabelProvider {
    /**
     * 记录显示的文字。不能返回null值
     */
    public String getText(Object element) {
        ITreeEntry entry = (ITreeEntry) element;
        return entry.getName();
    }
    /**
     * 记录显示的图像,可以返回null值
     */
    public Image getImage(Object element) {
        return null;
    }
}

```

```

    }
    //-----以下方法暂不用，空实现-----
    public void addListener(ILabelProviderListener listener) {}
    public void dispose() {}
    public boolean isLabelProperty(Object e, String p){return false;}
    public void removeListener(ILabelProviderListener listener) {}
}

```

### 15.3.4 内容器（实现ITreeContentProvider接口）

标签器还比较简单，在TreeViewer中最主要、最复杂的是内容器。本例把内容器和标签器一样写成单独的外部类，以便于后面的实例共用，其代码如下：

```

//-----文件名：TreeViewerContentProvider.java-----
/**
 * 内容器。由它决定那些对象记录应该输出在TreeViewer里显示
 */
public class TreeViewerContentProvider implements ITreeContentProvider {
    /**
     * 由这个方法决定树的一级显示那些对象
     * @param inputElement 是用tv.setInput()方法输入的那个对象
     * @return Object[]一个数组，数组中一个元素就是一个结点
     */
    public Object[] getElements(Object inputElement) {
        if (inputElement instanceof List) {
            List list = (List) inputElement;
            return list.toArray();
        } else {
            return new Object[0]; //生成一个空数组
        }
    }
    /**
     * 判断某结点是否有子结点。如果有子结点，这时结点前都有一个“+”号图标
     * @param element 需要判断是否有子的结点
     * @return true有子结点，false无子结点
     */
    public boolean hasChildren(Object element) {
        ITreeEntry entry = (ITreeEntry) element;
        List list = entry.getChildren();
        if (list == null || list.isEmpty()) //判断是否有子结点
            return false;
        else
            return true;
    }
    /**
     * 由这个方法决定父结点应该显示那些子结点。
     * @param parentElement 当前被单击的结点对象
     * @return 由子结点作为元素的数组
     */
    public Object[] getChildren(Object parentElement) {
        ITreeEntry entry = (ITreeEntry) parentElement;
        List list = entry.getChildren();
        if (list == null || list.isEmpty())
            return new Object[0];
        else
            return list.toArray();
    }
}
//-----以下方法无用，空实现-----

```

```

public Object getParent(Object element) {return null;}
public void dispose() {}
public void inputChanged(Viewer v, Object oldInput, Object newInput) {}
}

```

程序说明：

在容器中最关键的是getElements、hasChildren、getChildren这三个方法。

- getElements只在显示“一级”结点时有用。
- hasChildren主要用于判断当前所显示的结点是否有子结点，如果有子结点则前面显示一个“+”号图标，而有“+”号的结点则可以单击展开其下一级的子结点。
- 当单击有“子”的结点时，会执行getChildren方法。当然展开子结点后，少不了又执行一遍子结点的hasChildren方法，以决定其各子结点前是否显示“+”号图标。

图15.4给出了内容器在启动、单击结点、关闭窗口这三种情况时的方法执行的时序图：

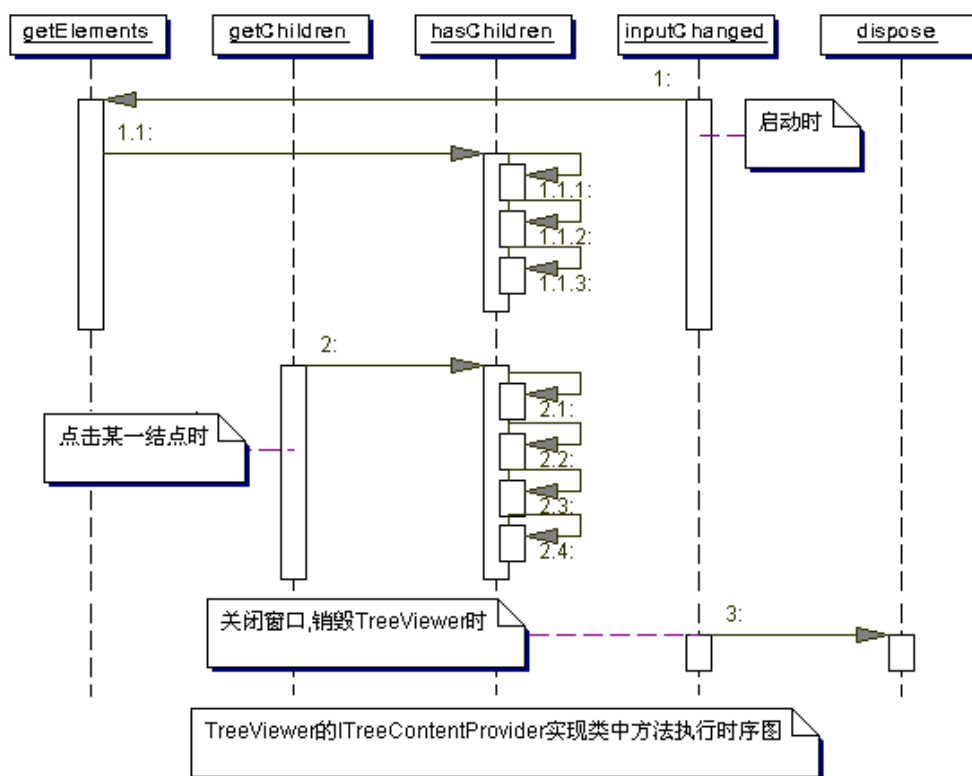


图15.4 内容器的方法的时序图

以本实例来解释此图如下：

- 启动时。先执行一次inputChanged方法，此方法为空实现，跳过。再执行一次“Object[] getElements(Object inputElement)”方法，参数inputElement实际上就是setInput的入口参数的值，也即DataFactory.createTreeData()的返回值：一个包含“中国、美国”两个实体对象的List。此List在getElements中被转化成数组返回，因此界面的一级结点显示的是“中国、美国”两个结点。接下来连续执行两次“boolean hasChildren(Object element)”方法，element参数分别传入“中国、美国”两个结点对象并进行判断其是否有子结点，毫无疑问，这两个结点都有子结点，因此返回True，界面上都显示有“+”图标。
- 单击某一结点时。比如单击“中国”结点，会先执行一次“Object[] getChildren(Object parentElement)”方法，parentElement参数传入的就是被单击的“中国”结点对象，

然后getChildren方法中的程序把“中国”下的子结点取出并转化成数组返回，此数组包含三个结点对象“北京、台湾、桂林”。接下来连续执行三次hasChildren方法，传入的参数分别为“北京、台湾、桂林”这三个结点对象，并判断它们是否有子，如有，则在结点前显示一个“+”图标。

- 关闭窗口。先后会执行inputChanged、dispose方法。如果有需要，可以在dispose方法中将一些树所要用到无法自动销毁的资源dispose掉，比如图像Image、字体Font、颜色Color等。

## 15.4 给树加上右键菜单及取得结点的值

本实例的效果如图15.5所示：



图15.5 右键菜单效果图

在树上加右键菜单和在TableViewer上加右键菜单的方法是一样的，也是要用到Action和ActionGroup以及MenuManager类，当然程序要根据树的特点做稍微改动。

1、实例的主程序代码如下：

```
shell.setLayout(new FillLayout());
Composite c = new Composite(shell, SWT.NONE);
c.setLayout(new FillLayout());
TreeView tv = new TreeView(c, SWT.BORDER);
tv.setContentProvider(new TreeViewContentProvider());
tv.setLabelProvider(new TreeViewLabelProvider());
Object inputObj = DataFactory.createTreeData();
//-----加入代码：START-----
//生成一个ActionGroup对象
MyActionGroup actionGroup = new MyActionGroup(tv);
//调用fillContextMenu方法将按钮注入到菜单对象中
actionGroup.fillContextMenu(new MenuManager());
//-----加入代码：END-----
tv.setInput(inputObj);
```

2、ActionGroup类的代码如下

```
//-----文件名：MyActionGroup.java-----
public class MyActionGroup extends ActionGroup {
    private TreeView tv;
    public MyActionGroup(TreeView treeViewer) {
        this.tv = treeViewer;
    }
    /**
     * 生成菜单Menu，并将两个Action传入
```

```

    */
    public void fillContextMenu(IMenuManager mgr) {
        /*
         * 加入两个Action对象到菜单管理器
         */
        MenuManager menuManager = (MenuManager) mgr;
        menuManager.add(new OpenAction());
        menuManager.add(new RefreshAction());
        /*
         * 生成Menu并挂在树Tree上
         */
        Tree tree = tv.getTree();
        Menu menu = menuManager.createContextMenu(tree);
        tree.setMenu(menu);
    }
    /**
     * “打开”的Action类
     */
    private class OpenAction extends Action {
        public OpenAction() {
            setText("打开");
        }
        /**
         * 继承自Action的方法，动作代码写此方法中
         */
        public void run() {
            IStructuredSelection selection = (IStructuredSelection)
tv.getSelection();
            ITreeEntry obj = (ITreeEntry) (selection.getFirstElement());
            if (obj != null)
                MessageDialog.openInformation(null, null, obj.getName());
        }
    }
    /**
     * 刷新的Action类
     */
    private class RefreshAction extends Action {
        public RefreshAction() {
            setText("刷新");
        }
        public void run() {
            tv.refresh();//调用TreeViewer的刷新方法
        }
    }
}
}

```

## 15.5 树结点的展开、收缩、新增、删除、修改

展开结点用expandToLevel或expandAll方法；收缩结点用collapseToLevel或collapseAll方法；新增结点用add方法；删除结点用remove方法；修改结点只需要取出结点对象，做相应修改后，用refresh方法将结点刷新一下即可。

本实例的效果如图15.6所示：

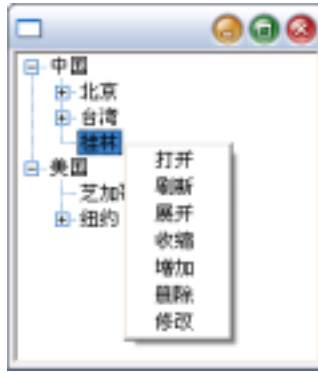


图15.6 实例效果图

这几个功能的增加只需要修改MyActionGroup类，增加相应的Action即可，代码如下：

```
//-----文件名：MyActionGroup.java-----
public class MyActionGroup extends ActionGroup {
    private TreeView tv;
    public MyActionGroup(TreeView treeViewer) {
        this.tv = treeViewer;
    }
    /**
     * 生成菜单Menu，并将两个Action传入
     */
    public void fillContextMenu(IMenuManager mgr) {
        /*
         * 加入两个Action对象到菜单管理器
         */
        MenuManager menuManager = (MenuManager) mgr; //类型转换
        menuManager.add(new OpenAction());
        menuManager.add(new RefreshAction());
        menuManager.add(new ExpandAction());
        menuManager.add(new CollapseAction());
        menuManager.add(new AddEntryAction());
        menuManager.add(new RemoveEntryAction());
        menuManager.add(new ModifyEntryAction());
        /*
         * 生成Menu并挂在树Tree上
         */
        Tree tree = tv.getTree();
        Menu menu = menuManager.createContextMenu(tree);
        tree.setMenu(menu);
    }
    /**
     * “打开”的Action类
     */
    private class OpenAction extends Action {
        public OpenAction() {
            setText("打开");
        }
    }
    /**
     * 继承自Action的方法，动作代码写此方法中
     */
    public void run() {
        ITreeEntry obj = getSelTreeEntry();
        if (obj != null)
            MessageDialog.openInformation(null, null, obj.getName());
    }
}
```

```

    }
}
/**
 * 刷新的Action类
 */
private class RefreshAction extends Action {
    public RefreshAction() {
        setText("刷新");
    }
    public void run() {
        tv.refresh();//调用TreeViewer的刷新方法
    }
}
/**
 * 展开当前结点的Action类
 */
private class ExpandAction extends Action {
    public ExpandAction() {
        setText("展开");
    }
    public void run() {
        ITreeEntry obj = getSelTreeEntry();
        if (obj != null)
            tv.expandToLevel(obj, 1); //只展开一层，设值可以超过实际树的层数
    }
}
/**
 * 收缩当前结点的Action类
 */
private class CollapseAction extends Action {
    public CollapseAction() {
        setText("收缩");
    }
    public void run() {
        ITreeEntry obj = getSelTreeEntry();
        if (obj != null)
            tv.collapseToLevel(obj, -1); //-1为将当前结点的所有子结点收缩
    }
}
/**
 * 给当前结点增加一个子结点的Action类
 */
private class AddEntryAction extends Action {
    public AddEntryAction() {
        setText("增加");
    }
    public void run() {
        ITreeEntry obj = getSelTreeEntry();
        if (obj == null || obj instanceof PeopleEntity)
            return;
        InputDialog dialog = new InputDialog(null, "给当前结点增加一个子结
点", "输入名称", "abc", null);
        if (dialog.open() == InputDialog.OK) { //如果单击OK按钮
            String entryName = dialog.getValue(); //得到Dialog输入值
            /* 根据单击结点的不同类型生成相应的子结点*/
            ITreeEntry newEntry = null;
            if (obj instanceof CountryEntity)
                newEntry = new CityEntity(entryName);

```

```

        else if (obj instanceof CityEntity)
            newEntry = new PeopleEntity(entryName);
        /*在增加子结点之前将父结点展开*/
        if (!tv.getExpandedState(obj))
            tv.expandToLevel(obj, 1);
        tv.add(obj, newEntry); //增加结点
    }
}
}
/**
 * 删除结点的Action类
 */
private class RemoveEntryAction extends Action {
    public RemoveEntryAction() {
        setText("删除");
    }
    public void run() {
        ITreeEntry obj = getSelTreeEntry();
        if (obj == null)
            return;
        tv.remove(obj);
    }
}
/**
 * 修改结点名称的Action类
 */
private class ModifyEntryAction extends Action {
    public ModifyEntryAction() {
        setText("修改");
    }
    public void run() {
        ITreeEntry obj = getSelTreeEntry();
        if (obj == null)
            return;
        InputDialog dialog = new InputDialog(null, "修改结点", "输入新名称", obj.getName(), null);
        if (dialog.open() == InputDialog.OK) {
            String entryName = dialog.getValue();
            obj.setName(entryName);
            tv.refresh(obj); //刷新结点
        }
    }
}
}
/**
 * 自定义方法：取得当前选择的结点
 */
private ITreeEntry getSelTreeEntry() {
    IStructuredSelection selection = (IStructuredSelection) tv
        .getSelection();
    ITreeEntry entry = (ITreeEntry) (selection.getFirstElement());
    return entry;
}
}
}

```

程序说明：

- 为了复用代码，将取得当前选择结点的几条语句写成了一个方法。
- 在对结点对象操作时，要记得进行空值判断，因为树的大部份方法的参数都不支持空值，会报异常并且中断程序。

## 15.6 ListViewer类

### 15.6.1 ListViewer简介

在前面介绍过TableViewer和TreeView之后，ListViewer就没有什么新鲜的知识了。如果把TableViewer设置成一列，就成了ListViewer的样子，当然ListViewer类并非由TableViewer类简化而来，它是基于SWT的List组件的一个扩展，其谱系如图15.7所示：

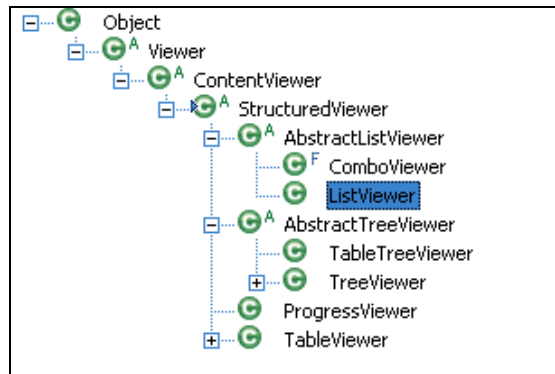


图15.7 ListViewer的谱系图

### 15.6.2 ListViewer的实例

ListViewer和TableViewer、TreeView一样也有自己的内容器和标签器。其内容器和TableViewer是一样的，甚至可以不用改动的挪用TableViewer的内容器，以及TreeView的标签器。在本实例输入setInput方法的数据是用TreeView一章的数据，但只借用了两个国家对象。实例的具体代码如下：

```
shell.setLayout(new FillLayout());
ListViewer lv = new ListViewer(shell, SWT.BORDER);
lv.setContentProvider(new TableViewerContentProvider());
lv.setLabelProvider(new TreeViewerLabelProvider());
lv.setInput(DataFactory.createTreeData());
```

实例运行的效果如图15.8所示：

可以用add方法往列表中加入新项。比如，在setInput方法后新增如下两句：

```
lv.add(new CountryEntity("英国"));
lv.add(new CountryEntity("法国"));
```

则运行效果如图15.9所示：



图15.8 ListViewer的效果图



图15.9 增加两项的ListViewer效果图

### 15.6.3 ListViewer常用方法

ListViewer常用方法，如下表15.1所示

表15.1 ListView常用的方法

void add(Object element)	添加一项
void add(Object[] elements)	通过数组加入多项
Object getElementAt(int index)	返回序号为index的项（是一个对象）
List getList()	ListViewer是List的扩展，此方法可得到内含的List对象
void remove(Object element)	从ListViewer移去一项
void remove(Object[] elements)	从ListViewer移去数组中的几项

## 15.7 本章小结

TreeViewer的使用方式与TableViewer有很多地方都是非常相似的，TreeViewer也有自己的过滤器和排序器，具体使用方式请参照TableViewer。TreeViewer中比较复杂的内容器，熟悉内容器是掌握TreeViewer的要点。

ListViewer也有过滤器和排序器，它们的使用方法和TableViewer是一样的，本章没有再详细介绍。笔者在实际开发中很少用到ListViewer，简单的开发需求，用SWT的List就够了。对于复杂的开发需求，把TableViewer设置成一行也能满足，而且还便于扩展成多列。如果开发需求很明确：就只要一个列表，而且数据操作比较复杂、数据量比较多，这时就比较适合用ListViewer。