文章编号：2096-1472(2017)-09-59-04

基于WPF的图书阅读器的设计与实现

邹 珺
(苏州农业职业技术学院，江苏 苏州 215008)

摘 要：随着互联网的兴起，各种图书阅读器也应运而生，它们能提供类似纸张阅读感受的功能。为了让用户方便、快捷地管理指定文件夹下的图书，图书阅读器能实现图书阅读，并可以读取压缩包中的图片文件，支持一些特定的图书格式。本文主要描述使用WPF这个UI端技术来生成一个图书阅读器，包括系统架构、系统核心类的实现，特别是在WPF中使用了多线程技术和事件路由技术，让用户能够灵活自如地使用该工具进行图书阅读。

关键词：图书；阅读器；WPF
中图分类号：TP312 **文献标识码**：A

Design and Implementation of the Book Reader Based on WPF

ZOU Jun
(*Suzhou Agricultural Vocational College*,*Suzhou* 215008,*China*)

Abstract:A variety of book readers have come into being with the rise of the Internet,providing similar experience as paper reading.In order to allow users to manage the books in the specified folder conveniently and quickly,the book reader is designed to implement book reading,read picture files in a compressed file and supports some specific book formats.This paper mainly describes the book reader generated by using the UI technology of WPF,including the system architecture and the implementation of the system kernel class.Additionally,the multi-threading technology and the event routing technology are applied in WPF,which enable users to read books through this tool with ease.

Keywords:book;reader;WPF

1 引言(Introduction)

图书阅读器能提供以下功能：

(1)管理指定文件夹下的图书，在图书封面区中显示图书的封面。

(2)阅读书时，可以指定书签，并可以跳到指定的书籍。

(3)保存所有的状态，以便在下次继续读取书籍。

(4)提供对于压缩的内容访问，实现图像缓存管理。

(5)一旦一本书被打开，显示里面的页面和文件夹的结构。

本文介绍使用WPF这个UI端技术开发图书阅读器，该阅读器可以读取压缩包中的图片文件，支持一些特定的图书格式[1]。

2 图书阅读器系统架构(Book reader system architecture)

图书阅读器仅包含一个WPF应用程序项目，由一个主窗体和多个用户控件组成。

在这个系统中出现的实体有图书目录、图书列表、图书、压缩格式的图书、图像缓存等。找出这些实体后，进行面向对象的抽象，找出一些有共性的实现未抽象基类或接口，以便于应对变化，而一些未变的可以直接定义为类，分析如下：

(1)文件夹可以直接定义为一个类。因为该对象相对固定，不同的文件夹除了名称和位置不一样之外，还可能会有一些其他变化的特性。

(2)每个文件夹包含多部书。因为图书的类型不是固定的，比如有压缩文件类型的图书和有其他格式的图书等，需要抽象出来实现为一个接口。

(3)每本图书包含多个页面。因为每个页面的格式是不同的，因此也需要进行抽象。

(4)每本图书会包含一个图像缓存，该缓存提供的功能相对固定，当然也可以进行进一步抽象，不过在本示例中将实现为一个单一的类，以求简化[2]。

经过上述分析，可以得出如图1所示的类结构图

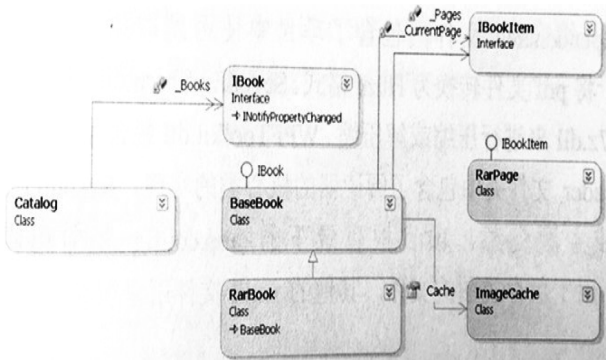

图1 类结构图

Fig.1 Structure chart of class

Catalog代表一个文件夹类，它包含代表该目录下所有图书的ObservableCollection<IBook>泛型集合类。IBook是抽象出来的代表一部图书的接口，它实现了INotifyPropertyChanged以便实现UI级别的绑定。BaseBook是一个实现了IBook接口的基类，提供了对于每本图书的基本实现，RarBook通过派生自BaseBook类，实现了压缩格式的图书对象。IBookItem接口是代表图书书页的接口，IBook接口包含一个类型为List<IBookItem>泛型集合，来表示一本书的所有图书页。RarPage实现了IBookItem接口，提供了对于RarBook类型图书的书页实现。ImageCache是每本书包含的图像缓存信息的对象[3]。

3 系统核心类的实现(Implementation of system kernel class)

3.1 实现图书目录Catalog类

Catalog类定义了三个属性，分别用于指定文件路径、用于保存图书的列表及一个布尔值获取和设定图书变更信息，属性定义代码如下：

```
private string_bookPath=string.Empty;
//文件路径
public string BookPath                    //文件路径属性
{
get { return _bookPath;}                  //返回值
set { _bookPath=value;}                   //设置值
}
private ObservableCollection<IBook>_Books=new
ObservableCollection<IBook>();            //图书集合
public ObservableCollection<IBook>Books
//图书集合属性
{
get { return _Books;}                     //返回图书列表
set { _Books=value;}                      //设置图书列表
}
private bool _IsChanged=false;            //是否变更
public bool IsChanged                     //是否变更属性
{
get { return _IsChanged;}                 //返回变更
set { _IsChanged=value;}                  //设置变更
}
```

3.2 定义图书接口IBook

IBook接口被Catelog引用，使用这种基于接口的方式可以实现程序间的解耦，使程序具有良好的可扩充性。IBook接口定义了一本书需要具备的基本契约，实现代码如下：

```
internal interface IBook:INotifyPropertyChanged
{
string FileName {get;}                    //文件名称
int NbPages {get;set;}                    //图书页数
long Size {get;set;}                      //文件大小
bool IsRead {get;set;}                    //是否阅读
string BookMark {get;set;}                //书签
BitmapImage Cover {get;set;}              //封面
IBookItem CurrentPage {get;set;}          //书页
string FilePath {get;set;}                //文件路径
List<IBookItem> Pages {get;set;}          //书页集合
ImageCache Cache {get;set;}               //图像缓存
BitmapImage GetCurrentPageImage();        //当前书页
void GotoMark();                          //定位书签
bool GotoNextPage();                      //转到下一页
bool GotoPage(IBookItem page);            //定位到指定页
bool GotoPreviousPage();                  //转到上一页
void Load();                              //加载图书
void ManageCache();                       //管理缓存
void SetMark();                           //设置书签
void UnLoad();                            //卸载图书
}
```

IBook接口定义了一本书的基本属性和方法，该接口派生自INotifyPropertyChanged接口，当图书的属性发生改变时，可以向UI触发属性变更通知[4]。

3.3 图书基类BaseBook

BaseBook实现了IBook接口，同时也要实现INotifyPropertyChanged接口的成员，BaseBook内部包含ImageCache实现图像缓存。BaseBook的Pages包含实现了IBookItem接口的对象集合，CurrentPage用于显示当前的图书页面[5]。

BaseBook定义了八个属性，这些属性除了Pages是一个包含多个图书页面的泛型集合外，其他的都来自IBook接口的实现。该类重载了构造函数，提供了一个接收文

件路径的构造函数，当文件路径发生改变时，会触发在INotifyPropertyChanged接口中定义的变更通知，构造函数代码如下：

```
public BaseBook(string filePath)
{
  _filePath=filePath;                    //得到文件路径
   RaisePropetyChanged("FilePath");      //触发文件路径变更通知
   RaisePropetyChanged("FileName");      //触发文件变更通知
}
```

3.4 实现Rar压缩文件格式的图书

该类引用SevenZip类库，并从BaseBook类中派生。由于SevenZipLib依赖于7z.dll这个类库，因此在RarBook的构造函数中，要先设置7z.dll类库的路径给SevenZipLib。RarBook类的构造函数代码如下：

```
public RarBook(string filePath,bool makeCover):base(filePath)//调用基类构造函数
{
  try
  {
    if(!_IsInitDone)
    {
      _IsInitDone=true;              //压缩库初始化参数
    string sevenZip=Assembly.GetExecutingAssembly().Location.Replace("BookReader.exe","Dependencies\\7z.dll ");          //获取7z.dll的路径
      SevenZipExtractor.SetLibraryPath(sevenZip);//设置依赖的7z.dll的路径
    }
  }
   catch(SevenZipLibraryException err)//如果触发异常
  {
    //显示异常信息
   ExceptionManagement.Manage("RarBook:RarBook",err);
  }
 if(makeCover)                    //如果要创建封面
GenerateCover();                  //生成图书封面
}
```

在代码中，RarBook的构造函数需要传递两个参数：一个用来表示图书的路径，另一个布尔值用来确定是否需要为压缩文件创建一个封面。首先调用基类的构造函数，然后获取7z.dll的路径，调用SevenZipExtrator类的静态方法SetLibraryPath()为SevenZip指定库路径。如果需要为图书创建封面的话，代码将调用GenerateCover生成图书封面[6]。

3.5 图书页面接口IBookItem的定义

图书页面类是包含在每一本图书中的页面的集合，因为BookReader将使用基于文件的页面，比如压缩包中的图片文件，那么图书页面类需要具有文件路径和文件名称属性。IBookItem提供了对于页面类的基本定义，代码如下：

```
internal interface IBookItem        //图书页面接口
{
string FilePath { get;}             //文件路径
string FileName { get;}             //文件名称
}
```

3.6 实现缓存管理核心类

ImageCache类是整个缓存功能的核心，该类的内部包含一个嵌套类ImgInfo用来保存图像信息。与多数缓存功能的实现一样，ImageCache在内部实际上也就是使用了Lise<ImgInfo>对象在内存中保存图像信息。因为过多的图像保存会占用系统太多的内存，所以ImageCache提供了一些机制来实现缓存数据的新增、修改和移除工作。ImgInfo类的定义和ImageCache类的构造函数代码如下：

```
internal class ImageCache
{
  internal class ImgInfo            //嵌套类ImgInfo用于保存图像缓存信息
  {
    public ImgInfo(BitmapImage image,object tag)//构造函数
    {
      _Image=image;                 //缓存的图像对象
       _Tag=tag;                    //缓存标签
    }
    public object _Tag=null;        //标签字段
  public BitmapImage _Image=null;   //图像字段
    public DateTime _LastAcces=DateTime.Now;//最后访问时间
  }
  public ImageCache()
  {
    _ImagesCache=new List<ImgInfo>();//实例化一个新的List<ImgInfo>泛型集合
  }
  private long _TotalSize;          //总尺寸
  private List<ImgInfo> _ImagesCache;   //保存对
```

象的私有字段

}

4 WPF关键技术(Key technology of WPF)

4.1 在WPF中使用多线程

WPF与Windows Forms一样，UI元素只能由创建该元素的线程来访问。此时需要借助于WPF中提供的全新的Dispatcher类，该类提供了BeginInvoke()方法。BeginInvoke是异步调用的方法，在示例中大多数都使用了同步的Invoke()方法，该方法直到UI线程实际执行完该委托后才返回。BeginInvoke是异步的，将立即返回。

Dispatcher按优先级对其队列中的元素进行排序。向Dispatcher队列中添加元素时可指定10个级别。这些优先级在DispatcherPriority枚举中维护，BookReader中在后台线程中显示异常信息的方法，该方法使用了Invoke()方法进行同步调用，代码如下：

```
catch(Exception err)                    //如果产生异常
{
  //在与UI相同的线程中调用异常显示窗口
    Application.Current.Dispatcher.Invoke(DispatcherPriority.Normal,
    (ThreadStart)delegate
    {
      //使用自定义的ExceptionManagement类
      ExceptionManagement.Manage("Catalog:LoadCovers",err);
    });
}
```

上述代码的Invoke调用中，首先使用DispatcherPriority枚举指定优先级别，然后使用了一个匿名委托来调用ExceptionManagement类的Manage()静态方法，该匿名委托要符合ThreadStart委托的方法签名[7]。

4.2 WPF中的事件路由技术

路由事件的定义是由公共的静态RoutedEvent成员加一个约定的Event后缀组成，路由事件需要在.NET事件系统中进行注册。然后路由事件也有一个和普通的.NET事件一样的事件定义，或者是一个事件包装器，使得可以像使用普通事件那样使用路由事件，也可以在XAML中使用事件特性语法添加事件。为WPF定义一个路由事件代码如下：

```
public static readonly RoutedEvent ZoomChangedEvent=EventManager.
RegisterRoutedEvent("ZoomChangedEvent",RoutingStrategy.Bubble,typeof(ZoomChangedEventHandler),typeof(PageViewer));
public delegate void ZoomChangedEventHandler(object sender,ZoomRoutedEventArgs e)
public event ZoomChangedEventHandler ZoomChanged
{
  add{AddHandler(ZoomChangedEvent,value);}
  remove{RemoveHandler(ZoomChangedEvent,value);}
}
```

在代码中，定义了一个ZoomChangedEventHandler类型的委托，首先调用定义一个名为ZoomChangedEvent的RoutedEvent，通过调用EventManager.RegisterRoutedEvent()方法向WPF的事件系统注册路由事件[8]。

5 结论(Conclusion)

本文介绍了使用WPF技术开发的图书阅读器，为了实现阅读逻辑，使用面向对象的设计方式设计了多个类，以处理文件的打开和阅读工作，对系统架构、系统核心类的实现，以及涉及的关键技术作了阐述，从中体现了WPF技术的强大功能。

参考文献(References)

[1] Yang L,et al.A bi-direction authentication protocol for RFID based on the variable update in IOT[J].Proceedings of the 2nd International Conference on Computer and Applications ASTL,2013(02):82-83.

[2] Xie L,et al.Continuous scanning with mobile reader in RFID systems:an experimental study[J].Proceedings of the Fourteenth ACM International Symposium on Mobile Ad Hoc Networking and Com putting,2009(08):167-168.

[3] Kazuya Sakai,et al.Wei-Shinn Ku,Roger Zimmermann,Min-Te Sun.Dynamic Bit Encoding for Privacy Protection against Correlation Attacks in RFID Backward Channel[J].IEEE Transactions on Computers,2013(04):212-213.

[4] 孙广霞,张秀兰.电子阅读器在图书馆的推广策略研究[J].图书馆学研究,2013(02):77-78.

[5] 赵慧真.电子阅读器盛行引发图书馆服务工作的变革[J].四川图书馆学报,2013(05):121-122.

[6] 胡昌文,唐振贵,陈金菊.图书馆电子阅读器内容推送模式的探讨[J].数字图书馆论坛,2016(10):32-33.

[7] 刘颂莉.电子阅读器在图书馆的应用探讨[J].科技视界,2012(22):55-56.

[8] 金红亚,周德明.电子阅读器应用与图书馆借阅业务的变革[J].图书馆杂志,2010(04):135-136.

作者简介：

邹 珺(1981-)，女，硕士，讲师.研究领域：软件开发.