Web 报表生成器的设计与实现

学　　院　计算机科学与技术

专　　业　计算机科学与技术

年　　级　2007 级

姓　　名　王银鹤

指导教师　张坤龙

2011 年　6 月　15 日

摘　　要

报表作为一种重要的信息载体在信息系统中有着重要的作用，而传统的报表系统主要基于 C/S 模式，存在开发周期长及扩展性差等缺点，不能满足用户需求，因此研究基于 Web 的报表系统具有重要的现实意义。

传统 Web 报表系统往往是先在单机版中设计好报表模板，然后再发布到 Web 中。而对于复杂打印，往往采用安全性不高的 ActiveX 方式。论文详细考察了传统 Web 报表系统的不足，在.NET 移动编码的基础上，设计并实现了一个有强大打印功能的 Web 报表生成器，能让用户在浏览器中进行所见即所得的可视化报表设计，而无须在客户端单独安装程序，这就提高了 Web 报表设计的灵活性。系统实现了多种报表格式自动调整功能，极大的提高了用户报表设计的效率。同时 Web 报表生成器依靠.NET 移动编码方式，在.NET 框架安全机制的控制下能进行复杂的报表打印操作，这比 ActiveX 方式有更好的安全保障。报表打印功能支持自动分页、用户自定义打印等。

关键词：web 报表生成器；移动编码；.NET；可视化报表设计

ABSTRACT

Report, as a significant information carrier, plays an important role in enterprise information system. However, traditional reporting system is based on C/S model, which has some weak points such as long period of development cycle and poor expansibility. Therefore, research on web reporting system is of great practical value.

Traditional Web reporting system often designs the report templates on single computer, then publishes the report templates to the Web. For complex print, traditional Web reporting system often uses the ActiveX which is not so safe. The paper examines the lack of traditional Web reporting system in detail, designs and implements a powerful Web report generator which has the capabilities of complex printing based on the .NET mobile coding, the reporting system supports the WYSIWYG visual report design, without a separate client installation program, improving the flexibility of the Web report. The system achieve a variety of report formats adjust automatically feature, which greatly improves the efficiency of the user's report design. At the same time, the system relies on .NET mobile encoding, can do complex reports print operation under the control of .NET Framework security mechanism, which means a better security than the ActiveX. Report printing feature supports automatic paging and custom printing.

Key words： web report generator ； mobile code； .NET； visual report design

目　　录

第一章　　绪论

1.1　开发背景

通常在开发 B/S 结构的应用程序时，最头疼的问题往往就是报表的设计和打印。理想的报表设计器是要能支持用户拖放、拉伸控件等可视化的设计操作。如果仅仅依靠浏览器对 HTML 标签的解释来实现这样的功能，开发的难度会很大。这时可选的方案有 ActiveX 技术，但 ActiveX 主要缺点就是需要客户端进行注册，而且不能得到很好的安全保障。基于这样的原因，传统的 Web 报表系统一般都包括一个 C/S 的设计器和一个运行引擎。这个设计器通常在 Windows 下运行，引擎则运行在服务器上。开发者在设计器中完成报表的设计，保存为配置文件。当一个报表被调用时，先读配置文件，然后读相应的数据，再按一定的格式生成报表，最后返回给浏览器。可以看出，在这样的 Web 报表系统中，设计器和运行引擎相互间是独立，浏览器在报表系统中主要只负责报表的展示，如果想在远程客户端进行报表的设计，则不得不在客户端安装一个设计器。

对报表的打印操作在报表系统中是一个很重要的功能。Web 报表系统由于只能采用浏览器来作为用户界面进行交互，所以不能精确控制客户端的打印机。而报表系统常常需要完成非常复杂的报表打印任务。仅仅靠 IE 自带的页面打印功能一般不能满足需要。为了能够精确控制打印格式，不能采用 Web 浏览器页面打印的方式进行报表打印工作，只能采取自编程控制客户端的打印工作。

本次课题最终的目的就是要实现一个这样的 Web 报表生成器：支持在浏览器中进行所见即所得的可视化报表设计；能完成复杂的报表打印任务；在远程客户端进行报表设计时不需要安装程序。为了达到这样的目的，最终选择了网页中嵌入 WinForm 的移动编码方式。

1.2　Web 报表生成器关键技术分析

1.2.1　技术选择

.NET Framework 的 Windows Forms 有一个优点就是，在与 IE6 及以上版本一起使用时，IE 在使用了 HTML 的<object>标签的 Web 页中支持控件的下载和显示。这意味着在浏览器中可以利用 Windows Forms 和.NET 环境提供的所有优点，自然也包括 WinForm 中强大的打印功能和绘图功能。这样就能够在浏览器中以 WinForm 的方式实现一个报表设计器了。同时依赖.NET Framework 的类库就能实现很强大的打印控制功能[9]。

.NET 平台以类似于 Java 小程序的方式在 Web 页中提供 Windows Forms 的功能，但主要有以下的优点：

1)　改善了 Java 小程序的性能。

2) 给 Web 环境带来丰富的 Windows 应用程序。
3) 在客户端自动地高速缓存即时编译的代码——客户端只下载有变化的程序。

在浏览器端，Web 报表生成器使用 Windows Forms 与使用 Asp.NET 页相比，有这样的优点：由于到服务器所需要完成的往返工作减少，因而应用程序的相应速度大大提高。另外由于大量的处理在客户端完成，从而减轻了服务器的负担

1.2.2 移动编码概述

微软把通过网络从远程服务器传输到本地计算机，然后在本地执行的软件定义为移动编码，上文中讨论的在网页中嵌入 WinForm 的技术指的就是移动编码。移动编码与传统编码的主要区别在于用户不需要显示地安装和执行移动编码。

Microsoft 的中间语言（MSIL）是与 CPU 无关的指令集，能够十分有效地转换成本地代码。C#、VB.NET 和许多其他的语言先编译成 MSIL，然后 MSIL 再通过 CLR 转换为本地机器码[6]。实现移动编码时，先把 WinForm 源程序编译到 DLL 文件的程序集里，然后在 Web 页上使用 HTML 的<object>标签来引用服务器上的该程序集。IE 将自动地下载 MSIL，然后在托管环境里高速缓存、编译和运行该 MSIL [7]。

移动编码不需要修改注册表，.NET 对此还设置了很多限制，在指定的时间里，移动编码只能驻留在指定量的高速缓存中。当到达限制量时，.NET Framework 将自动删除最近使用最少的程序集。

.NET 下的移动编码支持增量下载模型，不需要下载整个应用程序才开始运行，如果应用程序中的某些部分从不被运行，则这部分程序就不会被下载。

1.2.3 安全性

网页中嵌入 WinForm 相对于 ActiveX 的一大优点就是有更好的安全保障，这里托管环境起到了十分重要的作用，它提供了安全地运行代码所需要的很多安全性服务，代码在.NET Common Language Runtime 的控制之下才能运行，这能有效地防止各种各样的恶意代码[9]。

1.3 论文章节安排

本文从 Web 报表生成器的背景出发逐步讨论 Web 报表生成器的设计与实现

第二章叙述 Web 报表生成器需要实现的功能，重点阐述 Web 报表生成器最后是个怎样的系统。

第三章进一步叙述 Web 报表生成器的设计方案，重点介绍一些系统的基础知识。

第四章在第三章的基础上进一步叙述整个报表生成器的具体实现。

第五章是对已完成工作的总结和对未来工作的展望。

第二章　　Web 报表生成器的需求分析

本章重点介绍 Web 报表生成器所实现的功能，从用例分析、功能分析和使用场景分析三方面入手，对整个系统的功能做概括性的介绍，使读者对 Web 报表生成器有一个初步的认识。

2.1　用例分析

用例是外部可见的一个系统功能单元，可以被描述为参与者与系统之间的一次交互作用。用例视图是被称为参与者的外部用户所能观察到的系统功能的模型图。用例模型的用途是列出系统中的用例和参与者，并显示哪个参与者参与了哪个用例的执行。通过用例，用户可以看到系统提供的功能，先确定系统范围再深入开展项目工作。Web 报表生成器的用例分析如图 2-1 所示。

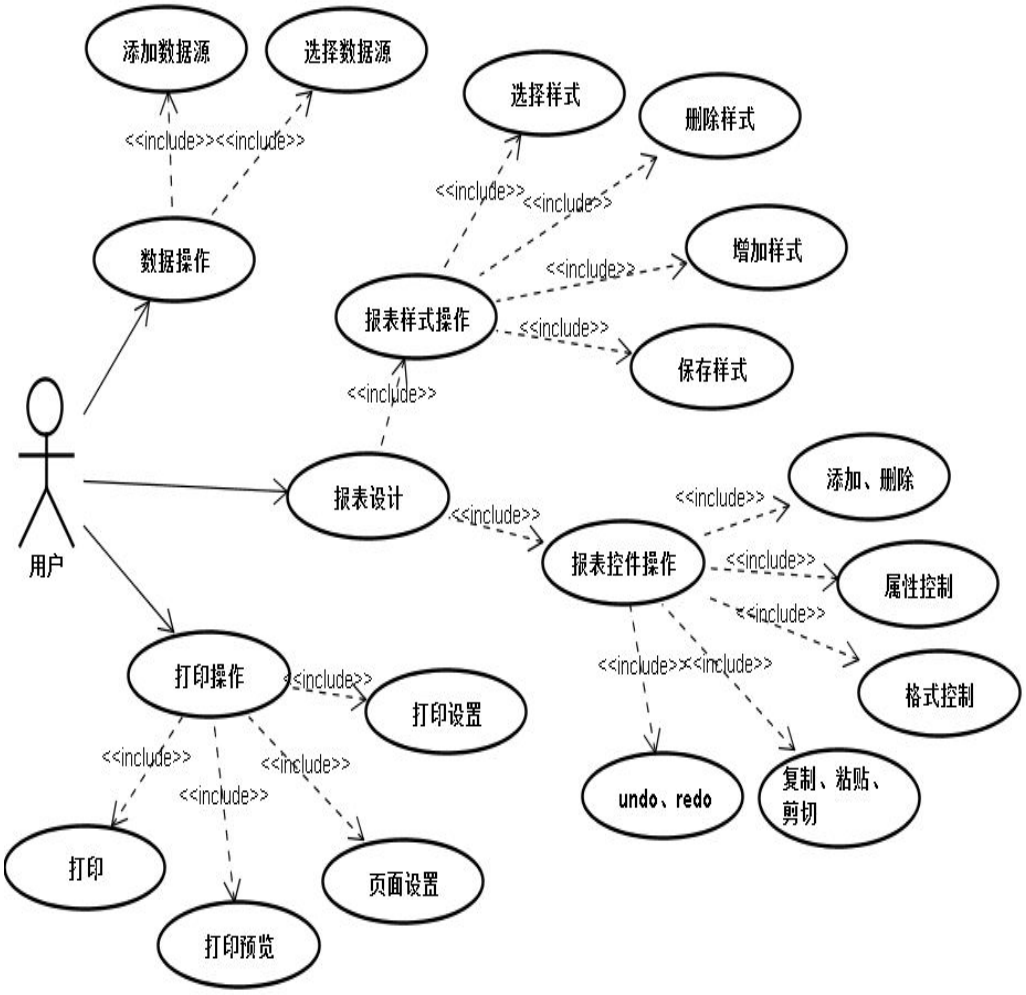

图 2-1　　Web 报表生成器用例图

2.2　功能分析

Web 报表生成器实现的主要功能如下：

1) 数据集操作：Web 报表生成器中，每个报表模板都对应着一个数据集，一个数据集则可以有多个报表模板，Web 报表生成器能添加数据集，然后把数据集的配置信息保存下来，供下次启动时初始化。有多个数据集时可以对数据集进行选择。
2) 报表设计：报表生成器中的报表设计过程完全是所见即所得的可视化操作过程，报表设计面板中的报表模板（即报表样式）都是由多个控件构成的。
   - 选择报表样式：用户可以为数据集选择不同的报表样式模板。
   - 删除报表样式：对于不符合要求或者已经过期的报表样式，用户可以将其删除。
   - 增加报表样式：对于新建的或者已经修改过的报表样式，用户可以将其保存在模板库中，在下次进入系统时便能进行选择。
   - 保存报表样式：对于已有的报表模板，当对其进行更改以后，用户可以将其更新保存。在下一次读取时便已经是更新后的样式。
   - 添加控件：在设计报表时，用户能选择不同的控件添加到设计面板中。
   - 删除控件：对于选中的控件，用户可以将其删除。
   - 属性控制：对于选中的控件，用户可以在属性框中对其属性进行更改。
   - 报表控件之间的格式控制：选中多个控件并选好初始控件之后，系统能根据用户的选择自动对选中的控件进行以下的格式控制。
     - 左对齐：所有选中控件都对齐到初始控件的左边界线处。
     - 居中对齐：所有选中控件都对齐到初始控件的左右平分线上。
     - 右对齐：所有选中控件都对齐到初始控件的右边界线上。
     - 顶部对齐：所有选中控件和初始控件的顶部对齐。
     - 中间对齐：所有选中控件都对齐到初始控件的上下平分线上。
     - 底部对齐：所有选中控件和初始控件的底部对齐。
     - 使宽度相等：所有选中控件和初始控件的宽度相等。
     - 使高度相等：所有选中控件和初始控件的高度相等。
     - 使大小相等：所有选中控件和初始控件的大小相等。
     - 使水平间距相等：所有选中控件和初始控件水平间距相等。
     - 使垂直间距相等：所有选中控件和初始控件垂直间距相等。
     - 使水平相邻：所有选中控件和初始控件水平相邻。
     - 使垂直相邻：所有选中控件和初始控件垂直相邻。
   - 控件在设计面板中的对齐方式：

- ✧ 水平居左：所有选中控件在设计面板上居左对齐。
- ✧ 水平居中：所有选中控件在设计面板上居中对齐。
- ✧ 水平居右：所有选中控件在设计面板上居右对齐。

- ➢ 能对选中控件进行复制、剪切，然后再粘贴。
- ➢ undo 操作：当设计面板上的控件状态发生改变时就能进行 undo 操作，每进行一次 undo 操作，都能把设计面板上的控件状态撤销到对应的上一次改变前的状态。
- ➢ redo 操作：当进行 undo 操作后便可以进行 redo 操作，每进行一次 redo 操作便能把设计面板上的控件状态重复到对应的一次 redo 操作之前的一个状态。

3) 打印操作：打印是报表生成器的一个重要功能，Web 报表生成器在报表打印上提供了强大的功能来满足用户的打印需求。

- ➢ 页面设置：用户通过此功能来设置打印的纸张大小。
  - ✧ 选择系统页面设置信息：Web 报表生成器已经提供了很多种纸张类型，用户选择相应的纸张类型后便能进行打印。
  - ✧ 添加页面设置信息：如果系统自带的页面类型无法满足用户需求，则用户可以自己定义纸张的大小，并可以保存定义好的纸张类型，方便下次打印时使用。
  - ✧ 删除页面设置信息：对于不需要的纸张类型用户可以进行删除，但不能删除系统自带的类型。
- ➢ 打印设置：通过此功能用户能够选择打印机、设定打印份数、设定打印时纸张为纵向打印还是横向打印等。
- ➢ 打印预览：设计好报表模板并设定后打印信息后可以预览打印效果。
- ➢ 打印：开始打印报表。

2.3 使用场景分析

用户在使用 Web 报表生成器的时候，使用步骤如下：

i. 进入 Web 报表生成器主页。
ii. 选择数据源。
iii. 选择报表模板。
iv. 进行报表设计操作。
v. 进行打印的设置操作，包括打印设置和页面设置。
vi. 打印报表。
vii. 退出报表生成器。

第三章　Web 报表生成器的设计

本章重点介绍 Web 报表生成器的一些基础知识，包括报表元素、报表文件、XML 报表控件、数据绑定模型、打印标记类等，同时介绍程序中主要的类和结构的设计方式。

3.1　报表元素设计

3.1.1　打印设计元素

在进行报表设计的时候，能够在报表设计器中使用的元素控件共有 6 种，它们是 Label、CheckBox、Line、Rectangle、PictureBox、TrueDBGrid，当在设计器面板上进行可视化设计时，操作的都是这 6 种类型的控件。这些控件有自定义类型的也有.NET 自带的。Label 是标签元素，是.NET 的 System.Windows.Form.Label 类型；CheckBox 是检查框元素，是.NET 的 System.Windows.Form.CheckBox 类型；Line 是线段元素，是自定义控件，类型为 Yinhe.XReport.ReportLine；Rectangle 是矩形元素，也是自定义控件，类型为 Yinhe.XReport.ReportRectangle；PictureBox 是图形元素，是.NET 的 System.Windows.Forms.PictureBox 类型；TrueDBGrid 是网格元素，是 C1.Win.C1TrueDBGrid.C1TrueDBGrid 类型。当用户在设计面板上进行拖放、拉伸报表元素时，和用户进行直接交互的就是这六种控件，把这六种控件定义为打印设计元素，关于打印设计元素的更多功能以及实现细节，将会在第四章中作详细介绍[3]。

3.1.2　属性替代对象

在进行报表设计的时候，属性框中会显示被选控件的所有属性，每一个打印报表设计器上的控件都有很多的属性。打印报表设计器的最终用户不是程序开发人员，因此很多属性对用户来说都是无用的，而且把所有的属性都显示在属性框中，这对没有程序开发经验的使用者来说会不知所措，因此需要一个中间对象，这个对象的属性都是对用户有用的，数量很少，而且都是中文的。当属性窗口中要显示控件的属性时其实显示的是这个中间对象的属性。这个中间对象叫做属性替代对象，它简化了打印报表设计元素的属性设置。每一种打印设计元素都会对应一种属性替代对象，由于 Web 报表生成器中用到的报表设计元素的控件只有 6 种，所以这样设计的工作量不是很大，是完全可行的。Label 控件的属性替代对象是 rptLable；CheckBox 的属性替代对象是 rptCheckBox；Line 的属性替代对象是 rptLine；PictureBox 的属性替代对象是 rptPictureBox；Rectangle 的属性替代对象是 rptRectangle；TrueDBGrid 的属性替代对象是 rptTrueDBGrid。这些属性替代对象不是控件，他们只是用来显示属性，真正显示在打印报表设计器面板上的

还是与之对应的控件，而不是这些属性替代对象。关于属性替代对象的更多实现细节以及它与设计元素间的交互方式将在第四章中作详细介绍。

3.2 XML 报表控件即插即用

3.2.1 XML 报表控件

XML 报表控件是本文提出的一个新概念，所谓 XML 报表控件就是报表控件的 XML 表示，其中报表控件就是上一节中介绍过的 6 种打印设计元素，这 6 中控件中有 4 种是.NET 平台中自带的控件，有两个是自定义控件，但是这两个自定义控件（Line 和 Rectangle）都继承自 System.Windows.Forms.Panel，所以说到底所有的控件其实都来自于.NET。而根据.NET 控件的特点，很显然是可以用 XML 标签元素来表示的，一个控件实例便对应一个标签元素。例如，一个 Label 控件，它的 XML 表示为：<Label></Label>标签对。一般说来，用类名作为它的标签名。那么对于报表的 XML 标签就是<Report></Report>，这里的 Report 通常用来做根标签[1]。

例如，图 3-1 的 XML 文档表示了一个报表模板。

```
<Report Margins="0,0,0,0" Landscape="false" PaperSize="A4,794,1122"
xmlns="">
  <Controls>
    <Label Name="Label7" Size="0,0" />
    <Label Name="Label1" Location="107,178" Size="531,23"
BackColor="White" BorderStyle="FixedSingle" TextAlign="MiddleCenter"
Text="姓名" DataBindings="Text, Age" Font="Times New Roman, 10.5pt" />
    <Label Name="Label4" Location="6,178" Size="102,23"
BackColor="White" BorderStyle="FixedSingle" TextAlign="MiddleCenter"
Text="年龄" Font="Times New Roman, 10.5pt, style=Bold" />
    <Label Name="Label2" Location="6,156" Size="102,23"
BackColor="White" BorderStyle="FixedSingle" TextAlign="MiddleCenter"
Text="姓名" Font="Times New Roman, 10.5pt, style=Bold" />
    <Label Name="Label18" Location="524,529" Size="117,23"
BackColor="White" BorderStyle="None" TextAlign="MiddleRight" Text="页
码" Font="宋体, 9pt" Tag="PageNum" />
  </Controls>
</Report>
```

图 3-1　　报表文件结构设计举例

在 Web 报表生成器中，报表的设计面板就是一个 Panel。由于在.NET 中，Panel 上的所有控件都放在属性 Controls 中，所以在设计报表格式文件时就加了这个标签，这个属性是一个集合类型属性，在 XML 文档中用 Controls 标签表示。实际上，它不是一个实体标签（并不代表控件实例），只是起到一个标记作用，报表中的每一个控件都是 Controls 标签的子标签，这样就正好与设计面板中所有控件都放在 Panel 的 Controls 属性中相对应。

3.2.2 控件属性的 XML 表示

- XML 属性概述

使用 XML 标签可以表示报表的控件对象，而控件的属性是用 XML 标签元素来表示的。在 3.2.1 节中的报表格式文档中，Name、Location、Size、BackColor、ForeColor 等这些控件的属性都是用 XML 元素的属性表示的。

那么什么是XML元素的属性呢？XML元素的属性就是标签内的名字/值对。例如，对于标签<Label Name="Label1" Text="姓名"/>，Name="Label1"、Text="姓名"都是属性。其中，Name 和 Text 是属性名，“Label1”和“客户编号”是属性值。在同一个报表中，每个标签元素的 Name 属性值不能相同，因为一个标签元素代表一个控件。属性值是用双引号括起来的字符串，根据属性值的数据类型不同，其值形式也不相同。

例如，一个 Label 控件定义如图 3-2。

```
<Label Name="Label1" Location="107,178" Size="531,23" BackColor="White" ForeColor="215,223,232" BorderStyle="FixedSingle" TextAlign="MiddleCenter" Text="姓名" DataBindings="Text, Age" Font="Times New Roman, 10.5pt" />
```

图 3-2　　Label 控件标签结构举例

在本例中，当在设计面板上创建该 Label 控件时，控件的 Name 属性其属性值是“Label1”；Location 属性的属性值是“107,178”；Size 属性的属性值是“531,23”；BackColor 属性的属性值是“White”等等。

- 类型转换器

在前面的实例中，Name 属性定义了该 Label 的名字 Label1，它是字符串类型；Text 属性是 Label 显示的内容，也是字符串类型；TextAlign 是定义显示文本的对齐方式，为枚举类型；BackColor 是背景颜色，Color 对象类型；ForeColor 是字体颜色，也是 Color 对象类型；Location 是位置，Point 对象类型；Size 是尺寸，Size 对象类型；Font 是字体，为 Font 类型。

从这个例子的几个属性中可以发现，这几个属性值的类型都是不相同的，有的是字符串类型；有的是 Color 类型；有的是 Size 类型；有的是 Font 类型。不同类型的属性值有不同的表示方式，那么怎样才能把控件的属性值转换成字符串

呢？比如，属性背景颜色的类型是 Color 类型，那么 Color 类型转换成 XML 属性值的表达式是什么样呢？从本例中我们可以看出，有两种形式：BackColor="White"和 ForeColor="215,223,232"。这两种形式都是合法的，这两种形式来源于 Color 类型的类型转换器。

在.NET 框架的类库中，有这样一个类 System.ComponentModel.TypeConvert，它有多个方法：CanConvertFrom()、CanConverTo()、ConvertFromInvariantString()、ConvertTo()。CanConvertFrom()方法返回的是 bool 类型，返回该转换器是否可以将给定类型的对象转换为此类型转换器的类型；CanConverTor()方法返回的也是 bool 类型，返回此转换器是否可将该对象转换为指定的类型；ConvertTo()方法使用参数将给定的值对象转换为指定的类型；ConvertFromInvariantString()将给定字符串转换为此转换器的类型；这几个方法决定了对于给定的类型与字符串类型能否进行转换，以及转换为字符串格式[5]。

3.2.3 即插即用举例

在上两节中已经介绍了如何用 XML 标签来表示一个报表控件，下面开始探讨 XML 控件即插即用思想在报表可视化设计中的应用。所谓的即插即用就是实现控件在报表上的任意插拔。即插即用有两种方式：一种是手工编辑；一种是可视化设计。

可视化设计方式是通过报表设计器来实现把控件从工具栏拖到报表设计表面上，然后通过设置属性方式来完成设计，并保存为 XML 文件。手工编辑方式是用文本编辑器直接编辑 XML 文件后再用报表生成器来打开 XML 文件。

两种方式各有优缺点，下面来看看这两种方式的具体操作[6]。

首先启动 Web 报表生成器，新建一个报表模板，然后在设计面板上添加一个 Label 和 CheckBox，如图 3-4 所示。保存之后构成的报表文件如图 3-3 所示。

```
<Report Margins="0,0,0,0" Landscape="false" PaperSize="A4,794,1122" xmlns="">
  <Controls>
    <Label Name="Label1" Location="77,161" Size="100, 23" BackColor="White"
        BorderStyle="FixedSingle" TextAlign="MiddleLeft" Text="文字" />
    <CheckBox Name="CheckBox1" AutoCheck="False" Location="209,163"
        Size="104, 24" FlatStyle="Flat" BackColor="White" Text="文字" />
  </Controls>
</Report>
```

图 3-3　报表文件结构

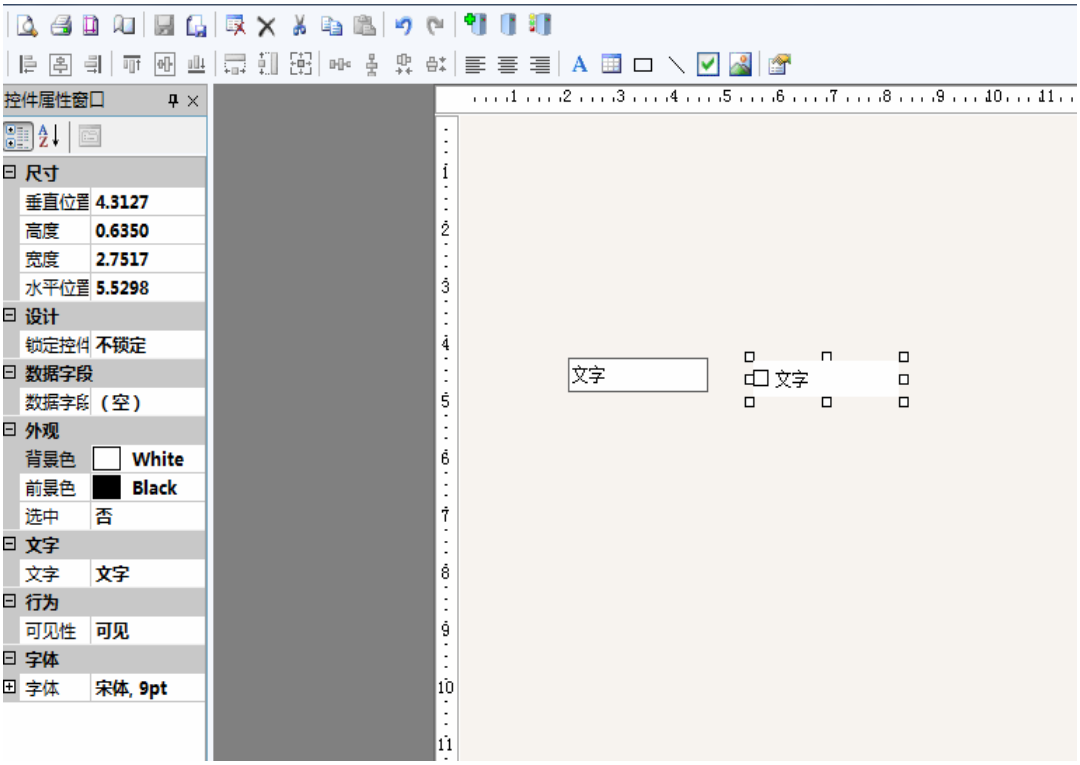

图 3-4　　新建的报表设计器界面

对于刚才的报表文件，可以把 CheckBox 去掉，报表文件代码变为如图 3-5 的形式。

```
<Report Margins="0,0,0,0" Landscape="false" PaperSize="A4,794,1122"
xmlns="">
  <Controls>
    <Label Name="Label1" Location="77,161" Size="100, 23"
        BackColor="White" BorderStyle="FixedSingle"
        TextAlign="MiddleLeft" Text="文字" />
  </Controls>
</Report>
```

图 3-5　　更改后的报表文件

接下来再用报表设计器打开这个文件，结果如图 3-6 所示。这样就实现了“即插即用”。

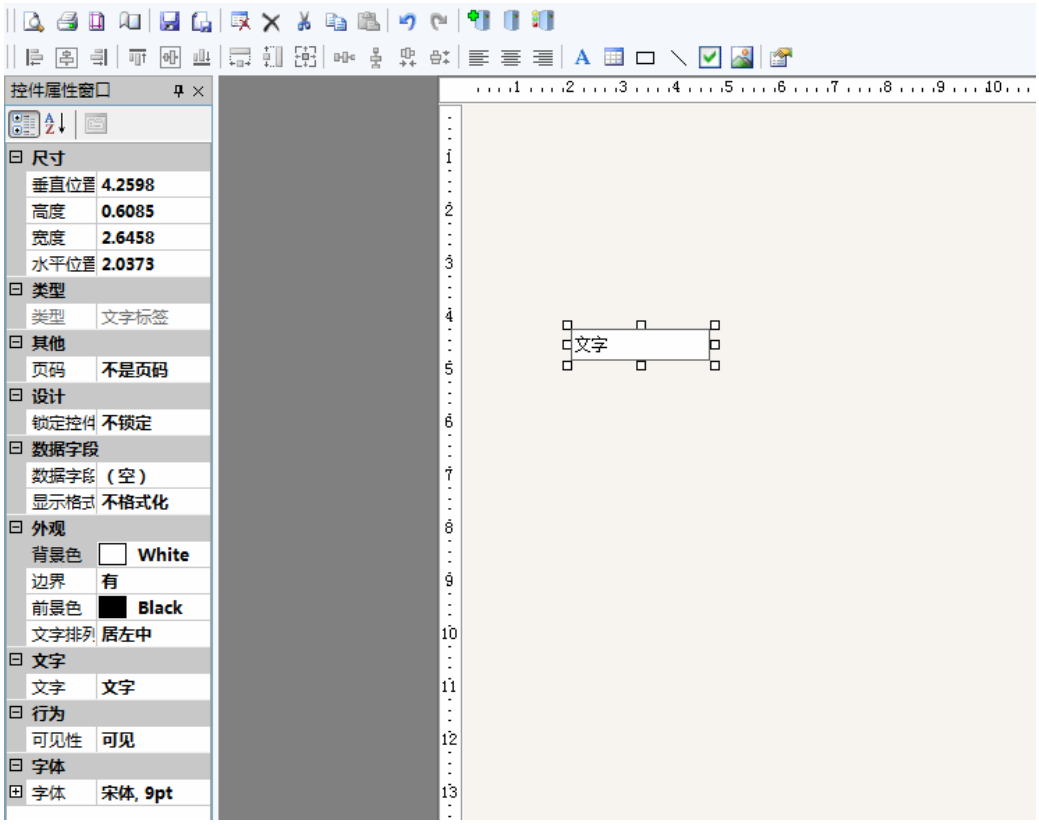

图 3-6　　更改后的报表设计器界面

从刚才的实例中可以看出，Web 报表生成器中设计面板上的可视化设计结果和 XML 报表文件是相互对应的。因此在设计报表模板时既可以用可视化方式也可以用手写方式，情况就如同在做 HTML 网页时一样。关于 XML 控件即插即用的更多实现细节，包括 XML 文件的读写操作将在第四章中做详细介绍。

3.3　报表标签定义文件

3.3.1　报表标签定义文件介绍

上文中已经介绍过，一个报表控件可以用 XML 标签来表示。当报表设计器打开一个 XML 格式的报表文件时，设计器是怎样把 XML 标签显示在设计面板上的呢？对于一个标签<Label Name="Label1" Text="姓名" />，设计器又是怎样知道在设计面板上建立一个 Label 呢？比如说，对于一个 CheckBox 标签，要创建一个 CheckBox 实例，那么这个实例是通过什么创建的呢？一般说来，想要创建一个对象都是用类的构造函数实例化这个对象。比如：CheckBox CheckBox1= new CheckBox();就创建了一个复选框的实例 CheckBox1。但首先必须要有 CheckBox 的类型（System.Windows.Forms.CheckBox）[8]。如果控件是纯粹的.NET 控件，那自然很容易得到控件的类型。但对于一个自定义的控件，在获取其类型的时候就会有问题了，为了解决这个问题，首先需要控件的类型信息，因此引入

了报表标签定义文件——Report.Markup.xml 来定义这些信息。如果没有这个文件就无法识别打印标签，更无法产生设计元素对象。Report.Markup.xml 也是 XML 格式的文件，最顶层元素标签必须是 Schema，而且只有唯一一个。每个元素都是 Schema 的子元素，一个元素代表一种打印类型，它们的结构如图 3-7 所示。

```
<?xml version="1.0" encoding="UTF-8" ?>
<Schema>5
  <Label Type="System.Windows.Forms.Label,System.Windows.Forms.dll" >
    <Property Name="Name" Tag="名字" />
    <Property Name="BackColor" Tag="背景颜色" />
    <Property Name="BorderStyle" Tag="边界风格" />
    <Property Name="Font" Tag="字体" />
    <Property Name="ForeColor" Tag="前景颜色" />
    <Property Name="Location" Tag="位置" />
    <Property Name="Size" Tag="尺寸" />
    <Property Name="Tag" Tag="别名" />
    <Property Name="Text" Tag="文本" />
    <Property Name="TextAlign" Tag="文本对齐" />
    <Property Name="Visible" Tag="可见性" />
    <Property Name="FlatStyle" Tag="风格" />
    <Property Name="Dock" />
    <Property Name="Locked" />
    <Property Name="DataBindings" Tag="绑定"/>
    <Property Name="ContextMenu " Tag="弹出菜单"/>
  </Label>
  ……
  </Schema>
```

图 3-7　　报表标签定义文件

打印标签元素有 4 个属性：Type、IsRoot、Children、Visible，其中 Type 是必需的，其他的是可选的。

- Type：“类型全名，动态链接库名”。
  比如：“System.Windows.Forms.Label,System.Windows.Forms.dll”，系统在读 XML 格式报表文件的过程中，当遇到<Label>后就会在动态链接库 System.Windows.Forms.dll 中查找类型 System.Windows.Forms.Label 并创建它。

- IsRoot：根属性，标识此标签在 XML 文件中是否允许作为根元素，默认值为 false，表示标签不能为根标签，根标签一定是 Yinhe.XReport.Report 类型。
- Children：子控件属性，表示此属性值是一个属性名，这个属性名是集合类型的属性，在这个集合中存放子控件。
- Visible：可见性，默认值为 true，如果为 false，则控件不可见。

在 XML 格式的报表模板文件中，控件标签的属性名必须是 Report.Markup.xml 中已经定义的。其中，控件标签的属性名在 Report.Markup.xml 中是通过 Property 标签来定义的，在 Property 标签中，只有两个属性 Name 和 Tag，Name 定义的是英文属性名，Tag 定义的是中文属性名。在写 XML 文件时不仅可以使用英文属性名，也可以使用中文属性名。

3.3.2 报表标签定义文件加载

如果没有 Report.Markup.xml 文件就无法识别打印元素，所有在系统启动时首先要加载该文件。加载过程大致如图 3-8。

```
Yinhe.XReport.XmlControl.ReadReportSchema(“Report.Markup.xml”);
```

图 3-8　　加载报表标签定义文件

ReadReportSchema 是 XmlControl 类的一个静态方法，读入报表标签定义文件后，便会把文件解析为一个静态的内存结构，放在内存中，供后续过程中解析报表模板文件之用。

3.4 数据模型设计

报表的最终目的其实就是展示数据，因此报表生成器中必须要解决的问题就是数据绑定模型。也就是数据库中的数据是怎样取到报表界面上的？本节将讨论这些相关内容。

3.4.1 报表数据源

在 Web 报表生成器中，定义了一种用来保存数据源信息的文本类型文件，里边保存的信息有数据库连接字符串、主表名称、列表名称、主表 SQL 语句、列表 SQL 语句。当启动 Web 报表生成器后，用户能选择不同的数据源其实就是在选择不同的数据集配置文件。当选定数据源以后，系统便根据配置文件中的信息与相应的数据库连接，并根据 SQL 语句取出数据，然后把数据装入定义好的数据集中。用户还可以添加数据源[10]，其实就是增加一个数据源配置文件以供下次选择。创建好数据集后，每个数据集都会有很多报表模板（即报表样式）与之对应，用户再选择一种报表样式，然后设计面板上便能显示出相应的报表了。这一对应关系如图 3-9 所示。

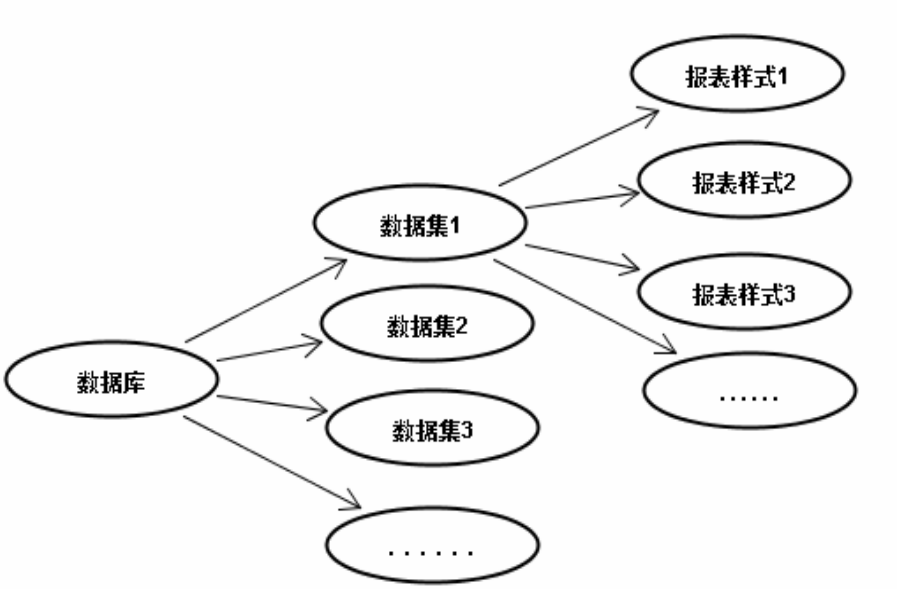

图 3-9　　报表文件对应关系

由刚才的讨论可知，报表生成器先将数据库中的数据取出来放到一个数据集中，接下来再选择与这个数据集相对应的报表样式。那么，报表模板是如何与数据集中的数据组合起来再共同显示在设计面板中的呢？答案是数据绑定。接下来就需要讨论一下这个概念。

3.4.2　数据绑定

在.NET 中，数据源与界面的交互是通过数据绑定来实现的。.NET 中的数据绑定有简单绑定和复杂绑定两种。像 Label 这样的简单控件的数据绑定采用的是简单绑定方式；像 DataGridView 这样的控件采用的则是复杂数据绑定方式。对于每一个简单控件，都存在一个 DataBindings 属性，这是一个集合类型的属性，它的成员是一个 Binding 类型。Binding 类用来创建和维护某控件的属性与某对象的属性或对象列表中当前对象的属性之间的简单绑定。还有一种非常重要的组件是 BindingSource，此组件有两个用途。

首先，它通过提供一个间接寻址层、当前项管理、更改通知和一些其他服务，简化了窗体中控件到数据的绑定[9]。这时通过将 BindingSource 组件附加到数据源，然后再将上层控件绑定到 BindingSource 组件来实现的。其次，与数据的所有进一步交互，包括定位、排序、筛选和更新，都通过调用 BindingSource 组件实现。

3.4.3　数据绑定模型和 XML 表示

由上两节的讨论已经知道，从数据库取来的数据放在数据集中，那么是否报

表控件就直接绑定到数据集上呢？答案是否定的。虽然控件是可以直接绑定到数据集的数据表上，但这样做不利于数据的控制，因此最终选择了一个中间媒介来完成数据绑定，这个中间媒介就是 BindingSource 组件。如图 3-10 所示。

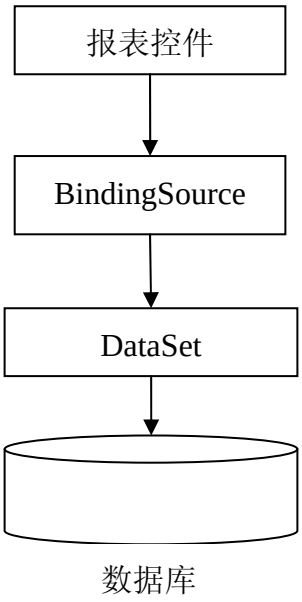

图 3-10　　报表控件数据绑定

从图 3-10 中可以看出，控件是绑定在 BindingSource 上的，而 BindingSource 又绑定在数据集（DataSet）上，而数据时通过数据集与数据库直接发生联系的。

在 Web 报表生成器中，发生数据绑定的有三种控件，标签元素、检查框元素、网格元素。

- 标签元素和检查框元素的数据绑定：这两个元素属于简单控件，所以数据绑定是通过 DataBinding 属性来实现的。它们只能绑定到主表上，没有选择的余地。主表中只有一条记录，所以，标签元素和检查框元素在进行数据绑定的时候是通过选择主表记录的不同字段来进行绑定的。当从数据字段下拉列表中选择一个字段时，对于标签元素而言，是把该字段绑定到标签元素的 Text 属性上；对于检查框元素而言，是把该字段绑定到检查框元素的 Checked 属性上。数据绑定格式如图 3-11。

```
DataBindings=“控件属性名，字段名”
```

图 3-11　　简单控件数据绑定格式

- 网格元素的数据绑定：网格元素一定绑定到列表数据源对象上。当从工具条上拖一个 Grid 放到设计面板上时，数据绑定自动完成，不需要人工干预。网格元素的数据字段也是自动建立的。

3.5 报表内存结构设计

在 Web 报表生成器中，有两种外部 XML 文件：报表标签定义文件和报表格式文件。在读取 XML 文件时，系统中定义了相应的内存结构来存储这些信息，先来讨论这些内存结构。

- XmlTagSchema 类：与报表标签定义文件相关，为标签结构。每一个标签对应着一个 XmlTagSchema，XmlTagSchema 中定义了标签的名字、类型、是否为根标签、是否可见、子控件属性和属性集等信息。其结构与报表标签定义文件中所定义的标签对应。属性集为 ArrayList 类型，其中的每一个节点为一个 PropertySchema 类型的对象。
- PropertySchema 结构：标签属性结构，存放的是属性名和别名对。
- XmlComponentSchema 类：报表生成器中很重要的一个类，用来存放 XML 类型的报表文件。成员变量主要有三个：string 型的 XML 控件类型，如 Label、CheckBox 等；ArrayList 类型的 XML 控件属性集，每个单元存放的是一个 XmlProperty 结构的对象；SortedList 类型的子控件集。重点在 SortedList 类型的子控件集中：如果当前节点为 Report，子控件集的 key 即为 Controls，value 为一个 ArrayList 类型的列表，其中 ArrayList 中放的是定义成了 XmlComponentSchema 类型的 Label、CheckBox、Line、C1TrueDBGrid 等等的节点；如果当前节点为 C1TrueDBGrid，则子控件集的 key 为 Columns，value 与刚才情形相同，只是其 ArrayList 中放的是 C1DataColumn 等的节点。上文中已经提过，这里作为 Key 的 Controls 和 Columns 在报表文件中虽然以节点形式存在，但其实只起到标识作用，如果当前节点没有子节点，则子控件集为空，除 C1TrueDBGrid 和 Report 外，其他类型的节点都没有子节点[12]。由此可以看出，这个类是递归定义的，用一个 XmlComponentSchema 类型的对象就能存放一个报表文件了。XmlComponentSchema 中还定义了一些重要的成员函数：
  a) 读取 XML 文件
  b) 遍历 XML 文件的每一个节点，并加载之。
  c) 记录当前节点的所有属性值。
  d) 添加控件。
  e) 移除控件。
  f) 修改控件。
  g) 创建 XML 文件。
- XmlProperty 结构：存放的是控件的属性名和属性值对。

3.6 打印标记类

Web 报表生成器的另一个重要功能就是进行报表的打印，打印操作基本就是向打印机绘制图形的操作。整个.NET 打印架构的核心就是打印文档对象（PrintDocument）。PrintDocument 的 PrintPage 事件通过使用 Graphics 对象将要打印的文档输出到打印机画面中。实际绘制操作就像在任何其他 Graphic 对象上绘制一样。因此在创建好 PrintDocument 的实例后，需要订阅它的 PrintPage 事件。然后通过调用 PrintDocument 对象的 Print 方法便能触发 PrintPage 事件，从而进行打印。

打印报表的所有设计元素均是通过 Tag 打印标记类来实现打印的。这个类实现了所有设计元素的属性。比如，它具备一般设计元素的宽度和高度，但也同时具备图形元素的 Image 属性。Tag 类有一个重要的方法 PrintSelf(Graphics g)，所有设计元素向打印机进行输出的操作都是通过这个方法实现的，它是通过判断不同设计元素类型进行打印输出的。在使用这个方法之前，先要把设计元素转换为为打印标记类的对象，这个过程由一组控件转换函数来实现，具体转换过程将在第四章中作介绍[17]。

第四章　Web 报表生成器的实现

前面的内容已经对 Web 报表生成器的基础知识进行了详细的介绍，本章将在前面内容的基础上，进一步讨论 Web 报表生成器的具体实现。在详细讨论之前，先介绍一下整个系统在实现过程中用到的一些比较重要的全局变量。

1) SortedList slRec = new SortedList() ；维护着被选控件的矩形信息。对于 slRec，key 为控件名，value 为控件对应的矩形。控件的大小和位置与这个矩形一致，在对控件进行拉伸、拖动时，首先是对这个矩形进行操作，完毕之后再把控件调整为与这个矩形大小、位置一致。slRec 中的控件肯定一直与 alControl 中的控件保持一致，因为这些可是化操作肯定是对被选控件进行的。
2) ArrayList alControl = new ArrayList()；为选中控件的集合。
3) XmlComponentSchema xcSchema = new XmlComponentSchema()；控件结构，读 xml 类型的报表文件时读出来的就是这个类的对象。
4) static SortedList XmlTagSchemaCollection；存放的是 Report.Markup.xml 文件的结构信息。键为节点名，值为 XmlTagSchema 类型的对象，要区分 XmlTagSchema 类型和 XmlComponentSchema 类型。具体的第三章中以有介绍。
5) Control PrimaryControl；为标识控件。所标识的控件四周的八个小矩形为黑色，多个控件的对齐、大小控制等命令都以此控件为准。
6) page：报表生成器的设计面板，为 panel 类型。
7) dragcontrol：string 类型，其值有“Label”、“CheckBox”、“C1TrueDBGrid”、“Line”、“PictureBox”、“Rectangle”、“Arrow”七种，需要画控件的时候，点击相应的控件按钮，就把 dragcontrol 的值设为了相应的控件，然后在设计面板的鼠标事件中，通过这个变量的值来判断要在 page 上画何种类型的控件，若其值为“Arrow”，则不画控件。
8) MouseArrow：string 类型，其值有“LeftUp”、“LeftMiddle”、“LeftDown”、“RightUp”、“RightMiddle”、“RightDown”、“UpMiddle”、“DownMiddle”、“None”九种情况，用来表示拉伸控件时鼠标所处的位置，为“None”时不进行拉伸。

4.1　XML 报表控件即插即用实现

从 XML 文件到设计面板上的报表控件，再从报表控件保存回 XML 文件的整体流程如图 4-1 所示。

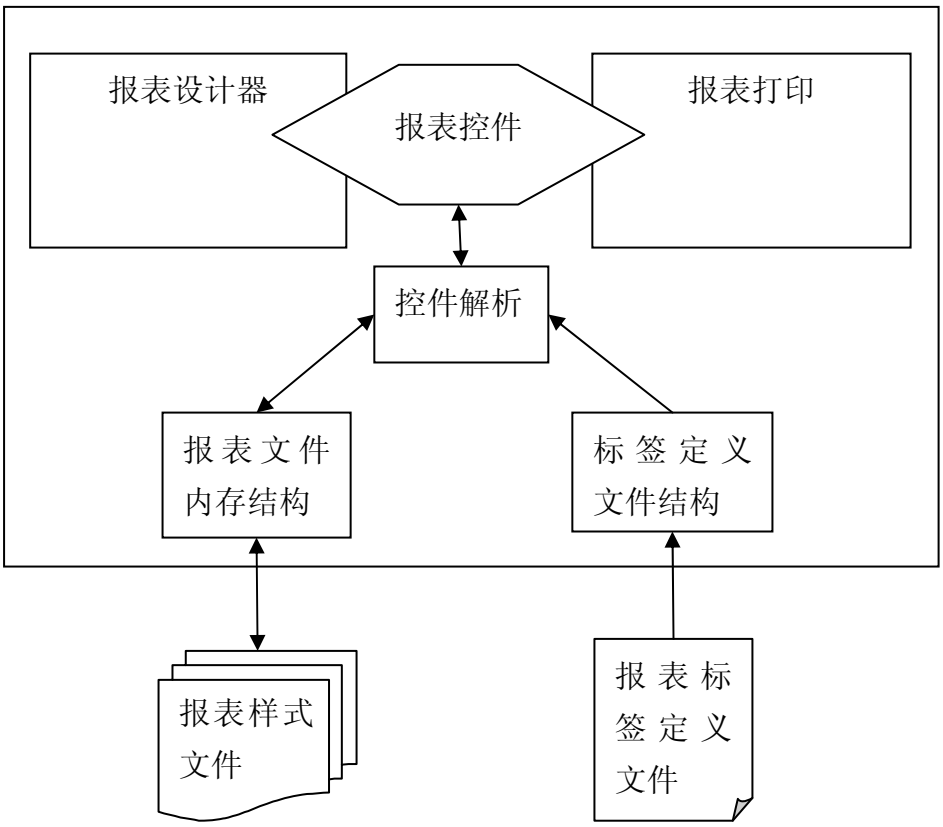

图 4-1　　系统总体流程

接下来的内容将对系统的总体流程作详细介绍。

4.1.1　加载报表标签定义文件

在上一章中已经介绍过 Report.Markup.xml 文件的结构和作用，现在来讨论此文件在系统初始化时的加载过程。

Report.Markup.xml 文件是通过 XmlControl 类的 ReadReportSchema 静态方法来加载的，最终把 Report.Markup.xml 文件中的报表标签结构信息存放在静态属性 XmlTagSchemaPro 中，该属性是 SortedList 类型。在 XmlTagSchemaPro 中，每个单元的值是 XmlTagSchema 类型，Key 是“Label”、“CheckBox”、“ReportLine”、“PictureBox”、“ReportRectagle”、“C1TrueDBGrid”等。

4.2.2　加载报表样式文件

报表设计器初始化的时候便会去读相应的 XML 文件，再把 XML 文件解析成控件。主要过程从 Initial()函数开始，调用 Choose_Print_Style()，这个函数内调用 XmlComponentSchema 类的 ReadXmlFile(string XmlName)函数，便得到一个 XmlComponentSchema 对象 xcSchema，xcSchema 是整个 XML 文件的结构，所有 XML 文件的信息都放到这个对象中，得到 xcSchema 后，page 中的所有控件

便从其中进行加载，当对报表进行更改之后，新的报表信息也是先存回 xcSchema 中，最后保存报表文件的时候[13]，直接把 xcSchema 写回成 XML。可见，xcSchema 在整个读写过程中起到一个桥梁作用，是个很重要的对象，而且把 XML 文件中的信息以 xcSchema 对象的行式放在内存中，比每次交互都去读磁盘中的 XML 文件要快。

XmlComponentSchema 类的主体结构是棵递归定义的树，关于这个类的定义已经在上一章中作过介绍了。

下面从 XmlComponentSchema 的 ReadXmlFile(string XmlName)开始详细介绍读取 XML 文件并得到 xcSchema 的过程，加载的 XML 文件以图 4-2 所示的为例，仅有一个 label 和一个 C1TrueDBGrid，省略了部分属性和子节点。

```
<?xml version="1.0" encoding="UTF-8"?>
<Report Margins="100,100,100,100" Landscape="false" PaperSize="A4,794,1122">
  <Controls>
    <Label Name="Label1" Location="159, 113" …… />
    <C1TrueDBGrid Name="C1TrueDBGrid1" ……  >
      <Columns>
        <C1DataColumn Tag="1" Caption="Id" …… />
        <C1DataColumnTag="2" Caption="Name" ……  />
        ……
      </Columns>
    </C1TrueDBGrid>
  </Controls>
</Report>
```

图 4-2　　Xml 文件

- 加载 XML 文件，加载的文件以只有一个 Label 和一个 C1TrueDBGrid 为例。先得到一个 doc，开始调用 LoadXmlComponent(object obj, ArrayList childCotrols, Junxian.XReport.XmlComponentSchema xcSchemaParent)，
- 首先传递的是 doc，对 doc 遍历，只有一个子节点，即 Report。判断 Report 是否是标签，为是，加载 Report 的属性，判断 Report 是否是根节点，也为是，所以让此层函数中新建的列表和传递进来的列表有相同的引用。判断该节点是否有子节点，根节点有子节点，所以把列表添加到其 ChildControls 中，判断此节点的子节点数目是否大于 0，因为根节点有一个 Controls 子节点，大于 0，所以转入下一次递归，
- 此时传递的是 Report，遍历 Report，有一个 Controls 子节点，判断 Controls

是不是标签，不是。判断父节点是不是根节点，为是，因此让此层递归中新建的列表与传递进来的列表有相同的引用，接下来让父节点（Report）的 ChildControls 中的列表与新建的列表有相同的引用，判断此节点的子节点数目是否大于 0，有两个子节点，满足。继续下一次递归

- 此时传递进来的是 Controls，遍历 Controls，有两个子节点：
  - ✧ 首先取得的是 label，判断 label 是不是标签，是。加载 Label 的属性。判断是不是根节点，不是，所以把它加进传递进来的列表中，然后让新建的列表引用到一个新的对象，因为此时的节点（label）也是可能有子节点的，有的话就要把新建的列表添加的其 Childcontrols 中，但 label 没有子节点，所以继续往下走。判断此时的节点（label）是否有子节点，没有，所以不往下递归，进行下一次循环。如果循环取得的和 label 一样没有子节点的节点，如还是 label，或者 CheckBox、Line、PictureBox 等等，但大多数时候还是 label，情况就与此相同了。
  - ✧ 取得 C1TrueDBGrid，判断其是不是标签，是。加载其属性。判断是不是根节点，不是，所以把它加进传递进来的列表中，然后让此层递归中新建的列表引用到一个新的对象，因为此时的节点（C1TrueDBGrid）也是可能有子节点的，有的话就要把新建的列表添加的其 Childcontrols 中，很明显有，为 Columns。所以添加到 C1TrueDBGrid 的 ChildControls 中，并且进入下一层递归。
- 此时传递进来的是 C1TrueDBGrid，如此继续下去。

整个过程结束后也就得到了一个 xcSchema。

4.2.3 报表内存结构转化为具体控件

这个过程分三个步骤来介绍。

i. 创建控件实例

Initial()函数中加载完 XML 并获得 xcSchema 对象后，便开始创建控件，此过程从 InitializeForm()开始。调用函数 CreateControls(SortedList slXmlControls, object ParentControl)，遍历 slXmlControls 里列表中的所有控件，每取到一个控件结构（加入此处刚取得的控件结构为 tempCtrl），就用 XmlControl.Create(tempCtrl.Type)函数创建控件的实例，现在先重点说一下创建控件实例的过程：

- ➢ 根据控件类型取得相应的标签结构（XmlTagSchema）。
- ➢ 根据标签结构取得控件的类型全名和所在的程序集名。
- ➢ 根据程序集名取得对应的程序集。在取程序集的过程中，有一点需

要引起特别注意：去读取程序集时，最终用到的函数是：Assembly.LoadWithPartialName(dllName)，从应用程序目录或全局程序集缓存中加载，但每次都去磁盘中取 dll 可能会很慢，因此，为了提高性能，定义了一个静态 SortedList 类型的 XmlAssembly。这里存放的是以动态连接库名为 Key 的 Assembly，遇到 Dll 时首先是查看 XmlAssembly 集合中是否是以前加载过的 dll，如果是则直接取出[15]。

- 获得程序集后，便通过程序集的 CreateInstance(TypeName)方法创建控件实例，传递的参数为控件的全名。

ii. 把控件加入到父控件中

首先调用的是函数：

XmlComponent.XmlComponent.AddChildControl(ParentControl, key, (object)obj)，其中的 key（Controls 或 Columns）就是 ParentControl(Page 或 C1TrueDBGrid)的一个属性值。以父控件 page，key 为 Controls 为例：

- 函数原型：AddChildControl(object obj, string PropertyName, object ChildControl)，先根据 PropertyName 得到 obj 的相应属性，主要为：
  - Type tObj = obj.GetType();
    得到父控件类型(此时为 Panel)
  - PropertyInfo PropInfo = tObj.GetProperty(PropertyName);
    得到属性。
  - Type tProp = PropInfo.PropertyType;
    得到属性的类型(为 ControlCollection)

  接着调用 AddItem(obj, tProp, strPropName, ChildControl)函数。
- 对于函数 AddItem(object obj, Type tProp, string PropertyName, object objItem)，主要代码为：
  - Type[] typeArray = new Type[]{objItem.GetType()};
  - MethodInfo methodInfo = tProp.GetMethod("Add", typeArray);
    发现属性的类型的方法：参数为方法名和发现的方法的参数类型的数组
  - object objProp = GetPropertyValue(obj, PropertyName);
    得到属性对象。
  - methodInfo.Invoke(objProp, Parameter);
    调用方法，objProp 为对方法进行调用的对象，Parameter 为参数数组。

进行到此便已经把控件 tempCtrl 加入到父控件中了，即加入到了 panel 类型的设计面板 page 中。

iii. 设置控件的属性

现在 page 中已经有了控件的初始化实例，接下来就是根据报表文件结构中记录的控件信息来初始化此控件的实例，当前取得的报表控件结构是 tempCtrl，对其 Properties 属性进行遍历，每遍历一次取得一个 XmlProperty 对象，然后就是根据这个 XmlProperty 对象来设置 tempCtrl 的属性[16]。

4.2 报表控件属性机制

报表生成器中有一个属性窗口类 PropertyForm，通过这个属性窗口就能很方便的对报表控件的属性进行控制，在属性窗口 PropertyForm 中，主要封装了一个 System.Windows.Forms.PropertyGrid 对象，并把其 SelectedObjects 属性封装为 object 类型的 Property 数组，外界要显示控件的属性时都对这个 Property 数组赋值，而 Report 中，这个操作都在 SetSelectedComponents()函数中进行，所以凡是需要显示控件属性的地方，都会调用这个函数，让它来赋值显示控件的属性。

在前文中已经介绍过，为了简化打印报表的设计元素的属性设置，每一个报表设计元素都对应着一个属性替代对象。属性窗口中显示的属性都是属性替代对象的属性，但属性替代对象仅仅是用来显示属性的，而不参与报表设计，因此当设置这个中间对象的属性时，如何转化为对实际控件属性值的设置呢？同样的，当实际控件的属性发生改变时，又如何转化为属性窗口中替代对象的属性改变呢？其实重点就在对替代元素初始化时，会给其传递当前实际控件的一个引用。要显示属性时，就通过这个引用获得实际控件的实际属性，然后进行替换。最后在属性窗口中显示的就是替换后的属性[13]。设置属性时，是把替代元素中获得的值通过实际控件的引用赋给实际控件的对应属性，从而使实际控件的属性得到改变。

4.3 图形系统实现

Web 报表生成器采用的是所见即所得的可视化方式来进行报表设计的，因此图形系统是必不可少的。本系统采用的是网页中嵌入 WinForm 的移动编码方式，因此在网页中也能充分利用 WinForm 的绘图功能。图形系统便是围绕着 WinForm 展开的。图形系统需要处理的是手柄矩形、框选矩形、动态控件矩形、设计元素移动、拉伸、对齐等操作。本节将介绍这些图形处理功能。首先介绍图形系统中频繁被调用的部分程序。

- 动态矩形

动态矩形包括虚线和实线两种：虚线矩形框是在打印报表设计器图面上按住鼠标左键拖动鼠标形成的虚线矩形，它划定选中区域。在其范围内的所有设计元素都会被选中；实线矩形框是在拖动或拉伸控件时产生的，对控件进行拖动或拉伸操作时，便会在面板上画出实线的矩形框，相应的操作都先在这个实线矩形上

进行，等拖动或拉伸完毕弹起鼠标左键后，再把控件移到实线矩形的位置或者把控件大小变为和实线矩形一样，这样就会让用户体验到很好的拖放设计效果。

画实现矩形和虚线矩形在准备阶段的步骤并不一样，但最终这两种动态矩形都是通过 Yinhe.XReport.Report 类中的 DrawRectangle 方法画成的（通过传递的样式参数来区分具体画哪种矩形）。在报表设计器面板上画动态矩形时需要进行特殊的处理，因为这个矩形可能跨越很多控件，如果用画矩形的一般方法可能会被控件盖住。因此为了把动态矩形画在最上面，并且还要反衬托动态矩形，则需要用 System.Windows.Forms.ControlPaint.DrawReversibleFrame 方法。照 msdn 的说法，这个函数的作用是在屏幕上的指定边界内[14]，按指定背景色绘制处于指定状态的可逆框架。这个函数最大的好处就是再次绘制同一框架会逆转该方法的结果，特别注意，进行逆转时引用的框架必须和进行绘画时引用的是同一个框架。就是说通过这个方法画出一个矩形后，用同样的参数再把这个矩形画一遍就能取消刚画出的矩形。利用这一点就能够实现画动态矩形的目的了。

- 手柄矩形

手柄矩形是依附在选中控件四周的八个小矩形，通过拖拽手柄可以调整控件的尺寸，以达到拉伸控件的效果。在选中控件四周画出八个小矩形，让用户知道控件已经被选中，只要想选择某个或某及个控件就会来调用这部分代码。选中某个控件在代码中是表现为把控件加入到 alControl 列表中，而在与用户交互的界面中，则表现为在控件四周画出八个小矩形，下面介绍如何实现：

- DrawGrabHandle_ChooseControlsExtreme()：真正在函数四周画矩形的操作不在这个函数中，此函数的工作是对 alControl 列表中的所有控件进行遍历，如果里边的控件数目大于 1 且标识控件为空时，就把 alControl 中的第一个控件设为标识控件。如果有标识控件了，那就通过调用函数画出此控件四周的八个小矩形，且让这些小矩形的背景为黑色。以上条件都不满足了自然就只画出控件形四周的八个背景为白色的小矩形。
- DrawControlRec(Control control,Color backColor,Color borderColor)，此函数根据特定控件的位置坐标算出四周每个小矩形的位置，然后调用八次 DrawRec(int x1,int y1,Color backColor,Color borderColor)函数画出八个小矩形。
- DrawRec(int x1,int y1,Color backColor,Color borderColor)是真正执行画矩形操作的函数。调用了 System.Drawing.Graphics 中的 DrawRectangle 函数。

图形系统能支持通过鼠标在设计面板上进行可视化的拖放操作，说到底就在处理两种鼠标按键事件：设计面板（即 page）上的鼠标事件，还有报表控件上的鼠标事件，只要把这两种鼠标事件讨论清楚，也就介绍清楚了整个 Web 报表

生成器的可视化设计功能。下面分别来讨论这两种鼠标事件。

4.3.1 报表设计面板中的鼠标事件

这部分主要完成添加控件、拉伸控件、框选控件等功能，鼠标事件分移动、按下左键、按键弹起三种。

1) **Panel_MouseMove 鼠标移动事件**。

这个事件有两种情况：

- 没有按键的移动：这时候仅仅负责鼠标形状的更改，且这种情况肯定在有按键同时移动的情况前发生，所以在进行可拉伸式的画控件时，鼠标形状已经通过 ChangeArrow 函数来改变：
  - ✧ dragcontrol != "Arrow"，即已经点了画控件按钮，则把鼠标在 page 上的形状改为十字行。
  - ✧ page 上有选中的控件时，控件四周已经有了八个小矩形，每个小矩形的边长为 6，当鼠标移动到相应小矩形范围内时，鼠标就变为相应的形状，分别有左上、上中、右上、左中、右下、下中、左下、左中八种状态。然后把标记的状态用字符串形式传给 MouseArrow，在对控件进行拉伸更改大小时便会用到。
- 有鼠标左键的同时移动：
  - ✧ MouseArrow == "None"时进行框选，在 page 上画一个虚线框的矩形，画的时候有边界维护，即鼠标移出 page 时矩形框在相应方向上不再增大。
  - ✧ MouseArrow 不为“None”时，即进行拉伸控件操纵，函数为：ConditionStretchControl(string MouseArrow, int addx, int addy)。拉伸控件时把拉伸位置分为 4 种情况：上、下、左、右。拉伸的时候要特别注意其边界控制。拉伸完后通过 ReiniRec(int adx,int ady,int adwidth,int adheight)重新初始化矩形组 slRec 的位置和大小。slRec 肯定是和被选控件列表对应的，因为控件只有被选中了才能进行拉伸操作。拉伸过程中 page 上动态移动的矩形框是 ReiniRec 函数调用 DrawRectangle(int x,int y,int width, int height, FrameStyle style)函数来实现的。等拉伸完毕之后总会有一个 page 上的 MouseUp 事件，接下来真正对控件的重绘便是在 MouseUp 时发生的。

2) **Panel_MouseDown 鼠标按下左键事件**

这个函数主要负责以下几个操作：

- MouseArrow == "None"：意味着仅仅在 page 上单击一下，这时就对所有选中的控件进行取消选择操作。alControl.Clear()，slRec.Clear()；
- MouseArrow != "None"：意味着有对控件进行拉伸的前兆，因此在单击

下的时候就对所有选中的控件进行备份，因为当下一次鼠标弹起的时候可能已经是另外一种状态了。备份的函数为 BackupSelectedComponents()；

- 此事件函数的另一个作用是记下鼠标按下左键时的坐标，此时记下的这个坐标在 Panel_MouseDown 和 Panel_MouseMove 两个函数中会用到。

3) **Panel_MouseUp 鼠标弹起事件**

这个事件处理函数内容相对比较多：

- 右键弹出菜单
- 框选控件：进行框选时，新建了一个坐标(x,y)，不管是向左、向右、向上、向下框选时，都保证这个坐标是框选矩形左上角的顶点，这样方便后面画控件和矩形，而且在进行框选矩形的边界限定时也用到了这对值。框选的最后，即弹起左键时，便对框选矩形范围内的控件进行选定。ControlContainer(x,y,width,height)；它调用的两个函数分别表示判断控件是否在此范围内，有两种情况：
  - ✧ InvoleControl(Control control, int x1, int y1, int x2, int y2)：控件是否被框选矩形压着
  - ✧ CircleControl(Control control, int x, int y, int width, int height)：控件完全在框选矩形内部。

  以上两个函数判断满足时便都会调用 alControl，slRec 的 add()函数然后加入控件。
- 画新控件：标志为 dragcontrol != "Arrow"；调用 Add_Control(int cx, int cy, int width, int height)函数：先清空 alControl 和 slRec，
  - ✧ dragcontrol == "Arrow" 返回；
  - ✧ dragcontrol == "Label" 使用了 DesignerTransaction；说明画了一个 Label，所以调用 CreateLabel(int cx, int cy, int width, int height)函数，然后此函数中按顺序进行以下工作：
    - ➢ 创建好控件对象。
    - ➢ 自动给控件命名，为了防止控件名重复，命名方式为在 Label 后加从 1 开始的数字，每得到一个名字后就去检查已有控件中是否有这个名字，如果有，则把数字加 1，再去检查。如果没有这个名字，则控件就以这个名字命名。
    - ➢ 为控件设定初始文本、初始背景颜色、初始位置、初始大小、初始宽度、高度、初始边框、初始对齐方式。
    - ➢ 为控件的鼠标按下、弹起事件挂载了事件处理函数(control_MouseDown 和 control_MouseUp)以为后续处理做好准

备。

- 接下来把控件添加到 page 的 Controls 属性中，同时添加到 alControl 中，再在控件的四周画出八个小矩形。
- 通过 WriteLabelXml(System.Windows.Forms.Label c)函数将这个 label 的属性序列化，然后添加到 xcSchema 中。

- 拉伸控件完毕，标志为 MouseArrow != "None"：进行拉伸就是在对控件进行更改操作，所以拉伸完毕后要报更改信息保存回 xcSchema 中。

4.3.2 报表控件上的鼠标事件

报表控件上的鼠标事件主要完成对控件的单选和对控件的拖放功能：

1) MouseDown 鼠标按下事件

- 未按下 Shift 或 Ctrl，意味着要选择单个控件，或者在多个控件之间切换标识控件，先判断选中控件集合 alControl 中是否包含当前控件
  - alControl 中包含当前控件的情况：如果 alControl 中就只有当前控件一个，则标识控件设为 null，因为一个控件的时候不需要标识控件，然后再把当前控件四周的八个小矩形重新画一遍。如果 alControl 中有多个控件，那要做的就是切换标识控件，所以先判断一下标识控件是否存在，存在的话把它四周的八个小矩形重画为白色背景。接着就把当前控件设为标识控件，然后把当前控件四周的八个小矩形重画为黑色背景，这样标识控件就切换为当前的控件了。
  - alControl 中不包含当前控件。这一过程的最终结果显然是要让 alControl 中只包含当前控件一个，界面上的效果是只让当前控件四周有八个小矩形。所以先要判断 alControl 列表中选中控件个数是否不为 0，不为 0 了自然就把 alControl 列表和 slRec 清空，然后刷新 page。接下来把当前控件添加到 alControl 中，同时在 slRec 中也进行相应的添加。然后判断当前控件是否包含在锁定控件列表 alControlLocked 中，如果不包含，则仅仅重画当前控件四周的八个小矩形。包含的话，就是在当前控件四周画锁定控件矩形框。
- 按下 Shift 或 Ctrl：添加选中控件或者去除选中控件，也是通过判断 alControl 中是否包含当前控件来确认是要添加控件还是取出控件：
  - alControl 中包含当前控件，所以目的是从 alControl 中去除当前控件，但这里还有个细节要处理，就是如果当前控件是标识控件，则把当前控件去了后该怎样确定标识控件。首先把当前控件从 alControl 和 slRec 中去除，接下来进行重绘，就是让当前

控件所在的一个矩形区域（同个当前控件的位置和宽、高，很容易找到一个适合的矩形区域）无效，然后对 alControl 中的所有控件进行重绘。进行到这一步后，如果 alControl 中控件个数为 1 或 0，则让标识控件为 null，否则把 alControl 中的第一个控件设为标识控件，然后把此控件四周的八个小矩形画为黑色背景。刚才的过程中，如果不把标识控件设为 null 的话会出现一个控件也有四周八个小矩形为黑色的情况。

- alControl 中不包含当前控件，此时目的是把当前控件加入到 alControl 列表中。如果当前选中控件数目为 0，则仅仅只是选中当前控件，即要让标识控件为 null，然后在当前控件四周画出八个小矩形。如果当前选中控件数目不为 0，则先判断标识控件是否为 null，然后把它四周的八个小矩形画为白色背景的。再把当前控件设为标识控件，并把它四周的八个小矩形画为黑色背景。最后不管 alControl 中控件数目是否为 0，都把当前控件加入到 alControl 和 slRec 中。

✧ 不管是否按下 Shift、Ctrl，都会在最后执行如下过程：求出所有控件四个脚的最大值和最小值：Canclulate_Control_MaxMinValue()，对选中的控件进行备份：BackupSelectedComponents()，显示选中控件的属性：SetSelectedComponents()。

2) MouseMove 事件

主要看移动的同时有没有按下鼠标左键，按下的话就是拖动选中的控件，拖动选中控件又是一个比较重要的过程，这个过程完全通动态的画一个矩形框来模拟， ControlPaint.DrawReversibleFrame(Rectangle rectangle, Color backColor, FrameStyle style)：在屏幕上的指定边界范围内，按指定背景色绘制处于指定状态的可逆框架。通过这个函数在每一次画出矩形框时取消上一次的矩形框，这样就有了拖动效果[11]。

3) MouseUp 事件

控件上的 MouseUp 事件主要标志着结束拖动控件操作，调用 ReiniControl()函数把控件放到最终的矩形框内。这样就产生了拖动控件的效果。在 MouseMove 的过程中，每移动一次便已经把 slRec 中控件对应的矩形框进行了重设，所以在 ReiniControl()函数就直接把控件放到 slRec 中所保存的矩形框中。也就是把控件的大小和位置与矩形框对应起来。对控件进行拉伸或移动的时候 slRec 的作用便充分体现了出来。进行完这个操作之后，自然又会调用到几个底层函数：对属性框的操作对应的是 SetSelectedComponents()；画出所有选中控件旁边的八个小矩

形对应的是 DrawGrabHandle_ChooseControlsExtreme()；

4.4 打印功能实现

4.4.1 设计元素转换

在进行报表打印时，重点是设计元素的 Tag 转换。Tag 是打印的通用类，由它完成每种标签的输出和打印。在此之前，应该把设计元素转换为 Tag 对象。系统中定义了一组控件转换函数，专门负责设计元素的 Tag 转换。

- ConvertLabel(System.Windows.Forms.Label label, int PagNum)：负责 Label 标签元素的转换，先判断是否是页码，如果是就进行页码处理；如果不是页码，则再判断是否是数据绑定，如果是则取出数据并格式化；如果不是则按一般 Label 处理，当做文本转换。
- ConvertLine(ReportLine line)：负责线段元素的转换，此过程比较简单。
- ConvertRect(ReportRectangle rect)：矩形元素的转换。
- ConvertPictureBox(System.Windows.Forms.PictureBox pic)：图形元素的转换。
- 检查框元素的转换分两部分：
  - ConvertCheckBox_Choose(CheckBox checkBox)：实现对图形的转换，对于选中还是不选中要采用两个不同的图像，这个转换其实是把 CheckBox 当做了图形元素 PictureBox 处理。
  - ConvertCheckBox_Main(CheckBox checkBox)：这个过程把 CheckBox 中的剩余部分当做 Label 来处理。包括字体、背景色、前景色等。
- ConvertTrueDataGrid(C1TrueDBGrid TruedataGrid, int startRow, int endRow, bool IsEndPage)：这个过程相对比较复杂，由于 DBGrid 是表格形式的，所以可以把每个表格单元看作一个 Label。最终到底需要产生多少个 Tag 呢？首先看是否有类标题，如果有，则说明存在一行列标题。列标题的 Tag 高度是 DBGrid 的 ColumnCaptionHeight 属性值，宽度是每列的 Width 属性值。接下来是单元格的转换，首先要知道的是需要生产多少个单元格，每页有多少个单元格，共多少页。由于总条目数（总行数）是知道的，它由 Count 属性确定。所以就得到了这几个公式：每页的行数=(DBGrid 高度-列标题高度)/行高，页数=Count/每页的行数，每页单元格数=列数*每页行数。如果 DBGrid 元素有页合计属性，则每页应该少一行数据，多一行页合计栏。页合计栏的高度由 ColumnFooterHeight 属性确定。DBGrid 元素还有一个总合计属性，总合计只能打印在最后一页。总合计的高度与页合计的高度相同。所有的过程中每次产生 Tag 时都需要考虑字体、边界、背景颜色、字体颜色、文

本对齐等[10]。

4.4.2 分页打印

当数据内容在一页内打不下时，就需要分多页才能打印完。因此，需要把数据按页面的大小分为一页一页的送往打印机打印。分页打印一般是对于带有网格设计元素的打印报表进行分页。

打印报表类中有几个 int 型变量用来控制分页，其中 totalPages 是总页数，startPage 是打印起始页，endPage 是打印结束页。可以通过控制 starPage 和 endPage 来实现任意页的打印[9]。

- 开始打印时，先通过 Convert(startPage)函数进行标签转换，然后再打印标签。
- 将 startPage 值加 1。
- 判断 startPage 是否大于 endPage，是的话结束，否的话继续刚才的过程，打印下一页。

进行分页时，标签转换的主要工作都针对的是网格元素，已经计算出 tatalPages 总页数和每页的行数后，根据不同的 startPage 就能按顺序对任意页进行打印。

4.4.3 页面设置

Web 报表生成器中的页面设置功能给用户的报表打印提供了很大的扩展性，系统已经提供了很多的纸张类型供用户选择，如果用户觉得不满意的话还能进行自定义纸张类型，比如，现在数据库中有张一千条记录的表，用户也有张 1m 长的纸，用户想把这一千条记录全都打印在这张纸上，而不进行分页。此时用户就能在页面设置对话框框中新建一种长度为 1m 的纸张类型，选定之后，报表设计面板也能自动调整到相应的大小，设计好报表之后就能进行打印了。

添加自定义纸张类型是通过 Report 类的 SetCustomPaperSize()方法实现的。删除自定义纸张类型使用的是 DelCustomPaperSize()方法。

第五章　总结与展望

传统 Web 报表系统中，很少有在浏览器中支持用户拖放、拉伸控件等可视化的设计操作。同时，传统 Web 报表系统仅仅依靠浏览器自带的页面打印功能，不能精确控制客户端的打印机，因此很难完成复杂的报表打印任务。或者通过 ActiveX 来完成，但其安全性不高。基于这样的情况，本文的主要研究工作如下：

1. 采用了.NET 的移动编码技术，把 WinForm 嵌入到网页中，很好的解决了浏览器中可视化设计和报表复杂打印的问题。
2. 提出了 XML 报表控件的概念，将 Web 报表可视化设计中使用的控件都以 XML 的形式保存，实现了 XML 报表控件的即插即用。
3. 设计并实现了图形系统，用来支持报表可视化设计。
4. 实现了很多在报表设计过程中用到的控制功能，包括多种对齐方式、大小控制、复制剪切粘贴、撤销和重做等等。
5. 设计并实现了数据绑定模型。
6. 打印功能，包括支持打印分页、支持用户自定义打印等。

一个完整的报表系统包含了从数据获取到数据分析，再到数据展示的一整套流程，由于时间和精力有限，在课题研究过程中是不可能把整个流程中所有的功能都完整实现的，因此，课题着重研究并实现了报表的设计和最终的报表打印两大功能。而对于报表中的某些功能，则相对弱化了，其中未能完整实现的功能包括：

1. 公式定义：现阶段只能进行简单的累加操作，还需进一步丰富 Web 报表系统的公式计算能力。
2. 报表样式相对比较简单：现阶段主要实现了简单表、列表、主从表三种报表，而对于分组表、嵌套表、交叉表等复杂的报表则还需进一步进行研究。

经过不断的改进与完善，相信 Web 报表生成器的功能将会越来越强大。

参考文献

[1] W3 Schools．XML Tutorial [EB/OL]．http://www.w3schools.com/, December 1999．

[2] Thiru Thangarathinam．Hosting .NET Windows Forms Controls in IE [EB/OL]．http://www.15seconds.com/issue/030610.htm．2004-11-14．

[3] Jack Herrington．Code Generation:The one page guide[EB/OL]．http://www.codegeneration.net,2003．

[4] Zvon．XPath Tutorial [EB/OL]．http://www.zvon.org/xxl/XPathTutorial/General/examples.html, 2000．

[5] 陈俊先，高阳．.NET 软件设计新思维——像搭积木一样搭建软件[M]．北京：电子工业出版社，2009．2—229．

[6] Chris Sells,Michael Weinhardt 著．汪泳 译．Windows Forms 2.0 程序设计[M]．北京：电子工业出版社，2008．259—289．

[7] Christian Nagel,Bill Evjen,Jay Glynn 等著．李铭 翻译．黄静 审校．C#高级编程（第 6 版）[M] ．北京：清华大学出版社，2008.10．21—529．

[8] Brian Noyes 著．汪泳 译．Windows Forms 2.0 数据绑定[M]．北京：电子工业出版社，2007．81—176．

[9] Jason Bell，Benny B，Johansen 等著．袁勤勇，郑魏 等译．Windows Forms 高级编程[M]．北京：清华大学出版社，2002．533—548．

[10] David Richard Kalkstein Deloveh，William Sempf 等著．陈安全，刘莉 等译．Viusal Studilo .Net 高效编程[M]．北京：清华大学出版社，2002．118—142．

[11] 康博．C#高级编程[M]．北京：清华大学出版社，2002．568—635．

[12] 张能立．Web 动态报表的实现[J]．计算机应用与软件，2004，21(4)：112—114．

[13] 毛尧飞，崔伟．C# XML 入门经典[M]．北京：清华大学出版社，2004．25—226．

[14] 杨宏伟等．C#程序员开发手册[M]．北京：科学出版社，2005．113—244．

[15] 毛勇，陈奇，俞瑞钊．XML：一种将广泛应用的数据格式描述语言[J]．计算机应用研究，1999，09．

[16] 高洪．XML 技术内幕[M]．北京：清华大学出版社，2006．1—18．

[17] 李勇平．.NET WINDOWS 应用开发教程[M]．北京：希望电子出版社，2004．221—268．

致　　谢

值此论文即将定稿之际，我谨向所有关心过我，帮助过我的老师、同学朋友、家人表示衷心的感谢。

首先感谢张坤龙老师，能让我接触到这样一个毕业设计课题，使我从中学到了很多很多的知识。张老师渊博的知识、敏捷的思维、踏实的工作作风、乐观的生活态度、谦和的为人、深厚的学术造诣、严谨的治学态度深深的感染了我，在此谨向张老师致以由衷的感谢和最崇高的敬意！

感谢我的父母，正是由于他们在我学习和生活中给予的无微不至的关怀和不断的鼓励，我才能够顺利完成十多年的学业。我无法用语言表达对他们的感谢之情。

感谢我的同学，在天津大学学习期间，与他们一起学习、成长的经历将使我终生难忘！