**智能报表系统的设计与实现**

**摘 要**

**关键词：**

**ABSTRACT**

**Key words：**

1. **绪论**
   1. **研究背景及意义**

在企业的信息化建设中，报表占据了非常重要的地位，越来越多的智能报表系统应运而生。企业管理的基本措施和途径，是企业的基本业务要求，也是实施BI战略的基础。报表可以帮助企业访问、格式化数据，并把数据信息以可靠和安全的方式呈现给使用者，使企业管理者可以深入地洞察企业的运营情况。

大型企业通常采用树状的组织架构，总部通常下辖多个分部，而各个分部又可能进一步管辖多个单位，每个单位甚至又可以进一步细分为部门和班组。在这种树状的组织架构当中，不同层次的机构之间存在着复杂的数据交换，协调企业的运营。显然，为了在构成庞大企业的众多细胞单位之间正确地共享数据，必须定义明确的数据交换和共享协议。

根据所采用的数据交换协议，组织机构之间的数据交换模式通常可以分为两类：一类是数据提供者和消费者根据系统提供的确定不变的协议共享数据；另一类是，数据提供者和消费者根据他们临时自定义的协议共享数据。例如，传统的基于关系数据库的解决方案预定义数据库模式，数据的生产者根据系统中给出的表模式进行数据填报，而数据的消费者则可通过结构化查询获得其感兴趣的信息。然而，企业的数据交换内容并非一成不变。例如，总部可能出于某种管理需要，要求各个分部上报数据库协定之外的数据。这时总部通常会下发一些表格，要求分部上报。如何支持用户定义、分发、汇总和管理这些灵活定制的表格，获取企业生产过程中的细节数据，是传统企业信息管理系统面临的难题。

该在线智能报表系统集报表模板的设计，报表的下发，数据的上报，数据的汇总和管理等功能于一体的灵活可定制报表的轻量级的在线智能报表系统，适合各种大中小企业的使用。该系统不但解决了传统的基于数据源的报表系统存在的问题，并且为后续在这一领域研究的学者作参考。

* 1. **国内外发展现状**

近年来，国内外各个领域对于智能报表的研究在逐渐增多，有主要采用C/S结构的传统报表工具，也有采用B/S结构的面向Web的报表系统。

2003年，郑州大学教授王文义等人针对信息管理系统开发中普遍存在大量动态变化的报表问题，利用PowerBuilder（简称PB）中Data Window方式和数据库管理模式，设计开发一个可由用户通过自定义生成任意形式报表的智能化系统[1]。它可对多表、多字段进行任意组合查询、统计，提供对内部数据源查询报表和外部数据源统计报表的支持。2004年，鲁东大学教授苏子林等人探讨了中国式报表的常见结构和实现过程的合理分区方法，详细阐述了智能报表系统实现的几个概念，并提出了系统框架和实现方案[2]。2005年，苏子林教授等人在介绍了智能报表系统的概念基础上，探讨了报表生成智能代理的集成知识表达方法，提出了报表框架匹配的评价函数和相应的推理算法，并给出了基于多代理的原型系统[3]。2007年，太原理工大学教授崔冬华等人介绍了报表的通用格式，阐述了基于Web的智能报表系统的设计与实现，重点分析了系统实现中的一些难点问题[4]。2012年，工程师胡佐和肖文研究了基于模板技术的智能报表系统[5]。他们在文章中描述了系统各个组成部分，并从报表实际需求出发，讨论了实现基于模板技术的智能报表系统的基本原理及技术架构。智能报表系统能够实现报表开发的可视化以及模板化，一定程度下降低了开发的难度，减少了实施和运维的成本，这在电力行业有着广阔的前景。

国外在智能报表方面的研究主要集中在报表工具的设计上。SAP公司开发的商务智能软件Crystal Report[6]，水晶报表是目前业内较专业功能较强的报表系统。它除了强大的报表功能外，最大的优点是实现了与绝大多数流行开发工具的集成和接口。但是就其目前使用的情况来看，又有许多的缺点。一是不能很好的处理多数据源关联交叉的报表样式，水晶报表的处理机制是通过写很复杂的视图或利用子报表来解决，这样的处理方式具有很大的局限性，无法满足格式上的要求。二是国内报表习惯于excel的表格格式，不能很好的处理大量的excel文件，所以这也成为水晶报表的一个限制因素[7]。微软在2003年推出为了代替Crystal Report的报表开发工具Reporting Service[8]。而且微软似乎还不满足于一个报表开发工具那么简单。Reporting Services是一种基于服务器的新型报表平台，部署在Microsoft SQL Server 2000基础上，可用于创建和管理包含来自关系数据源和多维数据源的数据的表格报表、矩阵报表、图形报表和自由格式报表[9]。可以通过基于Web的连接来查看和管理创建的报表。Reporting Services提供了一套完整的服务、工具和应用程序编程接口（API），即使不是程序员也可以使用Reporting Services。可以使用Reporting Services中包含的应用程序和工具来制作、发布和管理报表。此外，还提供了支持报表生存周期的各个阶段的工具或应用程序。程序员可以使用API将报表功能扩展或集成到自定义解决方案中[10]。

国内外关于智能报表的研究大多是基于关系数据库的，数据的生产者根据系统中给出的表模式进行数据填报，而数据的消费者则可通过结构化查询获得其感兴趣的信息。然而在支持用户定义、分发、汇总和管理这些灵活定制的表格，获取企业生产过程中的细节数据等方面，国内外的研究仍有所欠缺，在这一方面存在研究价值。

* 1. **研究内容**

本文主要研究内容为设计并实现一个基于B/S（浏览器/服务器）架构的web应用系统。将RESTful web services应用于系统服务端，将AngularJS框架应用于系统浏览器端，在二者的结合之下实现智能报表系统，以提高报表系统的整体性能，提高开发效率和系统的可伸缩性，同时降低客户端与服务器之间的耦合率，更好地将智能报表系统服务于企业。具体内容包括：

1. 介绍实现系统所涉及到的关键理论及技术，包括REST架构风格，AngularJS框架技术等。
2. 对系统进行总体分析，按照其功能模块进行需求分析。
3. 研究实现基于REST风格的服务，包括资源的划分，资源URI的设计等。
4. 设计客户端的整体架构，并研究使用AngularJS框架来实现系统的浏览器端。
   1. **论文组织结构**

本文主要分为七个章节，第一章为绪论，主要阐述本文研究背景和意义，后续各章节具体安排如下：

第二章为相关理论与技术。该章详细介绍了本文研究所涉及到的两个关键技术：RESTful web services和AngularJS技术。对比了传统的SOAP风格与REST风格的优劣后，阐述了选择基于REST风格的web services来实现系统的服务端的原因。并且研究了AngularJS框架，根据其特性，选择使用该框架设计浏览器端。

第三章为系统整体分析。该章对智能报表系统做了详细的需求分析，依据其功能模块的使用人员不同从两个大的方面分析了系统的需求，并对每个功能模块提出了模型状态图。

第四章为系统设计。该章首先提出了服务端的总体框架结构，然后根据该结构设计了REST风格的服务中的资源，资源的URI，提供给每类资源的HTTP方法等。并阐述了资源请求和处理以及返回响应的流程。结合AngularJS是基于JavaScript的MVC框架的特性，详细阐述了系统客户端在Model-View-Controller各层的设计。

第五章系统的实现。根据四、五章的设计，在这章详细介绍了系统的实现，包括具体的实验环境，实现平台以及系统的实现结果。

第六章为总结与展望。该章对本文在智能报表系统的设计与实现做了总结，分析了论文的不足之处，并对将来在智能报表领域的研究做了展望。

1. **相关理论及技术**

智能报表系统是基于B/S架构的应用，可利用Web Services技术设计系统的服务端，由于传统的基于SOAP的Web Services不断扩充其协议，导致在处理效率方面性能下降，同时降低了其易用性。而基于REST风格的Web Services则具有简单性、松散耦合、高度可伸缩性以及良好的性能等特点，因此利用它来实现报表系统的web服务。而近年来刚被提出的AngularJS框架作为一套成熟的JS框架，使用它来构建应用程序，同样可加快开发周期，并且能够提高系统的性能以及可伸缩性。本文将使用RESTful Web Services来设计智能报表系统的服务端，并且将AngularJS技术用于浏览器端的设计。

* 1. **REST架构风格的Web服务**
     1. **Web Services技术**

Web Services是一个平台独立的，低耦合的，自包含的，模块化的应用，它能够在网络上被描述、发布、定位和激活。开放通用的协议标准通常为HTTP，XML（标准通用标记语言下的一个子集）等协议使它对所有的平台和体系结构都是可行的。[Web Services技术应用与探讨]它在不同的软件应用之间提供了标准的交互方式，使原来各孤立的站点之间的信息能够相互通信、共享，而不用考虑应用程序的实现技术以及运行平台。[Web Service及其关键技术研究综述]

Web Services的体系结构是基于服务提供者、服务请求者、服务注册中心三个角色和发布、发现、绑定三个动作构建的。Web服务的体系结构如图2-1所示，

图2-1 Web服务的体系结构图

服务注册中心

服务请求者

服务提供者

发现

绑定

发布

服务提供者作为服务的拥有者等待其他服务和用户请求服务；服务请求者即客户向服务提供者发送请求以获取服务；服务注册中心担任服务中介的角色，将Web服务提供者和Web服务请求者联系在一起。服务提供者定义服务描述并将服务发布到服务注册中心；服务请求者通过发现操作在本地或服务注册中心检索服务描述，查找到后根据服务描述绑定到对应的服务提供者，调用所需的服务。

Web Services的主要目标是跨平台的可互操作性。为了实现这一目标，Web Services完全基于XML（可扩展标记语言）、XSD（XML Schema）等独立于平台、独立于软件供应商的标准，是创建可互操作的、分布式应用程序的新平台。

使用Web Services作为中间件，可以实现客户端和服务器之间跨防火墙通信。通过Web Services，应用程序可以用标准的方法把功能和数据"暴露"出来，供其它应用程序使用，实现应用程序的集成。Web Services也是B2B（Business to Business）集成成功的关键。使用Web Services来实现B2B集成可以轻易实现互操作性。只需将商务逻辑设计成为Web Services，就可以让任何指定的合作伙伴调用这些商务逻辑，而不用管他们的系统运行在何种平台上以及使用何种开发语言。这样就在极大程度上减少了B2B集成的时间和成本。并且利用Web Services技术能够实现代码重用的同时，重用代码背后的数据。

* + 1. **REST架构风格的Web Services**

Web Services有多种方式，其中有两种比较主流的方式，一种是传统的SOAP协议方式，另一种就是本文使用的REST方式。

SOAP（简单对象访问协议）是交换数据的一种协议规范，是一种基于HTTP和XML（标准通用标记语言下的一个子集）的协议，它被设计成在Web上交换结构化的和固化的信息。SOAP最早是针对RPC的一种解决方案，简单对象访问协议，很轻量，同时作为应用协议可以基于多种传输协议来传递消息（HTTP，SMTP等）。

REST(Representational State Transfer，表述性状态移交)是由Roy Fielding博士2000年在其博士论文中首次提出，作为分布式系统设计的一种架构风格，它使得HTTP协议最初的设计思想得到了最大的发挥，让人们真正理解HTTP本来面貌。在REST的名称“表述性状态转移”中，省略了主语。表述性指的是资源的表述性。所谓“资源”，就是网络上的一个实体，或者说是网络上的一个具体信息。可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的实在。用一个特定的URI（统一资源定位符）指向每种资源，要获取这个资源，只需访问它的URI。因此URI就成了每个资源的地址或独一无二的识别符[3]。对资源的获取、创建、修改和删除四种基本操作对应于HTTP协议提供的GET、POST、PUT和DELETE方法[4]。当一个客户端对一个资源发起GET请求时，服务器会以一种有效地方式提供一个采集了资源信息的文档作为回应。这就是表述——一种以机器可读的方式对资源当前状态的说明[5]。

总的来说，在REST的概念中，整个Web被看作一组资源的集合，资源由URI标识，对资源进行的操作由客户端指定的URI和HTTP协议动词的组合来实施，同时将资源和资源的表示分开，为构建可扩展、简单性、可移植和松耦合的Web程序提供了一个架构上的准则，REST这种基于资源的设计改变了传统的基于服务的设计思想，可以降低开发的复杂性，提高系统的可伸缩性，提高资源的访问效率[6]。

随着SOAP作为Web Services的广泛应用，在其后续的发展过程中，WS-\*一系列协议的制定，增加了SOAP的成熟度，但同时也增加了SOAP的负担，阻碍了Web Services的推广和应用。REST则是将HTTP协议的设计初衷作了诠释。HTTP协议诞生时设计者其实是把它作为一种应用协议来考虑的，但现今人们都将其作为传输协议来使用。而REST把四种对事物常用的操作创建（Create）、获取（Read）、更新（Update）和销毁（Delete）直接映射到HTTP协议中标准的POST，GET，PUT和DELETE方法。并且将这些事物抽象为资源，用URI来唯一标识，直接使用HTTP协议的响应状态码来标识服务器是否适当地处理了对资源执行的各种操作。这些都是将HTTP协议作为应用协议而非传输协议来使用的。表2-1为两种风格的web services在多个方面的比较，

表2-1REST和 SOAP系统的特性比较

Table 2-1 Comparison of feature between REST and SOAP system

|  |  |  |
| --- | --- | --- |
| 特性 | REST | SOAP |
| 复杂度 | 简单 | 复杂 |
| 耦合度 | 低 | 高 |
| 应用规模 | 大 | 中小 |
| 地址模式 | URL和服务一一对应 | URL和服务一对多 |
| 通用接口 | 4个HTTP方法 | 无 |
| 交互类型 | 人机交互 | 机机协作 |
| 缓存支持 | 是 | 很难 |

由上表可知，REST风格具有简单，易用，资源描述与视图的松耦合性等特性。因为其满足无状态通信的原则，可提高应用的水平扩展性。还可以通过缓存功能，提高服务器的性能。目前已经存在许多的Web服务开始采用REST风格设计和实现。例如，Amazon.com提供接近REST风格的Web服务进行图书查找；Newegg.com提供的Web服务也是REST风格的。因此，本文也选择REST风格的Web服务来设计智能报表系统的服务端。

* 1. **AngularJS技术**

AngularJS是一个基于JavaScript的开源MVC框架，它诞生于2009年，由Misko Hevery等人创建，后为Google所收购。它作为一款优秀的前端设计框架，主要用于支持单页面动态应用程序的开发，目的是为了简化开发过程和测试的复杂度。具体来说就是在编写web应用的过程中能够实时地观察到代码所产生出的效果，这样可以使得在开发大规模的应用时，程序依然可以扩展、维护和测试。使用AngularJS框架是非常方便的，只需将angular.js库在使用的页面中加载，并且在需要使用该框架的文档对象模型(DOM)上使用ng-app属性即可[动态web应用程序开发框架AngularJS的特性分析]。

AngularJS作为浏览器端的解决方案，主要是为了克服HTML在构建应用上的不足而设计的。在浏览器端的技术中，HTML通常被作为静态网页的解决方案。它是一种声明式标签语言，但要构建WEB应用的话，静态的HTML标签语言就如法满足用户需求。因此需要一种能够兼容HTML的技术，用来解决客户端实现动态网页的问题。

AngularJS是一种很适合用于编写大型应用的框架。使用它，可以加快应用程序的开发周期，而且它在组织应用的结构上也很有帮助。它作为Javascript框架，对模型层的完全掌控是它的特别的地方。并且AngularJS也是强大的，因为应用程序的核心就是数据，而各种应用之间的数据又有很大区别[简谈AngularJS在下一代Web开发中的应用]。

AngularJS可以对这些不同的、区别很大的数据进行整合，甚至于会用于数据的存储。动态Web应用设计是一个可以使用HTML作为模板的语言，通过扩展HTML的语法，能够更清楚、简洁地构建Web应用组件。它最大的创新点在于，利用依赖注入和数据绑定，可以使得开发过程中不再需要写大量的冗余代码，因而使得应用开发周期缩短，同时降低了开发的难度。

AngularJS存在如下的四个特性，也是本文选择利用该技术来开发系统浏览器端的主要原因：

1. 跨平台。能够构建渐进式Web应用和原生移动应用，结合访问原生操作系统API的能力，创造能在桌面环境下安装的应用，横跨Mac、Windows和Linux平台。
2. 速度和性能高。AngularJS可以把模板高效转换成代码，可在服务端渲染应用的首屏，支持多种服务器，加快了通过SEO来优化站点的进程；提供了自动拆分代码的功能，为用户单独加载它们请求的视图中需要的那部分代码，使得应用性能更高。
3. 高生产率。通过简单而强大的模板语法，快速创建UI视图；方便的命令行工具，快速进入构建环节、添加组件和测试，然后立即部署；在常用IDE和编辑器中获得智能代码补全、实时错误反馈及其它反馈等特性。
4. 完整开发故事。框架包含的Karma，Protractor等测试工具使得测试更简单快捷；通过AngularJS中直观简便的API只要非常少的代码即可创建高性能复杂编排和动画时间线；通过支持ARIA的组件、开发者指南和内置的一体化测试基础设施，能够创建出具有完备可访问性的应用。
   1. **本章小结**

本章主要介绍了系统开发中用到的相关理论以及关键技术。首先介绍了Web Services的概念，通过对比SOAP和REST这两种主流的Web Services设计风格，阐述了选择REST风格的原因。然后介绍了浏览器端使用到的AngularJS框架的概念以及使用它来开发系统带来的优势，并且阐述了该框架的四个特性。

1. **智能报表系统的需求分析**
   1. **功能需求分析**

智能报表系统用于企业总部与各个分支机构之间数据交换与汇总。在分析了满足使用系统的不同用户的功能需求后，对系统的功能需求分析分为两个部分进行：普通用户需求分析以及系统管理用户需求分析。

系统的主要功能包括企业中的各个单位设计创建报表模板、分发报表模板至下属单位或协作单位、对所获得的上报数据进行汇总并持久化封存，形成企业管理的信息资源库。各个单位可以填写来自上级或协作单位的报表模板、上报企业运行数据等。

**3.1.1普通用户需求分析**

**（1）用户个人信息管理**

用户只有登录系统后才可进行后续操作，登录系统需要使用个人的用户名及密码，若登录成功，此时系统会将用户名、用户角色、登录时间、所在单位等信息保存在全局会话当中，在整个功能范围内有效，便于在系统范围内访问用户信息以及判定用户权限；若登录失败，则给出通知。

为方便用户使用，系统提供自动保存用户登录历史功能，用户可下拉选择曾经录入的用户名。登录系统后，普通用户可查看并修改个人信息，系统会根据用户角色赋予相应的权限，对于用户不可查看或者修改的信息，系统采取不展示或者只读的方式将用户个人信息展现出来。

**（2）报表模板管理**

模板定义了用户上报数据的规范。模板具有类别，例如财务报表、生产报表，生产报表又可以细分为生产情况完成表、生产成本表等。即模板的类别形成一个层次结构，而模板则是最下层类别中的元素。报表模板的状态图如图3-1所示，

单位可基于系统创建的报表类别树新增自定义的模板。每个单位可自主创建一系列模板。并且每个单位可以查找并使用其自己创建的模板或系统提供的模板，创建周期性下发任务或单个下发任务。该部分拟采用用友Cell插件，以Excel的界面风格设计报表模板的创建界面。对于报表模板的管理部分的设计，可更改报表模板的类型，模板下发之前，用户可对自己创建的模板进行删除和修改。模板下发之后，任何用户都不能对其进行操作包括打开、修改以及保存。若模板错误，则需撤销与之相关的所有下发任务后删除该模板或直接停用该模板。通过与企业内部人员的沟通，根据其业务背景，需要为系统提供一部分常用报表模板以便用户使用。



图3-1 报表模板状态模型图

Fig. 3-1



**（3）数据采集任务管理**

数据采集任务，即报表下发任务，是用户管理其数据采集工作过程的核心概念。数据采集任务可以分为两类：一类随机发布的单个采集任务，一类是周期性下发的一系列采集任务。当采集任务下发后，还包含对任务查询以及任务进度的管理。

a 单任务的状态模型

单任务状态模型图如图3-2所示，



图3-2 单任务状态模型图

1）用户根据随机性数据采集的需求，创建单个数据采集任务。

2）任务下发之前，用户可以对其进行删、改、查；

3）任务下发之后，用户可对其进行查看、发送用户通知、增补下发对象、删减下发对象、修改上报期限以及撤销整个任务，但用户不能直接删除任务。

4）下方对象上报完成之后，用户可以对该任务进行汇总或上报（有些任务是从上面转发下来的，需要汇总后上报）

5）下发任务被撤销后，相关各单位针对该任务上报过的数据则被标注为“过期”。

6）各单位拥有权限删除本单位的“过期”报表数据。

b 周期性任务模板的状态模型

用户负责创建周期性任务模板。系统自动根据周期性任务模板周期性地发布一系列单个数据采集任务。用户可以以管理单任务的方式对在周期系列中的任务进行查看管理，包括查看数据、发布用户通知等等。状态模型图如图3-3所示，

1）周期性任务模板下发之前，用户可对其进行删除、修改和查看。下发之后只能对其进行终止操作；



图3-3 周期性任务状态模型图

2）若用户希望启动被停用的模板，可以以模板为基础，创建一个新的周期性任务模板。

对下发任务的管理包括新任务的创建以及任务的查询和任务进度管理。

创建单个任务的流程为：首先，指定报表模板、选择下发单位、指定是否短信通知、采集数据的起止日期、上报数据的截止日期、预览。其次，点击保存按钮，将该任务保存至表中，将每个单任务标志为待下发状态；最后，用户可以对任务进行反复修改，并点击保存并提交，或找到任务直接提交。提交时，系统将针对每个单位自动创建上报任务记录并保存在上报表当中；或者用户在预览之后，马上点击保存并下发按钮，将该任务保存至汇总表中，将每个单任务标志为已下发状态，同时针对每个单位创建上报任务保存在上报表当中。

创建周期性任务模板的流程为：

1）指定报表模板、选择下发单位、指定是否短信通知、指定周期性任务的启动日期、预览；

2）点击保存至TCyclicTaskTemplate。

3）系统每天检查TCyclicTaskTemplate表，根据每个周期性任务的周期类型，确定当天应启动的下发任务，创建该任务，将之添加到TCollectTask中，同时针对每个单位自动创建上报任务记录并保存在TReport当中。注意上述操作必须在同一事务内完成。

4）允许用户以已有模板为参照，创建新的周期性任务模板启用该模板之前，对其进行修改。

例如，服务端的精灵进程，可周期性地检查哪些待下发任务模板应被实例化，创建下发任务，并标记为已下发并自动生产一组上报任务记录。

任务查询及任务进度管理包含以下方面的功能：

1）用户应能够直观地浏览本单位已经下发的所有任务，包括未来可能下发的周期性任务。

2）用户可查看特定任务、发送用户通知、增补下发对象、删减下发对象、修改上报期限、甚至撤销整个任务。但是用户不能直接删除已下发的任务。

3）用户能够按时间、按编号、按对象、按报表类型、按周期性等条件检索一组性质类似的任务并对该组任务进行管理，例如统一发出催缴通知。

4）用户能够查看指定任务的明细，查看任务中已经上报的数据，可退回某些单位的数据，要求其重新上报，也可直接对某些存在明显错误或者上报单位已提出需要修改的上报数据进行修改。

5）用户能够撤销特定任务，同时需要通知上报单位该任务已撤销，相关数据已经“过期”，从而允许数据上报单位对其进行主动清除。

6）当存在上报任务但未在最晚上报时间之前上报该任务，或者当修改任务接近或超出期限后，按一定方式和频率向需要上报数据的单位发出催缴通知。

**（4）上报任务及上报数据管理**

企业各机构的单位既可以作为数据采集方，主动下发报表模板，采集数据，还可以作为数据提供方，填报数据。一般地，一个数据采集的报表模板会下发给多个不同的单位，即每个单位实际上接受了一个需要完成的数据上报任务，需要在系统中填报数据并将这些数据保存在数据库中。

单位的上报任务和数据有若干不同的状态。每个状态中所能接受的合法操作不同。其状态模型如图3-4所示，

上报任务的填写和管理功能有：



图3-4 上报任务状态模型图

1）系统根据数据采集任务中的采集对象列表，自动创建一系列数据上报任务。

2）单位能够查看本单位需要完成的所有上报任务

3）单位能够为指定任务填写数据，并上报。

4）单位能够自主清除无效的上报任务及相关数据

5）单位能够选择多个任务，集中上报。

6）允许用户清空现有填报数据，重新填写。

**(5)数据汇总管理**

针对同一报表模板进行填报所得的上报数据均可被汇总在一起，展现企业运营过程中的价值和规律。因此汇总表与报表模板以及汇总范围密切相关。报表模板和汇总范围就确定了一个汇总表。对符合同一模板的不同数据的报表汇总，就是将各单位数据报表中相同位置的单元格数据进行累加并保存在汇总表中的相应位置。

汇总数据报表的状态模型如图3-5所示，

用户创建汇总表之后，可以对其进行保存得到一个新汇总。可以审查新汇总，并提交封存申请。高级用户接到封存申请之后可以对普通用户提交的汇总进行审查、封存或者退回。用户可以对已经封存的数据提出解封申请，对解封后的数据经过修改后可以再次申请封存。



图3-5 汇总数据状态模型图

创建汇总时，可根据数据汇总方式可将汇总分为两类：即单任务汇总和单周期汇总。

1）单任务汇总。是针对一次特定采集任务所获得的上报数据的汇总。可定制参与汇总的上报数据单位。

2）单周期汇总。是针对一个周期性任务系列中所获得上报数据的汇总。可定制参与汇总的单个任务以及单个任务中的上报单位。

拟采用矩阵形式进行汇总的创建如表3-1所示，

表3-1 单任务、周期性汇总的界面设计示意图表

Table 3-1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 单位  任务 | A | B | C | D | E | F |
| T1 |  |  |  |  |  |  |
| T2 |  |  | [T2,C] |  |  |  |
| T3 |  |  |  |  |  |  |
| T4 |  |  |  |  |  |  |

上表中，每行的头表示一个以选定报表模板为模板的下发任务编号，例如T1表示第一个下发任务，T2，T3，T4就表示第2，3，4个下发任务。表的每一列的头表示参与上报数据的单位。例如A单位参与任务T1，T2，T3，T4。表中的每个单元格[T2,C]就表示C单位在T2上报任务中上报的数据。若单元格为空或其他特殊标志，则标示此单位在此次上报任务中的完成情况，例如空表示未参与上报。可在上表中选择制定单元格参与复合汇总或单任务汇总。

**3.1.2系统管理用户需求分析**

系统管理员负责系统的元数据的维护和管理。系统的元数据主要包括单位及其类别、用户权限指派策略、功能条目树以及角色定义策略、系统提供的默认报表模板类别以及相关报表模板、系统的状态等。系统中元数据的状态模型如图3-6所示，

系统的普通用户只能使用已经启用的元数据，而系统的管理员则能够在系统管理模块中管理所有元数据。



图3-6 系统元数据状态模型图

单位用户能够看到所有已经启用的报表模板，以及本单位定义的所有报表模板、下发任务、上报报表、汇总任务等，若模板已停用则高亮。

系统管理员将具体完成以下任务：1）单位及其类别管理；2）用户的单位权限指派；3）功能条目管理以及角色权限定义；4）报表模板类别以及报表模板管理；5）下发周期的定义与管理；6）系统状态和基本信息管理等；7）以超级管理员的身份而，完成用户所能完成的所有任务，包括下发、上报、汇总、封存等。

**（1）单位部门管理**

企业的单位包含一个或多个子单位或部门，而部门是仅包含一个或多个用户的末端单位。

对单位管理采用树形结构。在单位树形结构增加条目时，需指定新增条目为部门或单位。若在一个部门中创建子部门，则该部门升级为一个单位。部门升级为单位时，该单位中必须设置至少一个部门，默认将原部门中所有用户调整为部门中的用户。部门升级为单位后，则其新增的部门自动继承原部门的类别。系统正式启用之后，不能删除单位，可停用单位，允许增加停用单位的功能和标志。若单位停用，则其中的所有部门都将被停用。对单位和部门的管理，还需提供单位和部门的基本信息维护及查询功能。

企业用户所属的部门只能由系统管理员来修改。人员必须隶属于唯一的一个部门，允许管理员新增人员、修改其基本信息、根据其姓名或编号查询人员。

**（2）角色权限管理**

为了提高系统的可伸缩性，也为了能够更加方便用户的使用智能报表系统，在设计用户权限时，是以角色为单位分配权限的。用户被赋予角色后，自动拥有角色对应的权限。这样会更加方便系统管理员对用户权限的控制。

系统管理员可对角色进行管理，包括增加角色，删除角色或者修改某种角色所对应的权限等。除此之外，还需要对权限进行管理，以便用户为角色分配权限。

**（3）模板类别管理**

对模板的管理是智能报表系统的基础功能，模板设计的优劣直接影响数据收集的效率。因此首先系统会用提供大量的常用模板，用户在使用过程中也可能会新建各种模板。为报表模板分类能够更方便的管理模板，同时更易于用户使用。为此系统提供了模板类别管理的功能。模板类别也采用树形层次结构来进行管理。系统管理员可根据实际需要进行增加新的模板类别或者对现有模板类别进行修改和删除。当删除某个模板类别时，要将该类别下的所有模板所对应的类别修改为原始类别上一层的模板类别。

**（4）系统信息维护**

系统名称，系统的启用状态，系统数据的备份，系统日志以及其他系统信息都需要系统管理员来维护。

* 1. **非功能性需求分析**

虽然功能需求是对系统的一项基本要求，但并不是唯一需求[软件工程导论张海藩牟永敏]。本文设计的智能报表系统还应满足下述几个方面的综合要求。

**（1）性能需求**

智能报表系统集合了传统报表系统的诸多功能于一身，方便用户管理企业报表数据，但同时还需保持高效的数据访问效率。基于REST风格的服务，可提供缓存功能，能够有效地提高服务器的性能。而AngularJS框架能够将模板高效转化为代码，并且可以仅仅加载视图中的部分代码从而使得应用的性能更高。

**（2）可靠性需求**

智能报表系统作为企业内部数据交换的工具，在高性能的基础上，可靠性尤为重要。REST风格是基于HTTP协议的，由其设计的应用幂等性可以更好地实现可靠性。

**（3）扩展性需求**

企业的业务可能随着时间的变化而发生变化，为满足这一需求，系统应该拥有良好的扩展性。REST服务因为满足无状态通信，因而具有更高的扩展性。

* 1. **系统数据模型分析**

数据模型是连接现实世界和计算机世界的桥梁。它是以一定方式组织起来的，有足够的抽象性和概括性的，对客观事物及其联系的描述。这种描述包括数据内容的描述和各类实体数据之间联系的描述[地理信息系统数据模型分析参考文献1-3]。它描述了从用户角度看到的问题，它反映了用户的现实环境，而且与在软件系统中的实现方法无关。

通常，使用实体-联系图（entity-relationship diagram）简称E-R图来建立数据模型。根据对系统需求的详细分析，本文设计了报表模板，模板类型，下发任务，周期性任务模板，采集周期，上报任务，汇总数据，汇总数据明细，单位，单位类型，用户以及角色等总共17个实体，由这些实体以及它们之间的联系构成的E-R图如图3-7所示，

* 1. **系统功能结构分析**

根据本章前三节对系统整体的需求及数据分析，总结出整个系统的功能结构如图3-8所示，

图3-8系统总体功能结构图

Fig. 3-8

* 1. **本章小结**

本章对智能报表系统做了功能需求分析和非功能需求分析。根据使用系统的不同用户，从两个方面对系统的功能需求进行了详细地分析。对于普通用户，需要实现从报表模板的创建，下发报表到汇总报表数据等功能。对于系统管理用户，需要实现对企业中单位的管理，人员的管理，角色的管理以及系统基础信息维护等功能。最后提出了系统总体功能结构。除了功能需求，同时系统还应具备低耦合、高性能、高可靠性和高扩展等非功能性特点。

1. **智能报表系统的设计**
   1. **系统总体体系结构**

基于RESTful Web Services的报表系统即以按照REST原则创建的轻量级的Web服务作为服务端的智能报表系统，在服务端，智能报表系统的所有数据及其表现形式都被设计成资源对外呈现，用户通过资源的URI和统一的HTTP动词访问资源[自己的论文]。

系统总体体系结构图如图4-1所示，

客户端

资源的表述

JSON/XML

http请求

Web浏览器

桌面应用

移动设备

RESTful APIs

Web服务基本功能调用句柄

报表数据服务调用句柄

服务器

CRUD

DB

客户端

Web服务器

图4-1 服务体系结构图

Fig. 4-1

Web客户端可以为浏览器，桌面应用或者移动设备，本文设计的智能报表系统采用B/S架构，可以随时随地进行查询、浏览等业务处理，并且方便业务的拓展，因此采用浏览器作为Web服务的客户端，使用AngularJS框架搭建其架构。

服务器端主要由Web服务器、服务应用程序和数据库组成。通常当Web浏览器向服务器发出请求后，由RESTful API接收请求，对其解析并处理，然后对数据库执行相应的CRUD（Create，Retrieve，Update，Delete）操作，最后向客户端返回资源的某种表述。

* 1. **基于REST风格的服务端设计**

本文设计的智能报表系统是基于REST风格的Web服务的。REST是一种组织Web服务的架构风格，利用该架构可以创建具有良好可扩展性的分布式系统。作为一种软件架构，其提出了如下一系列约束与原则：

1. 将所有事物抽象为资源，并使用URI唯一标识。资源是数据和表现形式的组合，一份数据不同的表现形式就是不同的资源，需要不同的URI标识。
2. 使用标准的HTTP方法。使用HTTP协议原生的标准方法来提供统一的接口，完成子系统之间以及客户端与服务器之间的交互。
3. 资源多重表述。针对系统不同的需求提供资源的多重表述。
4. 分层系统。在一个REST系统中，浏览器端可能需要与不同的服务器进行交互。
5. 无状态通信。服务器端不保存客户的状态信息，从客户端发出的每次请求都必须包含理解请求的足够的信息。
6. 缓存机制。利用缓存机制可提高服务器的性能。

根据上述REST设计原则，本文总结了REST架构风格的服务设计步骤：

1. 划分数据集。分析系统的业务需求，划分不同的功能模块，建立数据集。
2. 识别资源。将已有的数据集抽象成资源。
3. 设计资源的URI，统一资源接口。设计命名风格一致的URI用以标识所有资源。并且确定对资源所需的操作，均使用HTTP标准方法来进行。
4. 确定资源的表现形式。根据需求确定服务要对外提供的资源的表现形式。

以下部分将按照上述设计步骤详细介绍服务端的具体设计。

* + 1. **REST服务层次结构**

将系统的整个业务应用划分为层次，能够实现“高内聚，低耦合”。采用“分而治之”的思想，把问题划分开来各个解决，易于控制，延展和分配资源。同时有利于系统的开发、维护、部署和扩展。本文将系统的服务端设计为三层结构，分别是Service（服务层），Business（业务逻辑层）和DataAccess（数据访问层）。

服务端的层次结构如图4-2所示，

Service层：Service层负责接收客户端发来的HTTP Request，依据URI来判断请求的资源以及对资源的操作，然后调用相应的Business层进行业务逻辑处理，将处理结果以请求中对应的数据表现形式进行封装并将其添加到对应的HTTP Response返回给客户应用程序。

图4-2 服务层次结构图

Fig 4-2 Server hierarchical structure diagram

**Service**

**Business**

**DataAccess**

**DB**

模板资源

下发任务资源

汇总数据资源

**…**

SQL Script

数据库

数据库访问接口

模板管理

数据汇总管理

**…**

下发任务管理

Response

Request

Business层：Business层负责实现对系统的各种操作，处理具体的业务逻辑，并调用DataAccess层访问数据库的数据。该层对系统的业务逻辑进行了封装，更好地为上层提供服务。

DataAccess层：DataAccess层负责访问数据库并向Business层提供数据。该层将对数据库一系列操作的SQL脚本封装起来，向上层提供访问数据库的接口，规范了对数据库的访问。

* + 1. **数据集的划分和数据库的设计**

设计一个基于REST风格的Web Services，首先需要根据系统的业务逻辑划分数据集。由上一章的需求分析可知，系统由两大部分组成，一部分为普通用户使用需求包含的五个功能模块，另一部分为系统管理用户使用需求包含的四个功能模块。

1. 普通用户需求功能模块

用户个人信息管理包含的数据集为用户。

报表模板管理包含的数据集主要为模板。

数据采集任务管理包含的数据集有报表模板，单下发任务，周期性下发任务模板，下发周期以及单位。

上报任务及上报数据管理包含的数据集为上报任务。

数据汇总管理包含的数据集有汇总报表和汇总明细。

1. 系统管理用户需求功能模块

单位部门管理包含的数据集有单位，单位类别和部门。

角色权限管理包含的数据集有角色和权限。

模板类别管理包含的数据集为模板类别。

系统信息维护包含的数据集为系统信息。

根据以上数据集以及第三章中对系统数据模型的分析，需要设计的数据库表包括用户表，数据采集模板表，下发任务表，周期性任务模板表，采集周期表，上报任务及上报数据表，数据汇总表，汇总明细表，单位表，单位类型表，单位类型映射表，角色表，功能条目表，功能角色分配表，数据采集模板类型表，数据采集模板类型映射表，系统信息表等17张表。本文将详细介绍其中与系统功能紧密相关的10张表的设计。

系统所有用户的数据都存储在用户表中，包括普通用户，系统管理用户以及超级用户。用户表的设计如表4-1所示，

表4-1 用户表

Table 4-1 User table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| UserId | varchar(6) | 是 |  | 否 | 六位用户编号，仅为数字 |
| UserName | nvarchar(50) |  |  | 否 | 用户姓名，可能为中文 |
| UserPassword | varchar(8) |  |  | 是 | 登录密码，仅为数字或英文 |
| DeptId | varchar(6) |  |  | 否 |  |
| DeptName | nvarchar(250) |  |  | 是 |  |
| RoleId | varchar(6) |  | 是 | 是 |  |
| RoleName | nvarchar(50) |  |  | 是 |  |
| UserEmail | varchar(150) |  |  | 是 | 用户Email地址，最大长度150 |
| UserMobile | varchar(50) |  |  | 是 | 手机号码，只能是数字或英文字符 |
| UserOfficeAddress | nvarchar(300) |  |  | 是 | 办公室地址 |
| UserOfficePhone | char(20) |  |  | 是 | 单位电话，仅为数字和字母 |
| UserZipCode | char(10) |  |  | 是 | 邮政编码 |
| UserQQ | varchar(20) |  |  | 否 | qq号码，仅为数字 |
| UserPicture | varbinary(MAX) |  |  | 是 | 用户照片 |
| isActive | bit |  |  | 否 |  |
| Bak9str | varchar(50) |  |  | 是 |  |
| Bak10str | varchar(50) |  |  | 是 |  |

由于用户登录系统时会根据角色，用户名及密码来验证用户是否存在，为加快查询速率，需要在RoleId和UserId这两个字段分别添加索引。

数据采集模板表在整个报表系统中占据非常重要的地位，它贯穿整个业务流转的始终。模板内容以二进制的格式存储，字段设计如表4-2所示，

表4-2 数据采集模板表

Table 4-2 Data acquisition template table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| templateId | varchar(6) | 是 |  | 否 |  |
| createDeptId | varchar(6) |  |  | 否 |  |
| createDeptName | nvarchar(250) |  |  | 否 |  |
| templateName | nvarchar(250) |  |  | 是 |  |
| templateData | image |  |  | 否 |  |
| createTime | datetime |  |  | 是 |  |
| creatorId | varchar(6) |  |  | 否 |  |
| creatorName | nvarchar(50) |  |  | 否 |  |
| templateDesc | varchar(300) |  |  | 是 |  |
| Bak8str | varchar(50) |  |  | 是 |  |
| Bak9str | varchar(50) |  |  | 是 |  |

下发任务表即采集数据的任务表，包括了单次下发任务和周期性下发任务。对于周期性任务，其单次下发与单任务下发时所需保存的信息相同，因此将其保存于下发任务表。使用字段isCyclic标明任务类型为单任务或是周期性任务。任务表的设计如表4-3所示，

表4-3 下发任务表

Table 4-3 Issue task table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| taskid | varchar(6) | 是 |  | 否 |  |
| templateId | varchar(6) |  | 是 | 否 |  |
| templateName | nvarchar(50) |  |  | 否 |  |
| deadline | datetime |  |  | 是 | 上报数据的截止日期 |
| isSendMessage | bit |  |  | 是 |  |
| collectDeptId | varchar(6) |  |  | 否 |  |
| collectDeptName | Nvarchar(250) |  |  | 是 |  |
| issueStatus | smallint |  |  | 否 |  |
| approvePlanId | varchar(6) |  |  | 是 |  |
| isCyclic | bit |  |  | 否 |  |
| cyclicTaskId | varchar(6) |  |  | 否 | 若isCyclic为false，则cyclicTaskId为000000 |
| startTime | datetime |  |  | 是 |  |
| endTime | datetime |  |  | 是 |  |
| Bakstr1 | nchar(10) |  |  | 是 |  |
| Bakstr2 | nchar(10) |  |  | 是 |  |

对于周期性下发任务，设计了一个周期性下发任务模板，任务开始执行后，系统会根据用户设置的周期，根据周期性下发任务模板保存的各种下发所需信息，定时发送给各单位。周期性任务模板表设计如表4-4所示，

表4-4周期性任务模板表

Table 4-4 Cyclic task template table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| cyclicTaskId | varchar(6) | 是 |  | 否 |  |
| templateId | varchar(6) |  |  | 否 |  |
| templateName | nvarchar(50) |  |  | 是 |  |
| cyclicId | varchar(6) |  | 是 | 否 |  |
| cyclicName | nvarchar(50) |  |  | 是 |  |
| isSendMessage | bit |  |  | 否 |  |
| collectee | varchar(10000) |  |  | 否 |  |
| isClosed | bit |  |  | 否 |  |
| startTime | datetime |  |  | 否 | 默认为当前时间 |

上报任务表与下发任务表是多对一的关系，下发任务表一次下发给多个单位，每个单位接收到任务后需要填写并上报，因此，上报报表是与单位对应的。设计如表4-5所示，

表4-5 上报任务表

Table 4-5 Report task table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| reportId | varchar(6) | 是 |  | 否 |  |
| deptId | varchar(6) |  | 是 | 否 |  |
| deptName | nvarchar(250) |  |  | 否 |  |
| reportedData | image |  |  | 是 |  |
| reportState | varchar(10) |  |  | 是 |  |
| reporterId | varchar(6) |  |  | 是 |  |
| reporterName | nvarchar(50) |  |  | 是 |  |
| reportTime | datetime |  |  | 是 |  |
| taskid | varchar(6) |  |  | 是 |  |
| templateId | varchar(6) |  |  | 是 | 指定上报数据所依据的模板 |
| approvePlanId | varchar(6) |  |  | 是 |  |
| startTime | datetime |  |  | 是 | 数据产生的时间起点 |
| endTime | datetime |  |  | 是 | 产生数据的时间终点 |
| Bakstr1 | varchar(6) |  |  | 是 |  |
| Bakstr2 | nvarchar(50) |  |  | 是 |  |
| Bakstr3 | datetime |  |  | 是 |  |

需要采集数据的单位，收到各单位上报的报表后，会对数据进行检查，确认无误后，可根据不同的汇总方式对报表数据进行汇总。用户可以使用不同条件对汇总表进行查询，这些条件包括汇总所用报表模板，汇总时间段，上报单位等等。为满足这一要求，需将这些信息都保存在汇总表中。汇总数据表的设计如表4-6所示，

表4-6 汇总数据表

Table 4-6 Data collection table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| collectionId | bigint | 是 |  | 否 |  |
| collectinoName | nvarchar(300) |  |  | 是 |  |
| collectionDesc | Nvarchar(1000) |  |  | 是 |  |
| collectionData | image |  |  | 否 |  |
| collectionTime | datetime |  |  | 是 |  |
| collecterId | varchar(6) |  | 是 | 否 |  |
| collecterName | varchar(50) |  |  | 否 |  |
| collectionStatus | smallint |  |  | 是 |  |
| isSealed | bit |  |  | 否 |  |
| sealerId | varchar(6) |  |  | 是 |  |
| sealerName | nvarchar(50) |  |  | 是 |  |
| sealedTime | datetime |  |  | 是 |  |
| Bakstr | nchar(10) |  |  | 是 |  |
| Bakstr1 | nchar(10) |  |  | 是 |  |

为方便用户使用，在查看汇总数据时，系统提供用户快速查看汇总明细的功能。即用户点击汇总数据报表中的某一栏的数据即可看到这一汇总数据的具体来源。本文设计了汇总明细表用以实现该功能。汇总明细表的设计如表4-7所示，

表4-7 汇总明细表

Table 4-7 Collection manifest table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| id | int | 是 |  | 否 |  |
| collectionId | bigint |  |  | 否 |  |
| reportId | varchar(6) |  | 是 | 否 |  |
| reportDeptName | nvarchar(50) |  |  | 否 |  |
| reportDeptId | varchar(6) |  |  | 否 |  |
| startTime | datetime |  |  | 否 |  |
| endTime | datetime |  |  | 否 |  |

以上数据库表均为满足系统普通用户的功能需求所设计，对于单位管理，用户管理等系统管理用户的功能需求，所需设计的数据表如下。

企业内部数据流转是基于单位的，各个单位之间存在层级关系，本文设计单位间的逻辑结构为树状结构，因此在存储单位时，除了单位的基础信息以外还需要存储单位的父、子单位id等信息。单位表的设计如表4-8所示，

表4-8 单位表

Table 4-8 Dept table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| DeptId | varchar(6) | 是 |  | 否 | 单位编号，六位数字 |
| DeptName | nvarchar(250) |  |  | 否 | 单位名称 |
| DeptShortName | varchar(50) |  |  | 是 | 单位简称，字母及数字 |
| DeptDesc | nvarchar(300) |  |  | 是 | 单位简介 |
| DeptOfficeAddres | nvarchar(300) |  |  | 是 | 单位地址 |
| DeptZipcode | char(10) |  |  | 是 | 邮政编码 |
| DeptParentId | varchar(6) |  |  | 是 |  |
| DeptEmail | varchar(50) |  |  | 是 |  |
| nLeft | int |  |  | 是 |  |
| nRight | int |  |  | 是 |  |
| isActive | bit |  |  | 否 |  |
| isLeaf | bit |  |  | 否 |  |
| BakStr1 | nchar(10) |  |  | 是 |  |
| BakStr2 | nchar(10) |  |  | 是 |  |

系统各类型用户的的权限是通过角色赋予的，即为不同的角色分配权限，再将角色赋予某用户，此时该用户拥有角色所拥有的所有权限。角色表的设计如表4-9所示，

表4-9 角色表

Table 4-9 Role table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| RoleId | varchar(6) | 是 |  | 否 | 角色编号 |
| RoleName | nvarchar(50) |  |  | 否 | 角色名字 |
| RoleDesc | varchar(300) |  |  | 是 | 角色描述 |
| IsActive | bit |  |  | 否 | 角色是否处于活动状态 |
| ParentId | varchar(6) |  |  | 是 | 父角色 |
| nLeft | int |  |  | 是 |  |
| nRight | int |  |  | 是 |  |
| Bak8str | varchar(50) |  |  | 是 |  |
| Bak9str | varchar(300) |  |  | 是 |  |

为便于系统扩展，将系统的一些基础信息存储于数据库中，由管理员维护这些基础信息。系统信息表的设计如表4-10所示，

表4-10 系统信息表

Table 4-10 System information table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 列名 | 数据类型 | 主键 | 外键 | 允许空 | 说明 |
| sysInfoId | varchar(6) |  |  | 否 |  |
| sysInfoName | varchar(50) |  |  | 否 |  |
| sysInfoDesc | nvarchar(250) |  |  | 否 |  |
| sysInfoValue | nvarchar(300) |  |  | 否 |  |

* + 1. **资源的分析与设计**

REST服务是面向资源的服务，以资源为中心，根据划分好的数据集抽象出系统资源是设计REST服务的关键步骤。

按照上一节中数据集的分类，对资源的分析同样从系统的两大用户需求功能模块来进行。

1. 普通用户需求功能模块

用户个人信息管理模块可以抽象出用户信息和用户集合这两类资源，对于普通用户只有查看以及修改个人部分信息的功能。在报表模板管理模块，包含模板信息和模板集合两类资源，用户可以对其进行创建，查询，修改和删除等操作。数据采集任务管理模块可以抽象出五类资源，包括报表下发任务信息，下发任务集合，周期性任务模板，周期性任务模板集合以及采集周期。上报任务管理模块可以抽象的资源有上报任务信息和上报任务集合两类资源。数据汇总模块可以抽象出三类资源：汇总数据信息，汇总数据集合以及汇总明细。

1. 系统管理用户需求功能模块

单位管理模块包含了单位信息，单位集合，单位类型，单位类型集合以及单位类型映射这五类资源。单位包含的用户信息，对应用户集合资源，对于系统管理用户可以对所有用户资源进行新建，查看，修改和删除等操作。角色权限管理模块包含角色，角色集合，功能条目和功能角色分配这四类资源。功能条目资源即为系统权限集合。模板类别管理模块包含模板类别，模板类别集合和模板类别映射三类资源。系统信息模块包含系统信息这一类资源。

经上述分析，系统总共涉及27类资源，具体的资源与资源描述如表4-11所示，

表4-11 资源与资源描述

Table 4-11 Resources and direction of resources

|  |  |  |
| --- | --- | --- |
| 序号 | 资源名称 | 资源描述 |
| 1 | 报表模板信息 | 单个模板的详细信息 |
| 2 | 报表模板集合 | 批量模板的集合，得到所有模板的宏观信息 |
| 3 | 下发任务 | 单个下发任务的详细信息 |
| 4 | 下发任务集合 | 批量下发任务的集合 |
| 5 | 周期性任务模板 | 单个周期任务模板详细信息 |
| 6 | 周期性任务模板集合 | 批量周期性任务的集合 |
| 7 | 采集周期 | 周期性任务中的周期列表 |
| 8 | 上报任务 | 单个上报任务的详细信息 |
| 9 | 上报任务集合 | 批量上报任务的集合 |
| 10 | 汇总数据 | 单个汇总数据的详细信息 |
| 11 | 汇总数据集合 | 批量汇总数据的集合列表 |
| 12 | 汇总明细 | 展示某个汇总数据的所有来源 |
| 13 | 用户信息 | 单个用户的具体信息 |
| 14 | 用户信息集合 | 批量用户的集合列表 |
| 15 | 单位信息 | 单个单位的具体信息 |
| 16 | 单位集合 | 批量单位的集合列表 |
| 17 | 单位类型 | 单个单位类型的具体信息 |
| 18 | 单位类型集合 | 批量单位类型的集合列表 |
| 19 | 单位类型映射 | 每个单位对应的单位类型集合列表 |
| 20 | 角色 | 单个角色包含的具体信息 |
| 21 | 角色集 | 批量角色的集合列表 |
| 22 | 功能条目 | 系统所有的功能条目 |
| 23 | 角色功能条目分配 | 每个角色对应的功能条目集合列表 |
| 24 | 模板类别 | 单个模板类别的详细信息 |
| 25 | 模板类别集合 | 批量模板类别集合 |
| 26 | 模板类别映射 | 每个模板对应的类别集合列表 |
| 27 | 系统信息 | 系统的基础信息 |

* + 1. **资源的URI和资源接口的设计**

REST原则中第一条原则就是每个资源都拥有一个资源标识。上一节中已经识别出了系统包含的所有资源，需要为这些资源分配其所对应的URI来标识。良好的URI设计应该拥有层次结构，并且每个URI都具有恰当的意义[基于REST和Andriod…]。

首先，所有的资源都应该存在于一个相对路径之下。一方面，对于从向相对路径发送请求才能得到各个资源，将它们置于相对路径之下是非常合理的；另一方面也便于REST服务API的版本更替。其次，URI代表的是一种资源，因此只能包含名词，而对资源的操作应该包含在HTTP请求头中。最后，资源在URI中使用单数还是复数表示的问题，应该根据具体资源的数量来决定。如果某类型资源有多个，则应使用复数表示，否则若该类型资源只有一个，则使用单数形式表示。而对于具有特定ID的某个实例时，应该对资源使用复数。因为在通过特定ID访问某个资源的实例其实就是从该资源的集合中取出特定实例。因此表示该资源集合的URI仍然需要使用复数形式，而其后所使用的ID则标明了其所访问的是资源中的单一实例，因此向这个URI发送GET请求将返回该资源的单一实例。

根据上述设计资源URI的方式，本文设计以http://localhost:60705作为服务根目录，相对路径为/smr。以模板资源为例，模板集合资源的URI为/smr/template，单个模板的URI为/smr/template/{templateid}，模板编号为00001的模板URI为/smr/template/00001。

在为每类资源定义URI之后，还需要为资源选择合适的HTTP动词来操作这些资源。HTTP协议提供了多种在URI上操作的动词，主要使用的有GET，PUT，POST以及DELETE等。在REST服务中，使用这些HTTP动词来表示对资源进行CRUD操作。HTTP协议对各个动词的定义如表4-12所示，

表4-12 HTTP动词

Table 4-12 HTTP verbs

|  |  |
| --- | --- |
| HTTP动词 | 定义描述 |
| OPTIONS | 描绘了Request-URI所指明的源端的请求/响应链的有效通信选项等信息 |
| GET | 取回Request-URI所指明的源端的一切内容 |
| HEAD | 与GET方法类似，只是服务器不会在响应中返回信息体( message-body) |
| POST | 让Request-URI所指明的源端接受请求信息中的实体，并让此实体内容成为源端的一部分，通常在表单中使用此方法 |
| PUT | 将请求信息中的实体内容保存在Request-URI所指明的源端，若有此实体的保存文件名在源端已经存在，则覆盖源端的该文件，也常用在表单中 |
| DELETE | 要求由Request-URI所指明的源服务器删除Request-URI所指明的源端路径处的内容 |
| TRACE | 使客户端能知道请求信息被接收的情况，常用于测试、诊断信息 |
| CONNECT | 使用代理的连接方法 |

通过对HTTP动词的分析，本文针对不同的资源在系统中的作用，设计了应为每个资源所提供的操作。

本文所有资源的URI以及为其提供的HTTP方法具体设计如表4-13所示，

表4-13 资源URI及HTTP方法

Table 4-13 URI of resources and HTTP method

|  |  |  |
| --- | --- | --- |
| 资源 | URI | HTTP方法 |
| 报表模板信息 | /smr/template/{templateid} | GET/POST/PUT/DELETE |
| 报表模板集合 | /smr/template | GET/DELETE |
| 下发任务 | /smr/issuetasks/{issuetaskid} | GET/POST/PUT/DELETE |
| 下发任务集合 | /smr/issuetasks | GET/DELETE |
| 周期性任务模板 | /smr/cyclictemplates/{cyclictemplateid} | GET/POST/PUT/DELETE |
| 周期性任务模板集合 | /smr/cyclictemplates | GET/DELETE |
| 采集周期 | /smr/cycles | GET/POST/PUT/DELETE |
| 上报任务 | /smr/reporttasks/{reporttasksid} | GET/PUT |
| 上报任务集合 | /smr/reporttasks | GET |
| 汇总数据 | /smr/collections/{collectionid} | GET/POST/PUT/DELETE |
| 汇总数据集合 | /smr/collections | GET/PUT/DELETE |
| 汇总明细 | /smr/collectionmanifest | GET/POST |
| 用户信息 | /smr/users/{userid} | GET/POST/PUT/DELETE |
| 用户信息集合 | /smr/users | GET/POST/DELETE |
| 单位信息 | /smr/departments/{departmentid} | GET/POST/PUT/DELETE |
| 单位集合 | /smr/departments | GET/DELETE |
| 单位类型 | /smr/departmenttypes/{departmenttypeid} | GET/POST/PUT/DELETE |
| 单位类型集合 | /smr/departmenttypes | GET/DELETE |
| 单位类型映射 | /smr/depttypemapping | GET/POST/PUT/DELETE |
| 角色 | /smr/roles/{rolesid} | GET/POST/PUT/DELETE |
| 角色集合 | /smr/roles | GET/DELETE |
| 功能条目 | /smr/functions | GET/PUT |
| 角色功能条目分配 | /smr/rolesfuncmapping | GET/POST/PUT/DELETE |
| 模板类别 | /smr/templatetypes/{templatetypeid} | GET/POST/PUT/DELETE |
| 模板类别集合 | /smr/templatetypes | GET/DELETE |
| 模板类别映射 | /smr/templtypemapping | GET/POST/PUT/DELETE |
| 系统信息 | /smr/systeminfo | GET/PUT |

* + 1. **设计资源的表现形式**

REST服务中资源可以使用任何格式来表示。表示资源时所能选取的表示形式是由实现REST所使用的协议决定的。一个基于HTTP的REST服务中，可以使用多种格式来表示资源，例如纯文本格式，JSON格式，XML格式，甚至是自定义的MIME类型。其中XML格式和JSON格式因为在各方面的优势，例如格式简单，良好的可读性以及支持多种平台等而得到广泛的应用。本文设计的服务为客户端同时提供这两种格式的资源表示形式。

在传输过程中具体的使用何种格式来表示资源，需要客户端和服务端按照HTTP协议所规定的标准协商过程来协商所使用负载类型，即使用Accept头来标示客户端所可以接受的数据传输格式。例如，客户端想要接收JSON格式的数据时，HTTP请求头如下，

GET /smr/templates HTTP/1.1

Host:localhost

Authorization: Basic ……

Accept:application/json

此时，服务端返回的响应就将使用该类型表示：

HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: ……

以浏览器端向服务端的模板资源发送GET请求为例，展示两种数据的请求格式与服务器的响应格式。上一节已经定义了模板资源的URI：/smr/templates/{templateid}，因此浏览器将对http://localhost:8080/smr/templates/{templateid}提交GET请求，请求提交的数据格式如下，

JSON格式：

{"templateid":"00001"}

XML格式：

<templateid>00001</templateid>

服务器对上述请求进行响应的数据如下，

JSON格式：

{

"template":[

{

"templateId":"00001",

"templateName":"test",

"templateData":"0xabhsksmakjzh",

"createDeptId":"10001",

"createDeptName":"Financial",

"createTime":"2017/01/21 10:46:20",

"creatorId":"00001",

"creatorName":"Sonya",

"templateDesc":"for test",

"userId":"20001"

}]

}

XML格式：

<template>

<templateId>00001</templateId>

<templateName>test</templateName>

<templateData>0xabhsksmakjzh</templateData>

<createDeptId>10001</createDeptId>

< createDeptName>Financial</createDeptName>

<createTime>2017/01/21 10:46:20</createTime>

<creatorId>00001</creatorId>

<creatorName>Sonya</creatorName>

<templateDesc>for test</templateDesc>

<userId>20001</userId>

</template>

* + 1. **HTTP响应状态码**

在与REST服务进行交互的时候，用户需要通过服务所返回的信息决定其所发送的请求是否被适当地处理。REST架构风格服务的一个标志性特征是直接利用HTTP协议的响应状态码标识服务器对于请求的响应状态。

本文设计的服务器就是利用HTTP协议机制向用户返回响应状态码和提示信息，其中HTTP状态码分为五类，每类及其含义如表4-14所示，

表4-14 HTTP响应状态码及其含义

Table 4-14 HTTP response status code and meaning

|  |  |
| --- | --- |
| HTTP响应状态码 | 含义 |
| 1XX | 信息，服务器收到请求，需要请求者继续执行操作 |
| 2XX | 成功，操作被成功接收并处理 |
| 3XX | 重定向，需要进一步的操作以完成请求 |
| 4XX | 客户端错误，请求包含语法错误或无法完成请求 |
| 5XX | 服务器错误，服务器在处理请求的过程中发生了错误 |

在实际设计服务时，仅仅使用这些预定义的HTTP状态码可能无法满足所有的情况。有时候一个REST服务所希望返回的错误信息能够更加精确地描述问题，例如在用户点击某个系统操作时，服务端需要验证用户的权限以判断是否能够执行该操作，若用户并无此权限，则应返回“您没有权限执行该操作”类似这样的提示消息。在HTTP协议中，并没有提供一个能够精确地表示该意义的状态码，也有可能提示并不友好，致使用户体验比较差。

因此在设计REST服务时，可以在响应中额外地提供一个说明性的节点，来告知用户具体的问题所在。例如对于上面的验证用户权限的情况，服务端可能会返回如下响应：

HTTP/1.1 400 Bad Request

Content-Type: application/json

Content-Length: xxx

{

"ResponseCode":"2017010001",

"Status":"Failed",

"ErrorMessage":"Permission denied!"

}

上面的示例响应中主要包含以下的说明性信息：

（1）服务端响应的状态码。页面逻辑可以通过判断该状态码是否是HTTP状态响应码中的第四类和第五类来判断是否请求出错，从而在页面中展示一个警告对话框。

（2）服务所提供的内部错误响应代码。一般情况下，该内部错误代码也需要在警告对话框中展示出来，从而允许浏览器端用户根据内部错误代码来获取支持的服务。

（3）服务端返回响应的状态以及错误信息。如果请求成功处理，则“Status”节点值为“Succeed”，“ResponseCode”节点以及“ErrorMessage”节点值为空。如果请求如上例所示失败的话，则服务端返回“Status”节点值为“Failed”，并将具体错误信息赋给“ErrorMessage”节点，通过该错误信息，浏览器端用户能够了解系统内部到底发生了什么，然后执行正确的操作。在整个响应中，最关键的部分为服务端的响应状态码。一

个响应代码不仅仅标示了用户发送的请求是否成功，更包含了指导用户该如何操作的含义。例如对于401 Unauthorized响应代码而言，其表示该响应没有提供一个合法的身份凭证，因此需要用户首先执行登录操作以得到一个合法的身份凭证，然后该资源可能就可以被访问了。而403 Forbidden响应代码则表示当前请求已经提供了一个合法的身份凭证，但是该身份凭证并没有访问该资源的权限，因此使用该身份凭证重新登录系统等操作并不能解决问题。

因此本文在设计服务端的响应时，首先根据具体的情况选择在服务器返回的响应中使用何种响应代码。由于HTTP响应状态码的数量比较多，在设计标示错误时只使用了一系列常用的响应代码，如400 Bad Request，401 Unanthorized，403 Forbidden，404 Not Found，405 Method Not Allowed，500 Internal Server Error，503 Service Unavailable等。在用户请求被处理时，系统将返回200 OK，表示请求已经被成功处理。而在处理发生错误时则使用这些响应代码来表示，并且在响应的负载中提供具体的错误信息以供用户改正。

* 1. **基于AngularJS的系统客户端设计**

本文设计的系统采用浏览器作为客户端，并使用AngularJS技术来实现。为提高开发效率缩短开发周期，在设计阶段就将AngularJS技术与浏览器端的架构结合起来进行设计。本节第一部分介绍了系统客户端的整体架构设计，第二部分根据系统的功能模块介绍了系统客户端的原型设计。

**4.3.1客户端架构设计**

智能报表系统的客户端采用MVC设计模式来设计开发，即模型（Model），视图（View）和控制器（Controller），而AngularJS框架就是一个基于Javascript的开源MVC框架，利用它可以更加快速地构建MVC架构的应用程序。AngularJS应用程序有三个主要组成部分，分别是模板，应用程序逻辑和行为以及模型数据。它们与模型-视图-控制器的映射为：

1. 模板（Templates）。

模板是由HTML和CSS编写的文件，展现应用程序的视图。

1. 应用程序逻辑（Logic）和行为（Behavior）。

应用程序的逻辑和行为是由Javascript编写的文件，对应于控制器。

1. 模型数据（Data）。

模型是从AngularJS作用域对象的属性引申的。模型中的数据可能是Javascript对象、数组或基本类型，这都属于AngularJS作用域对象。AngularJS通过作用域来保持数据模型与视图界面UI的双向同步。一旦模型状态发生改变，AngularJS会立即刷新反映在视图界面中，反之亦然。依据上述AngularJS组件与MVC架构的对应关系，本文设计的系统客户端总体框架如图4-3所示，

图4-3 客户端框架图

Fig 4-3 diagram

应用程序逻辑和行为

（控制器）

模板

（视图）

模型数据

（模型）

RESTful APIs

HTTP请求

HTTP响应

双向绑定

视图选择

用户请求

状态改变

视图负责向用户展示操作界面，当用户点击后，视图将会把用户请求传到后台控制器。控制器负责接收用户请求，调用处理逻辑来处理请求，并根据处理结果返回相应的视图。模型负责与服务器交互，向服务器提供的API发送请求并接受服务器返回的响应。在该过程中，模型与视图可通过AngularJS作用域双向绑定，实现数据的实时同步功能。

**4.3.2客户端功能模块设计**

根据本文在第三章中对系统功能需求以及系统功能结构的分析，本节对系统客户端的功能模块进行了详细设计，主要设计了各模块的原型。这种被用于设计阶段的快速原型模型为实验型原型，运用它可以证实设计方案的正确性。系统的功能模块原型设计包括普通用户需求功能模块和系统管理用户需求功能模块这两大模块，每个模块又由不同的子模块组成。

1. **普通用户需求功能模块**

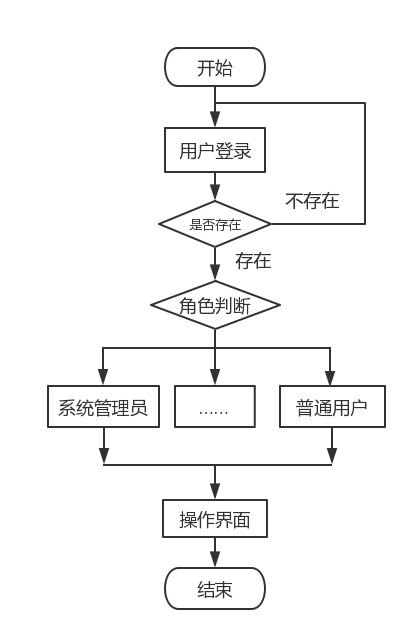
普通用户主要负责报表系统的各项业务处理，包括创建报表模板，下发任务，上报任务和汇总数据等。系统的业务处理流程图如图4-4所示，

用户个人信息管理模块。该模块主要为用户登录和用户个人信息查看及修改。用户只有在成功登录系统后才能进行下述其他模块的操作。由于本文设计的系统是为企业内部使用，因此不提供注册功能。新增用户，删除用户等操作均由系统管理人员进行。系统提供用户登录界面，用户需要选择角色并输入用户名与密码。点击登录后，浏览器端通过控制器将角色名，用户名以及密码这三个用户输入数据作为请求参数向服务的API发送验证请求，根据服务器端的返回结果来判断用户是否能够成功登录，若登录失败，则在用户名输入框后显示“用户名或密码错误”。否则根据用户所属角色提供不同的操作界面。普通用户与系统管理用户这两类角色登录系统后就对应不同的界面。用户在登录过程中输入的用户名会被记录下来并在第二次登录时回显。

用户登录系统流程图如图4-5所示，

图4-5 用户登录流程图

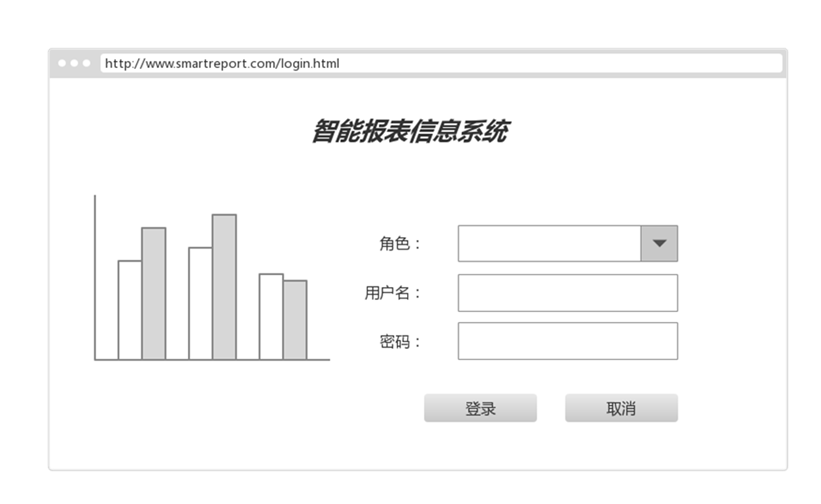
Fig. 4-5



用户登录过程的原型图如图4-6所示，

图4-6 用户登录原型图

Fig. 4-6



普通用户成功登录系统后的首页面原型如图4-7所示，

系统菜单设置在左上角，点击之后可看到其他功能模块按钮，整个浏览器页面显示当前模块的详细视图。

报表模板管理模块。该模块主要有创建，查询，修改以及删除报表模板的功能。为方便用户使用，本文以常用的Excel界面为参照设计了创建报表模板的界面。新建新报表模板的原型如图4-8所示，

当用户完成报表模板的新建工作，并点击界面上的保存按钮后，系统会自动将单元格中的所有已有内容设为只读，并将单元格颜色改变为灰色以表明其只读性。用户可通过不同条件对已有报表模板进行查询。查询条件包括报表模板Id，模板名称，模板所属单位，模板类型和模板状态。通过这些条件查询到的结果以列表的形式显示给用户。双击查询的结果列表中的一行即可查看该行所对应模板的详细信息。系统提供新建单个下发任务的按钮，方便用户在新建或查询到某个模板时新建下发任务。是否可对模板进行修改和删除操作要根据模板的状态来决定，只有当模板状态为新模板时，用户才可对自己创建的模板进行删除和修改。若模板状态为已启用或已停用，则任何用户都不能再对其进行操作包括打开、修改以及删除。如果用户发现模板出现错误，则需先撤销与之相关的所有下发任务后删除该模板或直接停用该模板，因此在模板管理视图上提供撤销按钮。所有对模板的操作都是通过客户端向服务器端发送HTTP请求来完成的。报表管理模块的原型图如图4-9所示，

数据采集管理模块。该模块应从两部分来设计，一部分为单下发任务管理模块，另一部分为周期性下发任务模板管理模块。单下发任务管理首先为创建单次采集任务，根据需求分析中提出的创建单个下发任务的流程，设计流程图如下图4-10所示，

还包括查询，更新以及删除单个任务，通过周期性任务模板下发的任务也由该模块来管理。根据需求可知，用户首先需要合适的权限才能对任务进行增、删、改、查等操作，除此之外，当任务处于各个不同状态时，用户对其可执行的操作也是不同的。为更好地向用户展示一个友好的界面，本文设计当用户无法执行当前操作时，仍旧向用户展示相应的操作按钮，但将其状态置为不可用。周期性下发任务模板管理模块不对任务进行管理，而是管理周期性下发模板。对周期性下发任务模板除了新建，修改，查看等基础操作，还需要终止操作。在需求分析中也提出了新建周期性下发任务模板的流程，设计流程图如下4-11所示，

周期性模板下发后，系统会根据其周期定时向模板中选定的单位发送选定的报表模板，因此在这个阶段只能对其进行终止操作。系统会为用户提供启用功能，如果用户希望启用已停用的模板，只需点击启用按钮，系统会自动复制原模板，然后创建一个新的模板，原周期性下发任务模板的生命周期就结束。用户也可手动复制原模板，并可以在复制前修改原模板的内容，如修改选定的待下发单位等。

单下发任务管理模块原型如图4-12所示，

周期性下发任务模板管理模块的原型如图4-13所示，

上报任务管理模块。各个单位接收到下发任务后，需要在任务规定的上报期限内填写上报报表数据。对上报任务的管理只有查询，填写和上报。由于上报任务的来源为下发任务被执行了下发操作，此时系统会自动创建一系列上报任务，因此无需提供创建上报任务的功能。查询上报任务时，也可根据多个查询条件进行筛选查询数据，查询结果仍旧显示在一个结果列表里。对于已经上报的任务或者由于下发报表的单位撤销了下发任务而使得上报任务失效的情况，上报单位可自行清除上报任务和上报数据。系统提供一键清除功能，在上报任务列表里的第一行添加复选框，用户可通过勾选复选框来表明是否要清除上报任务，点击一键清除按钮即可将已勾选任务清除。上报任务在上报前可多次修改填写的数据。填写，上报以及一键清除功能都可通过勾选任务列表中复选框对批量任务执行。上报任务及上报数据管理模块的原型如图4-14所示，

数据汇总管理模块。下发报表任务的单位收到各个单位的上报数据后，可以多种方式汇总数据。汇总的过程即为创建新汇总数据的过程，系统提供单次任务汇总和单个周期汇总两种汇总方式。对这两种汇总方式，用户都可按照本文在需求分析3.1.1节中提出的矩阵形式定制参与汇总的数据来源。用户可在汇总前先行校验各个上报数据，对于出现明显数据不合理的上报报表可退回至相应的上报单位去修改数据。查询汇总数据也可使用多个查询条件进行查询并将结果以列表的形式展示给用户。用户可点击列表中的一行来查看该条汇总的详细数据。打开的报表汇总数据，用户还可查看某个数据的来源明细，只需点击数据所在的单元格，系统将明细以表格的形式显示。对已经汇总的数据可进行封存，解封以及删除操作。在封存汇总报表前，用户可再次对汇总数据进行校验以确认数据无误方可汇总。一般用户并不能直接对汇总表进行封存与解封操作，需要向更高级的用户发出申请，由高级用户进行封存或者解封。数据汇总管理模块的原型如图4-15所示，

**2）系统管理用户需求功能模块**

系统管理人员主要负责系统的各项元数据的维护和管理，主要包括四个部分，分别是单位管理，角色管理，模板类别管理和系统信息维护。

单位部门管理模块。在系统业务处理部分的报表流转是基于单位的，因此良好的单位管理方法直接影响到系统的友好度。本文设计以树形结构来管理企业内的各个单位。树形结构能够清楚显示出单位的层级结构，便于用户管理单位或部门。若用户从树形结构中选择某个单位，系统会同时将该单位的详细信息展示给用户。提供多条件查询单位的功能，如果查询结果唯一，则系统同时在单位树中定位到相应的单位。单位树的终端节点为部门，单位可包含子单位，但部门不能包含子部门，所有部门均隶属于某个单位，所有系统用户均隶属于某个部门。当添加新单位时，用户可直接在单位树中选择新单位的父单位节点，右击该节点选择添加即可。修改和删除操作均可直接在单位树中对响应的节点进行操作。普通用户只可查看自己的个人信息，并可对其中某些信息进行修改。而对用户的管理是由系统管理员来完成的，因一个用户一定隶属于一个部门，因此将用户管理也包含在这一模块。对单位的管理还包括单位类别的管理，对单位进行分类有利于用户更方便快捷地查找特定单位，同时有助于用户更好地管理单位。单位管理模块的原型设计如图4-16所示，

角色权限管理模块。设计系统权限以角色为单位进行分配，即为每个角色赋予不同的权限，再根据用户应具有的权限为用户赋予角色。随着企业员工的增加，所应赋予的权限也会随之而增加，并且因员工之间存在层级关系，为每个用户赋予的权限也必然存在包含关系，例如某部门经理的权限可能除了包含该部门员工的所有权限外还具有其他更高级的权限。而权限是以角色来体现的，因此对角色的管理仍采用树形结构来管理。角色与角色之间存在的父子关系，能够清楚的表示在树状结构中。对角色的CRUD操作均可直接在角色树上点击节点进行。同时在界面上提供各个操作的按钮，可方便用户进行操作。可通过查询条件查询角色，查询结果仍旧采用list的形式显示。该部分还包括对权限的增、删、改、查操作。为角色赋予权限的过程，采取如图4-17的方式进行，

在对企业内部人员的分类进行充分调研后，目前提供给系统四类角色：普通用户，高级用户，系统管理员和超级用户。若企业新进员工，系统管理员则可根据该员工所应承担的责任赋予其相应的角色。系统角色权限管理模块的原型如图4-18所示，

模板类别管理模块。模板是报表系统的重要组成部分，为了使得用户更简单方便的管理模板，对模板进行了分类。模板类别存在层次关系，因此仍以树状结构管理模板类别。模板类别树的终端节点为模板，这样普通用户在使用模板时可快速定位到对应的模板。一个模板必须隶属于某个模板类别。若要删除模板类别，则要确保该类别下无模板方可进行。在模板的管理界面上仍旧需要提供模板类别树，模板查询条件，增加，修改，查询以及删除按钮，查询结果列表这几部分的内容。模板类别管理模块的原型如图4-19所示，

系统信息维护模块。系统管理员可查看并修改系统的各项配置，例如系统的名称，系统数据备份等。在需求中提到的系统预设报表模板也有系统管理人员进行管理。帮助部分和关于部分也包含于这一模块中。

* 1. **本章小结**

本章主要介绍了系统的总体架构，并分别阐述了系统服务端与系统浏览器端的详细设计。在服务端，根据所选技术，分节介绍了REST风格的服务器中的资源，URI，资源的表现形式以及HTTP状态码的设计。在客户端部分，首先介绍了客户端的设计架构，然后详细介绍了系统各个功能模块在客户端的原型设计。

1. **智能报表系统的实现**
   1. **系统开发环境**

系统是在Windows 10 64位操作系统下开发的，数据库采用My SQL。服务器端和浏览器端所用开发语言和实现环境有所不同，以下分别列出两部分的开发环境。

服务端开发环境如下：

开发平台：Visual Studio 2015

开发语言：C#

Web服务器：IIS 10.0

协议：HTTP/1.1

浏览器端开发环境如下：

开发平台：Sublime Text 3

开发语言：HTML、CSS、JavaScript

框架：AngularJS2

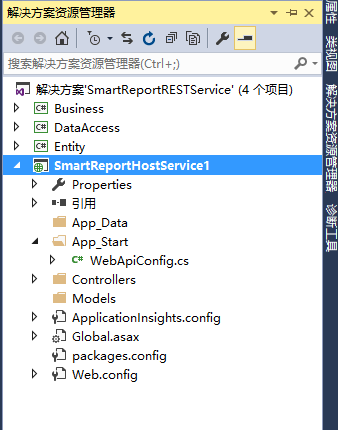
* 1. **系统服务端的实现**

**5.2.1 服务的架构搭建**

根据上一章的设计，服务端采用REST架构风格来实现。由4.2.1节可知，服务的层次结构包括Service层，Business层和DataAccess层。[使用Asp.Net WebAPI来构建REST服务](http://www.cnblogs.com/TianFang/p/3704221.html)，项目的工程结构如图5-1所示，

图5-1 工程结构图

Fig 5-1



将SmartReportHostService项目中的Controller作为服务的Service层接收请求并返回响应，调用Business处理请求，再调用DataAccess来访问数据库。Entity层中的各个实体类对应于数据库中的表，是表的映射。服务接收的request类和返回的response类也置于Entity层。其中Entity，Business，DataAccess均为类库项目，SmartReportHostService为Asp.Net WebAPI应用项目。由于各层之间的调用关系，需要将Entity类库引用进其他所有项目中，除此之外，还需在Controller中引用Business类库，在Business层引用DataAccess类库。

REST并没有像传统的RPC服务那样显式指定了服务器函数的访问路径，而是将URI根据一定的规则映射为服务函数入口，这个规则称之为路由。Asp.Net WebAPI支持两种路由方式，传统的路由映射和特性路由。根据上一章的设计，服务涉及的资源较多，因此同时使用这两种路由方式。路由规则在文件WebApiConfig.cs中定义，设置它的内容如下：

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API 路由

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "smr/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

其中第一行注册了特性路由，第二行注册了传统的映射路由。传统的Controller方式映射路由中URI格式设为"smr/{controller}/{id}"。其中，{controller}即为Controller名，{id}为可选项。当访问smr/templates/5时，系统会首先寻找名为TemplatesController的控制器，再根据其访问方式查找函数，最后根据参数id = 5来匹配具体的函数，调用其处理请求并返回响应。

默认的规则已经能够满足大多数需求，但是有些情况需要手动指定个性化的路由规则。例如，可以自定义一个按报表模板名称来查询的Uri：api/templates/templatename。此时则可使用特性路由快速实现：

[Route("smr/templates/templatename")]

public IEnumerable<TemplateEntity> Get()

其中[Route("")]为Asp.Net的一个Attribute，直接放置在操作上，通过路由将该操作公开。

服务的框架已经搭建完毕，下一节根据4.2节的设计实现服务的资源。

**5.2.2服务资源的实现**

资源是REST服务的核心内容，实现了所有的资源即实现了服务。本文设计的系统涉及的资源总共有27类，各类资源的实现方式基本相同，限于篇幅，本文只介绍其中一类资源的详细实现过程。由于系统的数据流转是基于报表模板的，并且根据需求本文设计为模板资源提供操作资源的所有的HTTP方法，因此选择模板资源为例，分析资源的实现过程比较具有代表性。

要实现资源首先需要创建数据库，根据4.2.2节中数据库表的设计，创建名为SmartReport的数据库，并创建所有的表。每张表的命名风格为以“T”开始，加上表所表示的意义，例如用户表的表名为“TUser”。因此报表模板表名为TTemplate。

报表模板资源的URL为：http://localhost:60705/smr/templates/{templateid}。它所支持的接口包括GET、POST、PUT和DELETE。

根据上一节搭建的服务架构，实现模板资源需要实现Service层，Business层，DataAccess层和Entity层。

首先在Entity层需要新建模板类的entity，request和response如下，

entity：

namespace Entity

{

public class TemplateEntity

{

public string TemplateId { get; set; }

public string CreateDeptId{ get; set; }

public string CreateDeptName{ get; set; }

public string TemplateName { get; set; }

public Object TemplateData { get; set; }

public DateTime CreateTime{ get; set; }

public string CreateUserId { get; set; }

public string CreateUserName { get; set; }

public string TemplateDESC { get; set; }

}

}

request：

public class TemplateRequest

{

public TemplateEntity template{ get; set; }

}

response：

public class TemplateResponse

{

public List<TemplateEntity> templates { get; set; }

}

Service层中Controller类的类名为资源URI的一部分，由4.2.4节的设计，模板资源的URI为template/{templateid}，因此将模板控制器类命名为TemplateController。成功创建该类后，项目自动使该类继承ApiController类。ApiController类定义了API控制器的属性和方法，并且提供了许多标准的工厂函数方便快速构建IHttpActionResult标准返回类型，如BadRequest，Conflict，Ok，NotFound等。

在控制器中实现模板资源的四种HTTP方法。详细代码如下，

public class TemplateController:ApiController

{

// GET: smr/Template

public TemplateResponse Get()

{

return TemplateBiz.Instance().GetTemplates();

}

// GET: smr/Template/00001

[Route("smr/template/{templateId}")]

public TemplateResponse Get(string templateId)

{

return TemplateBiz.Instance().GetTemplateById(templateId);

}

// POST: api/Template

public IHttpActionResult Post([FromBody]TemplateRequest request)

{

TemplateResponse response = TemplateBiz.Instance().Create(request);

if (response.Status == StatusEnum.Success) return Ok();

return BadRequest(response.ErrorMessage);

}

// PUT: api/Template/5

public IHttpActionResult Put([FromUri]string templateId, [FromBody]TemplateRequest request)

{

TemplateResponse response = TemplateBiz.Instance().Update(request);

if (response.Status == StatusEnum.Success)return Ok();

return BadRequest(response.ErrorMessage);

}

// DELETE: api/Template/5

public IHttpActionResult Delete(string templateId)

{

TemplateResponse response = TemplateBiz.Instance().Delete(templateId);

if (response.Status == StatusEnum.Success)return Ok();

return BadRequest(response.ErrorMessage);

}

}

其中Put方法中参数列表前的[FromUri]和[FromBody]用以指示该参数的来源，[FromUri]表示从请求的Uri中获取参数，[FromBody]表示从请求体中获取参数。每种方法都调用Business层相对应的方法处理请求。Post，Put和Delete方法只需返回请求处理成功与否，正如4.2.7节中对服务返回的HTTP响应设计，如果请求被成功处理，则返回ApiController类中的Ok()方法；如果处理出现错误则返回BadRequest(ErrorMessage)，向客户端返回具体的错误信息。

Service层实现后，继续编写Business层，该负责业务逻辑处理，为每个资源创建一个逻辑处理类，并使用单例模式实现该类。模板逻辑类文件为TemplateBiz.cs，代码如下，

namespace Business

{

public class TemplateBiz

{

private static TemplateBiz \_instance = new TemplateBiz();

public static TemplateBiz Instance()

{

return \_instance;

}

//add logic

}

}

对报表模板的操作包括了创建新报表模板，修改已有模板，查询模板和删除模板。对应Service层的每个标准方法，在该层实现了五个处理模板的函数，分别是根据条件查询模板的两个方法public TemplateResponse GetTemplates()，public TemplateResponse GetTemplateById(templateId)，新建模板的方法public TemplateResponse Create(request)，修改模板的方法public TemplateResponse Update(request)和删除模板的方法public TemplateResponse Delete(request)。其中两个查询方法中，检查要求不能为空的参数是否为空，例如IsNullOrEmpty(templateId)，若为空，则不做任何操作，直接返回NULL。若非空，则调用DataAccess层访问数据库读取数据并返回。新建模板的方法在该层也是判断是否非空参数为空，若为空，则将Response中的Status属性值赋为Failed，ErrorMessage属性值赋为“模板数据不能为空，请检查后重试。”，并返回到Service层。若均非空，则直接调用DataAccess层对应的写数据库方法进行插入操作。修改方法与删除方法均需先根据模板id查询模板的存在性，若存在则直接调用DataAccess层操作，否则返回Not Found错误。

完成Business层，需要实现DataAccess层，该层封装了对数据库的读写操作，当Business层调用该层时，DataAccess层通过SQL脚本访问数据库进行CRUD操作，不再对各项参数做检查。通过轻量级的ORM（Object Relational Mapping）框架Dapper来封装对数据库的访问。只需将GitHub上源码拷贝到DataAccess类库项目的solution下即可使用Dapper框架。以对模板的增、删、改、查操作为例，介绍通过Dapper访问数据库的具体实现。

创建数据库连接的核心代码如下，

public IDbConnection Getconnection(){

IDbConnection connection = new SqlConnection("Data Source=.;Initial Catalog=DataMip;Integrated Security=True;MultipleActiveResultSets=True");

return connection;

}

通过id查询模板的核心代码如下，

public TemplateResponse GetTemplateById(string templateId){

TemplateResponse response = new TemplateResponse();

response.Templates = new List<TemplateEntity>();

var query = Getconnection().Query<TemplateEntity>("select \* from TTemplate where templateid=@TemplateId", new { TemplateId = templateId });

if(query != null)

{

foreach(TemplateEntity en in query)

{

response.Templates.Add(en);

}

}

return response;

}

上述代码中Getconnection().Query<TemplateEntity>即使用Dapper来完成数据库表与实体对象的映射。它把一个匿名对象作为参数，传入SQL脚本，查询到数据后直接映射到TemplateEntity实体。

多条件查询模板时，基本实现原理与通过id查询模板相同，仅仅在sql脚本上有所不同。通过逻辑处理层传来的TemplateRequest类型的变量request中所包含的每个属性进行非空判断，若为空，则无需作为查询条件，若非空，则需要向sql脚本字符串添加该查询条件变量，具体实现代码如下，

String sql = " select \* from TTemplate where ";

If(!isNullOrEmpty(request.template.TemplateName))sql += " templateName = @TemplateName ";

If(!isNullOrEmpty(request.template.CreateDeptName)) sql += " AND CreateDeptName = @CreateDeptName ";

……

服务支持批量新建报表模板，同时支持新建单条记录。由于请求参数已由Business进行判空以及合理性等验证后才传入该层，因此直接将请求参数转换为List<TemplateEntity>类型的变量templateList，执行创建操作的核心代码如下，

var result = Getconnection().Execute("Insert into TTemplate values (@TemplateId,@CreateDeptId,@CreateDeptName,@TemplateName,@TemplateData,@CreateTime,@CreateUserId,@CreateUserName,@TemplateDESC

)", templateList);

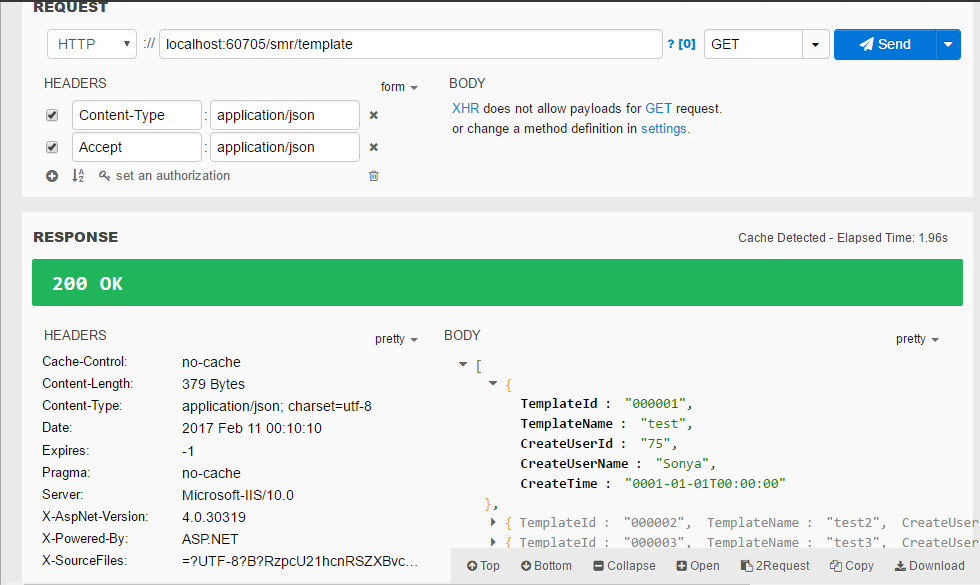
返回值result为int类型，根据其是否大于零来标识插入是否成功。更新和删除操作均使用Getconnection().Execute方法来执行。

通过以上的工作，就实现了模板资源的GET，POST，PUT以及DELETE接口，浏览器端可通过URI：http://localhost:60705/smr/template/{templateid}来操作模板资源。

使用Google测试REST服务的应用插件DHC工具来调用模板资源的GET方法，收到服务返回来的响应如图5-2所示，客户端通过Accept请求头来标明接收json格式的数据。

图5-2 模板资源Get请求与响应示例图

Fig. 5-2



* 1. **系统客户端的实现**

系统客户端的实现是基于AngularJS框架的。Angular是一个用HTML，CSS和JavaScript来构建浏览器端应用的框架。实现Angular应用的过程为，用Angular扩展语法编写HTML模板，用组件类管理这些模板，用服务添加应用逻辑，用模块打包发布组件与服务。然后通过引导根模块启动应用。

Angular应用是模块化的，并且Angular有自己的模块系统，它被称为Angular模块或NgModules。所有的模块为带有@NgModule装饰器的类，包括根模块。NgModule是一个装饰器函数，它接收一个用来描述模块属性的元数据对象。其中最重要的属性是：

Declarations：声明本模块中拥有的视图类。Angular有三种视图类：[组件](http://origin.angular.live/docs/ts/latest/guide/architecture.html#components)、[指令](http://origin.angular.live/docs/ts/latest/guide/architecture.html#directives)和[管道](http://origin.angular.live/docs/ts/latest/guide/pipes.html)。

Exports：declarations 的子集，可用于其它模块的组件[模板](http://origin.angular.live/docs/ts/latest/guide/architecture.html#templates)。

Imports：本模块声明的组件模板需要的类所在的其它模块。

Providers：[服务](http://origin.angular.live/docs/ts/latest/guide/architecture.html#services)的创建者，并加入到全局服务列表中，可用于应用任何部分。

Bootstrap：指定应用的主视图（称为根组件），它是所有其它视图的宿主。只有根模块才能设置bootstrap属性。

本文设计的根模块的模块名为AppModule。它的位于app.module.ts文件中，实现代码如下，

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import {FormsModule} from '@angular/forms';

import { AppComponent } from './app.component';

import {TemplateComponent} from './template.component';

@NgModule({

imports: [ BrowserModule ,FormsModule],

declarations: [ AppComponent, TemplateComponent],

bootstrap: [ AppComponent ]

})

class AppModule { }

* 1. **系统测试**
  2. **本章小结**

1. **总结与展望**
   1. **总结**

本文主要设计并实现了在线智能报表系统。根据使用系统的两类主要角色从啷个方面设计并实现了系统。普通用户主要负责为报表模板管理，数据采集管理，上报任务管理以及数据汇总管理等业务功能。系统管理用户主要负责单位部门管理，角色权限管理，模板类别管理和系统维护等系统管理功能。系统采用B/S架构，服务器端使用了REST风格架构来设计实现，浏览器端采用AngularJS框架来实现。

本论文主要完成了一下方面的工作：

（1）

* 1. **展望**