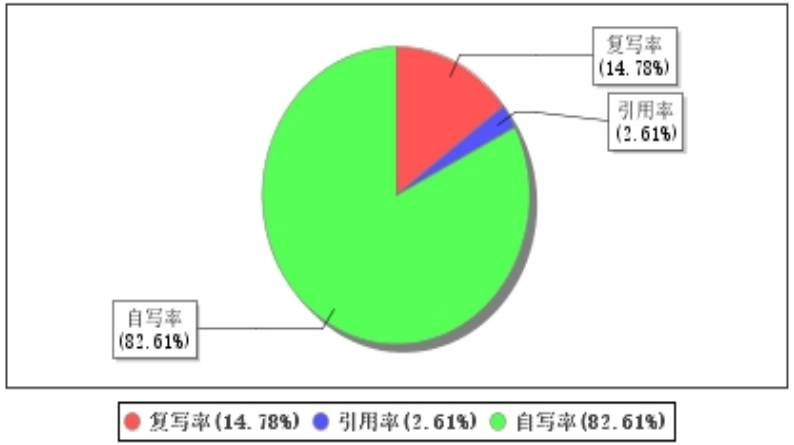
论文检测报告

报告编号： 20CA44C7B6584BEDA878DF4D5BC46A3C 真伪查询
送检文档： 1119-2013221104210097-肖锐- 在线的可视化网站设计-计算机科学与技术
论文作者： 肖锐

文档字数： 26715
检测时间： 2017-05-05 15:51:32
检测范围： 互联网，中文期刊库（涵盖中国期刊论文网络数据库、中文科技期刊数据库、中文重要学术期刊库、中国重要社科期刊库、中国重要文科期刊库、中国中文报刊报纸数据库等），学位论文库（涵盖中国学位论文数据库、中国优秀硕博论文数据库、部分高校特色论文库、重要外文期刊数据库如Emerald、HeinOnline、JSTOR等），共享资源库，参考文献库。

一、检测结果：

总相似比： 17.39% [即复写率与引用率之和]
检测指标： 自写率 82.61%复写率 14.78%引用率 2.61%



二、相似文献汇总：

序号	标题	文献来源	作者	出处	发表时间
1	网络控制系统调度与控制的协同设计	学位论文	王洁	硕博学位论文	2007
2	Domino/Notes作为后台数据库-设计/艺术-图宝贝文档搜索	互联网		互联网	0
3	兽药生产企业药品不合格的原因分析与建议	学术期刊	石留娥 张江宏	河南畜牧兽医	2004
4	servlet的执行原理与生命周期 - javaloveiphone的专栏 - 博客频道...	互联网		互联网	
5	Component组件的详细说明和生命周期-梦家之家-博客频道-...	互联网		互联网	0
6	五强争霸浏览器世界杯	学术期刊	孙洪志 程武伟	计算机应用文摘	2010
7	React入门到精通(三)——定义组件和css样式-ALLEN的博客-博客...	互联网		互联网	0
8	1.3React组件-LeanReact-知乎专栏	互联网		互联网	0
9	互联网时代网站建设的意义是什么-搜狐	互联网		互联网	0
10	-9hvCmX8G_"target="_blank">信息可视化研究及其在人才网站建设中的实现	互联网		互联网	0
11	人民法院诉讼档案鉴定销毁工作的现状与对策	学术期刊	殷艳梅	兰台内外	2004
12	Node.js快速入门-编程语言-IT问道	互联网		互联网	0

13	React入门03-react中操作组件的dom节点-博客频道-CSDN.NET	互联网		互联网	0
14	...nodejs、mongodb、expressjs的全栈开发实例教程1_JavaScript_...	互联网		互联网	0
15	js执行一个耗时操作时,比如一个大的遍历,如何才能真正..._百度知道	互联网		互联网	0
16	中储物流管理信息系统应用与网络系统设计	学位论文	郝勇	硕博学位论文	2002
17	专栏:前端-博客频道-CSDN.NET	互联网		互联网	0
18	数据库访问中间件在多服务器模型中的研究与实现	学位论文	洪君景	硕博学位论文	2003
19	RethinkDB分布式数据库-PHPERZ中文资讯站	互联网		互联网	0
20	Reactjs深入了解教程-前端开发博客	互联网		互联网	0
21	EAST超导托卡马克装置主机装配及其测量系统的研究	学位论文	赵庆荣	硕博学位论文	2004
22	建筑质量员自我评价、工作成果及工作总结_百度知道	互联网		互联网	0
23	组件化的Web王国-文章-伯乐在线	互联网		互联网	0
24	javaweb学习总结(七)——HttpServletResponse对象(一)-..._博客园	互联网		互联网	0
25	React数据获取为什么一定要在componentDidMount里面调用?-Clark...	互联网		互联网	0
26	基于CATIA的飞机驾驶舱布局设计的软件开发	学术期刊	艾玲英 王正平 王黎明	飞机工程	2004
27	NoSQL非关系型数据库技术和应用_图文_百度文库	互联网		互联网	0
28	react更新组件componentWillReceiveProp里面setState无效,未触发...	互联网		互联网	0
29	HttpServletRequest对象(一)-陆伟-博客园	互联网		互联网	0
30	桂林市有条件接收数字电视前端CA系统原理	学术期刊	黄克 戴石晓	有线电视技术	2004
31	基于移动互联网的手机互动广播系统	学术期刊	陈兆强	世界广播电视	2012
32	JavaScript运行机制详解:再谈EventLoop-阮一峰的网络日志	互联网		互联网	0
33	基于j2ee的cscw开发平台的研究与实现	学位论文	顾俊	硕博学位论文	2003
34	基于JSP开发的网上购物系统	学术期刊	蒋蕾	电脑知识与技术：认证考试	2004
35	Node.js高效分布式实时应用的技术措施6000字_毕业论文网	互联网		互联网	0

三、全文相似详情： （红色字体为相似片段、浅蓝色字体为引用片段、深蓝色字体为可能遗漏的但被系统识别到与参考文献列表对应的引用片段、黑色字体为自写片段）

目 录

在线的可视化网站设计

摘要

未来，Web应用所处的环境将会越来越开放，编写Web应用的代码也越来越复杂难懂。使得系统的构建与维护的成本不断增加。传统开发网站的方法明显不能适应网站应用系统的设计需求。如何快速响应需求，降低应用开发成本，构建可扩展和可维护的 Web 应用成为了新的挑战。

本文对在线的可视化网站设计进行了详细设计并给出实现方案。采用React.js、Node.js和RethinkDB设计应用开发环境的总体框架，并将各个功能模块封装成React组件，插拔灵活，扩展方便，易于维护。基于本系统框架，集成可视化的界面设计、预览工具，同时也可以界面元素设置属性、添加事件绑定。基于事件驱动机制，设计后台业务逻辑的执行环境。

【关键词】网站设计可视化Node.jsReact.jsRethinkDB

The design of online visualization website

Abstract

In the future, the environment in which Web applications will be more and more open, and the code that makes up the Web system is becoming more and more difficult. Making the system construction and maintenance costs continue to increase. The traditional Web application software development method obviously cannot meet the design requirements of Web applications. How to respond to requirements quickly, reduce application development costs, building scalable and maintainable Web applications has become a new challenge.

In this paper, the design of online visualization website is designed and the implementation scheme is given. Based on Node.js, React.js and RethinkDB design the overall framework of the application development environment, and the various functional modules packaged into React components, plug and play flexible, easy to facilitate, easy to maintain. Based on the system framework, integrated visual interface design, preview tool, but also interface elements can be set properties, add event binding. Based on the event-driven mechanism, the design environment of the background business logic is designed.

【Key words】Website Design Visualization Node.js React.js RethinkDB

绪论

有别于传统的可视化网站设计通常基于IDE的开发模式，本系统将web的开发与设计迁移到了云端。基于可复用组件库的代码生成、可视化界面建模及实时预览的开发工具，云端系统提供可视化的Web UI基础组件、开发者通过组合、嵌套操作结合事件绑定、属性设置来设计系统。这是保证效率与产品质量的一种重要手段,它的关键技术是组件复用技术和可视化建模设计方法.实现组件复用的方法主要是组件开发技术,通过Web系统架构设计把系统各模块组件化,组合、嵌套Web 组件,从而构建一个完整的 Web 应用系统.

1在线的可视化网站设计概述

1.1在线的可视化网站设计的概念[1]

可视化的网站设计，它的历史相对于互联网的应用时间是较长的。随着技术的进步，可视化网站设计工具也在不断地增强功能。这类工具大多所见即所得网页编辑器，能够简单快速地制作出充满动感的网页[2]。在线的可视化网站设计，就是将传统的网站设计迁移到云端，通过Web UI进行网站构建。

1.2在线的可视化网站设计的发展和趋势

进入21 世纪以后，由于国内移动互联网的飞速发展，以及政府大力推行“ 互联网+ ”，这导致政府和企业对网站设计需求快速增长，可视化网站设计又有了新的生命和机遇。国内专门从事互联网软件开发的公司也逐渐成长起来。由于各公司需求不同，导致需求量急剧增加，从程序员的角度来看，日复一日的编码，未免有重复造轮子之嫌。可视化网站设计工具虽多，却少有创新的设计。

目前可视化网站设计的发展比较成熟，大名鼎鼎的，如Adobe Dreamweaver和Microsoft Visual Studio。它们的出现都提高了网站的开发的效率。这些工具提供的控件拖拽，数据绑定，语法检查等功能，简化了开发流程，降低了web开发的门槛。但这些产品的开发效率还可以被进一步挖掘，前有微信小程序，后有Google Android Instant Apps。

综上所述，未来网站设计将向着可视化、组件化和平台化的方向发展。

1.3课题的背景及意义

本课题的价值在于：第一，简化了web开发的难度，降低了web开发的门槛。第二，简化了web开发流程，提高了web开发效率。第三，能够更好的展示自。同时也会使自身的生活变得更加的充实[3]。第四，增加了互动的时间，人们可以随时随地的了解到自己想了解的东西，不需要担心关门的客观因素。只要想了解，进行网站访问就可以了解。

1.4本文的主要工作

本系统的设计基于B/S架构，有别于传统的IDE，采用所见即所得，所见即可用的开发模式，在本系统的平台上开发网站，省去了传统开发模式的部署步骤，可大大提高开发效率。采用组件组合式开发模式，大大减少了开发出错的概率。

文所做的主要工作有：

整体的分析、设计系统，规划系统的体系结构。

对比现有的开发工具，确定有利于系统开发的工具，并选择适合本系统的数据库。

进行用户需求分析，确定系统的总体功能结构。

实现在线的可视化网站设计的部分基本功能模块。

2可视化网站设计相关技术的研究

2.1可视化网站设计的开发模式

系统结构的选择

（1）B/S模式[4]

B/S模式是当前环境下广泛使用的模式。它以服务器为核心，所有的功能都在服务器上实现，提供大量的限定权限的API接口。开发者通过调用这些API来完成自定义组件的设计，并通过自定义组件，完成网站的设计与开发。B/S模式简单实用，容易维护，可移植性强，灵活性较差。

（2）C/S模式[5]

C/S模式系统用服务器作为数据处理和存储的核心，在客户机设计相应的应用，必须使用应用才能对数据进行操作。C/S模式的专业化程度很高、安全性很好、开发手段也特别灵活、运行速度快、用户体验很好，但开发成本比B/S模式高，维护起来相当复杂，升级也很麻烦。

（3）单体式应用

单体式应用已经相当普及，界面友好、便于共享、易于测试、易于部署。在本系统的设计中，缺点也十分明显：不够灵活，开发周期长，维护复杂，学习成本高。现有的Eclipse、Visual Studio均采用这种架构。

（4）结论

本系统将开发与生产环境融为一体，具备在线协同开发，无需部署，开发完成即可投入使用。根据本网站应用开发系统的属性来确定，将在线的可视化网站设计的开发结构定为B/S模式。

开发环境的选择

（1）Node.js[6]

Node.js是服务端的JavaScript运行时环境。基于Google Chrome V8 JavaScript引擎，它具有无I/O阻塞和事件驱动的特性。Node.js实现了类似Apache的web服务。结合React.js和Angular.js等前端框架来使用，可以构建体验良好的JavaScript全栈跨平台应用。

（2）ASP.NET[7]

ASP.NET具有良好的扩展性和灵活性。与Windows操作系统紧密结合，使得它可以与其他Windows应用方便地交换数据。ASP.NET框架下有很多可以选择的开发工具，主要是VB、 VC、ASP.NET、C#。特别是 ASP.NET框架，它应用广泛，但它的专业性不强，在规模大的应用中，稳定性表现的不够好。

（3）JAVA EE[8]

JAVA EE适用于企业级应用的开发、以及大规模数据处理。它的安全性和稳定性十分优越，特别适合开发B/S模式的Web应用系统。使用Java 开发系统十分灵活，它的开发模式与ASP.NET极为相似。

Node.js能够在响应大量的数据请求的同时，避免造成大量的阻塞。Node.js高效利用并行I/O，解决了服务端的性能瓶颈。

鉴于Node.js的上述优点，本课题采用Node.js进行在线的可视化网站设计系统的开发工作。

2.2Node.js 技术[9]

Node.js是一个基于Google Chrome V8 JavaScript运行时环境建立的平台。它可以方便地构建响应速度快、易于扩展的互联网应用。Node.js可以应付分布式设备上运行的数据密集型的实时应用，因为它采用的是轻量且高效的事件驱动机制、非阻塞的I/O 模型。

2.2.1Node.js工作原理[10]

（1）使用Chrome V8引擎，解析JavaScript脚本。

（2）在使用Chrome V8 引擎解析代码之后，调用Node API。

（3）LIBUV库将会负责Node API的执行。采用一个事件循环来处理分配任务之后的不同的线程，这些线程会以异步执行的方式将执行结果返回给Chrome V8引擎。

（4）V8引擎得到执行结果后，将结果返回给用户。Node.js的工作原理如图2.1所示：

图 2.1 Node.js 工作原理

2.2.2Node.js Express框架[11]

Express 是一个基于node.js的Web应用框架。它内置功能强大的 HTTP 工具，用于帮助开发者快速构建功能完整的Web 网站应用。

2.2.2.1Request对象

Request 对象代表了一次HTTP 请求。它的常见属性如下：

(1) req.app：当callback为外部文件时，用req.app访问express的实例

(2) req.baseUrl：获取路由当前安装的URL路径

(3) req.body / req.cookies：获得请求的主体/本地Cookies缓存

(4) req.fresh / req.stale：判断请求是否还新鲜

(5) req.hostname / req.ip：获取主机名和IP地址

(6) req.originalUrl：获取原始请求URL

(7) req.params：获取路由的请求参数

(8) req.path：获取请求路径

(9) req.protocol：获取协议类型

(10) req.query：获取URL的查询参数串

(11) req.route：获取当前匹配的路由

(12) req.subdomains：获取子域名

(13) req.accepts()：检查可接受的请求的文档类型

(14) req.acceptsCharsets / req.acceptsLanguages / req.acceptsEncodings：返回请求头里面指定字符集的第一个可接受字符的编码方式

(15) req.get()：获取指定的HTTP请求头

2.2.2.2Response对象

Response 对象代表了 一次HTTP 响应。它的常见属性如下：

res.app：同req.app一样

res.append()：追加指定HTTP头

res.set()在res.append()后将重置之前设置的头

res.cookie(name，value [，option])：设置Cookie

opition: domain / expires / httpOnly / maxAge / path / secure / signed

res.clearCookie()：清除Cookie

res.download()：传送指定路径的文件

res.get()：返回指定的HTTP头

res.json()：传送JSON响应

res.jsonp()：传送JSONP响应

res.location()：只设置响应的Location HTTP头，不设置close response或者状态码
res.redirect()：设置响应的Location HTTP头，并且把状态码设置成302
res.send()：传递HTTP的响应
res.sendFile()：传输指定路径的文件给浏览器，通常会根据文件扩展名自动设定Content-Type
res.set()：设置HTTP头，传入object可以一次设置多个头
res.status()：设置HTTP状态码
res.type()：设置Content-Type的MIME类型
2.2.2.3路由
路由决定了哪段特定的脚本代码会去响应特定客户端的请求。下面是路由示例：

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
console.log("主页 GET 请求");
res.send('Hello GET');
})
app.post('/', function (request, response) {
console.log("主页 POST 请求");
response.send('Hello POST');
})
var svr = app.listen(8080, function () {
var hst = svr.address().address
var pt = svr.address().port
console.log("应用的访问地址是 http://%s:%s", hst, pt)
})
```

2.2.3Node.js访问RethinkDB数据库的主要步骤
（1）引入RethinkDB模块

```
var r = require('rethinkdbdash')({
port: 28015,     // Port用于指定rethinkDB数据库的端口
host: 'localhost'// host用于指定rethinkDB数据库的IP
db: ’onlineSystem ’// db用于指定数据库的名称
});
```

（2）创建数据库

```
r.dbCreate('myDB').run()
.then(function(response){
console.log(response);
})
.error(function(err){
console.log('error occured ', err);
});
```

（3）创建表

```
r.tableCreate('Employee')
.run()
.then(function(response){
console.log(response);
})
.error(function(err){
console.log('error while creating table ', err);
})
```

（4）在rethinkDB数据库中插入数据

```
r.table("Employee")
.insert({
name: "Jay",
company: "SitePoint"
}).run()
.then(function(response){
console.log('Successed ',response);
})
.error(function(err){
console.log('error occurred ',err);
})
```

（5）实时推送与更新数据

```
r.table('Employee')
.changes().run()
.then(function(cursor){
```

cursor.each(console.log);
}).error(function(err){
console.log(err);
});

2.3React.js技术简介

React.js工作原理[12]

使用react进行开发的时候，几乎所有的DOM操作都是通过操作虚拟DOM实现的。每次更新数据的时候，react会重新构建DOM树，并将新的DOM树与上一次生成的DOM树进行比较，然后仅仅渲染变化的DOM节点。每次对比使用的是React特有的Diff算法，这个算法是React高性能的关键之所在。并且虚拟的DOM节点存在于内存之中，这也是保证React高性能的关键因素之一。

通常，服务端的任务是将渲染好的HTML文档发送到浏览器。大部分情况下，用户只需要修改一部分的状态或者属性，这也会导致整个页面的刷新。因为服务端并不知道哪个状态或者属性需要改变。而React不同，每一次的UI更新，用户都可以认为刷新了整个页面，但是React实际上只刷新了局部UI。同时，React也保证了性能。这样，在保证性能的同时，开发者不必再关心如何操作DOM元素，而只需要关注在任意一个数据状态下，整个界面是如何渲染的。

React.js的组件化[13]

组件化的开发思想指导了虚拟DOM的研发。虚拟DOM使得UI开发的逻辑更为简洁。这些组件，就是一些封装起来的、功能相对集中的Web元素。React大力推荐以组件的方式去思考Web UI的构建。其实，每个UI部件都是一个React 组件，你可以使用React通过组合、嵌套等方式来完成Web UI的开发。相对MVC的思想，它做到了模型、视图、控制器的分离，而组件化的思想则让你做到了功能模块之间的分离。图2.2是一个评论框组件的示例：

图2.2 典型的评论框组件

React组件应该具有下面的这些特征：

（1）可组合：一个React组件应该无障碍地和其它React组件组合、嵌套使用。通过可嵌套这个特性，每一个功能复杂的UI组件都可以被拆分；

（2）可重用：每一个React组件都是功能独立的，并且可以应用在多个UI场景；

（3）可维护：每一个小小的React组件都应该是逻辑清晰的。同时它应该是易于维护和理解的。

2.3.3 React.js的生命周期

在React.js的生命周期，React组件会经历以下四个阶段：创建阶段、实例化阶段、更新阶段、销毁阶段。如图2.3：

图2.3 React组件的生命周期

（1）创建阶段：在调用React.createClass时触发getDefaultProps方法，getDefaultProps方法会返回一个对象，然后与父组件的props合并，组成新的this.props。

（2）实例化阶段：在组件类被调用的时候会触发如下的一系列方法：

getInitialState：它的返回值会传递给组件的this.state。

componentWillMount：组件渲染之前调用的方法。

render：组件渲染方法，每当组件状态更新时会调用该方法，此方法会生成虚拟DOM，并返回该虚拟DOM。

componentDidMount：组件渲染完成之后调用的方法。在这个方法内部，使用ReactDOM.findDOMNode(this)获取当前组件的DOM节点，并且可以像操作真实DOM节点那样操作DOM节点元素，但是React.js不推荐这样操作DOM节点。

（3）更新阶段：这个阶段会根据用户行为来重新调整DOM结构：

componentWillReceiveProps：当这个方法被触发时，表示这个组件接受到了新的props。你可以使用this.setState这个方法来完成对state的更新。

shouldComponentUpdate：这个方法会拦截props或state的变化，根据返回的true或者false来决定组件的更新。

componentWillUpdate：当上面那个方法返回true的时候，就可以在这个方法中做一些更新之前的操作。

render：根据React.js内置的一系列diff算法，渲染那些需要更新的虚拟DOM数据。

componentDidUpdate：这个方法会在组件完成更新之后触发，你可以在这个方法中做一些DOM操作。

（4）销毁阶段：销毁阶段只有一个方法，它就是componentWillUnmount。通常会在这个方法里面做一些取消事件绑定、移除React组件、清理定时器等工作。

2.4RethinkDB数据库简介[14]

RethinkDB 是一个实时的可扩展的开源JSON数据库。通过暴露出简单易用的访问接口，它反转了传统的数据库体系结构，不需要通过轮询更改，开发者可以使用RethinkDB暴露出的接口，将更新之后的查询结果实时地推送到你的应用程序。本质上，RethinkDB是一种非关系型数据库[15]。具有如写特征：

（1）没有预定义的模式：不必预先定义好数据模式和表结构。每一条记录都可以有自己的属性和格式。

（2）没有共享架构：传统数据库将所有数据集中存储在存储区，使用网络来连接集中的存储区域。RethinkDB将数据划分后存储在多个本地服务器上，数据的读写全部都在本地数据库进行。从本地磁盘读写数据的性能明显优于通过网络传输读写数据的性能。因此RethinkDB的性能高于传统关系型数据库的性能。

（3）弹性可扩展：RethinkDB可以动态地改变结点数量。数据在数据库运行的同时也可以自动迁移。

（4）分区： RethinkDB会将数据分区，把数据拆分并存储在多个节点上面。分区的同时可以做复制。这样做可以确保不会有单点失效，也可以提高并行性能。

（5）异步复制：和传统存储系统不同的是，RethinkDB中的复制是基于log的异步复制。因此，数据可以快速的写入本地的一个节点，且不会因为网络传输速度引起延迟。但是，这样做的缺点也很明显。它不能保证数据的一致性，在出现故障的时候会造成少量的数据丢失。

（6）BASE：传统数据库的事务采用的是严格的ACID特性，而RethinkDB保证的是软事务和最终一致性。

3系统分析

系统分析的基本思想是从系统的整体出发，通过对系统各部分进行综合的系统分析，为系统设计提供理论依据。系统分析分为用户需求分析、系统功能分析、系统性能分析、系统结构设计及系统配置分析这几个步骤。

3.1用户需求分析

用户需求分析就是对项目进行全面的调研，以确定在线的可视化网站设计的功能和目的，这是构建在线的可视化网站设计的基础。

企业构建一般的门户网站，一般不需要复杂的工具。在线的可视化网站设计构建的理念应该是方便客户进行网站设计，而且是越简单越好。构建完全在服务端完成，使企业内部人员能够方便高效地协同工作。

系统对软硬件要求不高，一台够用的服务器和可供安装使用Google Chrome浏览器的客户端即可供系统的正常运转。服务端需要连接互联网，并能够正常安装node.js和RethinkDB软件。客户端只要求系统安装了Google Chrome浏览器。

3.2系统功能分析

本系统是采用Node.js、React.js结合RethinkDB数据库平台开发的一个在线的可视化网站设计平台系统。本系统具有以下功能：

快捷按钮。提供菜单、新建页面、撤销修改、删除当前页面等功能。

基础组件。提供可选的基础的内置组件。

属性配置。配置组件的属性和事件方法。

路由配置。配置当前页面的访问路由。

页面设计。综合使用上述工具进行页面设计。

页面预览。点击页面上的Preview按钮，开发者可以预览设计好的页面。

在系统操作方面需要说明的是，本系统用到的RethinkDB数据库，网站设计者的修改会及时的推送到其他正在设计的设计者界面之中。

3.3系统性能分析

3.3.1在线的可视化网站设计的规范化

目前基于本地IDE的可视化网站设计占据了市场的大部分份额，在线的可视化网站设计基本可以参考本地IDE的可视化网站设计的模式。

3.3.2系统设计的先进性

可视化网站设计系统在开发的时候会使用最新的技术，以此来确保技术的先进性。这样可以保证开发出来的系统，不会因为技术太过陈旧、性能过于低下而需要进行大规模的调整或者重构。同时，设计好的系统也可以进行逐步的更新，不会因为对硬件的要求过高而造成系统的崩溃。

3.4系统结构设计

在设计在线的可视化网站设计系统的时候，采用的是B/S网络结构。图3.1是典型的B/S网络结构。

B/S网络结构将在线的可视化网站设计系统中的数据、功能、行为分离，分别表示为客户层，逻辑表达层，应用层。设计者应该隔离共享数据和控制业务逻辑；设计后端时，隔离服务层，这一层专门用来负责处理服务端-数据库、服务端-浏览器之间的数据传递。B/S网络结构体系层次分离的优势体现在统一的传输协议、语言格式和界面风格。

图3.1 B/S网络结构

3.5系统配置分析

本文采用B/S（浏览器/服务器）模式。后端开发采用Node.js技术，前端开发采用React.js技术,数据库采用RethinkDB。

系统的软硬件需求如下：

（1）服务器端

硬件配置：内存：8G，CPU：Core i7 2. 以上

软件配置：操作系统：Win Server 2012

安装程序：node.js 6.2.1；

Google Chrome 57.0.2987.133；

分辨率：800*600

（2）客户端

硬件配置：内存：2GB，CPU：Core i3 2.7G以上

软件配置：操作系统：Windows 7

安装程序：Google Chrome 57.0.2987.133；

分辨率1920*1080

4系统主要模块的实现与技术关键

4.1系统功能模块图

经过详细的系统需求分析和功能分析，可以得到系统功能模块图，分别是：快捷按钮、基础组件、属性配置、路由配置、页面设计、页面预览。如图4.1所示：

图4.1 系统功能模块图

4.2数据库设计

本课题设计时，数据库采用的是可扩展的RethinkDB，它是一种高性能的非关系型数据库。RethinkDB具有高可靠性和高性能，下面是它的一些主要应用场景：

（1）处理大量数据、处理不稳定的读写压力、快速启动服务；

（2）能够在数据稳定性与性能之间进行细微的调控；

（3）可以大大提key-value和高缓存层存储的性能，在现有的硬件条件下，更高的应用负载可以被轻松支持。

注：RethinkDB不需要预定义模式和表结构。

数据库连接设计

本系统的数据库连接设计定义了三个适配器graphql-data-source、ds-adaptor-rest、ds-adaptor-rethinkdb，便于前端通过restAPI来使用RethinkDB数据库。其原理图如下：

图4.2 数据库连接示意图

4.3系统模块设计

快捷按钮模块设计

快捷按钮模块如图4.3所示,从左到右依次是页面列表、可用组件、撤销、重做、向上移动选定组件、向下移动选定组件、克隆当前页面、删除当前页面、页面选项、定义页面类型。

图4.3 快捷按钮

该页面的主要处理代码如下：

```
class Container extends Component {
constructor(props) {
super(props);//继承父组件属性
this.state = {leftPanelOpen: false};
}
state = {
event:'',//初始化event状态
}
handleSaveAll(){
const that=this;
const {local,param,backend} = this.props.fields;
const savedProperties = {local:local,param:param, backend:backend};
this.props.saveProjectAndPageData(savedProperties)
}
render(){//每次状态改变的时候会重新调用render方法
const { popHistory, pushHistory } = this.props;//取出对应属性
```

```
return (
 〈div {...this.props}〉
 〈div style={{width: 0, height: 0}} ref="popoverAnchor"/〉
 〈div style={containerStyle}〉
 〈div style={leftContainerStyle}〉
 〈a href="/studio/home"〉 〈img src="/assets/images/demo_logo.png" /〉 〈/a〉
 〈/div〉
 〈div style={centerContainerStyle}〉
 〈IconButton//Page List按钮，其他按钮逻辑类似
style={iconBtnStyle}
iconStyle={iconStyle}
onTouchTap={(e) =〉 { e.preventDefault();
e.stopPropagation();
let leftPanelOpen = !this.state.leftPanelOpen;
this.setState({leftPanelOpen: leftPanelOpen});
toggleLeftPanel(
leftPanelOpen ? this.props.leftPanelWidth : 0,
( 〈PageListPanel /〉 )
);
}}
tooltip="Page List"
touch={true}
〉
 〈List
color={hpeGray} hoverColor={hpeGreen}
onClick={this.handleClick.bind(this)}
/〉
 〈/IconButton〉
 〈/div〉
 〈/div〉
 〈/div〉
)
```

```
}
}
export default connect(modelSelector, containerActions)(Container);
```

点击相应的按钮即可激活相应按钮的功能或者配置页面。

基础组件模块设计

基础组件。提供可选的预定义的内置基础组件，也可根据组件的名字搜索可选组件，可选组件选择框如图4.4所示：

图4.4 预定义的基础组件

注：该页面的组件大部分来自开源的Material UI、Grommet，代码将不再一一贴出。

属性配置模块设计

属性配置。你可以在这里设置组件元素的样式、属性、事件和布局。如图4.5所示：

图4.5 属性配置

其代码如下：

```
class Container extends React.Component {
constructor(props) {
super(props);//继承父组件属性并绑定组件内部的自定义方法
this.handleAddNewProp = this.handleAddNewProp.bind(this);
//还有其他绑定方法……
}
//以下是事件处理方法示例，只写一例，其它方法不再重复列出：
handleChangeOption(path, value) {//修改当前选择的组件
let optionObject = {};
set(optionObject, path, value);
const {currentComponent, changeOption} = this.props;
if (this.handleChangeOptionTimer) {
clearTimeout(this.handleChangeOptionTimer);
delete this.handleChangeOptionTimer;
}//设置定时器，自动保存修改
this.handleChangeOptionTimer = setTimeout(() => {
delete this.handleChangeOptionTimer;
changeOption(currentComponent, optionObject);
}, 500);
}
```

```
render() {//渲染方法
//初始化数据示例：
const {currentComponent} = this.props;
let panelContent = null;
return (
 〈Tabs id="top-panel-tabs" tabItemContainerStyle ={labelStyle} inkBarStyle={inkBarStyle}〉
 〈Tab label="Property" style={secondTabStyle}〉
 〈div style={{"overflow-y":"auto","overflow-x":"hidden","height":"900px"}}〉
 〈Group active={[0]}〉
 〈GroupItem heading="Basic" tag="h5"〉
 〈div style={labelStyle}〉
{
list.length〉0?
list.map(l =〉 {
return (
 〈FlexibleEdit//这个是自定义的组件，下面有定义
events={events}
name={l.name}
value={l.value}
type={l.type}
valueType={l.valueType}
properties ={listProperties}
/〉
);
})
:
 〈div〉There is no Property on selected Item〈/div〉
}
 〈/div〉
 〈/GroupItem〉
 〈/Group〉
 〈/div〉
```

```
〈/Tab〉
〈Tab label="Event" style={secondTabStyle}〉
〈div style={{"overflow-y":"auto","overflow-x":"hidden","height":"875px"}}〉
〈Group active={[0]} style={overflowStyle}〉
〈GroupItem heading="Basic" tag="h5"〉
{
eventData.length〉0?
eventData.map(
l =〉 {
return (
〈ComponentEventPanel
events = {events}
field = {l}
/〉
)
}
)
:
〈div〉There is no Events on selected Item〈/div〉
}
〈/GroupItem〉
〈/Group〉
〈/div〉
〈/Tab〉
〈Tab label="Style" style={secondTabStyle}〉 〈/Tab〉
〈Tab label="Layer" style={secondTabStyle}〉 〈/Tab〉
〈/Tabs〉
);
}
else {
return (
〈div style={labelStyle}〉
```

```
〈h4 className='text-center'〉Properties are not available〈/h4〉
〈/div〉
)
}
}
}
class FlexibleEdit extends React.Component{//自定义的可变组件
constructor(props){
super(props);//状态中保存了值类型，可根据值类型用不同的组件展示
this.state = {valueType:this.getValueType(this.props.valueType), showPopover:false};
}
componentWillReceiveProps(nextProps){
this.setState({valueType:this.getValueType(nextProps.valueType)})
}
handleChange = (event, index, value) => this.setState({valueType:value});
handleClick = ()=〉this.setState({"showPopover":true})
handleRequestClose = () => {//关闭请求
this.setState({
showPopover: false,
});
};
render(){//渲染方法，主要看return语句块
let events = this.props.events;
const type = this.props.type;
var renderField;
switch(type){
case 'date': renderField =//日期组件
〈DateField
events={events}
name={this.props.name}
type={this.props.type}
value={this.props.value}
```

```
/>
break;
//case其他组件代码类似….
}
return (
 〈div style={{"marginLeft":"40px", "marginRight":"20px"}}〉
 〈/div〉
 〈Box direction="row"  style={{"border":"1px"}} full="horizontal" 〉
{
this.state.valueType=="constant"?
renderField:false
}
 〈/Box〉
{
this.state.valueType=="variable"?
 〈TriggerCell
events={events}
name={this.props.name}
type={this.props.type}
value={this.props.value}
selectValue={this.handleClick.bind(this)}
/〉 :false
}
 〈div ref="item1"/〉
 〈Popover
open={this.state.showPopover}
anchorEl={this.refs.item1}
anchorOrigin={{horizontal: 'left', vertical: 'bottom'}}
targetOrigin={{horizontal: 'left', vertical: 'top'}}
style={{width:"300px", height:"500px" }} onblur={this.closePopover}
onRequestClose={this.handleRequestClose}
〉
```

```
〈SelectableList style={{"z-index":"10", webkitAppearance:"none"}}
onChange={this.handleRequestChange}〉
"{ this.props.properties}"
〈/SelectableList〉
〈/Popover〉
{
this.state.valueType=="expression"?
〈div className="property" style={{"margin-left":"-38px"}}〉
〈CodeMirrorCell
events={events}
name={this.props.name}
type={this.props.type}
/〉 〈/div〉:false
}
〈/div〉
)
}
}
//代码编辑组件、用到了开源的CodeMirror
class CodeMirrorCell extends React.Component{
constructor(props){
super(props);
}
state ={
value:this.props.value
}
handleCodeChange = (newCode) =〉 {
this.setState({value:newCode});
this.props.events.onChange(this.props.name,{__DATASOURCE:"expression", value:newCode});
}
componentWillReceiveProps(nextProps){
this.setState({value:nextProps.value})
```

```
}
render(){
let options = {
lineNumbers: true,
readOnly: false,
mode:'javascript'
};
return (//代码编辑组件，来自github
 〈CodeMirror value={this.state.value} onChange={this.handleCodeChange.bind(this)} options={options} style={{height:"100px"}}  className="CodeMirror_Custom" />
)
}
}
//其他自定义组件
//class TriggerCell extends React.Component//bool组件
//class TimeField extends React.Component//时间组件
//class DateField extends React.Component//日期组件
//class NumberField extends React.Component//数字组件
//class ColorFieldText extends React.Component//颜色组件，暂未完全实现
//class CheckBoxField extends React.Component//组合复选框组件
//class SelectField extends React.Component//选择框组件
//class EditText extends React.Component//文本编辑组件
//class OwnField extends React.Component//自定义的基础组件
export default connect(modelSelector, containerActions)(Container);
```

路由配置模块设计

路由配置。路由，也就是组件或HTML页面的访问路径，组件名称与路由绑定，组件既可以单独访问，也可以当作组件被其他组件使用。路由配置如图4.6所示：

图4.6 路由配置

注：路由配置的逻辑并不复杂，在此将不再贴出代码。

页面设计模块设计

页面设计。它是进行网站设计的主要模块，通过选中需要调整的组件进行组件属性、事件、样式的调整。

图4.7 页面设计模块

代码如下：

```
class DeskPage extends Component {//主页面
```

```
constructor(props) {
super(props);//继承父组件的属性和方法
this.handleComponentClick = this.handleComponentClick.bind(this);
}
componentDidMount(){//组件完成加载之后调用的方法
const domNode = ReactDOM.findDOMNode(this);
const { componentModel:{pages, currentPagePath} } = this.props;
loadPage();
this.contentDocument = domNode.contentDocument;
this.contentWindow = domNode.contentWindow;
this.setupShortcuts();
domNode.onload = ( () => {//dom节点初次加载时调用的方法
this.contentWindow.__pages = pages;
this.contentWindow.onPageDidMount = (page, pathname) => {
this.page = page;//页面加载完成之后，初始化数据
//下面定义了页面设计时，相关的操作方法
page.bindToState('onCut', (key, isModifier) => { setForCuttingKeys([key]); });
page.bindToState('onCopy', (key, isModifier) => { setForCopyingKeys([key]); });
page.bindToState('onClone', (key, isModifier) => { cloneSelected(); });
const initPage = () => {//页面初始化方法
this.contentWindow.__createPageDesk();
wait(() => this.contentWindow.pageReadyState === 'initialized', pageLoaded);
};
wait(() => this.contentWindow.pageReadyState === 'ready', initPage);
});
}
componentWillUnmount(){//组件即将销毁之前调用的方法
this.removeShortcuts();
this.contentDocument = undefined;
this.contentWindow = undefined;
this.page = undefined;
}
```

```
componentWillReceiveProps(nextProps){//组件属性被更新之后调用的方法
const { componentModel } = this.props;
const { componentModel: newComponentModel } = nextProps;
if(newComponentModel.reloadPageCounter != componentModel.reloadPageCounter){
var domNode = ReactDOM.findDOMNode(this);
domNode.src = '/app' + newComponentModel.currentPagePath;
} else if(newComponentModel.pagePathToChange != null &&
newComponentModel.pagePathToChange!= componentModel.pagePathToChange){
if(this.contentWindow){
this.contentWindow.__switchToPath(newComponentModel.pagePathToChange);
}
}
}
shouldComponentUpdate(nextProps, nextState){//判断组件是否应当更新
const { componentModel } = this.props;
const { componentModel: newComponentModel } = nextProps;
return (
nextProps.style.width !== this.props.style.width
newComponentModel.isEditModeOn !== componentModel.isEditModeOn
newComponentModel.markedUpdateCounter!==componentModel.markedUpdateCounter
newComponentModel.modelUpdateCounter !== componentModel.modelUpdateCounter
);
}
componentDidUpdate(){//组件更新之后
this.setupShortcuts();
if(this.page){
if(this.doUpdatePageModel){
const { componentModel } = this.props;
this.page.updatePageModel({
pathname: componentModel.currentPagePath
});
}
```

```
if(this.doUpdateMarks){
this.page.updateMarks();
}
}
}
handleComponentClick(key, isModifier){//组件单击之后调用的方法
const { setSelectedKey } = this.props;
setSelectedKey(key, isModifier);
}
handlePathnameChanged(pathname){//路由改变
const { changePageRouteFeedback } = this.props;
changePageRouteFeedback(pathname);
}
handleKeyDown(e){//快捷键事件处理
let contentEditableElement = document.activeElement.attributes['contenteditable']
? document.activeElement.attributes['contenteditable'].value : false;
let elementNameUpperCase = document.activeElement.tagName.toUpperCase();
if(elementNameUpperCase !== 'INPUT'
&& elementNameUpperCase !== 'TEXTAREA'
&& elementNameUpperCase !== 'SELECT'
&& !contentEditableElement && !window.getSelection().toString()){
if (e.which === 8  e.which === 46) { // Del or Backspace key
const { deleteSelected } = this.props;
e.stopPropagation();
e.preventDefault();
deleteSelected();
} else if (e.metaKey  e.ctrlKey) {
const { selectionBreadcrumbsModel:{selectedKeys} } = this.props;
switch (e.which) {
case 68: // D key
e.stopPropagation();  //禁止按键默认行为
e.preventDefault();
```

```
cloneSelected();
break;
//其他按键
// case 67: // C key
// case 65: // A key
// case 73: // I key
// case 86: // V key
// case 88: // X key
// case 90: // Z key
default:
break;
}
}
}
}
setupShortcuts(){//预览页面
const { componentModel:{isEditModeOn} } = this.props;
if(isEditModeOn){
if(this.contentWindow){
this.contentWindow.addEventListener('keydown', this.handleKeyDown, false);
}
window.addEventListener('keydown', this.handleKeyDown, false);
} else {
this.removeShortcuts();
}
}
removeShortcuts(){//取消预览
if(this.contentWindow) {
this.contentWindow.removeEventListener('keydown', this.handleKeyDown, false);
}
window.removeEventListener('keydown', this.handleKeyDown, false);
}
```

```
render(){//渲染方法
return ( <iframe {...this.props} src="/app/deskpage" /> );
}
}
export default connect(modelSelector, containerActions)(DeskPage);
```

4.3.6页面预览模块设计

页面预览。这一模块可以预览设计好的页面。

图4.8 页面预览

注：代码和上一模块极为类似，在此不再贴出。

5 可视化网站设计示例

下面是一个典型的Web应用配置过程：

步骤1，配置服务的基本信息（数据库可选Postgresql、Oracle），如图5.1：

图5.1 基础配置

步骤2，填写数据库相关信息，如图5.2：

图5.2 数据库连接配置

步骤3，填写管理员账号与密码，如图5.3：

图5.3 管理员配置

步骤4，配置LDAP，可选配置外部LDAP，如图5.4：

图5.4 LDAP配置

步骤5，开始安装，如图5.5：

图5.5 开始安装

结论

本文基于react.js、node.js和RethinkDB提出了一个在线的可视化网站设计系统的开发方案，以B/S的模式来支持设计平台的运行。在本系统的设计与开发过程中，我查阅大量的中英文资料、阅读各类教材。同时和老师同学进行探讨，对可视化网站设计有了相对深入的了解和认识。这个课题的研究使我更加深入的了解了JavaScript和NoSql。本人水平所限，所做的工作有很多疏漏和不足。最后总结全文，这段时间的工作成果有以下几点：

广泛查阅相关资料，跟踪当前和未来国内外可视化建站技术的发展趋势。提高了文档阅读能力，增强了解决问题的能力。

获得了有关Web服务、非关系型数据库和编程方面的知识。

采用JavaScript全栈开发技术，完成了可视化网站设计系统的整体设计，实现了在线的可视化网站设计系统。

本文所设计的在线的可视化网站设计系统主要具有以下几个特点：

本系统简单实用，具备一点JavaScript知识即可很快地掌握如何使用系统中提供的大多数功能。

本系统具有组件自定义功能，通过组合已有组件来构建新的组件。组件代码会自动生成，保证了代码的质量，降低了出错的概率。

本系统完全通过浏览器来操作，为设计者提供了方便，客户端安装广泛使用的Google Chrome浏览器就可以了。

本系统在服务器端执行维护，同时实现了多人协作构建，不需要C/S模式那样麻烦的系统维护。
本系统使用了react.js、node.js、RethinkDB和GraphQL等前沿技术，保证了系统的可扩展性和可维护性。

参考文献

[1] 周慧林. 可视化网站内容管理系统的设计[D].中国地质大学（北京）,2013.
[2] 刘一田,刘士进. 可视化Web设计器[J]. 计算机系统应用,2015,(10):80-84.
[3] Chiuhsiang Joe Lin,Tsung-Ling Hsieh. Exploring the Design Criteria of Website Interfaces for Gender[J]. International Journal of Industrial Ergonomics,2016,:.
[4] 查修齐,吴荣泉,高元钧. C/S到B/S模式转换的技术研究[J]. 计算机工程,2014,(01):263-267.
[5] 葛鼎新. 基于web的产品配置系统研究及实现[D].河北工业大学,2010.
[6] 姚立. IBM云计算平台下NodeJS应用支持环境的设计与实现[D].哈尔滨工业大学,2013.
[7] 仰燕兰,金晓雪,叶桦. ASP.NET AJAX框架研究及其在Web开发中的应用[J]. 计算机应用与软件,2011,(06):195-198.
[8] 陆洲. Java EE核心模式研究[J]. 计算机与数字工程,2014,(01):81-84.
[9] 张恺. 可视化跨平台移动应用开发环境的设计与实现[D].北京邮电大学,2015.
[10] 许会元,何利力. NodeJS的异步非阻塞I/O研究[J]. 工业控制计算机,2015,(03):127-129.
[11] 程桂花,沈炜,何松林,张珂杰. Node.js中Express框架路由机制的研究[J]. 工业控制计算机,2016,(08):101-102.
[12] 占东明,洪家伟,陈希杨,徐礼飞,辛鄢放. Web新兴前端框架与模式研究[J]. 电子商务,2016,(10):65-66.
[13] 戴翔宇. Web前端工程组件化的分析与改进[D].吉林大学,2016.
[14] Walsh L, Akhmechet V, Glukhovsky M. Rethinkdb -- rethinking database storage[J]. 2009.
[15] 范凯. NoSQL数据库综述[J]. 程序员,2010,(06):76-78.

致 谢

在论文即将完成之际，向所有给我指导、帮助和支持的人表示由衷的感谢。尤其要感谢我的论文指导老师蔡建宏，他在学习方面给了我很多帮助，在毕业设计上为我们作了认真的指导，使我从中学到了很多知识。他高度的敬业精神、严谨求实的治学态度、孜孜以求、兢兢业业的工作作风和锐意的进取精神对我产生了重要影响。他渊博的知识、敏锐的思维和开阔的视野给了我深深的启迪。在此祝愿他身体健康，全家幸福！
我要特别感谢帮助我写作论文的同学。在系统设计过程中，感谢他们提供的资料及建议，他们为我完成这篇论文提供了很大的帮助。还要感谢大学四年来的所有老师，为我打下了软件开发专业知识的基础。
感谢我的母校湖北大学。通过大学阶段的学习，提高了我们适应社会和自我发展的能力，为我们提供了一个绝佳的知识平台。

四、指标说明：

1. 总相似比即类似于重合率。总相似比即送检论文中与检测范围所有文献相似的部分（包括参考引用部分）占整个送检论文的比重，总相似比=复写率+引用率。
2. 引用率即送检论文中被系统识别为引用的部分占整个送检论文的比重（引用部分一般指正确标示引用的部分）。
3. 自写率即送检论文中剔除雷同片段和引用片段后占整个送检论文的比重，一般可用于论文的原创性和新颖性评价，自写率=1-复写率-引用率。
4. 复写率即送检论文中与检测范围所有文献相似的部分（不包括参考引用部分）占整个送检论文的比重。
5. 红色字体代表相似片段；浅蓝色字体代表引用片段、深蓝色字体代表可能遗漏的但被系统识别到与参考文献列表对应的引用片段；黑色字体代表自写片段。

五、免责声明：