[**Abstract** IV](#_Toc480643421)

[绪论 1](#_Toc480643422)

[1 在线的可视化网站设计概述 1](#_Toc480643423)

[1.1 在线的可视化网站设计的概念 1](#_Toc480643424)

[1.2 在线的可视化网站设计的发展和趋势 1](#_Toc480643425)

[1.3 课题的背景及意义 1](#_Toc480643426)

[1.4 本文的主要工作 1](#_Toc480643427)

[2 可视化网站设计相关技术的研究 2](#_Toc480643428)

[2.1 可视化网站设计的开发模式 2](#_Toc480643429)

[2.1.1 系统结构的选择 2](#_Toc480643430)

[2.1.2 开发环境的选择 2](#_Toc480643431)

[2.2 Node.js 技术 2](#_Toc480643432)

[2.2.1 Node.js工作原理 3](#_Toc480643433)

[2.2.2 Node.js Express框架 3](#_Toc480643434)

[2.2.3 Node.js访问RethinkDB数据库的主要步骤 5](#_Toc480643435)

[2.3 React.js技术简介 6](#_Toc480643436)

[2.3.1 React.js工作原理 6](#_Toc480643437)

[2.3.2 React.js的组件化 6](#_Toc480643438)

[2.4 RethinkDB数据库简介 7](#_Toc480643439)

[3 系统分析 8](#_Toc480643440)

[3.1 用户需求分析 8](#_Toc480643441)

[3.2 系统功能分析 8](#_Toc480643442)

[3.3 系统性能分析 8](#_Toc480643443)

[3.3.1 在线的可视化网站设计的规范化 8](#_Toc480643444)

[3.3.2 系统设计的先进性 9](#_Toc480643445)

[3.4 系统结构设计 9](#_Toc480643446)

[3.5 系统配置分析 9](#_Toc480643447)

[4 系统主要模块的实现与技术关键 10](#_Toc480643448)

[4.1 系统功能模块图 10](#_Toc480643449)

[4.2 数据库设计 10](#_Toc480643450)

[4.2.1数据库连接设计 10](#_Toc480643451)

[4.3 系统模块设计 11](#_Toc480643452)

[4.3.1 快捷按钮模块设计 11](#_Toc480643453)

[4.3.2 基础组件模块设计 11](#_Toc480643454)

[4.3.3 属性配置模块设计 11](#_Toc480643455)

[4.3.4 路由配置模块设计 12](#_Toc480643456)

[4.3.5 页面设计模块设计 12](#_Toc480643457)

[4.3.6 页面预览模块设计 12](#_Toc480643458)

[结论 12](#_Toc480643459)

[参考文献 13](#_Toc480643460)

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，构成网站系统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。传统的Web应用软件开发方法明显不能很好地满足 Web 应用的设计需求。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。基于Node.js、React.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.js React.js RethinkDB 异步请求

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 Asynchronous request

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、开发者通过自定义、组合、嵌套等操作并结合事件绑定、属性设置设计网站系统。这是保证开发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]

可视化网站设计，它的发展历史不长，随着技术的进步和人们认识过程的深化，可视化网站设计工具也在不断地增强功能。这类工具大多是集[网页](http://baike.baidu.com/item/%E7%BD%91%E9%A1%B5)制作和管理网站于一身的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]。

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。由于各个政府部门、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。它们的出现都大大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，只要想了解打开互联网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：

① 进行系统的整体分析、设计，确定了系统的B/S模式体系结构和开发工具。

② 通过比较，确定了系统的开发工具和后台数据库。

③ 进行用户需求分析，确定系统的总体功能结构。

④ 本文实现了在线的可视化网站设计的部分基本功能模块，包括：

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

2.1.1 系统结构的选择

（1）B/S模式[4]

B/S模式的系统以服务器为核心，把所有的功能都实现在Web服务器上，并就不同的功能为开发者和用户设置相对应的权限。开发者通过调用服务器上的API和自定义组件完成网站设计，用户通过HTTP请求在权限范围内调用Web服务器上不同API完成相应的工作。B/S模式易升级，易维护，易使用，可移植性强。同时，B/S模式灵活性较差、应用模式简单。目前B/S模式是被广泛应用的系统模式。

（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模式专业化程度高、安全性好、开发手段灵活、运行速度快、交互性强，但开发成本高，维护复杂，升级麻烦。

（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。根据系统的属性确定，将在线的可视化网站设计的开发结构确定为B/S模式。

2.1.2 开发环境的选择

（1）Node.js[6]

Node.js是服务端的JavaScript运行时环境。它基于Chrome V8 JavaScript引擎，具有无阻塞和事件驱动的特色。Node.js也实现了类似Apache和Nginx的web服务。结合React.js和Angular.js等前端框架，可以使用JavaScript开发全栈应用。

（2）ASP.NET[7]

ASP.NET框架具有很好的扩展性和灵活性。它与Windows操作系统紧密结合，可以与其他应用方便地进行数据交换。ASP.NET框架下有很多开发工具，主要是VB、 VC、ASP.NET、C#，特别是 ASP.NET框架应用广泛，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适用于大规模数据处理，具有很强稳定性和安全性，适合开发浏览器形式的应用系统，具有很好的灵活性，在开发模式上与ASP.NET相似。

Node.js能够应付大量的数据请求，而不会造成大量的阻塞。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。

2.2 Node.js 技术[9]

Node.js是一个基于Chrome V8 JavaScript运行时环境建立的平台，用于方便地构建响应速度快、易于扩展的互联网应用。Node.js 采用事件驱动机制，非阻塞I/O 模型，因而轻量和高效，特别适合在分布式设备上运行数据密集型的实时应用。

2.2.1 Node.js工作原理[10]

（1）V8引擎解析JavaScript脚本。

（2）解析后的代码，调用Node API。

（3）[LIBUV库](https://github.com/joyent/libuv)负责Node API的执行。它将不同的任务分配给多个不同的线程，形成一个Event Loop（事件循环），以异步执行的方式将执行结果返回给V8引擎。

（4）V8引擎再将执行结果返回给用户。

Node.js的工作原理如图2.1所示：

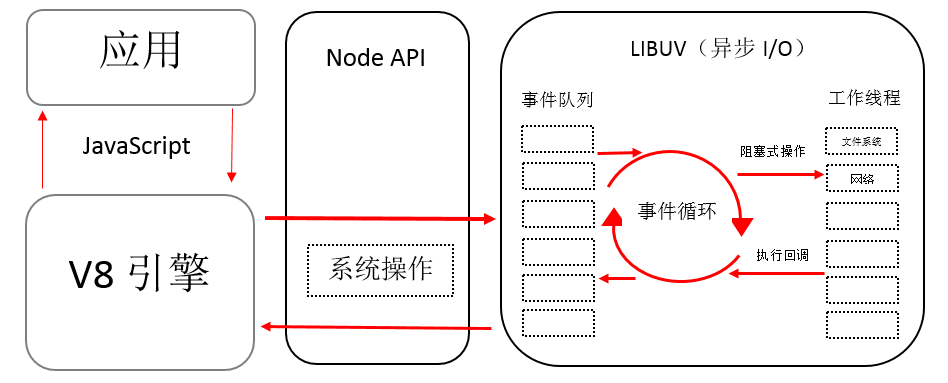


图 2.1 Node.js 工作原理

2.2.2 Node.js Express框架[11]

Express 是一个简洁而灵活的 node.js Web应用框架, 提供了强大特性和丰富的 HTTP 工具帮助开发者创建Web 应用。使用 Express 框架可以快速搭建一个功能完整的网站。

2.2.2.1 Request对象

 Request 对象代表了 HTTP 请求，包含了请求查询字符串，内容，参数，HTTP 头部等属性。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「请求主体」/ Cookies

(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parameters

(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：返回指定字符集的第一个可接受字符编码

(15) req.get()：获取指定的HTTP请求头

(16) req.is()：判断请求头Content-Type的MIME类型

2.2.2.2 Response对象

  Response 对象代表了 HTTP 响应，在接收到请求之后向客户端发送HTTP响应数据。常见属性如下：

1. res.app：同req.app一样
2. res.append()：追加指定HTTP头
3. res.set()在res.append()后将重置之前设置的头
4. res.cookie(name，value [，option])：设置Cookie
5. opition: domain / expires / httpOnly / maxAge / path / secure / signed
6. res.clearCookie()：清除Cookie
7. res.download()：传送指定路径的文件
8. res.get()：返回指定的HTTP头
9. res.json()：传送JSON响应
10. res.jsonp()：传送JSONP响应
11. res.location()：只设置响应的Location HTTP头，不设置close response或者状态码
12. res.redirect()：设置响应的Location HTTP头，并且设置状态码302
13. res.send()：传送HTTP响应
14. res.sendFile(path [，options] [，fn])：传送指定路径的文件，通常会根据文件extension自动设定Content-Type
15. res.set()：设置HTTP头，传入object可以一次设置多个头
16. res.status()：设置HTTP状态码
17. 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。在HTTP请求中，我们可以通过路由提取请求的URL和GET/POST参数。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 (req, res) {

console.log("主页 POST 请求");

res.send('Hello POST');

})

var server = app.listen(8081, function () {

var host = server.address().address

var port = server.address().port

console.log("应用实例，访问地址为 http://%s:%s", host, port)

})

2.2.3 Node.js访问RethinkDB数据库的主要步骤

（1）引入RethinkDB模块

Port用于指定需要连接的rethinkDB数据库的端口，host用于指定需要连接的rethinkDB数据库的IP，db用于指定需要连接的数据库的名称。

var r = require('rethinkdbdash')({

port: 28015,

host: 'localhost'

db:’myDB’

});

（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('Success ',response);

//响应成功

})

.error(function(err){

console.log('error occurred ',err);

//响应失败

})

（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);

});

2.3 React.js技术简介

2.3.1 React.js工作原理[12]

在Web开发中，我们经常需要将变化的数据实时反应到Web UI上，这时就需要对DOM（文档对象模型）进行操作。而复杂或频繁的DOM操作通常是性能瓶颈产生的原因。

React引入了虚拟DOM（Virtual DOM）的机制：在浏览器端用Javascript实现了一套DOM API。基于React进行开发的时候，所有的DOM构造都是通过虚拟DOM进行的，每当数据变化时，React都会重新构建整个DOM树，然后React将当前整个DOM树和上一次的DOM树进行对比，对比DOM结构的区别，然后只要将需要变化的部分进行实际的浏览器DOM渲染。而且React能够批处理虚拟DOM的更新，在一个事件循环（Event Loop）内的两次数据变化会被合并，例如：你连续的先将节点内容从X变成Y，然后又从Y变成X，React会认为UI没有变化，如果通过手动操作，这种逻辑通常是相当复杂的。尽管每一次都需要构造完整的虚拟DOM树，但是因为虚拟DOM是内存数据，性能是非常高的，而对实际DOM进行操作的仅仅是Diff部分，从而达到提高性能的目的。这样，在保证性能的同时，开发者将不必关注某个数据的变化会更新到一个或多个具体的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Render的。

通常，服务器端的任务的就是根据数据Render出HTML文档并送到浏览器端。这时，用户如果要改变某个状态文字，也必须刷新整个页面。服务器端并不需要知道是哪段HTML被改变了，而只需要根据更新之后的数据来刷新整个页面。React以高性能的方式将这种开发模式带到了前端，UI界面每一次的更新，你都可以认为刷新了整个页面。如何进行局部更新以保证性能呢？这就是React框架要做的事情！

2.3.2 React.js的组件化[13]

虚拟DOM(virtual-dom)不仅带来了组件化开发的思想，同时也带来了简单的UI开发逻辑，所谓组件，就是封装起来的、具有独立功能的UI部件。React推荐以组件的方式去构建UI界面，每一个功能相对独立的模块都是UI组件，将小的组件通过嵌套或组合的方式构成大的组件，最终完成整体UI的构建。Facebook的instagram.com整站点都采用了React来开发，整个页面就是一个大大的组件，它嵌套了大量的其他组件。MVC的思想让你做到视图-数据-控制器的分离，组件化的思考方式带来了UI功能模块之间的分离。下面是一个评论框组件的示例：



图2.2 典型的评论框组件

React认为一个组件应该具有如下特征：

（1）可组合（Composeable）：一个组件应该易于和其它组件一同使用，或者嵌套在另一个组件内部。如果一个组件内部嵌套了另一个组件，那么说父组件拥有（own）它嵌套的子组件，通过这个特性，一个复杂的UI能够分拆成多个简单的UI组件；

（2）可重用（Reusable）：每个组件都是具备独立功能的，它可以应用在多个UI场景；

（3）可维护（Maintainable）：每个小的组件只包含自己的逻辑，容易维护和理解。

2.4 RethinkDB数据库简介[14]

RethinkDB是一个基于实时Web构建的、开源的、可扩展的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）不需要预定义模式：不用预定义数据模式、表结构。每条记录都可以有不同的属性和格式。每当插入数据时，开发者不需要预先定义它们的模式。

（2）无共享架构：不同于传统数据库将所有数据存储在存储区域网络中的全共享架构。RethinkDB将数据划分后存储在多个本地服务器上。从本地磁盘读取数据的性能明显优于通过网络传输读取数据的性能，因此RethinkDB数据库系统的性能高于传统数据库的性能。

（3）弹性可扩展：RethinkDB可以动态地增加或者删除结点。不用停机维护，数据也能够自动迁移。

（4）分区： RethinkDB数据库会将数据分区，把数据分散在多个节点上面。分区的同时会做复制。这样既可以保证不会有单点失效的问题，又可以提高并行性能。

（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。这样，数据就可以快速地写入一个节点，而不会因为网络传输引起迟延。缺点是并不能保证一致性，出现故障的时候，可能会导致少量的数据丢失。

（6）BASE：不同于传统数据库事务严格的ACID特性，RethinkDB数据库保证的是BASE特性。BASE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分析，找出可行的方案，为系统设计提供理论依据。在系统分析中分为用户需求分析、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。

在软、硬件方面对系统的需求，只需要一台稍微强大一点的服务器和若干客户端。服务端需要连接互联网，并能够正常安装node.js和RethinkDB软件。客户端对软件配置的要求是系统最好安装了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：

① 快捷按钮。提供菜单、新建页面、删除页面、撤销修改、回退修改等功能。

② 基础组件。提供可选的内置基础组件。

③ 属性配置。配置组件的属性和事件方法。

④ 路由配置。配置当前页面的访问路由。

⑤ 页面设计。综合使用上述工具进行页面设计。

⑥ 页面预览。预览设计得到的页面。

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

在线的可视化网站设计在技术上应具有一定的先进性，代表在线的可视化网站设计的先进水平，这样系统在构建完成后一段时间内，才不会因为技术落后而需要大规模的调整，并且能够随着逐步的升级而保持整个系统的先进性。同时要求系统对计算机硬件要求不能太高，运行速度要快。

3.4 系统结构设计

本设计采用基于B/S模式的网络结构体系，来设计在线的可视化网站设计系统。图3.1是典型的B/S体系结构。

B/S结构将在线的可视化网站设计系统中的三要素(数据、功能、行为)分离，形成客户层，逻辑表达层，应用层。设计者设计应用隔离而共享和控制业务逻辑；后端的设计隔离和服务层，处理客户端与数据库间的数据流，提供对专门数据服务的访问。B/S层次分离的优势体现在具有统一的传输协议和语言格式，以及界面风格统一为浏览器，。

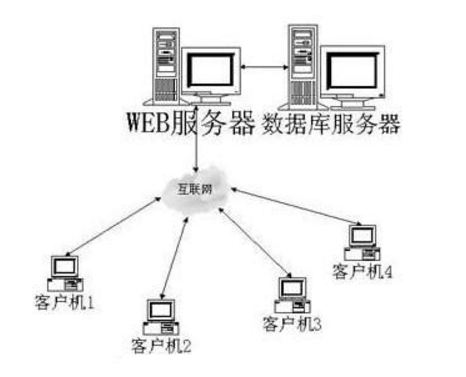


图3.1 B/S体系结构

3.5 系统配置分析

本文采用B/S（浏览器/服务器）模式，易于维护，并使前台客户机的负载减到最少。实现该模式的开发手段采用Node.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. 9G以上

软件配置：操作系统：Windows 2012 SERVER

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

根据需求分析和功能分析，对系统各功能进行集中、分块，按照结构化程序设计的要求，本系统的系统功能模块图如图4.1所示。

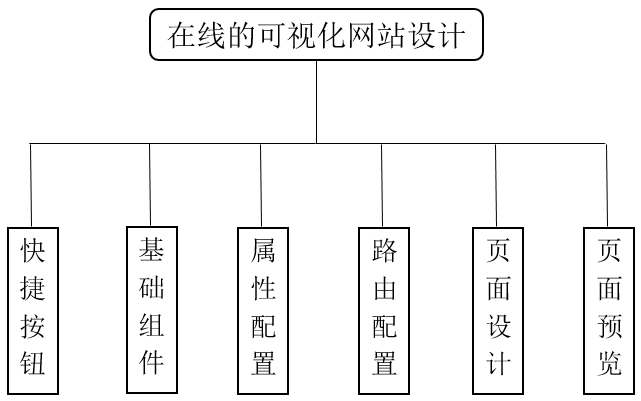


图4.1 系统功能模块图

4.2 数据库设计

本课题设计时，数据库采用的是RethinkDB，它是一种可扩展的、高性能的非关系型数据库。可以用来替代Memcached作为数据缓存层。或替代一些类似Memcached的存储，比如Membase，MemcacheDB，TokyoTyrant及Schooner Membrain等。RethinkDB为各种应用压力下提供高性能和高可靠性 支持，主要应用在下面一些场景：

（1）用来处理一些其它方案无法解决的服务负载（如：要求快速的服务启动，在数据量超出内存大小后还要有高的性能，处理各种不稳定的读写压力，处理大量数据等）

（2）提供性能与数据稳定性之间的精细调控

（3）大大提高你的缓存层和key-value存储的性能，在你现有的硬件条件下，能支持更高的应用负载。

4.2.1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：

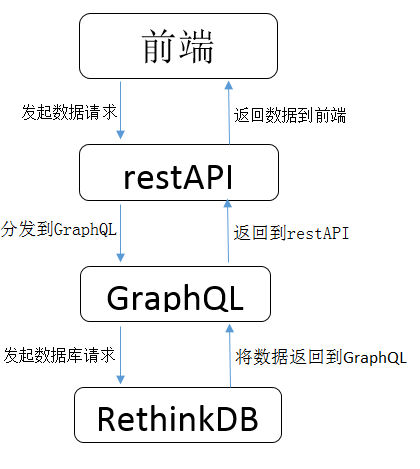


图4.2 数据库连接示意图

4.3 系统模块设计

4.3.1 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、复制当前页面、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：

import React, { Component, PropTypes } from 'react';

import { connect } from 'react-redux';

import { modelSelector } from './selectors.js';

import { containerActions } from './actions.js';

import { CHANGE\_OPTIONS, ADD\_NEW, DUPLICATE } from '../PageOptionsModal/actions.js';

import { SAVE\_PAGE\_PROPERTIES, SAVE\_PAGE\_TYPES } from '../DefinitionModal/actions.js';

import RaisedButton from '@hpe/chang-e-ui/components/Button';

import IconButton from 'material-ui/IconButton';

import DropDownMenu from 'material-ui/DropDownMenu';

import MenuItem from 'material-ui/MenuItem';

import Popover from 'material-ui/Popover';

import ClickToClose from 'material-ui/svg-icons/content/clear';

import StarIcon from 'material-ui/svg-icons/action/stars';

import SettingsIcon from 'material-ui/svg-icons/action/settings';

import SettingsIcon2 from 'material-ui/svg-icons/action/settings-applications';

import PageListPanel from '../PageListPanel';

import HistoryPanel from './HistoryPanel';

import LibraryPanel from '../LibraryPanel';

import { Code, Copy, Cut, Desktop, Down, Gears, History, List, Plus, Redo, Refresh, Trash, Undo, Up } from '../../../../framework/themes/SvgIcons';

import { hpeGray, hpeGreen } from '../../../../framework/themes/Themes';

import \* as actions from './actions.js';

const supportedWidths = [

'100%', '1800px', '1200px', '1100px', '1024px',

'1000px', '900px', '770px', '700px', '340px'

];

class Container extends Component {

static propTypes = {

leftPanelWidth: PropTypes.number

}

static defaultProps = {

leftPanelWidth: 300

}

constructor(props) {

super(props);

this.state = {leftPanelOpen: false};

}

state = {

event:'',

}

handleClick = (e) =>{

}

handleSaveAll(){

const that=this;

const {local,param,backend} = this.props.fields;

const savedProperties = {local:local,param:param, backend:backend};

this.props.saveProjectAndPageData(savedProperties)

}

render(){

const { componentModel, deskPageModel, deskModel, selectionBreadcrumbsModel: {selectedKeys}, sidebarModel } = this.props;

const { toggleSidebar, hideLeftPanel, toggleLeftPanel, hidePreviewComponent, togglePageTreeview, toggleQuickOptions } = this.props;

const { popHistory, pushHistory } = this.props;

const { moveSelected, setForCopyingKeys, setForCuttingKeys, deletePage, showPageOptionsModal } = this.props;

const { showDefinitionModal } = this.props;

const { changeViewportWidth } = this.props;

const { setLivePreviewModeOn, setEditModeOn, reloadPage } = this.props;

const { saveProject, exportModel ,saveProjectAndPageData} = this.props;

let containerStyle = {

display: 'flex',

flexDirection: 'row',

flexWrap: 'nowrap',

justifyContent: 'space-between',

alignItems: 'center',

zIndex: 1

};

let leftContainerStyle = {

display: 'flex',

flexDirection: 'row',

flexWrap: 'nowrap',

justifyContent: 'center',

alignItems: 'center',

'margin-left':'38px',

'margin-top':'15px'

};

let centerContainerStyle = {

flexDirection: 'row',

flexWrap: 'nowrap',

justifyContent: 'space-between',

alignItems: 'center',

width: '50%',

'margin-top':'15px'

};

let rightContainerStyle = {

display: 'flex',

flexDirection: 'row',

flexWrap: 'nowrap',

justifyContent: 'flex-end',

alignItems: 'center',

width: '30%'

};

let rightIconContainerStyle = {

display: 'flex',

flexDirection: 'row',

flexWrap: 'nowrap',

justifyContent: 'space-between',

alignItems: 'center',

width: '40%',

marginRight: 20

};

let rightBtnContainerStyle = {

display: 'flex',

flexDirection: 'row',

flexWrap: 'nowrap',

justifyContent: 'flex-end',

alignItems: 'stretch',

};

let iconBtnStyle = {

width: 28,

height: 28,

padding: 2,

'margin-right':'28px'

};

let iconStyle = {

width: 24,

height: 24

};

let closeBtnWidth = 24;

let closeBtnStyle = {

display: this.state.leftPanelOpen ? 'flex' : 'none',

flexDirection: 'row',

flexWrap: 'nowrap',

justifyContent: 'flex-end',

alignItems: 'flex-start',

width: '100%',

height: closeBtnWidth,

zIndex: 1500

};

let sidebarWidth = '0px';

let sidePanelWidth = '0px';

let widthsList = [];

supportedWidths.forEach((width)=> {

widthsList.push(

<MenuItem

key={width}

value={width}

primaryText={width}

/>

);

});

const saveAllStyle={

'background-color':'#00A982',

height:"52px",

'font-size':'16px',

'border':'none',

'color':'#FCFEFB'

}

const previewStyle={

height:"52px",

'font-size':'16px',

'border':'none'

}

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

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>

<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

disabled={!deskPageModel.isEditModeOn}

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

hidePreviewComponent();

let leftPanelOpen = !this.state.leftPanelOpen;

this.setState({leftPanelOpen: leftPanelOpen});

toggleLeftPanel(

leftPanelOpen ? this.props.leftPanelWidth : 0,

<LibraryPanel />

);

}}

tooltip="Show the list of available components"

touch={true}

>

<Plus

color={hpeGray} hoverColor={hpeGreen}

onClick={this.handleClick.bind(this)}

/>

</IconButton>

<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

popHistory();

}}

tooltip="Undo"

touch={true}

>

<Undo color={hpeGray} hoverColor={hpeGreen}

onClick={this.handleClick.bind(this)}

/>

</IconButton>

<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

pushHistory();

}}

tooltip="Redo"

touch={true}

>

<Redo color={hpeGray} hoverColor={hpeGreen}

onClick={this.handleClick.bind(this)}

/>

</IconButton>

<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

moveSelected(true);

}}

tooltip="Move up selected components within their parents"

touch={true}

>

<Up color={hpeGray} hoverColor={hpeGreen}

onClick={this.handleClick.bind(this)}

/>

</IconButton>

<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

moveSelected(false);

}}

tooltip="Move down selected components within their parents"

touch={true}

>

<Down color={hpeGray} hoverColor={hpeGreen}

onClick={this.handleClick.bind(this)}

/>

</IconButton>

<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

showPageOptionsModal(DUPLICATE);

}}

tooltip="Clone current page"

touch={true}

>

<Copy color={hpeGray} hoverColor={hpeGreen}

onClick={this.handleClick.bind(this)}

/>

</IconButton>

<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

onTouchTap={(e) => { e.stopPropagation(); e.preventDefault();

if(confirm('Are you sure you want to delete current page?')) {

deletePage();

}

}}

tooltip="Delete current page"

touch={true}

>

<Trash color={hpeGray} hoverColor={hpeGreen}

onClick={this.handleClick.bind(this)}

/>

</IconButton>

<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

onTouchTap={(e) => { e.stopPropagation(); e.preventDefault();

showPageOptionsModal(CHANGE\_OPTIONS);

}}

tooltip="Page options"

touch={true}

>

<Gears color={hpeGray} hoverColor={hpeGreen}

onClick={this.handleClick.bind(this)}

/>

</IconButton>

<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

onTouchTap={(e) => { e.stopPropagation(); e.preventDefault();

showDefinitionModal(SAVE\_PAGE\_PROPERTIES, 'Page Properties');

}}

tooltip="Define Page Types"

touch={true}

>

<SettingsIcon color={hpeGray} hoverColor={hpeGreen}

onClick={this.handleClick.bind(this)}

/>

</IconButton>

{/\*<IconButton

style={iconBtnStyle}

iconStyle={iconStyle}

onTouchTap={(e) => { e.stopPropagation(); e.preventDefault();

showDefinitionModal(SAVE\_PAGE\_TYPES, 'Page Types');

}}

tooltip="Define Types"

touch={true}

>

<SettingsIcon2 color={hpeGray} hoverColor={hpeGreen} />

</IconButton>\*/}

</div>

<div style={rightContainerStyle}>

<div style={rightIconContainerStyle}>

<DropDownMenu

value={deskModel.iframeWidth}

onChange={(e, index, value) => { e.preventDefault(); e.stopPropagation();

changeViewportWidth(value);

}}

iconStyle={{top: 0}}

style={{height: 24}}

labelStyle={{lineHeight: '24px'}}

>

{widthsList}

</DropDownMenu>

<IconButton

style={iconBtnStyle}

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

reloadPage();

}}

tooltip="Reload current page. State will be lost."

touch={true}

>

<Refresh color={hpeGray} hoverColor={hpeGreen} />

</IconButton>

<IconButton

style={iconBtnStyle}

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

/\* TODO: Switch between Desktop/Mobile mode. \*/

}}

tooltip="Screen Mode"

touch={true}

>

<Desktop color={hpeGray} hoverColor={hpeGreen} />

</IconButton>

</div>

<div style={rightBtnContainerStyle}>

<RaisedButton

style={previewStyle}

label="Preview"

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation(); toggleSidebar(0);

setLivePreviewModeOn();

}}

disabled={false}

/>

<RaisedButton

style={saveAllStyle}

label="Save"

primary={true}

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

this.handleSaveAll();

// saveProject();

}}

disabled={false}

/>

</div>

</div>

</div>

<Popover

open={this.state.leftPanelOpen}

anchorEl={this.refs.popoverAnchor}

anchorOrigin={{horizontal: 'right', vertical: 'top'}}

targetOrigin={{horizontal: 'left', vertical: 'top'}}

useLayerForClickAway={false}

style={{

width: this.props.leftPanelWidth,

height: '100%',

overflow: 'auto',

marginTop:'53px',

}}

>

<div style={closeBtnStyle}>

<IconButton

onTouchTap={(e) => { e.preventDefault(); e.stopPropagation();

this.setState({leftPanelOpen: false});

hideLeftPanel();

}}

tooltip="Close"

tooltipPosition="bottom-left"

touch={true}

style={{padding: 0, width: closeBtnWidth, height: closeBtnWidth}}

iconStyle={{width: closeBtnWidth, height: closeBtnWidth}}

>

<ClickToClose />

</IconButton>

</div>

{componentModel.leftPanelContent}

</Popover>

</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。

4.3.2 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，如图4.4所示：

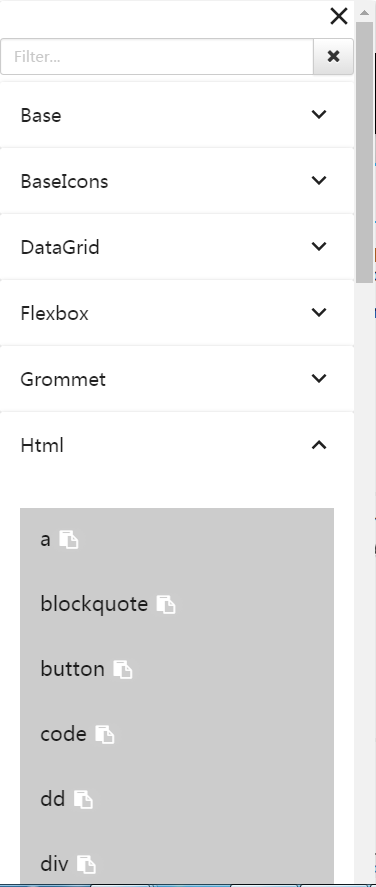


图4.4 预定义的基础组件

注：该页面的组件大部分来自开源的Material UI、Grommet，在此不再将代码一一贴出。

4.3.3 属性配置模块设计

属性配置。提供对组件元素的属性、事件、样式、布局的设置。如图4.5所示：

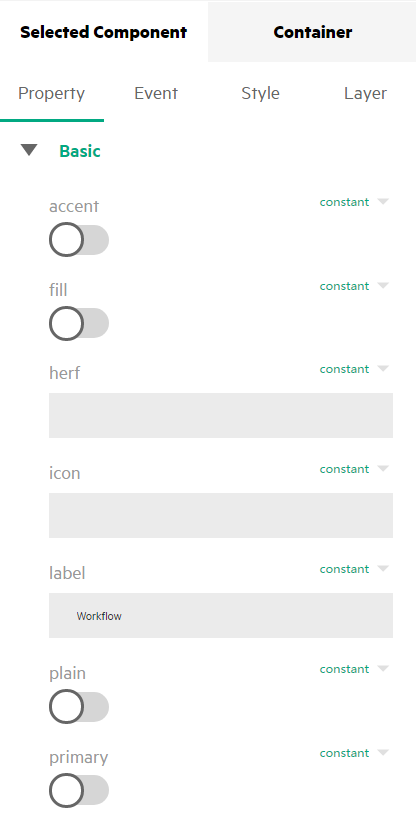


图4.5 属性配置

其代码如下：

import React, { Component, PropTypes } from 'react';

import TextInput from 'grommet/components/TextInput';

import FormField from 'grommet/components/FormField'

import DownIcon from 'grommet/components/icons/base/Down';

import AddIcon from 'grommet/components/icons/base/Add';

import SubtractIcon from 'grommet/components/icons/base/Subtract';

import CheckBox from 'grommet/components/CheckBox';

import DateTime from 'grommet/components/DateTime';

import Button from 'grommet/components/Button';

import { connect } from 'react-redux';

import {forEach, set, forOwn, isObject} from 'lodash';

import {Group,GroupItem,Box} from '@hpe/chang-e-ui'

import { Tabs, Tab } from 'material-ui';

import { modelSelector } from './selectors.js';

import { containerActions } from './actions.js';

import DropDownMenu from 'material-ui/DropDownMenu';

import MenuItem from 'material-ui/MenuItem';

import {List, ListItem , MakeSelectable} from 'material-ui/List';

import {Popover} from 'material-ui';

import \* as commondataApi from '../../../api/model/commonDataApi.js';

import CodeMirror from 'react-codemirror';

import 'codemirror/mode/javascript/javascript';

import './codemirror\_custom.scss';

import ComponentEventPanel from './ComponentEventPanel'

let SelectableList = MakeSelectable(List);

let processedTypeList = [];

class Container extends React.Component {

constructor(props) {

super(props);

this.handleAddNewProp = this.handleAddNewProp.bind(this);

this.handleChangeOption = this.handleChangeOption.bind(this);

this.handleDeleteOption = this.handleDeleteOption.bind(this);

this.state = {pageTypes:commondataApi.getTypeDefinitions()};

}

handleAddNewProp(path) {

this.handleChangeOption(path, null);

}

handleChangeOption(path, value) {

let optionObject = {};

set(optionObject, path, value);

const {currentComponent, changeOption} = this.props;

if (this.handleChangeOptionTimer) {

clearTimeout(this.handleChangeOptionTimer);

delete this.handleChangeOptionTimer;

}

this.handleChangeOptionTimer = setTimeout(() => {

delete this.handleChangeOptionTimer;

changeOption(currentComponent, optionObject);

}, 500);

}

handleDeleteOption(path) {

const {currentComponent, deleteOption} = this.props;

deleteOption(currentComponent, path);

}

getListRawData(){

const pageData = this.props.pageDataModel;

let properties = [];

properties.push({name:"local",type:"object",properties:this.\_getProperties(pageData.local)});

properties.push({name:"param",type:"object",properties:this.\_getProperties(pageData.param)});

properties.push({name:"remote",type:"object",properties:this.\_getBackendData(pageData.backend)})

return properties

}

\_getBackendData(properties=[]){

let propertyArray = [];

forEach(properties, (property)=> {

let temProp = {name:property.name};

temProp['type'] = 'object';

let temProperties =[];

forEach(property.backList, (backField)=> {

let tempBackendQuery = {name:backField.name,type:"object"};

let temMap = {};

tempBackendQuery["properties"] = this.\_getPropertiesBySourceType(commondataApi.getBaseTypeForList(backField.sourceType), temMap);

temProperties.push(tempBackendQuery);

})

temProp['properties'] = temProperties;

propertyArray.push(temProp);

})

return propertyArray;

}

\_getPropertiesBySourceType(graphqlType, tempMap){

if(tempMap[graphqlType]){

return null;

}

tempMap[graphqlType]=graphqlType;

let propertyArray = [];

let fields = commondataApi.getGraphQLTypeDetail(graphqlType)?commondataApi.getGraphQLTypeDetail(graphqlType).fields:null;

if(fields == null){

return null;

}

forEach(fields, (field)=> {

let tempObj = {name: field.name};

if (field.type.kind == "OBJECT") {

tempObj['type'] = "object";

tempObj['properties'] = this.\_getPropertiesBySourceType(field.type.name,tempMap);

}

propertyArray.push(tempObj);

})

return propertyArray;

}

\_getProperties(properties=[]){

let propertyArray = [];

forEach(properties, (property, index)=> {

let temProp = {name:property.name};

if(property.sourceType !="int" && property.sourceType !="date" && property.sourceType !="number" && property.sourceType !="string" && property.sourceType !="") {

temProp['type'] = 'object';

temProp['properties'] = this.\_getTypeDetails(this.state.pageTypes[property.sourceType], this.state.pageTypes);

}

propertyArray.push(temProp);

})

return propertyArray;

}

getListItems(properties, path) {

let items = [];

if (!properties) {

return items;

}

if (path) {

path = path + '.';

}

else {

path = '';

}

forEach(properties, (property, index)=> {

let key = path + property.name;

let nestedItems = undefined;

if (property.type === 'object' && property.properties) {

nestedItems = this.getListItems(property.properties, key);

}

items.push(

<ListItem

value={key}

key={key}

primaryText={property.name}

nestedItems={nestedItems}

style={{ webkitAppearance:"none"}}

/>

)

});

return items;

}

\_getTypeDetails(type,soureTypes) {

let properties = [];

forEach(type.arr, (type, index)=> {

let rawType = {name:type.name}

if(type.isComplex){

rawType['type'] = 'object';

rawType['properties'] = this.\_getTypeDetails(soureTypes[type.sourceType])

}

properties.push(rawType);

});

return properties;

}

render() {

const {currentComponent} = this.props;

let panelContent = null;

let style = {

width: '100%',

height: '100%',

paddingTop: '5px',

paddingRight: '5px',

paddingLeft: '5px',

paddingBottom: '10px',

};

if (currentComponent && currentComponent.component && currentComponent.component.propsSchema) {

if (this.state.pageTypes == null){

this.state.pageTypes = commondataApi.getTypeDefinitions();

}

const listProperties = this.getListItems(this.getListRawData());

const {key, sourceFilePath, componentName, props, component, text} = currentComponent;

let styleSchema = {};

let propsSchema = {};

let eventSchema = {};

forOwn(component.propsSchema, (value, name) => {

if (value.type === 'event') {

eventSchema[name] = value;

}

else if (value.type === 'style') {

styleSchema[name] = value;

}

else {

propsSchema[name] = value;

}

});

let values = currentComponent.props || {};

let events = {

onChange: this.handleChangeOption,

onAdd: this.handleAddNewProp,

onDelete: this.handleDeleteOption

}

let list = [];

for (var index in propsSchema) {

let obj = propsSchema[index];

let valueType="constant";

let value = values[index] || '';

if(isObject(value)){

valueType = value.\_\_DATASOURCE;

value =value.value;

}

list.push({name: index, type: obj.type, value:value , valueType:valueType});

}

let eventData = [];

for(var index in eventSchema){

let value = values[index] || {};

eventData.push({name:index,value});

}

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

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>

)

}

}

}

const overflowStyle = {

overflowY:'hidden',

}

class FlexibleEdit extends React.Component{

constructor(props){

super(props);

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})

handleRequestChange = (event, value) => {

this.props.events.onChange(this.props.name, {\_\_DATASOURCE:value.split(".")[0], value:value.substring(value.indexOf('.')+1)});

};

closePopover =() =>{

this.setState({showPopover:false})

}

getValueType =(valueType) => {

if(valueType=="local" || valueType=="param" || valueType=="remote"){

return "variable";

} else {

return valueType;

}

}

handleRequestClose = () => {

this.setState({

showPopover: false,

});

};

render(){

let events = this.props.events;

const type = this.props.type;

var renderField;

switch(type){

case 'date': renderField =

<DateField

events={events}

name={this.props.name}

type={this.props.type}

value={this.props.value}

/>

break;

case 'bool': renderField =

<CheckBoxField

events={events}

name={this.props.name}

type={this.props.type}

value={this.props.value}

/>

break;

case 'number': renderField =

<NumberField

events={events}

name={this.props.name}

type={this.props.type}

value={this.props.value}

/>

break;

case 'string':

default: renderField =

<EditText

events={events}

name={this.props.name}

type={this.props.type}

value={this.props.value}

/>

}

return (

<div style={{"marginLeft":"40px", "marginRight":"20px"}}>

<div style={{"marginBottom":"-10px"}} >

<div style={{"fontSize":"16px","color":"#999","float":"left", "marginTop":"20px"}}>

{this.props.name}

</div>

<div style={{"float":"right","fontSize":"12px","color":"#01a982", marginRight:"-29px"}} >

<DropDownMenu value={this.state.valueType} onChange={this.handleChange} openImmediately={false} style={{"padding-right":"-20px"}}

labelStyle={{"color":"#01a982","font-size":"6px"}} iconStyle={{"right":"25px"}} menuItemStyle={{"font-size":"6px"}} underlineStyle={{"display":"none"}} >

<MenuItem value={"constant"} primaryText="constant" style={{ webkitAppearance:"none"}}/>

<MenuItem value={"variable"} primaryText="variable" style={{ webkitAppearance:"none"}} />

<MenuItem value={"expression"} primaryText="expression" style={{ webkitAppearance:"none"}} />

</DropDownMenu>

</div>

</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>

)

}

}

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 (

<CodeMirror value={this.state.value} onChange={this.handleCodeChange.bind(this)} options={options} style={{height:"100px"}} className="CodeMirror\_Custom" />

)

}

}

class TriggerCell extends React.Component{

constructor(props){

super(props);

}

state ={

value:this.props.value

}

handleTextChange = (event) => {

var value=event.target.value;

this.setState({value});

this.props.events.onChange(this.props.name,value);

}

componentWillReceiveProps(nextProps){

this.setState({value:nextProps.value})

}

render(){

return (

<FormField style={field.outerStyle}>

<TextInput

style={field.commonStyle}

name={this.props.name}

value={this.state.value}

type="text"

onClick={this.props.selectValue}

onDOMChange={this.handleTextChange.bind(this)}

/>

</FormField>

)

}

}

class TimeField extends React.Component{

constructor(props){

super(props)

}

state ={

value:this.props.value

}

handleTime = (value) => {

this.setState({value})

this.props.events.onChange(this.props.name,value);

}

componentWillReceiveProps(nextProps){

this.setState({value:nextProps.value})

}

render(){

let events = this.props.events;

return (

<FormField style={field.outerStyle}>

<DateTime

style={field.commonStyle}

id="id"

name="name"

value={this.state.value}

onChange={this.handleTime}

format="H:mm:ss"

/>

</FormField>

)

}

}

class DateField extends React.Component{

constructor(props){

super(props)

}

state ={

value:this.props.value

}

handleDate = (value) => {

this.setState({value})

this.props.events.onChange(this.props.name,value);

}

componentWillReceiveProps(nextProps){

this.setState({value:nextProps.value})

}

render(){

return (

<FormField style={field.outerStyle}>

<DateTime

style={field.commonStyle}

id="id"

name="name"

value={this.state.value}

onChange={this.handleDate.bind(this)}

format="M/D/YYYY"

/>

</FormField>

)

}

}

class NumberField extends React.Component{

constructor(props){

super(props)

}

state={

value:this.props.value || 0

}

handleChange = (value) => {

this.setState({value})

if (this.props.handleChange)

this.props.handleChange(value);

this.props.events.onChange(this.props.name, value);

}

handleTextChange = (event) => {

let reg = new RegExp("^[0-9]\*$");

let value = event.target.value;

if(reg.test(value)) {

this.setState({value})

if (this.props.handleChange)

this.props.handleChange(value);

this.props.events.onChange(this.props.name, value);

}

}

handleAdd = () => {

let value = parseInt(this.state.value)+1;

this.setState({value});

if(this.props.handleChange)

this.props.handleChange(value);

this.props.events.onChange(this.props.name,value);

}

handleSub = () => {

let value = parseInt(this.state.value)-1;

this.setState({value});

if(this.props.handleChange)

this.props.handleChange(value);

this.props.events.onChange(this.props.name,value);

}

componentWillReceiveProps(nextProps){

this.setState({value:nextProps.value||0})

}

render(){

return (

<FormField style={field.outerStyle}>

<span style={field.commonStyle} className="grommetux-number-input">

<input

type="number"

className="grommetux-input grommetux-number-input\_\_input"

value={this.state.value}

onChange={this.handleTextChange.bind(this)}

/>

<Button className="grommetux-number-input\_\_add"

onClick={this.handleSub}

icon={<SubtractIcon />}>

</Button>

<Button className="grommetux-number-input\_\_subtract"

onClick={this.handleAdd}

icon={<AddIcon />}>

</Button>

</span>

</FormField>

)

}

}

class ColorFieldText extends React.Component{

constructor(props){

super(props)

}

state={

value:this.props.value,

}

handleTextChange = (event) => {

var value=event.target.value;

this.setState({value})

this.props.events.onChange(this.props.name,value);

}

componentWillReceiveProps(nextProps){

this.setState({value:nextProps.value})

}

render(){

return (

<FormField style={field.outerStyle}>

<TextInput

style={field.commonStyle}

//placeHolder={"Hint Text(# CCCCCC)"}

name={this.props.name}

value={this.state.value}

type={this.props.type}

onDOMChange={this.handleTextChange}

/>

{/\*colorIcon\*/}

</FormField>

)

}

}

class CheckBoxField extends React.Component{

constructor(props){

super(props)

}

state = {

value:this.props.value

}

handleCheckBoxChange = (event) => {

var value=event.target.checked;

this.setState({value})

this.props.events.onChange(this.props.name,value);

}

componentWillReceiveProps(nextProps){

this.setState({value:nextProps.value})

}

render(){

return (

<div style={field.outerStyle}>

<CheckBox

style={field.commonStyle}

toggle={true}

defaultChecked={this.state.value}

checked={this.state.value}

onChange={this.handleCheckBoxChange}

/>

</div>

)

}

}

//needs triangle with right and add icon

class SelectField extends React.Component{

constructor(props){

super(props)

}

state = {

value:this.props.value

}

handleTextChange = (event) => {

var value=event.target.value;

this.setState({value})

this.props.events.onChange(this.props.name,value);

}

componentWillReceiveProps(nextProps){

this.setState({value:nextProps.value})

}

log = () =>{

console.log("svg clicked")

}

render(){

return (

<FormField style={field.outerStyle}>

<span style={field.commonStyle} className="grommetux-number-input">

<input

type="number"

className="grommetux-input grommetux-number-input\_\_input"/>

<Button className="grommetux-number-input\_\_add"

icon={<DownIcon />}>

</Button>

<Button className="grommetux-number-input\_\_subtract"

icon={<AddIcon />}>

</Button>

</span>

</FormField>

)

}

}

class EditText extends React.Component{

constructor(props){

super(props)

}

state={

value:this.props.value,

}

handleTextChange = (event) => {

var value=event.target.value;

this.setState({value})

this.props.events.onChange(this.props.name,value);

}

componentWillReceiveProps(nextProps){

this.setState({value:nextProps.value})

}

render(){

return (

<FormField style={field.outerStyle}>

<TextInput

style={field.commonStyle}

//placeHolder={"Hint Text(# CCCCCC)"}

name={this.props.name}

value={this.state.value}

type={this.props.type}

onDOMChange={this.handleTextChange.bind(this)}

/>

</FormField>

)

}

}

class OwnField extends React.Component{

constructor(props){

super(props)

}

state={

value:this.props.value

}

handleAdd = () => {

this.setState({value:this.state.value+1});

if(this.props.handleChange)

this.props.handleChange(value);

}

handleSub = () => {

this.setState({value:this.state.value-1});

if(this.props.handleChange)

this.props.handleChange(value);

}

render(){

return (

<FormField style={field.outerStyle}>

<span style={field.commonStyle} className="grommetux-number-input">

<input

type="number"

className="grommetux-input grommetux-number-input\_\_input"

value={this.state.value}

/>

<Button className="grommetux-number-input\_\_add"

onClick={this.handleAdd}

icon={<SubtractIcon />}>

</Button>

<Button className="grommetux-number-input\_\_subtract"

onClick={this.handleSub}

icon={<AddIcon />}>

</Button>

</span>

</FormField>

)

}

}

const field={

commonStyle:{

backgroundColor:'#EBEBEB',

borderColor:'#EBEBEB',

padding:'none',

},

outerStyle:{

border:'none',

paddingTop:"0px",

}

}

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

const labelStyle = {

"background-color":"white",

}

const inkBarStyle= {

"background":"#01a982",

};

const secondTabStyle = {

"color":"#666",

"fontFamily":"Metric",

"fontSize":"16px",

"textTransform":"none",

}

4.3.4 路由配置模块设计

路由配置。路由，也就是HTML页面或组件的访问路径，路由与组件名称绑定，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：

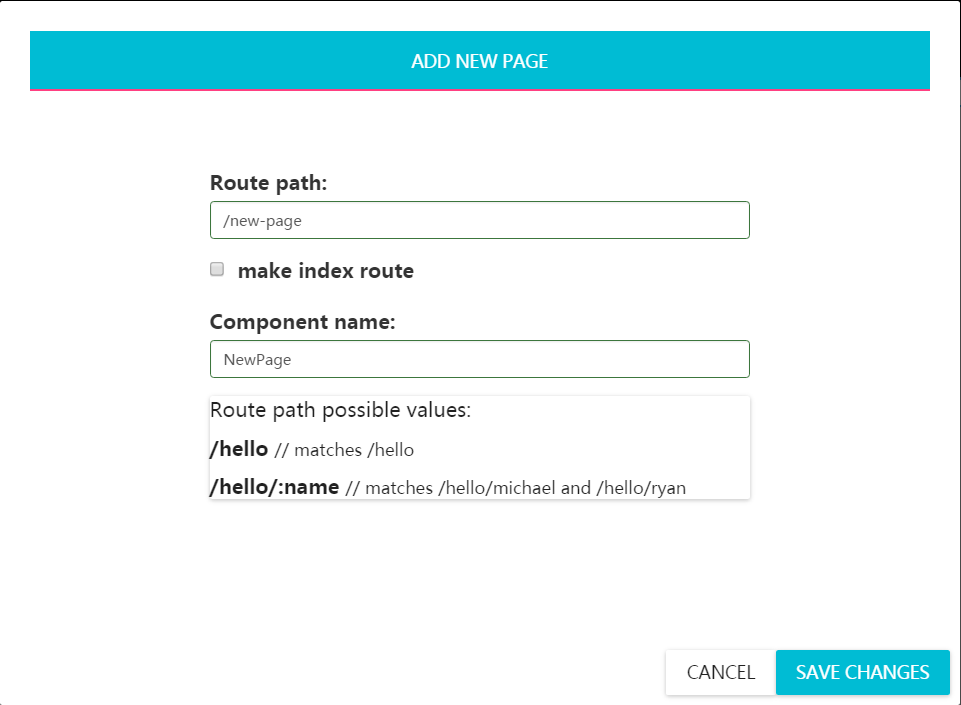


图4.6 路由配置

注：路由配置的逻辑并不复杂，在此将不再贴出代码。

4.3.5 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。

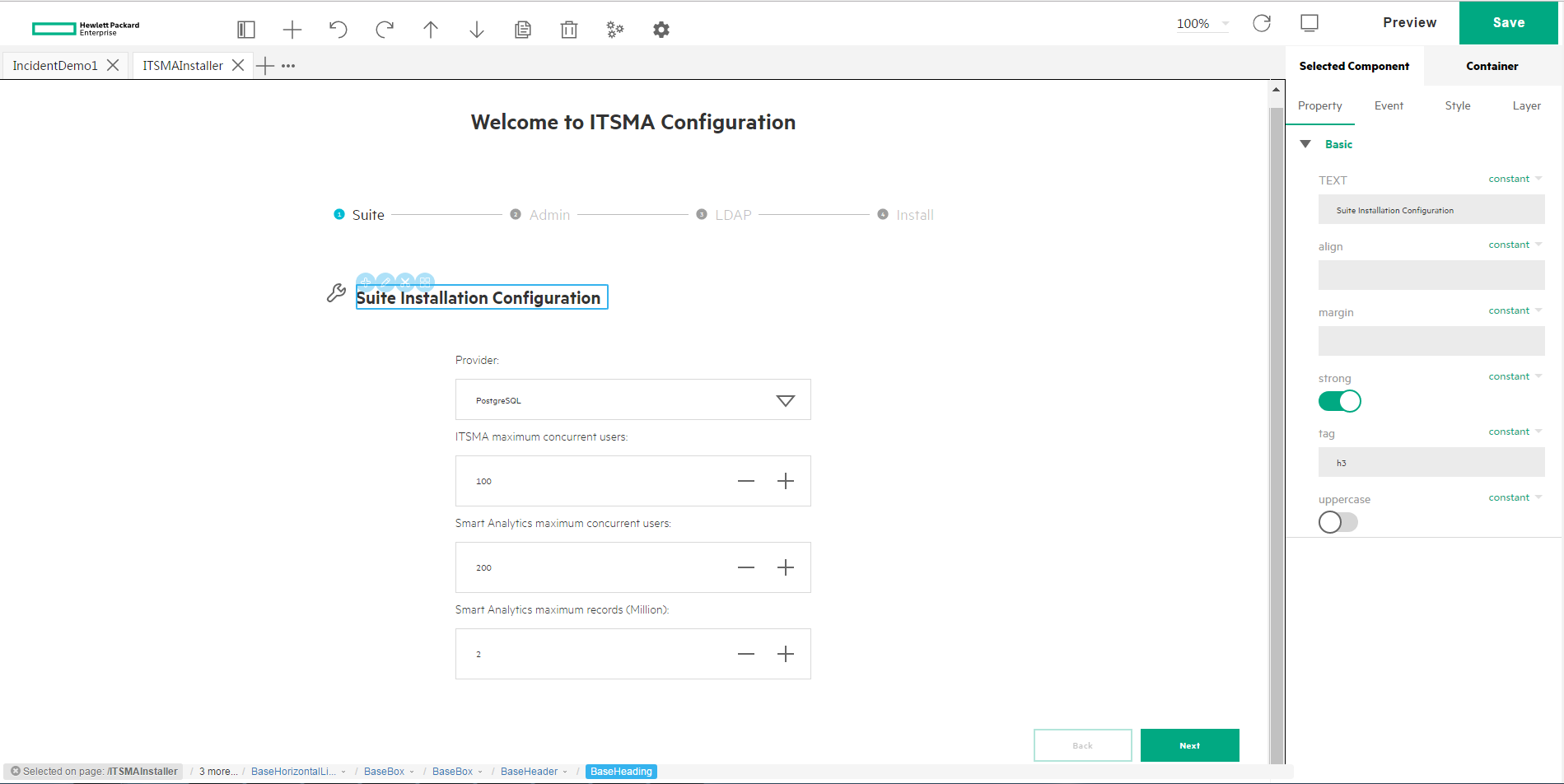


图4.7 页面设计模块

代码如下：

import { forOwn } from 'lodash';

import React, { Component, PropTypes } from 'react';

import ReactDOM from 'react-dom';

import { connect } from 'react-redux';

import { modelSelector } from './selectors.js';

import { containerActions } from './actions.js';

import { utilsStore, graphApi, previewGraphApi } from '../../../api/index.js';

import { CLIPBOARD\_CUT } from '../ClipboardControls/actions.js';

let lastWaitTimer = undefined;

const wait = (testFunc, launchFunc) => {

if(lastWaitTimer){

clearTimeout(lastWaitTimer);

lastWaitTimer = undefined;

}

if(!testFunc()){

lastWaitTimer = setTimeout(() => { wait(testFunc, launchFunc); }, 3000);

} else {

launchFunc();

}

};

class DeskPage extends Component {

constructor(props) {

super(props);

this.handleComponentClick = this.handleComponentClick.bind(this);

this.handlePathnameChanged = this.handlePathnameChanged.bind(this);

this.handleKeyDown = this.handleKeyDown.bind(this);

this.setupShortcuts = this.setupShortcuts.bind(this);

this.removeShortcuts = this.removeShortcuts.bind(this);

}

componentDidMount(){

const domNode = ReactDOM.findDOMNode(this);

const { componentModel:{pages, currentPagePath} } = this.props;

if(currentPagePath){

domNode.src = '/app' + currentPagePath;

}

const { loadPage, pageLoaded } = this.props;

const { setForCuttingKeys, setForCopyingKeys } = this.props;

const { pasteBefore, pasteAfter, pasteFirst, pasteLast, pasteReplace } = this.props;

const { cloneSelected, deleteSelected } = this.props;

const { setDefaultVariant, hidePreviewComponent, selectVariant } = this.props;

const { quickBefore, quickAfter, quickFirst, quickLast, quickReplace } = this.props;

const { loadOptions } = this.props;

loadPage();

this.contentDocument = domNode.contentDocument;

this.contentWindow = domNode.contentWindow;

this.setupShortcuts();

domNode.onload = ( () => {

this.contentWindow.\_\_pages = pages;

this.contentWindow.onPageDidMount = (page, pathname) => {

this.page = page;

page.bindOnComponentMouseDown(this.handleComponentClick);

page.bindOnPathnameChanged(this.handlePathnameChanged);

page.bindGetPagePath(pathname => graphApi.getPagePath(pathname));

page.bindGetPageModel(pathname => graphApi.getWrappedModelByPagePath(pathname));

page.bindGetComponents(() => graphApi.getComponents());

page.bindGetMarked(pathname => graphApi.getMarkedKeysByPagePath(pathname));

page.bindGetMode(() => {return this.props.componentModel.isEditModeOn;});

page.bindGetComponentInPreview(() => {

const { libraryPanelModel: {componentInPreview, variantsInPreview, defaultVariantMap}} = this.props;

if(componentInPreview){

const defaultVariantKey = defaultVariantMap[componentInPreview].key;

const previewModel = previewGraphApi.getWrappedModelForVariant(defaultVariantKey);

return {componentInPreview, variantsInPreview, previewModel, defaultVariantKey};

} else {

return undefined;

}

});

page.bindToState('onLoadOptions', (key, isModifier) => {

const { currentComponent } = this.props;

loadOptions(currentComponent);

});

page.bindToState('onCut', (key, isModifier) => { setForCuttingKeys([key]); });

page.bindToState('onCopy', (key, isModifier) => { setForCopyingKeys([key]); });

page.bindToState('onClone', (key, isModifier) => { cloneSelected(); });

page.bindToState('onDelete', (key, isModifier) => { deleteSelected(); });

page.bindToState('onBefore', (key, isModifier) => { pasteBefore(key); });

page.bindToState('onAfter', (key, isModifier) => { pasteAfter(key); });

page.bindToState('onFirst', (key, isModifier) => { pasteFirst(key); });

page.bindToState('onLast', (key, isModifier) => { pasteLast(key); });

page.bindToState('onReplace', (key, isModifier) => { pasteReplace(key); });

page.bindToState('isMultipleSelection', () => {

const { selectionBreadcrumbsModel: {selectedKeys} } = this.props;

return selectedKeys && selectedKeys.length > 1;

});

page.bindToState('isAvailableToPaste', key => {

const { clipboardIndicatorModel: {clipboardMode} } = this.props;

return clipboardMode !== CLIPBOARD\_CUT || graphApi.isCutPasteAvailable(key);

});

page.bindToState('isClipboardEmpty', () => {

const { clipboardIndicatorModel: {clipboardKeys} } = this.props;

return !clipboardKeys || clipboardKeys.length <= 0;

});

page.bindToState('setDefaultVariant', (componentName, key) => {

setDefaultVariant(componentName, key);

});

page.bindToState('selectVariant', (key) => { selectVariant(key); });

page.bindToState('hidePreview', () => { hidePreviewComponent() });

page.bindToState('getComponentsList', () => {

const { libraryPanelModel: {componentsList}} = this.props;

return componentsList;

});

page.bindToState('quickBefore', (componentName, selectedKey) => {

quickBefore(componentName, selectedKey);

});

page.bindToState('quickAfter', (componentName, selectedKey) => {

quickAfter(componentName, selectedKey);

});

page.bindToState('quickFirst', (componentName, selectedKey) => {

quickFirst(componentName, selectedKey);

});

page.bindToState('quickLast', (componentName, selectedKey) => {

quickLast(componentName, selectedKey);

});

page.bindToState('quickReplace', (componentName, selectedKey) => {

quickReplace(componentName, selectedKey);

});

};

const initPage = () => {

this.contentWindow.\_\_createPageDesk();

wait(() => this.contentWindow.pageReadyState === 'initialized', pageLoaded);

//pageLoaded();

};

wait(() => this.contentWindow.pageReadyState === 'ready', initPage);

});

}

componentWillUnmount(){

this.removeShortcuts();

this.contentDocument = undefined;

this.contentWindow = undefined;

this.page = undefined;

}

componentWillReceiveProps(nextProps){

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);

}

}

}

componentWillUpdate(nextProps, nextState){

this.removeShortcuts();

this.doUpdatePageModel = false;

this.doUpdateMarks = false;

const { componentModel } = this.props;

const { componentModel: newComponentModel } = nextProps;

if(newComponentModel.modelUpdateCounter !== componentModel.modelUpdateCounter) {

this.doUpdatePageModel = true;

} else if(newComponentModel.markedUpdateCounter !== componentModel.markedUpdateCounter) {

this.doUpdateMarks = true;

}

}

shouldComponentUpdate(nextProps, nextState){

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(){

this.setupShortcuts();

if(this.page){

if(this.doUpdatePageModel){

const { componentModel } = this.props;

//console.log('Updating page model: ' + componentModel.currentPagePath);

this.page.updatePageModel({

pathname: componentModel.currentPagePath

});

}

if(this.doUpdateMarks){

this.page.updateMarks();

}

}

}

handleComponentClick(key, isModifier){

const { setSelectedKey } = this.props;

setSelectedKey(key, isModifier);

}

handlePathnameChanged(pathname){

const { changePageRouteFeedback } = this.props;

changePageRouteFeedback(pathname);

}

handleKeyDown(e){

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;

const { clipboardIndicatorModel:{clipboardKeys} } = this.props;

const { setForCuttingKeys, setForCopyingKeys } = this.props;

const { pasteBefore, pasteAfter, pasteFirst } = this.props;

const { cloneSelected, popHistory } = this.props;

switch (e.which) {

case 68: // D key

e.stopPropagation();

e.preventDefault();

cloneSelected();

break;

case 67: // C key

e.stopPropagation();

e.preventDefault();

if(selectedKeys && selectedKeys.length > 0){

setForCopyingKeys(selectedKeys);

}

break;

case 65: // A key

e.stopPropagation();

e.preventDefault();

if(selectedKeys && selectedKeys.length === 1 && clipboardKeys && clipboardKeys.length > 0){

pasteBefore(selectedKeys[0]);

}

break;

case 73: // I key

e.stopPropagation();

e.preventDefault();

if(selectedKeys && selectedKeys.length === 1 && clipboardKeys && clipboardKeys.length > 0){

pasteFirst(selectedKeys[0]);

}

break;

case 86: // V key

e.stopPropagation();

e.preventDefault();

if(selectedKeys && selectedKeys.length === 1 && clipboardKeys && clipboardKeys.length > 0){

pasteAfter(selectedKeys[0]);

}

break;

case 88: // X key

e.stopPropagation();

e.preventDefault();

if(selectedKeys && selectedKeys.length > 0){

setForCuttingKeys(selectedKeys);

}

break;

case 90: // Z key

popHistory();

e.stopPropagation();

e.preventDefault();

break;

default:

break;

}

}

}

}

setupShortcuts(){

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(){

if(this.contentWindow) {

this.contentWindow.removeEventListener('keydown', this.handleKeyDown, false);

}

window.removeEventListener('keydown', this.handleKeyDown, false);

}

render(){

return (<iframe {...this.props} src="/app/deskpage" />);

}

}

export default connect(modelSelector, containerActions)(DeskPage);

4.3.6 页面预览模块设计

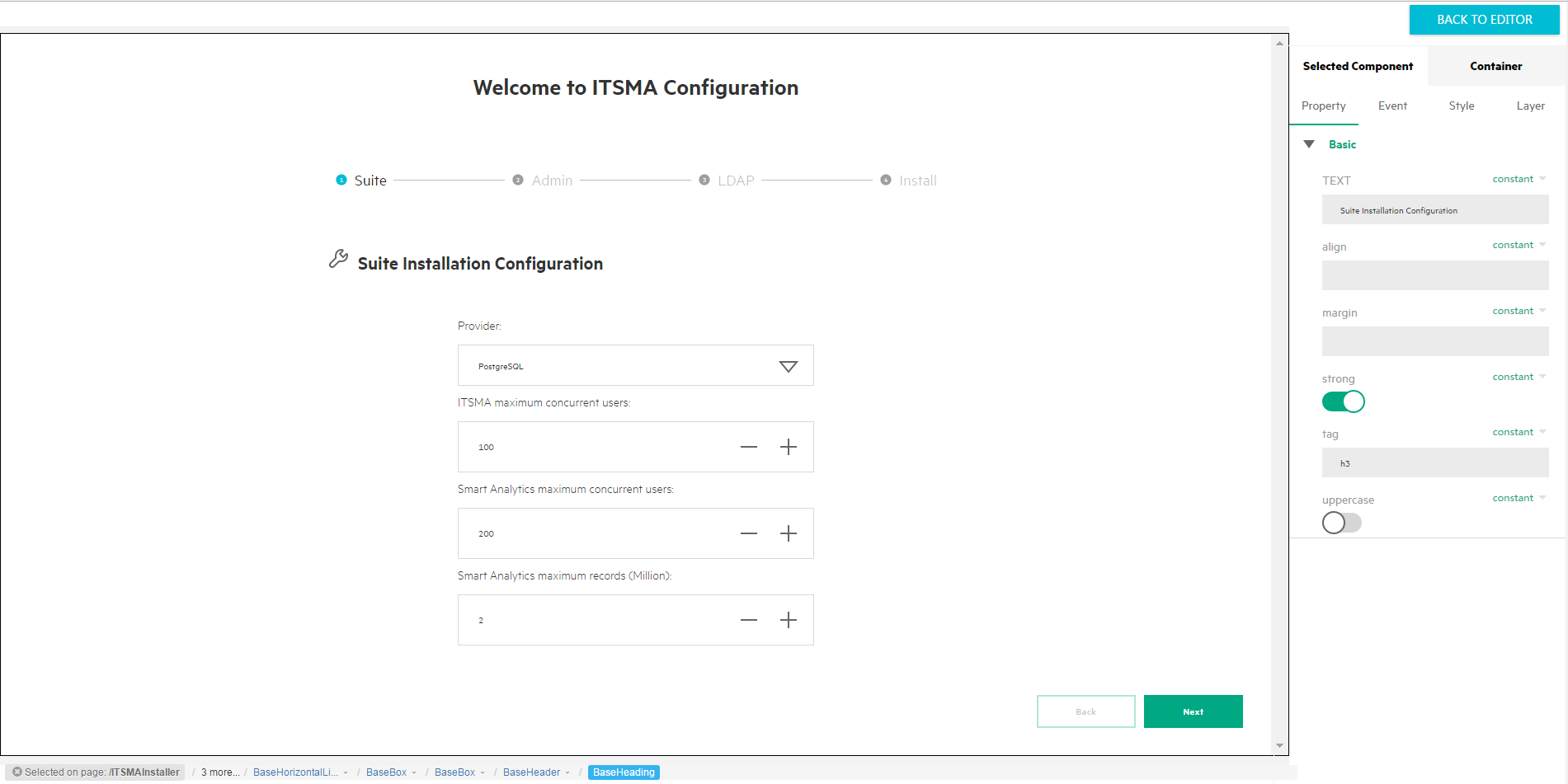


图4.8 页面预览

注：代码和上一模块极为类似，在此不再贴出。

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。在对在线的可视化网站设计的研究和实际开发过程中，通过查阅大量的中英文资料、阅读各种教材、和老师同学进行探讨，对可视化网站设计有了比较深入的了解和认识。这一课题的研究让我收获很大。由于时间及本人水平所限，所做的工作不可避免的存在有很多疏漏和不足。最后总结全文，前段时期的工作取得的成果主要有以下几点：

1. 了解并掌握了当前国内外在线的可视化网站设计的应用情况和发展趋势及系统构建过程。增强了提出问题、分析问题和解决问题的科研能力。

② 获得了有关Web服务、非关系型数据库和编程方面的知识。

③ 采用JavaScript全栈技术，完成了可视化网站设计系统的分析和整体设计，各个功能模块的划分，数据库的设计并最终实现了系统。

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：

① 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。

② 本系统具有组件自定义功能，通过组合已有组件、设置不同参数来构建新的组件。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。

③ 本系统完全基于浏览器的Web方式为设计者提供了方便，客户端只需安装大众的Google Chrome浏览器即可。

④ 本系统维护在服务器端执行，实现客户端零维护，并同时实现了多人协作构建，避免了C/S模式麻烦的系统维护过程。

⑤ 本系统引入了node.js、react.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教授，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。