对myblog-backbone的开发总结;

前端模块;

view/myblogIni.html;

定义了页面HTML源文件的架构, 主要包括 , 和从外部加载require.js脚本这三个部分;

public/main.js;

require.js的入口文件, 在加载完requirejs的主代码后会立即加载main.js;
其中定义了对require的config(定义模块加载的默认路径等);
加载了backbone和semantic这两个重要模块后, 设置了三个全局常量(title,description,tpl);
加载了router/router.js, 初始化了路由对象并且对路由路径开始监听;

public/router/router.js;

定义了路由规则:

```
''':'init',
'posts':'posts',
'posts?author=:authorid':'posts',
'posts?post=:postId':'posts',
'posts/create':'posts_create',
'posts/:postId/edit':'editPost',
'posts/:postId/remove':'removePost',
'comments/:commentId/remove':'removeComment',
'signin':'signin',
'signout':'signout',
'signup':'signup',
'*other':'default'
```

在路由规则对应的回调函数中加载了待加载视图的template模块(header部分和内容部分), 定义了如:

```
require(['template/component/header','template/posts','view/blogView'],function(header,posts,blogView){
    var tplHeader  = header.html;
```

```
            var tplPosts  =  posts.html;
            tpl = tplHeader + tplPosts;
            window.App = new blogView(authorid);
        });
```

取出了template模块中的html属性中定义的字符串, 拼接header和内容部分的html字符串将其赋值给全局变量tpl, 然后初始化待加载的视图对象,同时传入一些关键参数给视图模块的initialize函数;

public/view/404View.js
public/view/blogView.js
public/view/createPostView.js
public/view/editPostView.js
public/view/postView.js
public/view/signinViewjs
public/view/signupView.js

public/view中保存的是各个视图模块, 以blogView.js为例:

这个模块会返回一个Backbone的View模块对象, 它的initialize方法中将使用_.template()函数将全局函数tpl转换为一个根据传入tpl定义的模板方法, 然后调用render方法;
render方法中首先会将当前视图模块对象依赖的model模块或者collection模块使用fetch方法获取保存在后端数据库中的数据, 将异步获取到的这些数据整合后传递给刚才使用_.template()函数生成的模板方法, 将整合后的数据传入这个方法,方法将返回一个根据传入数据渲染后的html模板内容, 将这段内容插入页码中的element元素, 页面渲染完成,如:

```
var content = App.template(finalData);
$('#element').empty().append(content);
```

接着执行afterRenderProcess模板返回的函数;

public/afterRenderProcess.js

afterRenderProcess模块返回一个函数, 这个函数将首先利用e.preventDefault();禁止所有<a>链接的默认行为(单页面项目不能通过a链接跳转并刷新页面), 根据a链接的href属性获取到其目标url后使用Backbone提供的router.navigate(href, true); 来让跳转的地址通过Backbone单页面路由系统来处理; 并且指定了a链接如果指向一个删除操作(删除评论或者删除文章), 那么将会弹出询问框询问用户是否继续操作, 用户确认继续才会执行删除;
接着, 指定了表单中submit按钮的行为: 取消默认跳转行为, 用户点击后前端来验证表单中各个项的有效性, 如果全部符合发送请求的格式要求则通过Ajax发送表单请求, 一旦

Ajax请求响应成功则使用router.navigate()跳转到某个指定的url;
需要注意的是, 这里表单submit按钮对应的提交请求其实可以在此处拦截(就是取消默认行为)然后重新发送指定url的请求通过Backbone路由系统(router.js)再来发送Ajax请求更好, 因为这样的设计才是高内聚的;
最后设置延时清除掉成功、失败提示信息的代码和semantic框架需要对指定元素行为设置的js代码;

public/css/style.css
public/img/.....

项目的css样式文件和保存的用户注册头像图片;

public/deps/backbone-min.js
public/deps/jquery.min.js
public/deps/r.js
public/deps/semantic.min.js
public/deps/underscore-min.js

项目的依赖库;

public/model/blogComment.js
public/model/blogPost.js
public/model/blogUser.js

model模块中主要指定了获取指定model内容的url, 如:

```
return Backbone.Model.extend({
            url:'/getPost',
            initialize:function(postId){
      if(postId){
          this.url = this.url + '?' + 'postId=' + postId;
      }
     }
     // defaults:{
     //   author:'',
     //   title:'',
     //   content:'',
     //   pv:0
     // }
 });
```

public/collection/blogCollection.js
public/collection/commentCollection.js

collection模块中主要指定了获取指定collection内容的url, 如:

```
return Backbone.Collection.extend({
        url:'/getPosts',
        model:blogPost,
        initialize:function(authorid){
          if(authorid){
              this.url = this.url + '?' + authorid;
          }
        }
  });
```

public/template/404.js
public/template/createPost.js
public/template/editPost.js
public/template/post.js
public/template/posts.js
public/template/signin.js
public/template/signup.js
public/template/component/header.js

主要指定了需要用到的template模板的内容, 使用对象的形式保存(模仿app-content-content,使用khaki来实现), 如:

```
define([],function(){
      return {"html":"<div class='errorPage'>This is a 404 myblog page!</div>"}

})
```

后端模块;

index.js;

初始化了node express框架;
指定了需要的所有中间件, 数据库的设置;

router/index.js;

后端路由文件, 主要用来接收和处理Ajax请求, 如果请求不符合任何Ajax请求的路径并且未设置并发送任何响应header就默认是获取项目的主页面'/posts', 返回主页面.html文件给浏览器:

```
app.use(function (req, res, next) {
      if (!res.headersSent) {
          var clientui = require('fs').readFileSync('views/myblogIni.html');
          res.writeHead(200,{'Content-Type':'text/html;charset=utf-8;'}); //注意, 如果这里不添加charset=utf-8响应, 页面会显示中文乱码;
          res.write(clientui);
          res.end();
          return next();
      }
  })
```

lib/mongo.js

MongoDB的初始化设置, 并且初始化了User, Post, Comment 三张主表;

models/comments.js
models/posts.js
models/users.js

定义了三张主表的对应查询方法, 如:

```
const User = require('../lib/mongo').User

module.exports = {
  // 注册一个用户
  create: function create (user) {
    return User.create(user).exec()
  },

  // 通过用户名获取用户信息
  getUserByName: function getUserByName (name) {
    return User
      .findOne({ name: name })
      .addCreatedAt()
      .exec()
  }
}
```

TODO:
1.在myblog-backbone中直接加载页面: http://localhost:3000/posts/create 会报错:Unexpected token < 并且无法加载main.js,
这可能是由于requirejs加载main.js的地址为: "http://localhost:3000/posts/main.js" 从而找不到这个文件而导致的, 原因是当前页面的地址在/posts后又添加了子路径/create, 如果没有子路径或者只使用'?...'query字符串的形式就不会有这个问题;
参考: https://stackoverflow.com/questions/19682610/requirejs-does-not-follow-relative-path-for-data-main-with-baseurl-set
2.在点击链接时某些时刻可能会出现: **Error: Can't set headers after they are sent** 这样的后端报错, 这可能与node的版本有关, 也可能是因为在调用next()方法前没有使用return;
参考: https://cnodejs.org/topic/53774ffecbcc396349ca1155