# 门户改造前端方案对比

## DOM结构

页面结构为上中下模式

**顶部区域**



**中部区域**



**底部区域**



## 实现方式

* + 采用了前端多页面的形式, 每次进行页面跳转都会把整个页面重新加载.
  + 前端框架采用jQuery
  + 页面之间数据共享采用cookie
  + 页面跳转采用的纯连接跳转(即页面重刷)
  + 页面加载一次性加载整个页面
  + 各个模块之间联系紧密.

* 1. 现有存在的问题

采用的多页面形式实现, 每次页面跳转会存在一段时间的空白加载时间.

数据绑定才用的页面模板或直接操作dom的形式.

页面公共部分没有做缓存(头部和底部在页面跳转的时候是重新加载)

页面之间公共的UI部分没有抽离, 导致样式的全局修改不好实现

页面没有做首屏优化, 导致第一次访问页面图片不能立刻加载.

页面js, css, html没有做压缩处理, 不利于安全性保障.

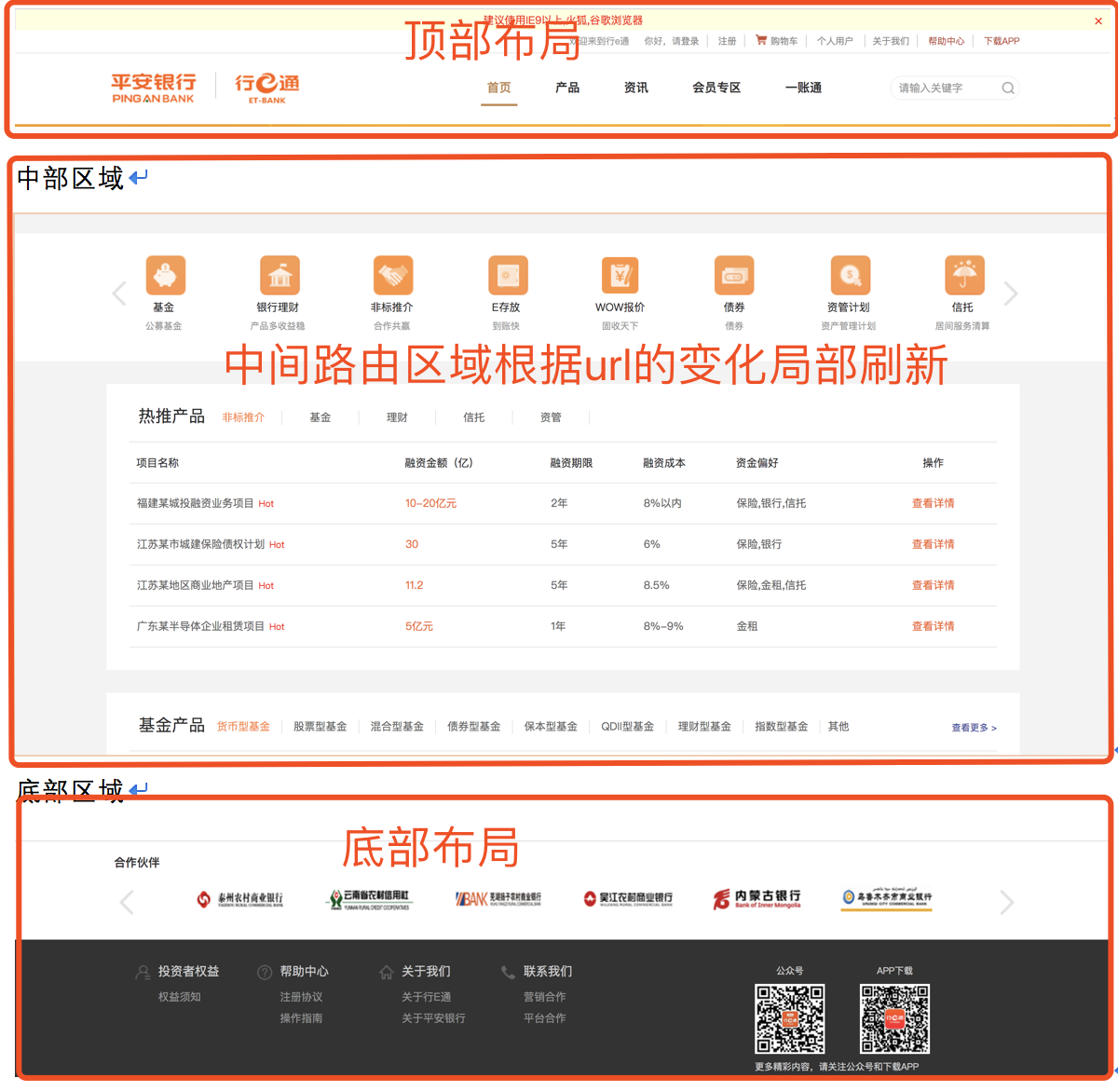


页面的样式用的是css文件, 可能会导致后期的全局样式修改难以实现.

* 1. 代码结构

# React解决方案

## DOM结构采用上中下的结构



## 技术选型

底层采用React ReactDOM 采用React的最新技术方案hooks api 编写页面代码.

路由自己封装, 采用hash路由模式实现路由跳转

UI样式依赖antd框架, 同时在antd框架上面封装所有的公共部分.

JavaScript语法采用ES6的语法规则.

Css样式预处理采用less

打包技术采用webpack

## 特点

React框架的可操作性和可扩展性比较强. 项目后期维护成本低.

React速度很快：它并不直接对DOM进行操作，引入了一个叫做虚拟DOM的概念，安插在javascript逻辑和实际的DOM之间，性能好。最大限度减少DOM交互。

对浏览器的兼容性好可兼容到IE8

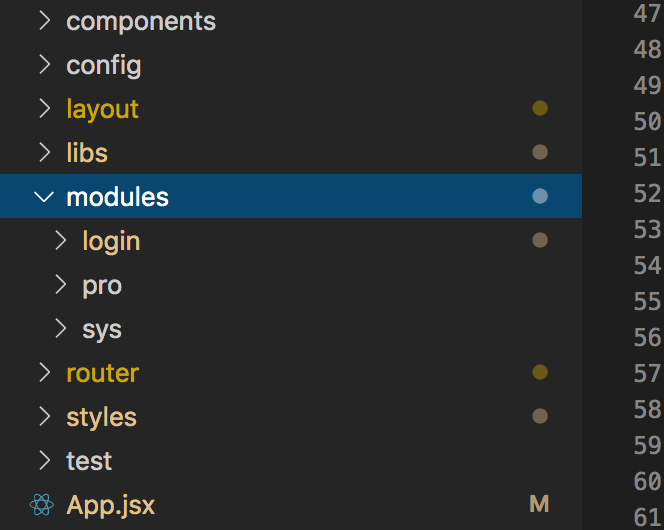
每个页面独立运行, 不依赖其他页面, 打包也会处于不用的文件夹里面.

页面切换仅仅进行局部刷新. 不会进行整个页面的刷新, 页面跳转会有缓存(可在页面单独配置是否使用缓存.), 下次进入该页面的时候依然是原来的页面. 用户的操作依然保留.

采用最新的hooks模式编写代码, 使逻辑更加细粒度, 同样的逻辑不用多个地方编写, 一个逻辑处理都可以写在一块. 方便代码维护.

采用less进行样式预处理. 可以将主题色, 元素大小等公共样式提取出来, 保存在本地. 以便用户自定义设置.

## 文件结构



components文件夹存放各类文件组件文件.

Config文件夹存放各类配置文件

Layout文件夹是整个页面的布局代码.

Modules文件管理各个分离处理的模块(各个模块采用文件夹方式管理. 如现在有三个模块login, pro, sys ) , 每个模块管理自己独立的代码块, 比如: 路由, 样式, 页面, 请求对象等代码. 只有当模块真正被加载了, 这些代码才会自动加载内存中.

Router模块管理整个路由的实现, 以及动态路由对象和静态路由对象列表的封装

Styles文件夹管理项目需要那个的公共样式

Test项目后期引入前端自动化测试.

## 缺点

react的学习成本高, 上手比较困难.

# Vue的解决方案.

## 项目结构

同recact

## 技术选型

底层采用vue框架

路由采用vue-router

数据交互采用vuex

样式预处理采用less

打包技术采用webpack

UI自己封装.

JavaScript语法采用ES6的语法规则.

## 特点

项目搭建快速( 有管理台为模板, 根据现有代码修改升级)

每个页面独立运行, 不依赖其他页面, 打包也会处于不用的文件夹里面.

页面切换仅仅进行局部刷新. 不会进行整个页面的刷新, 页面跳转会有缓存(可在页面单独配置是否使用缓存.), 下次进入该页面的时候依然是原来的页面. 用户的操作依然保留.

页面间切换可加入动画, 提高用户体验

所有模块和页面可配置化, 通过配置文件可以控制那些对那些页面/模块进行打包, 那些页面/模块不打包.

## 缺点

vue比较大, 占用内存会比react多