**理财服务平台前端渐进改造实践**

# 背景

近年来前端开发领域的技术演进很快，比起jQuery时代的传统技术，新兴前端技术带来更好的用户体验、更高的开发效率，以及更系统的工程化能力，公司内部也已经在多个新项目中使用react框架，但是与此同时，也有不少老系统出于提升用户体验、提高开发效率的目的，希望在老系统中使用react框架，这就为产品研发带来一个新挑战：如何在新旧框架替换过程中，保证业务平稳运行。

本文将以在理财服务平台旧系统中应用react框架为例，着重介绍其间前端所遇到的挑战和解决方案。

# 技术选型

## 是否需要前后端分离

前后端分离在过去几年里越来越普及，企业里前后端分离的项目逐步增多，但是如何判断我们是否需要这样的架构？主要有以下几个角度：

1. 页面交互是否复杂？复杂页面意味着传统后端人员无法兼职前端界面工作，需要更具体的前后端分工，前后端分离可解耦前后端协作。
2. 是否需要搜索引擎优化？如果需要的话，那么从一开始我们就需要考虑后端渲染。
3. 是否多终端应用？以后是否可能会扩展到移动端？前后端分离架构使得后端可以通过一套服务加上适配层来适配多终端接口。
4. 前端的修改是不是非常频繁？如果不需要经常修改的话，那么这种优化便没有优势。

结合理财服务平台的实际业务特点：交互复杂，无需搜索引擎优化，多终端界面等特点，最终我们选择前后端分离的技术架构。

## 技术栈选型

公司内部多数系统均为中后台管理系统，本质上为满足用户工作需要,包括理财服务平台，其业务特点可总结为：

1. 面向内部人员，最好有类似桌面端应用的用户体验，提高工作效率。
2. 用户体验强调高效性和一致性。
3. 公司内部系统之间相互协作，业务管理程度高，强度组件的复用性。
4. 不追求炫酷效果，无需考虑搜索引擎优化等。

针对以上业务特点，整体上我们选择**SPA单页应用** + **组件化**作为大的方向，具体来说, UI层我们选择以一切皆组件为核心思想的React作为核心技术，以ES6作为开发语言，利用Webpack和Babel进行编译打包,除此之外，我们在路由管理、状态管理、开发服务器、构建部署等方面，根据业务实际情况，结合社区最佳实践，产生了一系列的解决方案，更详细的前端框架选型如下：

|  |  |
| --- | --- |
| 领域 | 技术方案 |
| 开发语言 | ES6+ |
| UI层 | React |
| 状态管理 | Redux |
| 组件 | 以Ant Design为主 |
| 框架 | Dva |
| 路由 | react-router |
| 编译打包 | Babel + Webpack |
| 代码检查 | Eslint + StyleLint |

关于框架选型，还应注意以下几点：

1. 应了解社区各主流框架的特点、适用场景，结合自身业务特点做出选择，比如Angular作为前端框架，提供包括路由、状态管理、构建部署整套方案，而React、Vue只是UI层解决方案，路由、构建部署等还需要根据需求来整合，各自适用不同的场景。

2. 应考虑开发团队的实际情况，比如我们公司主要以开发商为主，在推行UI单元测试，使用TypeScript强类型语言时，要考虑开发商的适应能力，谨慎使用。

3. 上述技术选型涉及UI框架、开发语言、状态管理、构建部署等各方面，配置繁琐，因此我们整理了一个可快速上手开发的种子站点，公司内部gitlab地址：http://gitlab.htzq.htsc.com.cn/components/react-seed，此种子站点目前已应用到公司20多个项目中，除整合上述技术栈以外，也针对华泰场景做了一些优化：

1. 内置华泰中后台典型场景：侧栏 + 标签窗口工作台
2. 内置基于神策数据分析系统的日志收集中间件
3. 门户统一登录超时处理
4. 云桌面兼容性视图设置处理
5. 搭建华泰私有NPM源
6. 基于容器云和jenkins进行持续构建部署的最佳实践

欢迎Fork和Merge Request。

# 渐进改造

## 主要挑战

虽然我们在信息技术部的多个全新业务系统中已经开始大规模应用react相关技术栈，但是在旧有系统上应用react技术栈，仍然存在诸多挑战：

1. 现存系统应用体积大，复杂度高，我们很难在一个较短的时间内彻底完成前后端分离改造，同时，业务部门不断提出新的业务需求，技术演进和业务连续性必须同时兼顾，即无法推翻旧系统进行彻底重构。
2. 新旧系统要做到无缝融合，即从用户角度，不应感知到系统是“拼凑”起来的，除了风格一致外，新旧系统在交互上也应该做到完全融合，比如：点击旧页面上的某个按钮，可以打开新框架实现的dialog，而不是简单的使用Iframe将新页面嵌套了事。
3. H5开发人员不熟悉原后端环境部署，为减少开发过程中H5开发人员对环境依赖，应彻底隔离新旧系统，即H5人员采用新框架开发新页面时，无需感知旧系统环境。
4. 中长期来看，我们最终会彻底淘汰旧有系统，全部采用新的技术框架，因此，我们需要规划一整套灰度方案，通过渐进方式“和平演变”原有系统，在保障业务连续的同时，彻底完成技术改造。

下面我们主要就新旧系统整合、灰度替换两个方面阐述为解决上述挑战我们采取的解决方案：

## 新旧系统整合

如何在旧的单体应用中嵌入新框架？最直观的想法就是在旧系统中通过iframe方式嵌入新页面，这种方式简单直接，但是也存在比较大的问题，比如新页面中的全局对话框、Toast提示等，放在iframe中无法做到正常的居中、顶部显示，这个还只是展示的问题，更重要的是，iframe方式导致新旧系统无法直接交互，比如点击旧系统某个按钮，弹出新系统的对话框，虽然可以在新旧系统中相互做接口进行通信完成此类需求，但是这种做法导致新旧系统无法真正解耦，不要忘了我们最终目标是要完全替换旧系统，这种做法在不久的将来又将是一笔技术债。

iframe方式存在局限性，有没有其他方案可以实现真正的新旧系统融合？答案是肯定的。SPA应用本质上是在页面上提供一块区域作为容器，容器内具体展示什么内容，由路由决定，因此，我们可以在旧系统中开辟出一块容器，当展现旧系统内容时容器隐藏，当展现新系统内容时容器显示，而容器内具体显示什么内容，我们托管给具体的路由模块来负责，这样我们就实现了新旧系统无缝融合，具体参考下图进行说明：

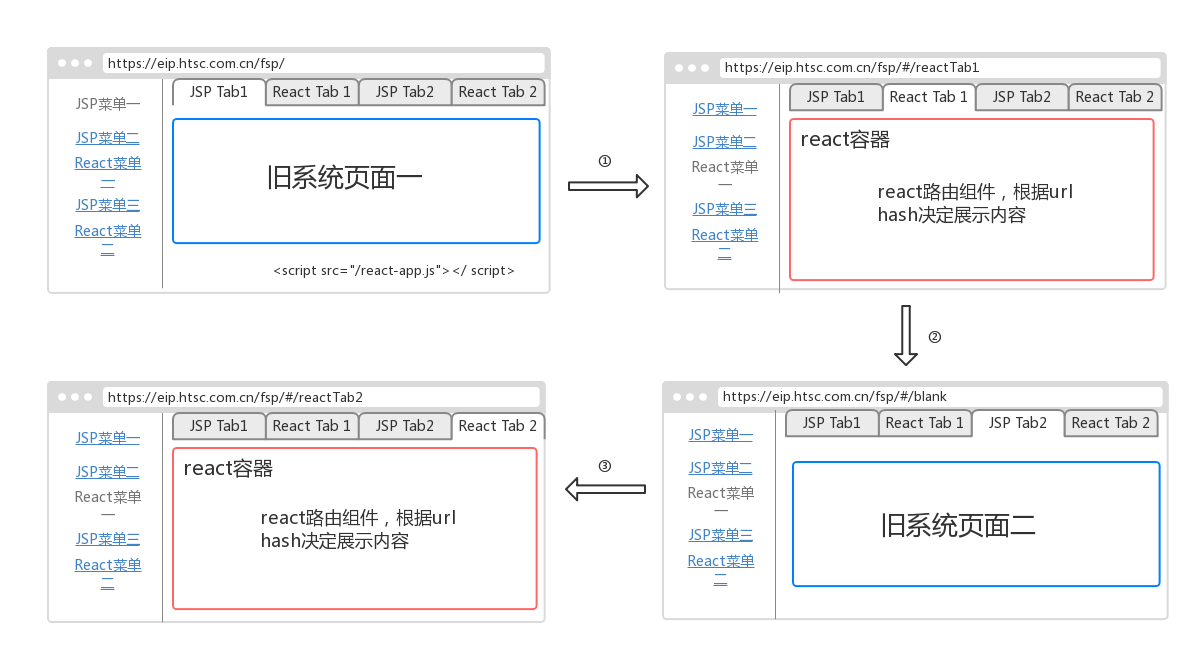


图1 理财服务平台新旧系统整合

1. 上图是理财服务平台页面的基本结构：左侧菜单+右侧TAB工作区，首先我们在原系统页面引入/fspa/react-app.js文件，这个文件是新旧系统唯一的联系，这里面存在一些设计考虑:
   1. 从页面加载性能优化和缓存利用角度，一般React项目构建产物至少包括三个文件，依赖库、业务包以及样式文件，同时文件命名加上hash值以便设置永久缓存，比如lib.018722a2.js / app.f48a8574.js / app.6e3ad99f.css;
   2. 新旧两个项目应能够独立部署，但是由于react产物文件名根据内容hash生成，是不确定的，所以react项目在构建时，需要同时生成一个给旧系统使用的入口文件，该文件包含当前构建中的所有产物，同时该文件缓存设置为“Cache-Control:no-cache“,强制要求每次加载均需要到服务端验证，从而保证react项目部署后能够立即生效，该文件体积很小，对页面性能影响不大。
2. 如图所示，当页面切换到React Tab时，我们首先更新地址栏的url hash部分，同时将预埋的React容器切入到可视区域，并隐藏上一个JSP Tab页面，至于React容器内应该显示什么内容，由React-router托管，React-router根据地址栏的hash值决定显示具体的路由组件；
3. 如图所示，当页面从React Tab切到JSP Tab页面，我们切出React容器，并按照系统原有逻辑显示JSP Tab页面，同时注意将地址栏hash切换到一个空白地址，以保证react组件的正常卸载；上述内容只是新旧系统融合的基本原理，在具体实现过程中，有许多细节问题需要解决，如： 如何保证JavaScript全局变量、CSS样式不冲突？如何保证前端开发人员对旧系统环境完全无依赖？单页应用如何处理门户超时？新旧系统如何交互等,由于篇幅有限，细节问题不一一阐述，如果有兴趣可以线下交流。

## 灰度替换

前面说过，新旧系统共存是过渡方案，最终我们希望通过逐步改造的方式，实现系统完全替换，改造主要面临几个问题：

1. 理财服务平台体量较大，有5个开发小组，每月新增react代码量10K+。
2. 历史遗留jsp页面较多，全面替换成本高。
3. 重构系统变动太大，为了稳妥起见，须采用灰度发布方案，逐步开放。

基于上述问题，我们在确定改造方案时须满足：

1. 新开发代码能同时满足在新旧系统中正常工作，既避免重复开发，也保证向新系统平稳过渡；
2. 逐步进行新旧替换，具体方案：
3. 首先采用上述的新旧系统整合方案，在现有系统中无缝嵌入react页面；
4. 同时采用react相关技术栈，根据业务部门新首页改造需求，平行开发全新入口，在新入口中可复用react页面；
5. 针对历史遗留的jsp页面，我们封装了HTMLComponent和IframeComponent组件作为适配层，将jsp页面封装成react组件在新入口中使用，后续jsp改造将综合考虑业务需要和成本进行逐步替换；
6. 新入口开发完成后，新旧系统在线上共存，可以采用灰度发布方式，逐步开放新入口。

# 总结

技术演进必须以业务平稳运行作为前提，本文以理财服务平台前端技术改造为例，主要阐述了改造过程中遇到的一系列问题以及相应解决方案，希望对类似场景的其他旧系统改造有所帮助。