Egg-React-SSR 深度解析

如何搭建一个成熟的企业级同构项目开发环境

分享者：十忆
阿里巴巴前端工程师
ykfe项目组核心开发者，Umijs官方成员

目录

- 介绍我们开源的egg-react-ssr项目
- 分享开发过程中的一些思考以及个人认为的一些最佳实践方式
- 在Serverless场景下，我们应该如何定义规范来进行页面的渲染服务

为什么服务端渲染又流行了

什么是同构

一套代码(逻辑)即运行在服务端又运行在客户端

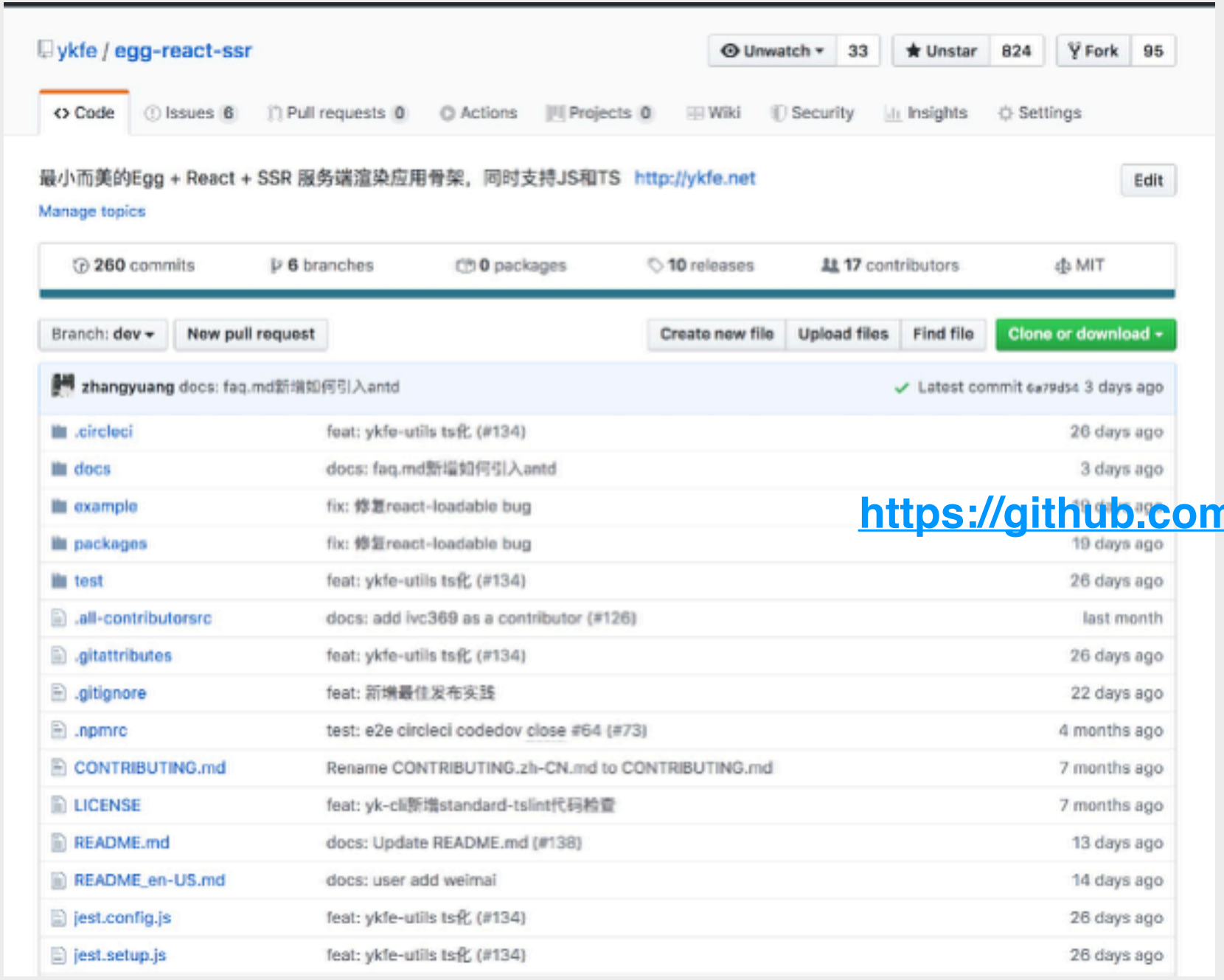
优点

前后端逻辑一致

维护成本降低

可以使用现代框架

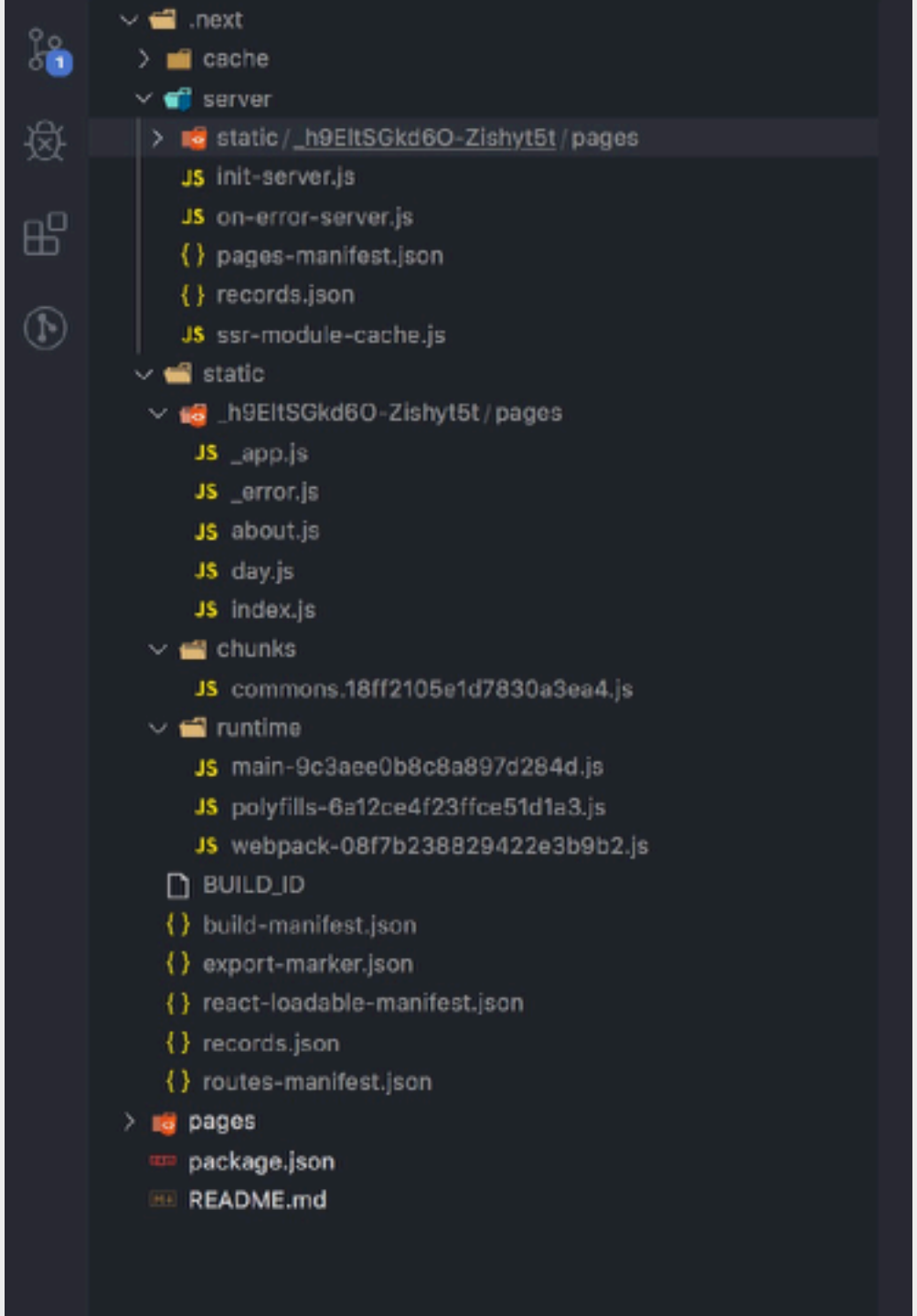
1、egg-react-ssr



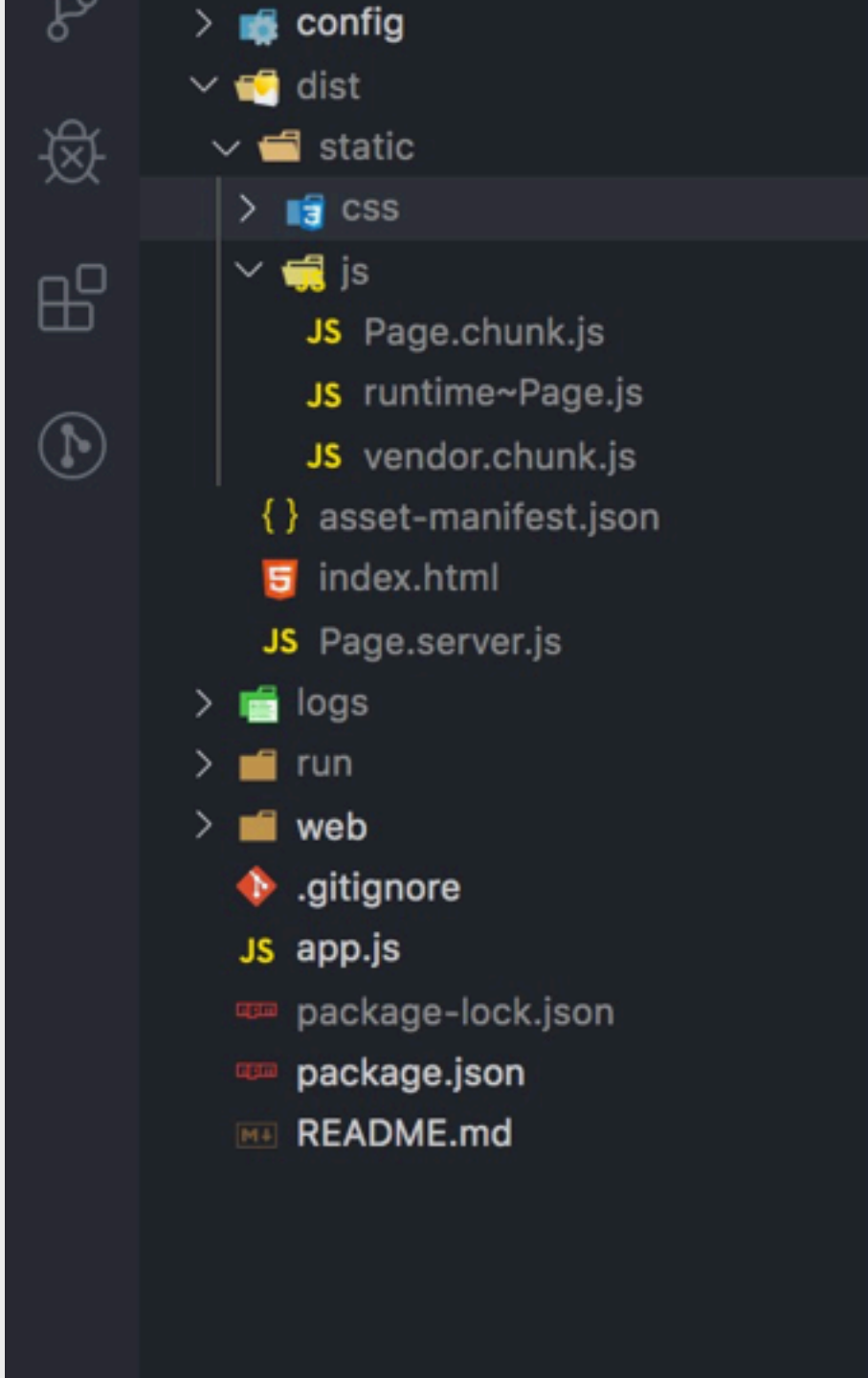
https://github.com/ykfe/egg-react-ssr

小：实现方式简洁，生产环境构建出来的bundle为同等复杂度的next.js项目的0.7倍，生成文件数量相比于next.js减少非常多

next.js



egg
react
ssr



：支持HMR，配套结合antd,mobx,dva,fc多个example，

支持CSR/SSR两种渲染模式无缝切换一键降级，同时支持TypeScript

服务端渲染

<!DOCTYPE html><html lang="en"><head><meta charSet="utf-8"/><meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"/><meta name="theme-color" content="#000000"/><title>React App</title><link rel="stylesheet" href="/static/css/Page.chunk.css"/></head><body> <h1 class="title">Egg + React + SSR by ykfe </h1> <ul class="list"> 文章标题：<!-- -->Racket v7.3 Release Notes 点击查看详情 文章标题：<!-- -->Free Dropbox Accounts Now Only Sync to Three Devices 点击查看详情 文章标题：<!-- -->Voynich Manuscript Decoded by Bristol Academic 点击查看详情 文章标题：<!-- -->Burger King to Deliver Whoppers to LA Drivers Stuck in Traffic 点击查看详情 文章标题：<!-- -->How much do YouTube celebrities charge to advertise your product? 点击查看详情 <script>window.__USE_SSR__=true; window.__INITIAL_DATA__ ={"news":[{"id":"1","title":"Racket v7.3 Release Notes"},{"id":"2","title":"Free Dropbox Accounts Now Only Sync to Three Devices"},{"id":"3","title":"Voynich Manuscript Decoded by Bristol Academic"},{"id":"4","title":"Burger King to Deliver Whoppers to LA Drivers Stuck in Traffic"},{"id":"5","title":"How much do YouTube celebrities charge to advertise your product? "}]}</script> <script src="/static/js/runtime~Page.js"></script><script src="/static/js/vendor.chunk.js"></script><script src="/static/js/Page.chunk.js"></script> </body></html>

一键切换，http://xxx.com?csr=true

客户端渲染

<!DOCTYPE html><html lang="en" data-reactroot=""><head><meta charSet="utf-8"/><meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"/><meta name="theme-color" content="#000000"/><title>React App</title><link rel="stylesheet" href="/static/css/Page.chunk.css"/></head><body> <script src="/static/js/runtime~Page.js"></script><script src="/static/js/vendor.chunk.js"></script><script src="/static/js/Page.chunk.js"></script> </body></html>

美：基于React和Eggjs框架，拥有强大的插件生态，配置非黑盒，核心代码简洁明了，无需改动源码即可实现业务的自定义需求

已拥有多个线上应用，获得所有使用者的一致好评

简体中文 | English

Egg + React + SSR boilerplate

最小而美的服务端渲染应用模板，特点

- 小：实现方式简洁，生产环境构建出来的bundle为同等复杂度的next.js项目的0.4倍，生成文件数量相比于next.js减少非常多
- 全：支持HMR，支持本地开发以及生产环境CSR/SSR两种渲染模式无缝切换，支持定制组件的渲染模式，同时支持TypeScript版本
- 美：基于React和Eggjs框架，拥有强大的插件生态，配置非黑盒，且一切关键位置皆可通过config.ssr.js来配置

正在使用这个项目的公司(应用)，如果您正在使用但名单中没有列出来的话请提issue，欢迎推广分享

2、开发过程的思考和最佳实践

如何搭建一个成熟的同构应用的开发环境?

你至少需要想清楚以下九点问题要怎么处理:

1、Node.js环境如何加载前端组件

- 2、组件的数据如何获取

3、HMR怎么做

4、CSS如何处理

5、如何拼接成完整的html结构返回

6、双端渲染结果不一致怎么办

7、如何进行路由分割

8、如何降级为客户端渲染

9、生产环境如何发布应用

1、Node.js环境如何加载前端组件

❌ es6 moudles

❌ jsx

都不支持怎么办，使用babel, webpack?

如何打包?

只打包前端组件?

还是连Node.js代码一起打包?

偷懒直接全部一起打包？你会遇到下列问题

1、引入babel会导致你要加载的模块数量变得巨大

2、服务端代码追求稳定，绝大部分语法Node8已原生支持，除非要用静态分析上TS，否则编译服务端文件意义不大

3、编译后的代码，对你是黑盒，不可读，debug困难

我们只处理前端组件

```
const serverRender = async (ctx) => {
  // 服务端渲染 根据ctx.path获取请求的具体组件，调用getInitialProps并渲染
  const ActiveComponent = getComponent(Routes, ctx.path)()
  const Layout = ActiveComponent.Layout || defaultLayout
  const serverData = ActiveComponent.getInitialProps ? await ActiveComponent.getInitialPr
  ctx.serverData = serverData
  return <StaticRouter location={ctx.req.url} context={serverData}>
    <Layout layoutData={ctx}>
      <ActiveComponent {...serverData} />
    </Layout>
  </StaticRouter>
}
```

Webpack配置怎么写?

```
const webpack = require('webpack')
const merge = require('webpack-merge')
const nodeExternals = require('webpack-node-externals')
const baseConfig = require('./webpack.config.base')
const paths = require('./paths')
const isDev = process.env.NODE_ENV === 'development'

const plugins = [
  new webpack.DefinePlugin({
    '__isBrowser__': false //eslint-disable-line
  })
]
module.exports = merge(baseConfig, {
  devtool: isDev ? 'eval-source-map' : '',
  entry: {
    Page: paths.entry
  },
  target: 'node',
  externals: nodeExternals({
    whitelist: /\.(css|less|sass|scss)$/
  }),      yuang.zya, 9 months ago • feat: 文件初始化
  output: {
    path: paths.appBuild,
    publicPath: '/',
    filename: '[name].server.js',
    libraryTarget: 'commonjs2'
  },
  plugins: plugins
})
```

重要选项

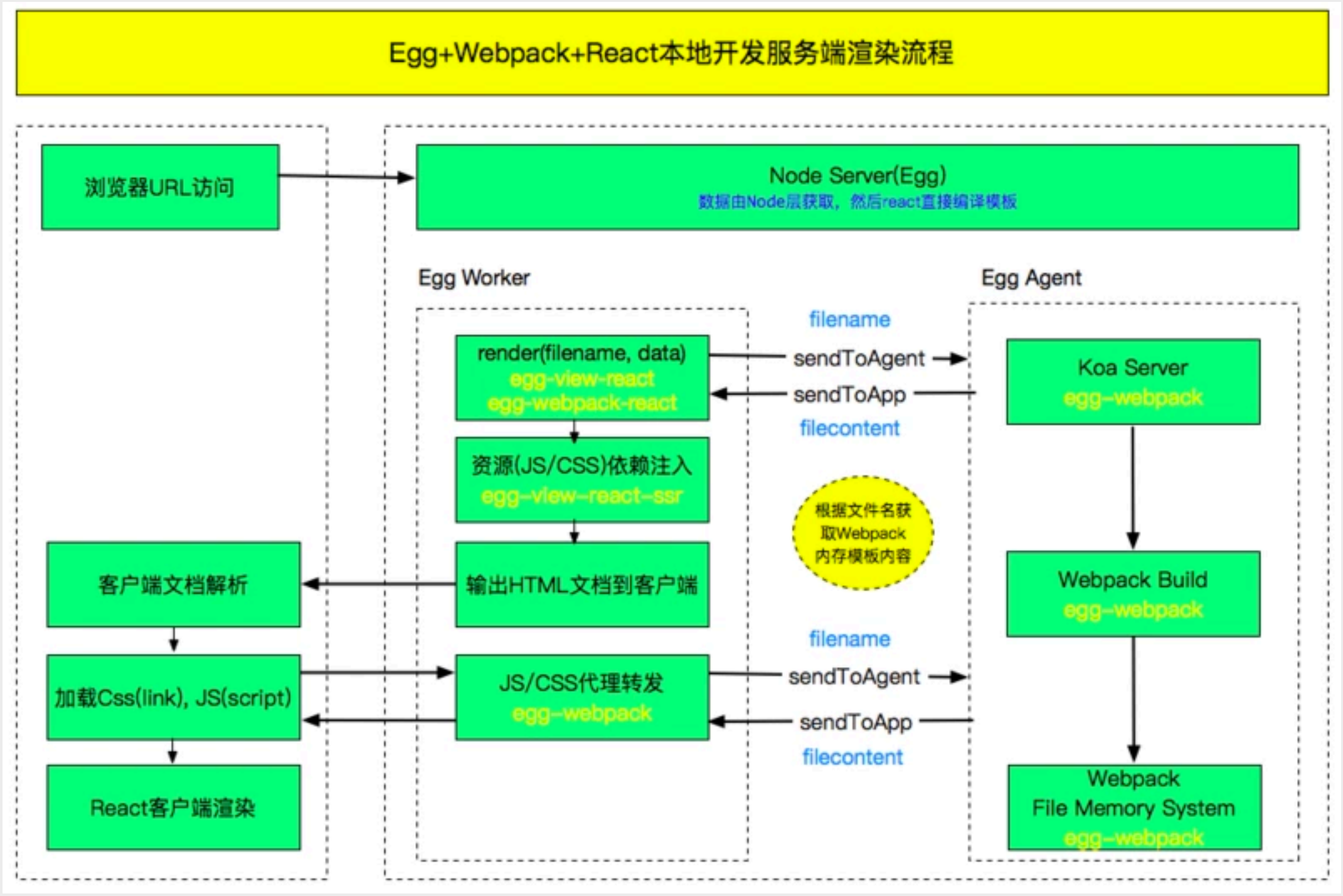
target

externals

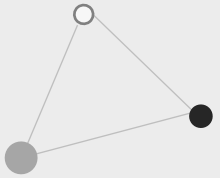
打包结果如何存放

文件存放于内存的优势	文件存放于内存的劣势
加载速度快 ✅	1、读取麻烦，需要用到memory-fs，同时涉及到进程通信 ❌ 2、loadable路由分割场景处理麻烦还是得落地磁盘 ❌ 3、开启externals后文件大小大多在几十kb之内，对比存放内存中速度优势并不明显 ❌

放在内存 ❌，本地硬盘 ✅


Egg+Webpack+React本地开发服务端渲染流程
浏览器URL访问
Node Server(Egg)
数据由Node层获取，然后react直接编译模板
Egg Worker
Egg Agent
filename
render(filename, data)
egg-view-react
egg-webpack-react
sendToAgent
sendToApp
filecontent
Koa Server
egg-webpack
资源(JS/CSS)依赖注入
egg-view-react-ssr
根据文件名获取Webpack内存模板内容
客户端文档解析
输出HTML文档到客户端
Webpack Build
egg-webpack
filename
加载Css(link), JS(script)
JS/CSS代理转发
egg-webpack
sendToAgent
sendToApp
filecontent
React客户端渲染
Webpack
File Memory System
egg-webpack

组件数据如何获取



1、放在componentWillMount?

2、定义静态方法获取?

componentWillMount中获取有什么问题?

1、客户端会重复获取数据

2、异步请求无法在render方法调用之前取得结果

在服务端如何调用静态方法getInitialProps

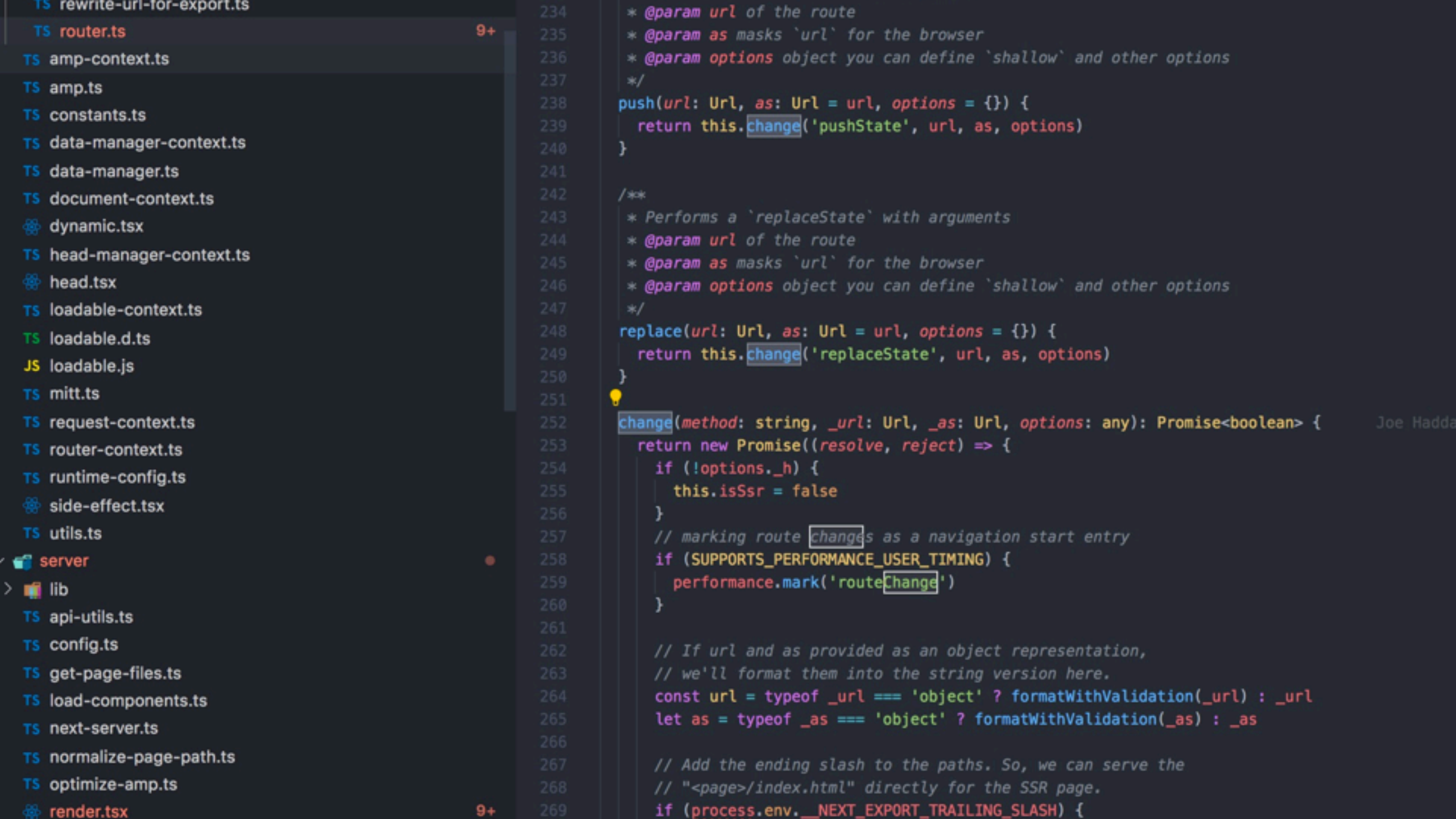
```
const serverRender = async (ctx) => {
  // 服务端渲染 根据ctx.path获取请求的具体组件, 调用getInitialProps并渲染
  const ActiveComponent = getComponent(Routes, ctx.path)()
  const Layout = ActiveComponent.Layout || defaultLayout
  const serverData = ActiveComponent.getInitialProps ? await ActiveComponent.getInitialProps(ctx) : {}
  ctx.serverData = serverData
  return <StaticRouter location={ctx.req.url} context={serverData}>
    <Layout layoutData={ctx}>
      <ActiveComponent {...serverData} />
    </Layout>
  </StaticRouter>
}
```

注意：只适用于页面级组件, 使用Suspense可解

在客户端切换路由时如何调用静态方法getInitialProps

next.js如何做的?

魔改react-router,自己封装了一套router，监听你的各种路由跳转动作


rewrite-uri-for-export.ts
router.ts 9+
amp-context.ts
amp.ts
constants.ts
data-manager-context.ts
data-manager.ts
document-context.ts
dynamic.tsx
head-manager-context.ts
head.tsx
loadable-context.ts
loadable.d.ts
loadable.js
mitt.ts
request-context.ts
router-context.ts
runtime-config.ts
side-effect.tsx
utils.ts
server
lib
api-utils.ts
config.ts
get-page-files.ts
load-components.ts
next-server.ts
normalize-page-path.ts
optimize-amp.ts
render.tsx 9+
234 * @param url of the route
235 * @param as masks `url` for the browser
236 * @param options object you can define `shallow` and other options
237 */
238 push(url: Url, as: Url = url, options = {}) {
239 return this.change('pushState', url, as, options)
240 }
241
242 /**
243 * Performs a `replaceState` with arguments
244 * @param url of the route
245 * @param as masks `url` for the browser
246 * @param options object you can define `shallow` and other options
247 */
248 replace(url: Url, as: Url = url, options = {}) {
249 return this.change('replaceState', url, as, options)
250 }
251
252 change(method: string, _url: Url, _as: Url, options: any): Promise<boolean> {
253 return new Promise((resolve, reject) => {
254 if (!options._h) {
255 this.isSsr = false
256 }
257 // marking route changes as a navigation start entry
258 if (SUPPORTS_PERFORMANCE_USER_TIMING) {
259 performance.mark('routeChange')
260 }
261
262 // If url and as provided as an object representation,
263 // we'll format them into the string version here.
264 const url = typeof _url === 'object' ? formatWithValidation(_url) : _url
265 let as = typeof _as === 'object' ? formatWithValidation(_as) : _as
266
267 // Add the ending slash to the paths. So, we can serve the
268 // "<page>/index.html" directly for the SSR page.
269 if (process.env.__NEXT_EXPORT_TRAILING_SLASH) {
Joe Hadda

一个HOC搞定

```
function GetInitialProps (WrappedComponent: FC): React.ComponentClass {
  You, 20 days ago | 3 authors (LeonCheung and others)
  class GetInitialPropsClass extends Component<RouteComponentProps<{}>, IState> {
    constructor (props: RouteComponentProps) {
      super(props)
      this.state = {
        extraProps: {},
        getProps: false
      }
    }       LeonCheung, 7 months ago • feat: 新增官方文档，以lerna来管理应用，新增脚手架 (#18)

    async componentDidMount () {
      const props = this.props
      if (window.__USE_SSR__) {
        _this = this // 修正_this指向，保证_this指向当前渲染的页面组件
        window.addEventListener('popstate', popStateFn)
      }
      const getProps = !window.__USE_SSR__ || (props.history && props.history.action === 'PUSH')
      if (getProps) {
        await this.getInitialProps()
      }
    }

    async getInitialProps () {
      // csr首次进入页面以及csr/ssr切换路由时才调用getInitialProps
      const props = this.props
      if (WrappedComponent.preload) {
        // react-loadable 情况
        WrappedComponent = (await WrappedComponent.preload()).default
      }
      const extraProps = WrappedComponent.getInitialProps ? await WrappedComponent.getInitialProps(props) : {}
      this.setState({
        extraProps,
        getProps: true
      })
    }
```

```
const dev = async (argv?: Argv) => {
  const PORT = (argv && argv.PORT) || 8000
  const compiler = webpack(clientConfig)
  const server = new WebpackDevServer(compiler, {
    quiet: true,
    disableHostCheck: true,
    publicPath: clientConfig.output.publicPath || '/',
    hotOnly: true,
    host: '0.0.0.0',
    sockPort: PORT,
    contentBase: cwd + '/dist',
    hot: true,      LeonCheung, 5 months ago • feat: 使用jsx代替
    port: PORT,
    clientLogLevel: 'error',
    headers: {
      'access-control-allow-origin': '*'
    },
    proxy: {
      '/api': 'http://localhost:7001'
    },
    before (app: any) {
      app.get('/', async (req: any, res: any) => {
        res.write('<!DOCTYPE html>')
        const stream = await renderLayout()
        stream.pipe(res, { end: false })
        stream.on('end', () => {
          res.end()
        })
      })
    },
    after (app: any) {
      app.get(/^\//, async (req: any, res: any) => {
        res.write('<!DOCTYPE html>')
        const stream = await renderLayout()
        stream.pipe(res, { end: false })
```

3、HMR怎么做

热替换不是热重载 ❌

直接用成熟的webpack-dev-server

HMR原理浅析:http://ykfe.net/guide/hmr.html

4、CSS如何处理

style-loader ❌	css-hot-loader ✅	isomorphic-style-loader ❌
不支持服务端渲染	几乎无需配置 同样支持HMR 统一开发/生产环境渲染方式	配置麻烦，对代码入侵性强 需要手动收集样式依赖

5、如何返回完整的html结构

返回string

```
app.get("*", (req, res) => {
  const htmlMarkup = renderToString(<App />);
  res.send(`
      <!DOCTYPE html>
      <head>
        <title>Universal Reacl</title>
        <link rel="stylesheet" href="/css/main.css">
        <script src="/bundle.js" defer></script>
      </head>

      <body>
        <div id="root">${htmlMarkup}</div>
      </body>
    </html>
  `);
});

app.listen(process.env.PORT || 3000, () => {
  console.log("Server is listening");
});
```

变成stream呢？如何返回?

```
// using Express
import { renderToNodeStream } from "react-dom/server"
import MyPage from "./MyPage"
app.get("/", (req, res) => {
  res.write("<!DOCTYPE html><html><head><title>My Page</title></head><body>
  res.write("<div id='content'>");
  const stream = renderToNodeStream(<MyPage/>);
  stream.pipe(res, { end: false });
  stream.on('end', () => {
    res.write("</div></body></html>");
    res.end();
  });
});
```

十分的ugly ✖

如何编译成字符串

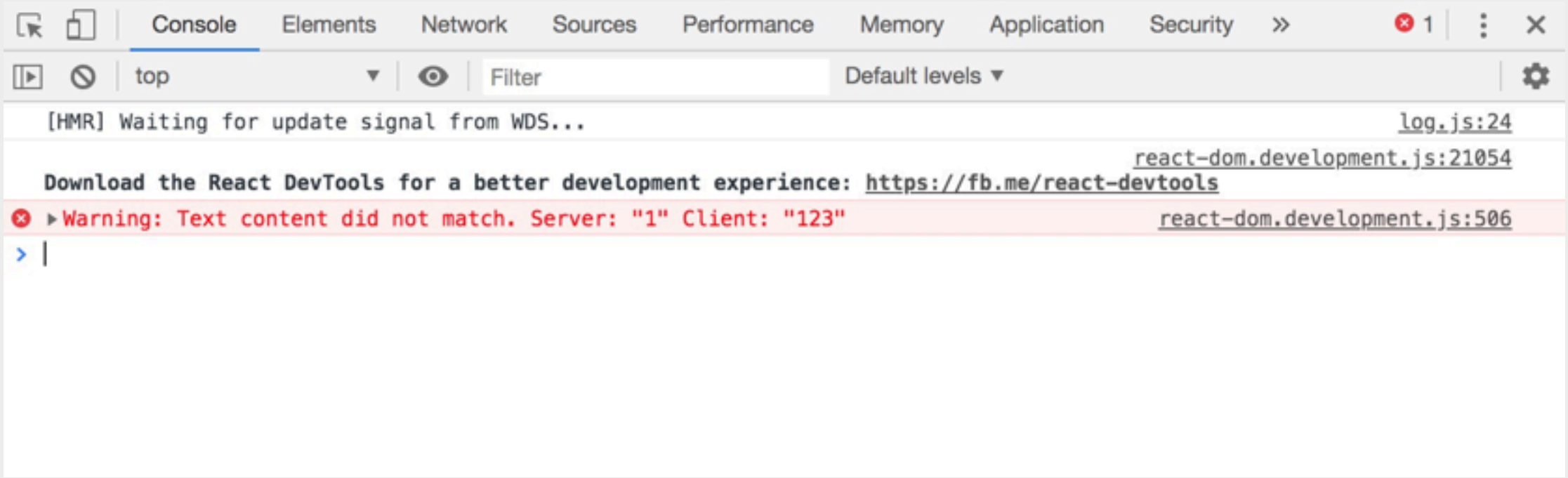
html文件 ❌	JSX ✅	模版引擎 ❌
不够灵活	功能强大 编辑器错误提示友好 无需引入新的三方库 支持HMR 支持TS	调试麻烦 需要学习特定的语法 可读性差

All JSX

```jsx
const Layout = (props) => {
  if (__isBrowser__) {
    return commonNode(props)
  } else {
    const { serverData } = props.layoutData
    const { injectCss, injectScript } = props.layoutData.app.config
    return (
      <html lang='en'>
        <head>
          <meta charSet='utf-8' />
          <meta name='viewport' content='width=device-width, initial-scale=1, shrink-to-fit=no' />
          <meta name='theme-color' content='#000000' />
          <title>React App</title>
          {
            injectCss && injectCss.map(item => <link rel='stylesheet' href={item} key={item} />)
          }
        </head>
        <body>
          <div id='app'>{ commonNode(props) }</div>
          {
            serverData && <script dangerouslySetInnerHTML={{
              __html: `window.__USE_SSR__=true; window.__INITIAL_DATA__ =${serialize(serverData)}`
            }} />
          }
          <div dangerouslySetInnerHTML={{
            __html: injectScript && injectScript.join('')
          }} />
        </body>
      </html>
    )
  }
}
```

```
const stream = await renderToStream(ctx, ctx.app.config)
ctx.res.write('<!DOCTYPE html>')
ctx.body = stream ✅
```

6、注水后双端渲染结果不一致怎么办



在大多数情况下你都需要把这个warning当作错误来处理
除非你清楚的知道自己确实要渲染不一样的内容

解決方式

jsdom

它仅仅帮你绕过了错误，但对你没有实际帮助在大多数情况下
你必须清楚服务端与客户端的代码编写差异

只在客户端加载模块

1. 使用本应用提供的 `__isBrowser__` 常量来判断，例如引入jquery可以使用以下引入方式

```js
import $ from 'jquery' // error
const $ = __isBrowser__ ? require('jquery') : {} // true
```

2. 在 `didMount` 生命周期加载模块

```js
class Page {
    this.state = {
        $: {}
    }
    componentDidMount () {
        this.setState({
            $: require('jquery')
        })
    }
}
```

编写一个onlyCsr的高阶组件，只在客户端进行渲染

```tsx
// 通过使用该HOC使得组件只在客户端进行渲染
import React, { Component } from 'react'

LeonCheung, a month ago | 1 author (LeonCheung)
interface State {
  isCsr: boolean
}

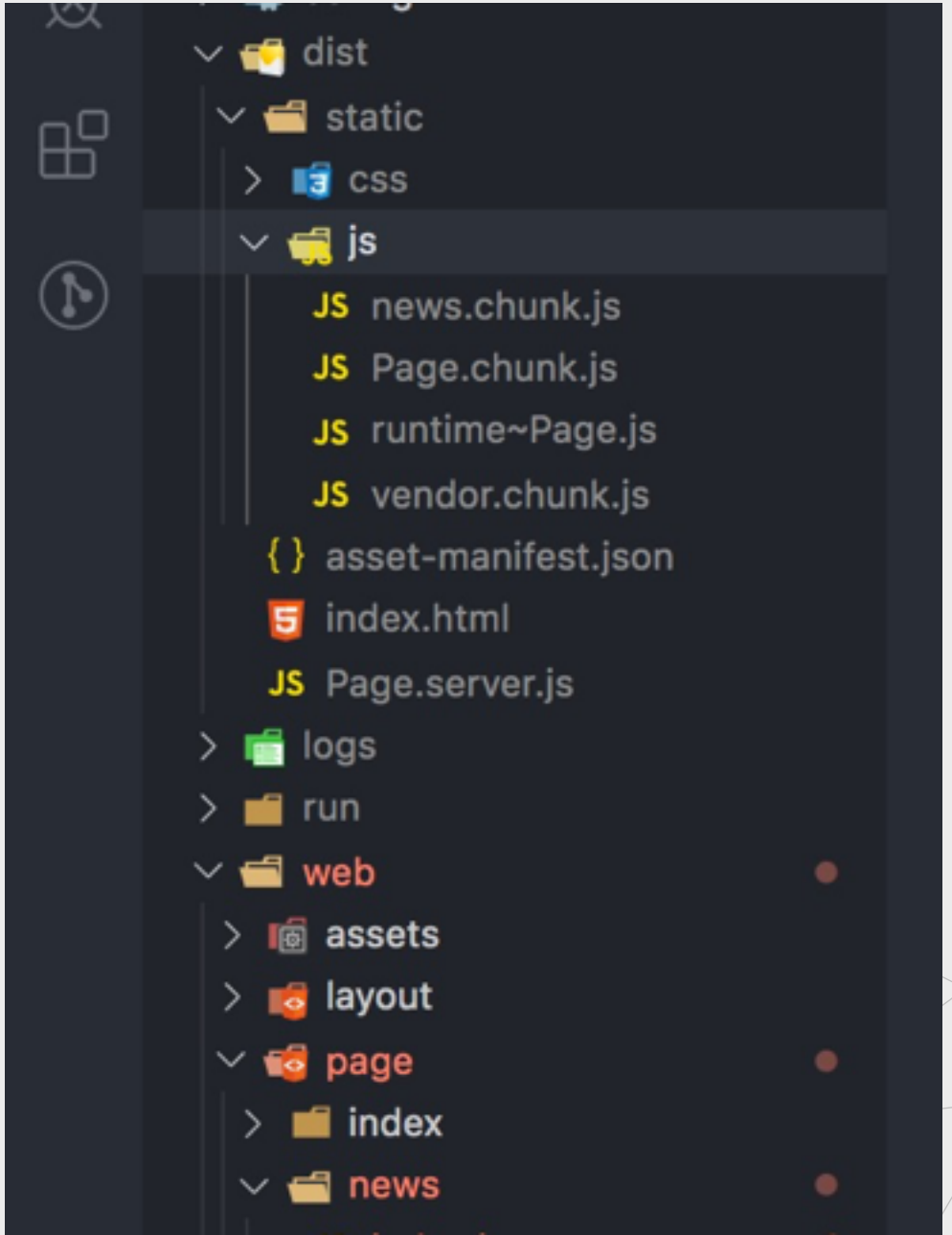
function onlyCsr (WrappedComponent: React.FC): React.ComponentClass {
  LeonCheung, a month ago | 1 author (LeonCheung)
  class OnlyCsrClass extends Component<any, State> {
    constructor (props: any) {
      super(props)
      this.state = {
        isCsr: false
      }
    }

    componentDidMount () {
      this.setState({      LeonCheung, a month ago • feat: ykfe-utils ts化 (#134)
        isCsr: true
      })
    }

    render () {
      return this.state.isCsr ? <WrappedComponent {...this.props}></WrappedComponent> : <div></div>
    }
  }
```

7、路由分割怎么做

将页面拆成多个chunk.js



react-loadable

A higher order component for loading components with dynamic imports.

```
import Loadable from 'react-loadable';
import Loading from './my-loading-component';

const LoadableComponent = Loadable({
  loader: () => import('./my-component'),
  loading: Loading,
});

export default class App extends React.Component {
  render() {
    return <LoadableComponent/>;
  }
}
```

react-loadable官方SSR做法

```
import Loadable from 'react-loadable';
import { getBundles } from 'react-loadable/webpack'
import stats from './dist/react-loadable.json';

app.get('/', (req, res) => {
  let modules = [];

  let html = ReactDOMServer.renderToString(
    <Loadable.Capture report={moduleName => modules.push(moduleName)}>
      <App/>
    </Loadable.Capture>
  );
```

```
import Loadable from 'react-loadable';

app.get('/', (req, res) => {
  let modules = [];

  let html = ReactDOMServer.renderToString(
    <Loadable.Capture report={moduleName => modules.push(moduleName)}>
      <App/>
    </Loadable.Capture>
  );

  console.log(modules);

  res.send(`...${html}...`);
});
```

仔细思考官方推荐做法的目的，我们一定要按照它来做吗?

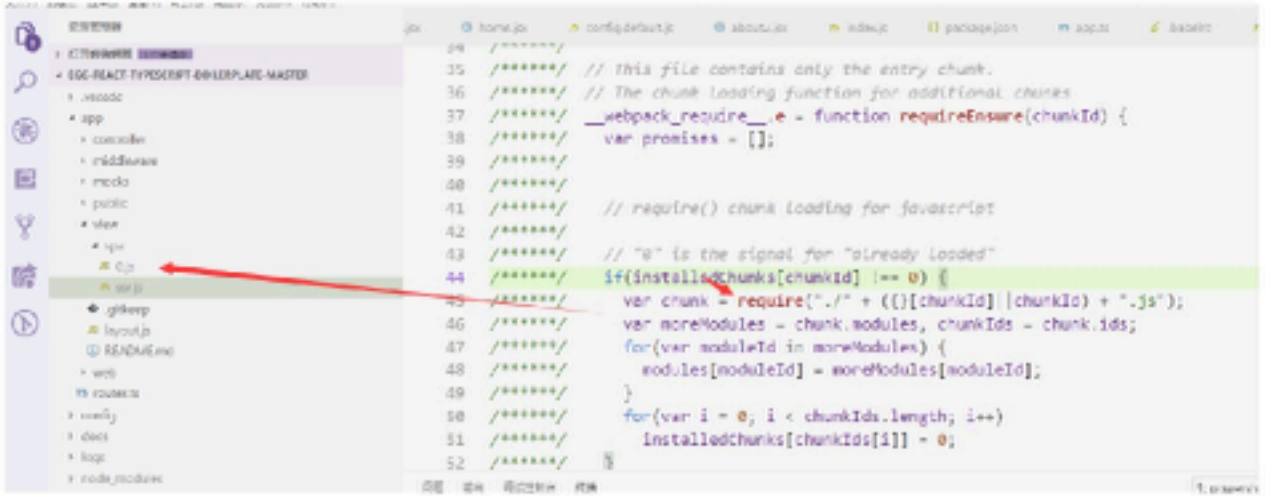
必须要求文件落地磁盘

w567675 commented on 6 Sep 2018 • edited ▾ Author +😀 ···

@hubcarl

spa的项目如果加入react-loadable. 在dev模式下，服务端运行的webpack会生成分离出一个0.js的一个chunk。 然后ssr.js里面会使用require（）的方式去加载这个0.js， 然后就会报错"can't fand ./0.js"。 因为node的是磁盘查找模式。

ssr.js



如果通过build的模式让文件落磁盘就没问题了。

不知道用什么思路去解决这个。

hubcarl commented on 1 Nov 2018 Collaborator +😀 ···

@w567675 这个可以在开发模式时，栓选出 异步chunk 模块，然后落地磁盘

yuxiuj commented on 5 May 2019 • edited ▾ +😀 ···

服务端Bundle无需分块

开启externals选项后，我们的Bundle天生就很小，再分块获得的性能收益与配置复杂度相比性价比极低

```
routes: [
  {
    path: '/',
    exact: true,
    Component: () => (require('@/page/index').default), // 这里使用一个function包裹为了让它延迟require
    controller: 'page',
    handler: 'index'
  },
  {
    path: '/news/:id',
    exact: true,
    Component: () => (__isBrowser__ ? require('react-loadable')({
      loader: () => import(/* webpackChunkName: "news" */ '@/page/news'),
      loading: function Loading () {
        return React.createElement('div')
      }
    }) : require('@/page/news').default
    ),
    controller: 'page',
    handler: 'index'      You, 4 months ago • feat: 新增配置文件config.ssr 分离前后端配置 close #81
  }
```

8、如何降级为客户端渲染

服务端渲染要做什么?

服务端获取数据 ✅

渲染完整页面 ✅

加载js ✅

注水绑定事件 ✅

客户端渲染要做什么?

服务端获取数据 ❌

渲染完整页面 ❌

加载js ✅

客户端获取数据✅

客户端渲染页面 ✅

绑定事件 ✅

结论：我们只需要服务端返回一个html空骨架即可

如何降级为客户端渲染

```
if (config.type !== 'ssr' || csr) {
  const renderLayout = require('yk-cli/lib/renderLayout').default
  const str = await renderLayout(ctx)
  return str
}
```

无需获取数据，直接把layout组件编译成空html返回 ✅

服务端渲染

<!DOCTYPE html><html lang="en"><head><meta charSet="utf-8"/><meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"/><meta name="theme-color" content="#000000"/><title>React App</title><link rel="stylesheet" href="/static/css/Page.chunk.css"/></head><body> <h1 class="title">Egg + React + SSR by ykfe </h1> <ul class="list"> 文章标题：<!-- -->Racket v7.3 Release Notes 点击查看详情 文章标题：<!-- -->Free Dropbox Accounts Now Only Sync to Three Devices 点击查看详情 文章标题：<!-- -->Voynich Manuscript Decoded by Bristol Academic 点击查看详情 文章标题：<!-- -->Burger King to Deliver Whoppers to LA Drivers Stuck in Traffic 点击查看详情 文章标题：<!-- -->How much do YouTube celebrities charge to advertise your product? 点击查看详情 <script>window.__USE_SSR__=true; window.__INITIAL_DATA__ ={"news":[{"id":"1","title":"Racket v7.3 Release Notes"},{"id":"2","title":"Free Dropbox Accounts Now Only Sync to Three Devices"},{"id":"3","title":"Voynich Manuscript Decoded by Bristol Academic"},{"id":"4","title":"Burger King to Deliver Whoppers to LA Drivers Stuck in Traffic"},{"id":"5","title":"How much do YouTube celebrities charge to advertise your product? "}]}</script> <script src="/static/js/runtime~Page.js"></script><script src="/static/js/vendor.chunk.js"></script><script src="/static/js/Page.chunk.js"></script> </body></html>

客户端渲染

<!DOCTYPE html><html lang="en" data-reactroot=""><head><meta charSet="utf-8"/><meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"/><meta name="theme-color" content="#000000"/><title>React App</title><link rel="stylesheet" href="/static/css/Page.chunk.css"/></head><body> <script src="/static/js/runtime~Page.js"></script><script src="/static/js/vendor.chunk.js"></script><script src="/static/js/Page.chunk.js"></script> </body></html>

如何发布应用

发布Node.js应用与传统的页面应用发布有什么区别?

传统页面

改个发布平台的版本号即可 ✅

2分钟生效，舒服

Node.js应用

发布

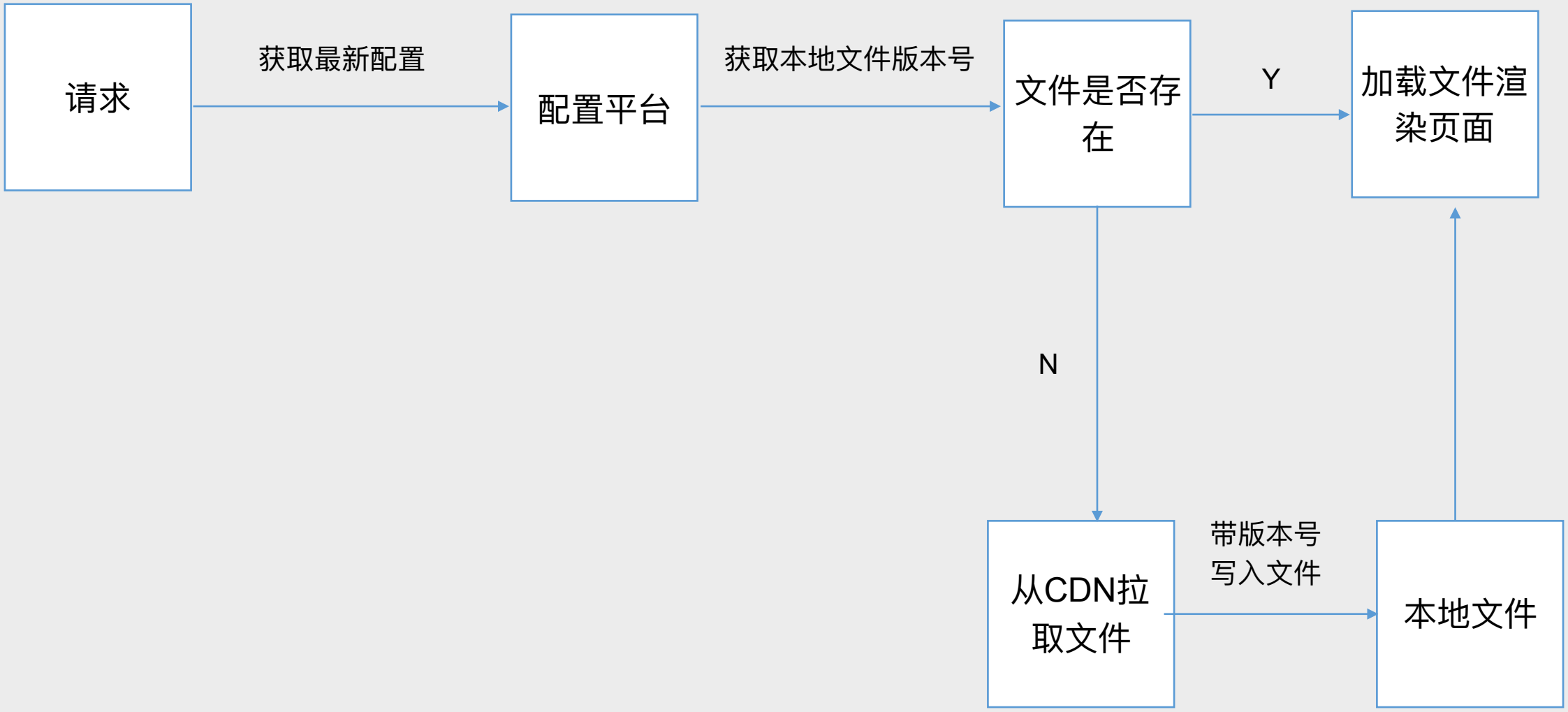
构建

分批重启

至少需要10分钟

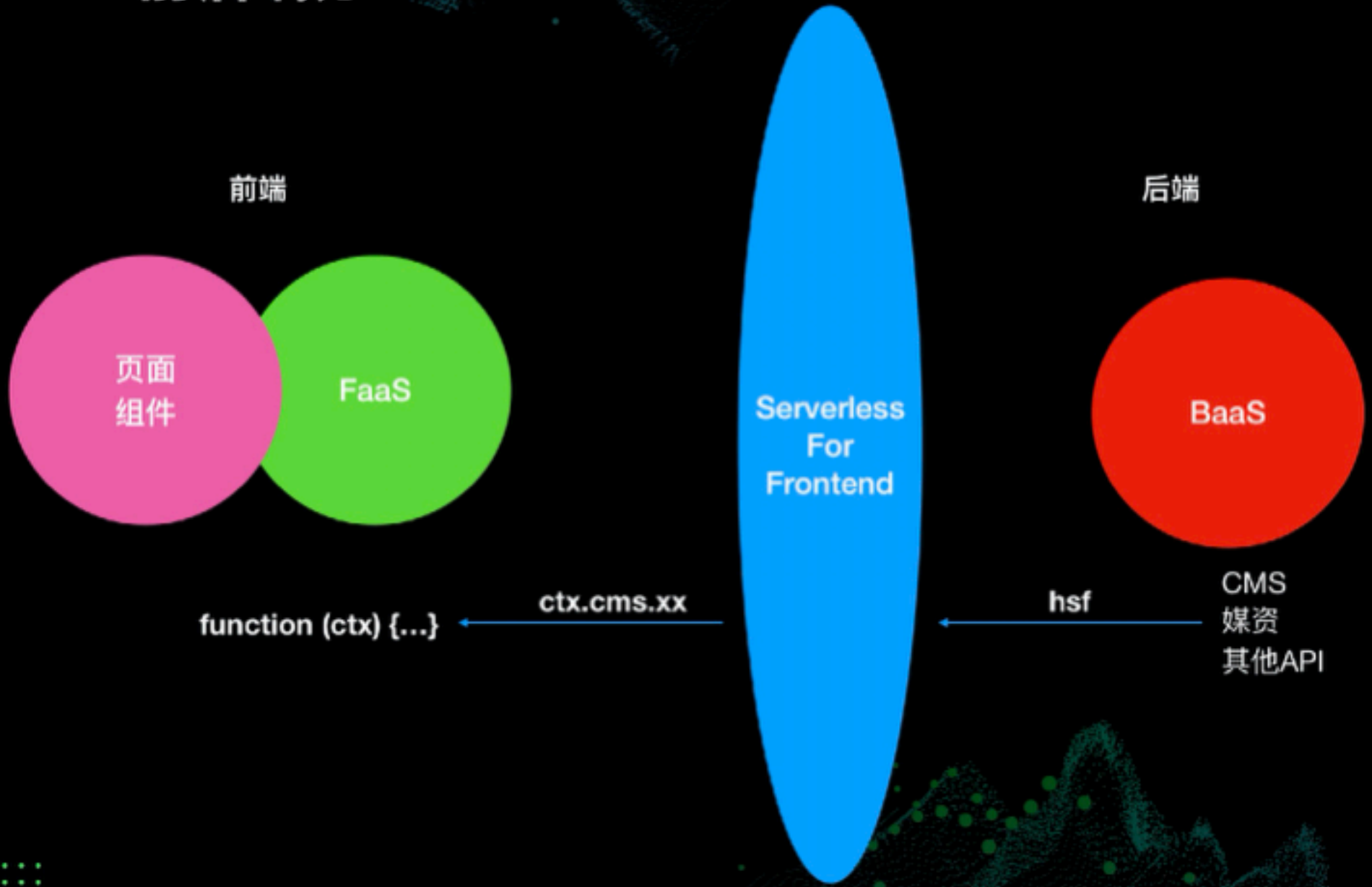
我们也想像传统的客户端渲染页面一样发布

所有文件全部上CDN

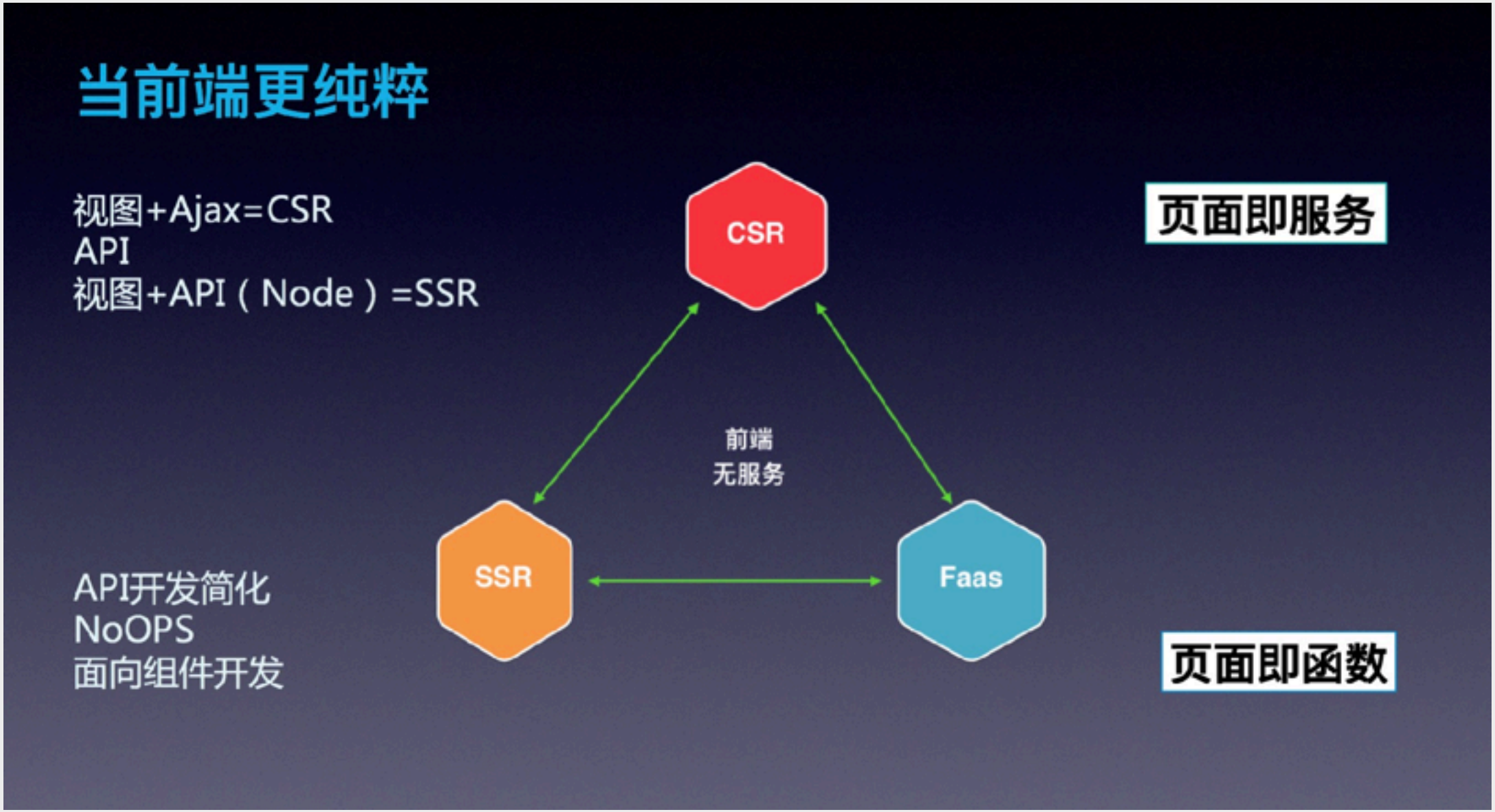

请求
获取最新配置
配置平台
获取本地文件版本号
文件是否存在
Y
加载文件渲染页面
N
从CDN拉取文件
带版本号写入文件
本地文件

3、serverless-side-render

SFF前后端分工


前端
后端
页面
组件
FaaS
Serverless
For
Frontend
BaaS
ctx.cms.xx
function (ctx) {...}
hsf
CMS
媒资
其他API

Serverless场景下如何做?



定义规范

- 提供服务器无关、函数无关的页面描述规范。
- 为不同页面的渲染，提供标准化组件描述，统一写法。
- 用法简单，易实现，可扩展。

组件即函数

- render是函数
- layout是函数
- fetch是函数

```
### 组件写法

```js
function Page(props) {
  return <div> {props.name} </div>
}

Page.fetch = async (ctx) => {
  return Promise.resolve({
    name: 'Serverless side render'
  })
}

Page.Layout = (props) => {
    const { serverData } = props.ctx
    const { injectCss, injectScript } = props.ctx.app.config
    return (
      <html lang='en'>
        <head>
          <meta charSet='utf-8' />
          <meta name='viewport' content='width=device-width, initial-scale=1, shrink-to-fit=no' />
          <meta name='theme-color' content='#000000' />
          <title>React App</title>
          {
            injectCss && injectCss.map(item => <link rel='stylesheet' href={item} key={item} />)
          }
        </head>
        <body>
          <div id='app'>{ commonNode(props) }</div>
          {
            serverData && <script dangerouslySetInnerHTML={{
              __html: `window.__USE_SSR__=true; window.__INITIAL_DATA__ =${serialize(serverData)}`
            }} />
          }
          <div dangerouslySetInnerHTML={{
            __html: injectScript && injectScript.join('')
          }} />
        </body>
      </html>
    )
}

export default Page
```
```

f.yml

```yaml
functions:
  home:
    handler: index.handler
    render:
      - component: src.home.index
      - layout: src.home.layout
      - fetch: src.home.fetch
      - mode: ssr | csr (默认ssr)
      - injectScript
          - runtime~Page.js
          - vendor.chunk.js
          - Page.chunk.js
      - injectCSS
          - Page.chunk.css
      - serverBundle: Page.server.js
    events:
      - http:
          path: /
          method:
            - GET
  news:
    handler: index.handler
    render:
      - component: src.news.index
      - layout: src.news.layout
      - fetch: src.news.fetch
      - mode: ssr | csr (默认ssr)
    events:
      - http:
          path: /
          method:
            - GET
```

配置都在yml文件中维护

一键发布应用

```
> fun deploy

using template: template.yml
using region: cn-shanghai
using accountId: ***********6972
using accessKeyId: ***********wekE
using timeout: 100

Waiting for custom domain ssr-fc.com to be deployed...
custom domain ssr-fc.com deploy success

Waiting for service ssr to be deployed...
        Waiting for function page to be deployed...
                Waiting for packaging function page code...
                The function page has been packaged. A total of 33 files files w
ere compressed and the final size was 726.19 KB
                Waiting for HTTP trigger http-test to be deployed...
                methods: [ 'GET' ]
                url: https://1812856288776972.cn-shanghai.fc.aliyuncs.com/2016-0
8-15/proxy/ssr/page/
                function http-test deploy success
        function page deploy success
service ssr deploy success
```

ssr-fc.com

概览 | 代码执行 | 触发器 | 日志查询 | 函数指标

ARN - acs:fc:cn-shanghai:1812856288776972:services/ssr.LATEST/functions/page

计量数据

本月资源使用量

534 次

本月资源使用量

8.338 CU-S

监控数据每小时更新并尽最大可能推送, 准确计量请参考 费用中心 最后统计时间 2020年1月11日 05:00

函数属性

配置 导出

函数名称 page

所属区域 华东2 (上海)

代码大小 743621 字节

创建时间 2019年10月7日 21:14

修改时间 2020年1月11日 11:58

函数入口 ./dist/FC.server.handler

运行环境 nodejs8

函数执行内存 128 MB

超时时间 3 秒

实例并发度 1

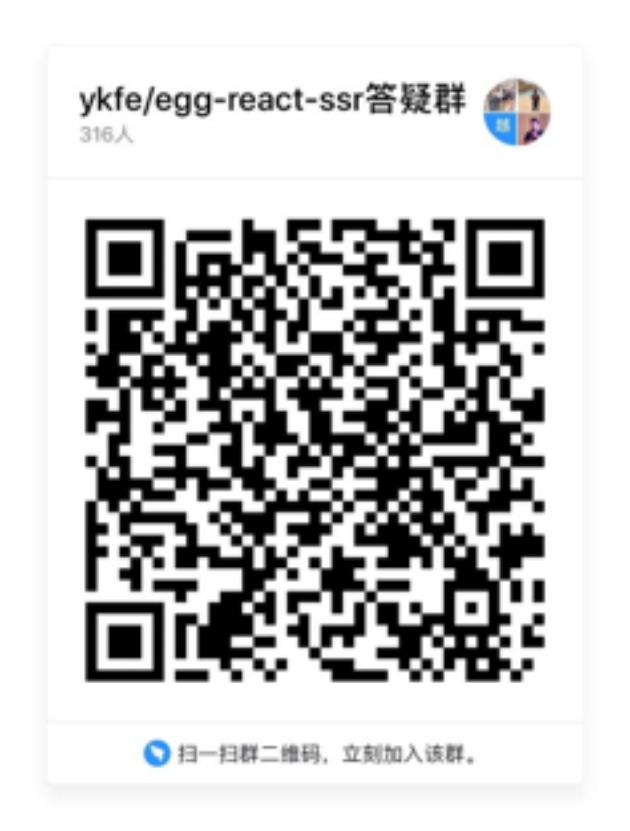
代码校验 7865114326325805989

描述信息 fc ssr demo with nodejs8!

环境变量 LD_LIBRARY_PATH: /code/.fun/root/usr/lib:/code/.fun/root/usr/lib/x86_64-linux-gnu:/code:/code/lib:/usr/local/lib

PATH: /code/.fun/root/usr/local/bin:/code/.fun/root/usr/local/sbin:/code/.fun/root/usr/bin:/code/.fun/root/usr/sbin:/code/.fun/root/sbin:/code/.fun/root/bin:/code/.fun/python/

钉钉交流群



Thanks!