

2021

YUANG

SSR框架V5.0

任意组合服务端框架/前端框架/与SERVERLESS 高度定制

TECHNOLOGY SHARING

作者介绍

ABOUT THE AUTHOR



张宇昂 (yuuang) 97年

2018 年毕业 加入阿里巴巴大文娱事业群优酷狼叔团队

2020 年 加入微信小程序开发团队

专注于 Node.js 与前端结合在企业的落地实践

专注于 服务端渲染 与 Serverless 的结合

目录

C O N T E N T S

01

框架介绍

Framework introduction

02

技术实现

Technical realization

03

未来展望

Future outlook

PART 01. SSR框架介绍

Framework introduction



<https://github.com/ykfe/ssr>

zhangyuan chore: vue example update App.vue			8ca154a 10 minutes ago	418 commits
github/workflows	fix: ci cache			2 days ago
.vscode	Feat/context close #8 (#9)			2 months ago
cypress	ci: combine cypress.spec			11 days ago
example	chore: vue example update App.vue			10 minutes ago
images	docs: update readme.md			4 days ago
packages	docs: update readme.md			19 hours ago
scripts	ci: publish readme.md on ci			9 days ago
.eslintignore	fix: types (skip ci)			13 days ago
.eslintrc.js	chore: update types			13 days ago
.gitignore	Feat/e2e (#14)			2 months ago
CHANGELOG.md	chore(release): publish v5.3.0			6 days ago
CONTRIBUTING.md	chore(release): publish			13 days ago
LICENSE	feat: init			12 months ago
README.md	chore: vue example update App.vue			10 minutes ago
cypress.json	chore: update e2e			13 days ago
jest.config.js	test: init jest config			11 months ago
jest.setup.js	Feat/context close #8 (#9)			2 months ago
lenna.json	chore(release): publish v5.3.2			2 days ago
package.json	feat/hest (#22)			7 days ago
tsconfig.cjs.json	Feat/context close #8 (#9)			2 months ago
tsconfig.esm.json	Feat/context close #8 (#9)			2 months ago
tsconfig.json	chore: add createRouter			13 days ago
tsconfig.lint.json	Feat/context close #8 (#9)			2 months ago
yarn.lock	fix: ci cache			2 days ago

Watch

8

★ Unstar

432

Fork

41

About



A most advanced ssr framework on Earth that implemented serverless-side render specification for faas and traditional web server.

github.com/ykfe/ssr

node

serverless

ssr

faas

midway-faas

ssr-spec

Readme

MIT License



介绍

INTRODUCTION

SSR 框架是为前端框架在服务端渲染的场景下所打造的开箱即用的服务端渲染框架。

在最新的 v5.0 版本中，同时支持 React 和 Vue 的服务端渲染框架，且提供一键以

Serverless 的形式发布上云的功能。

我们可以非常有自信说它是地球上最先进的ssr框架。如果你希望获得开箱即用的体验且

能够一键部署上云，请选择 ssr 框架。

Features

- 🌱 灵活组合：服务端支持 FaaS, Midway, Nestjs 前端支持 React/Vue 等现代 Web 框架
- 🚀 开箱即用：内置 10+ 脚手架配套扩展，如Antd、TS、Hooks等
- 🧩 插件驱动：基于插件架构，用户更加专注于业务逻辑
- 🏆 Serverless 优先：一键发布到各种Serverless平台，也支持传统Web Server，比如Egg、Midway、Nest等
- 🛡️ 高可用场景，可无缝从SSR降级到CSR，最佳容灾方案

支持创建多种场景的 example

\$ npm init ssr-app my-ssr-project --template=serverless-react-ssr

\$ npm init ssr-app my-ssr-project --template=serverless-vue-ssr

\$ npm init ssr-app my-ssr-project --template=midway-react-ssr

\$ npm init ssr-app my-ssr-project --template=midway-vue-ssr

\$ npm init ssr-app my-ssr-project --template=nestjs-react-ssr

\$ npm init ssr-app my-ssr-project --template=nestjs-vue-ssr

演变历史

EVOLUTION HISTORY

2019

egg-react-ssr

1.6k Star

数十个公司线上应用

2020

ssr v4.0

Serverless + React
+ SSR

2021

ssr v5.0

cover 所有场景

插件化组合服务端框架/前端框架

Serverless 场景高度优化

生态 monorepo 组织形式

ECOLOGY

Project	Status	Description
ssr	npm v5.3.2	cli for ssr framework
ssr-core-vue	npm v5.3.2	core render for vue
ssr-core-react	npm v5.3.2	core render for react
ssr-plugin-faas	npm v5.3.2	provide start deploy feature by midway-faas
ssr-plugin-midway	npm v5.3.2	provide start and build fetature by midway@2.0
ssr-plugin-nestjs	npm v5.3.2	provide start and build feature by Nestjs
ssr-plugin-react	npm v5.3.2	develop react application only be used in development
ssr-plugin-vue	npm v5.3.2	develop vue application only be used in development
ssr-server-utils	npm v5.3.2	server utils in Node.js environment
ssr-client-utils	npm v5.3.2	client utils in browser environment
ssr-hoc-react	npm v5.3.2	provide hoc component for react
ssr-types	npm v5.3.2	provide common types
ssr-webpack	npm v5.3.2	start local server and build production bundle by webpack

插件使用

PLUG-IN USE

我们目前提供了如下插件, 参考现有插件来开发一个新的插件是非常容易的事情。你可以根据自己的应用类型来自行开发对应的插件

例如 plugin-nest、plugin-egg, plugin-koa 等

服务端框架插件

- plugin-faas 基于 [Midway-faas](#)
- plugin-midway 基于 [Midwayjs@2.0](#)
- plugin-nestjs 基于 [Nestjs](#)

前端框架插件

- plugin-react 基于 React
- plugin-vue 基于 Vue

```
// plugin.js
const { faasPlugin } = require('ssr-plugin-faas')
const { reactPlugin } = require('ssr-plugin-react')
module.exports = {
  serverPlugin: faasPlugin(),
  clientPlugin: reactPlugin()
}
```

在example中
使用插件

```
"scripts": {
  "start": "ssr start",
  "build": "ssr build",
  "deploy": "ssr build && ssr deploy",
  "deploy:tencent": "ssr build && ssr deploy --tencent",
  "lint": "eslint . --ext .js,.tsx,.ts --cache",
  "lint:fix": "eslint . --ext .js,.tsx,.ts --cache --fix"
}
```

统一开发命令

```
$ ssr deploy
please input aliyun config
? Aliyun Account ID 1812856288776972
? Aliyun Access Key ID [REDACTED]
? Aliyun Access Key Secret [REDACTED]
? Default region name cn-shanghai
? The timeout in seconds for each SDK client invoking 10
? The maximum number of retries for each SDK client 3
? Allow to anonymously report usage statistics to improve the tool over time? Yes
Collecting your services information, in order to caculate development changes...

Resources Changes(Beta version! Only FC resources changes will be displayed):



| Resource | ResourceType                 | Action | Property |
|----------|------------------------------|--------|----------|
| ssr      | Aliyun::Serverless::Function | Modify | CodeUri  |



? Please confirm to continue. Yes
Waiting for service serverless-vue-ssr-spa to be deployed...
  Waiting for function ssr to be deployed...
    Waiting for packaging function ssr code...
      The function ssr has been packaged. A total of 4781 files were compressed and the final size was 8.9 MB
    Waiting for HTTP trigger http-ssr to be deployed...
      triggerName: http-ssr
      methods: [ 'GET', 'PUT', 'POST', 'DELETE', 'HEAD' ]
      trigger http-ssr deploy success
    function ssr deploy success
  service serverless-vue-ssr-spa deploy success

Detect 'DomainName:Auto' of custom domain 'midway_auto_domain'
Fun will reuse the temporary domain http://43579723-1812856288776972.test.functioncompute.com, expired at 2021-02-28 00:35:23, limited by 1000 per day.

Waiting for custom domain midway_auto_domain to be deployed...
custom domain midway_auto_domain deploy success

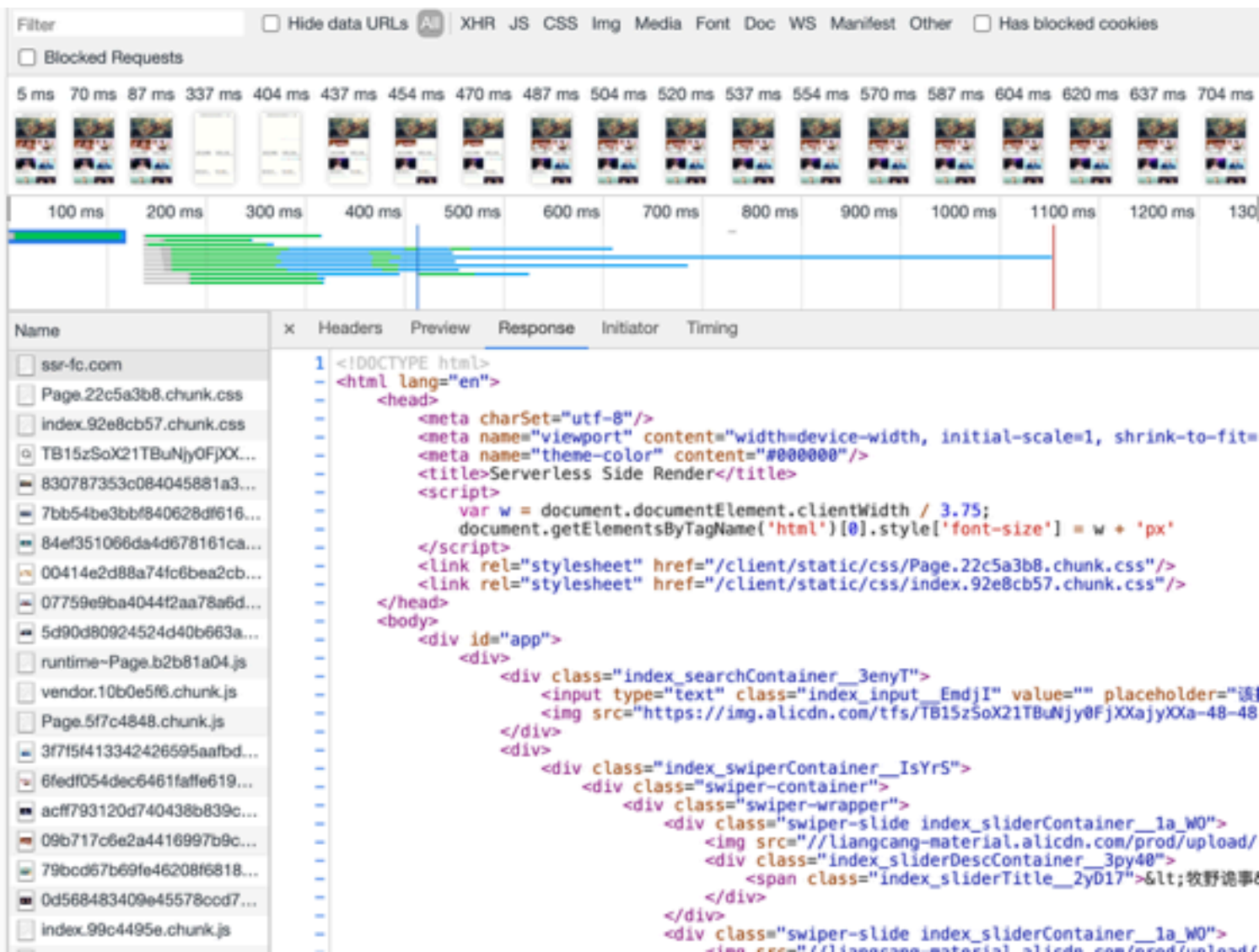
Deploy success
```

一键部署

支持部署到
阿里云、腾讯云，
无需修改任何配置文件
部署文件 size 优化

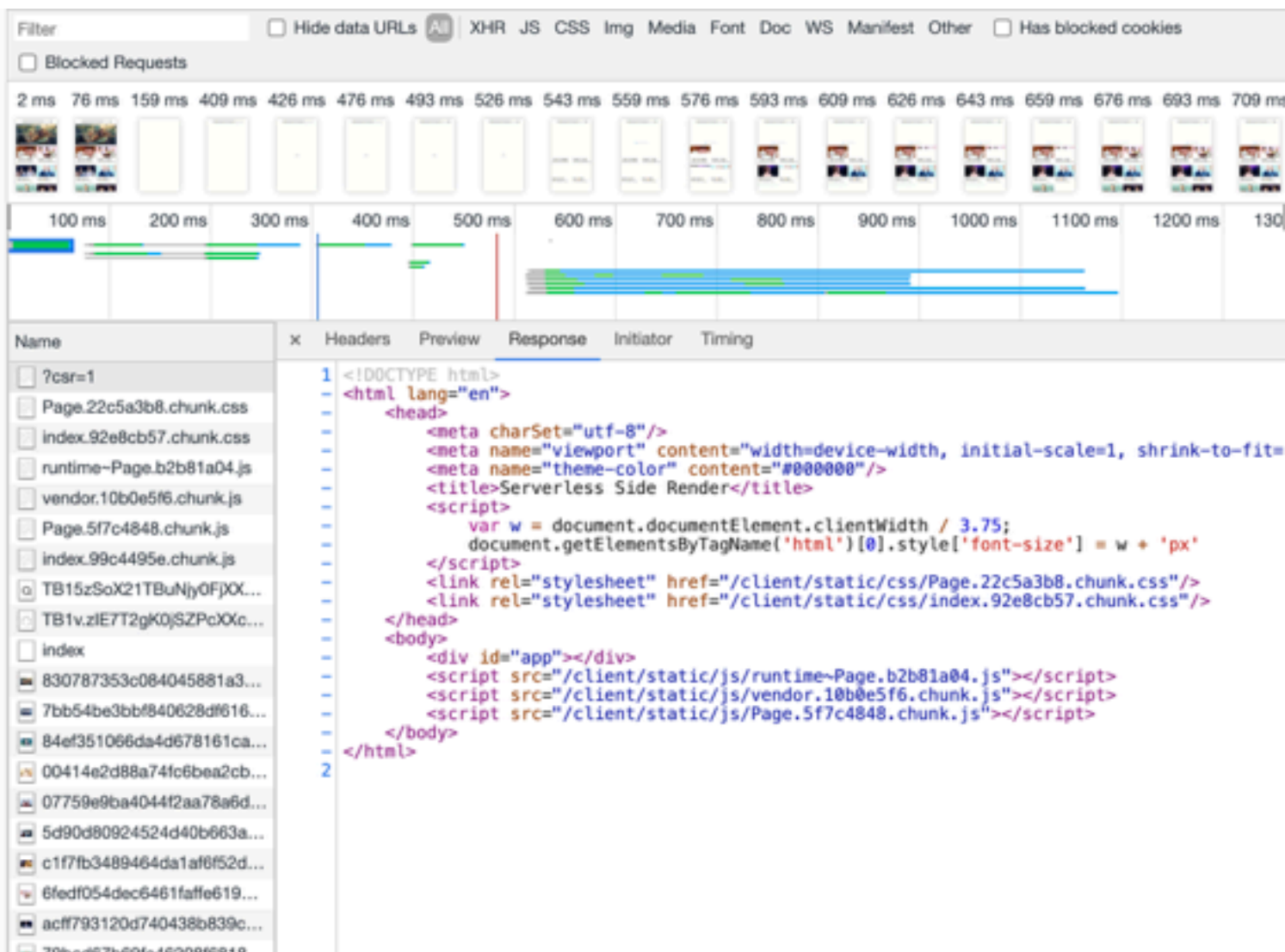
<http://ssr-fc.com/>

<http://ssr-fc.com/>



降级为客户端渲染

<http://ssr-fc.com?csr=1>



完整案例

CASE STUDY

- <http://ssr-fc.com/> 部署到阿里云的 React SSR 应用
- <http://ssr-fc.com?csr=true> 部署到阿里云的 React SSR 应用, 以 CSR 模式访问
- <http://tx.ssr-fc.com> 部署到腾讯云的 React SSR 应用
- <http://tx.ssr-fc.com?csr=true> 部署到腾讯云的 React SSR 应用, 以 CSR 模式访问
- <http://vue.ssr-fc.com> 部署到阿里云的 Vue SSR 应用
- <http://vue.ssr-fc.com?csr=true> 部署到阿里云的 Vue SSR 应用, 以 CSR 模式访问

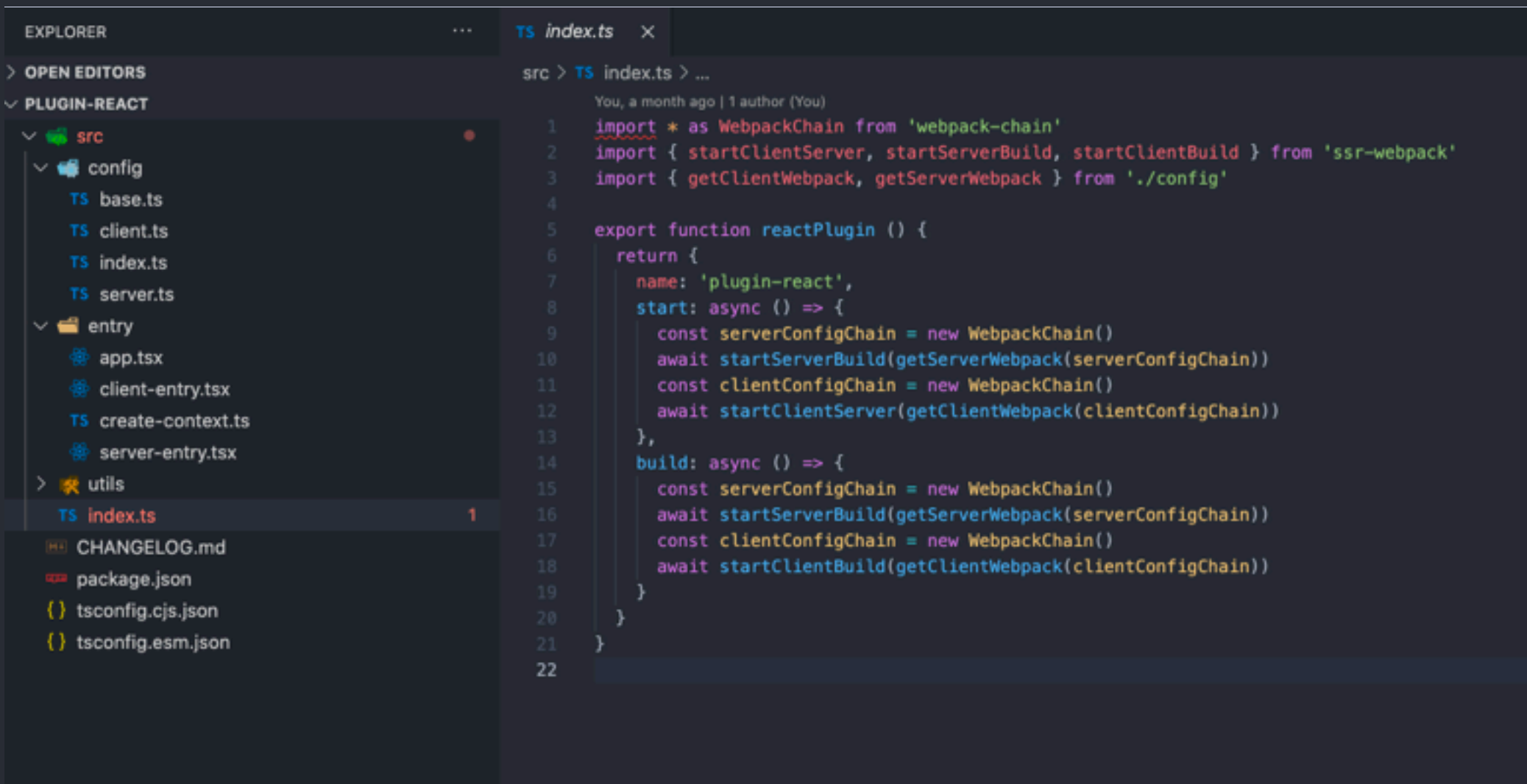
PART 02. 技术实现

Technical realization



客户端插件

ssr-plugin-react

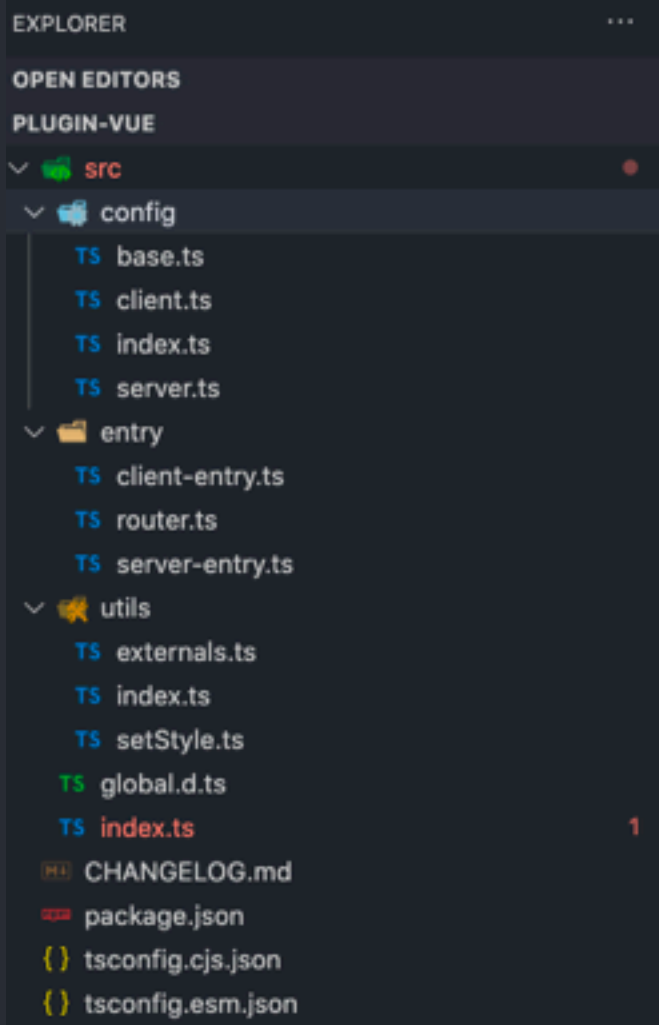


The image shows a VS Code editor interface. On the left, the Explorer sidebar is open, showing a project structure. The 'src' folder is expanded, revealing a 'config' subfolder containing 'base.ts', 'client.ts', 'index.ts', and 'server.ts'. The 'entry' folder contains 'app.tsx', 'client-entry.tsx', 'create-context.ts', and 'server-entry.tsx'. The 'utils' folder is also visible. The 'index.ts' file in the 'config' folder is selected and highlighted in red. Below the Explorer, the file 'CHANGELOG.md' is visible. The main editor area shows the content of 'src > TS index.ts'. The code is a TypeScript file defining a React plugin. It imports 'WebpackChain' from 'webpack-chain' and 'startClientServer', 'startServerBuild', 'startClientBuild' from 'ssr-webpack'. It also imports 'getClientWebpack' and 'getServerWebpack' from './config'. The plugin is defined as a function 'reactPlugin' that returns an object with 'name', 'start', and 'build' properties. The 'start' and 'build' methods are asynchronous functions that use the imported functions to build the client and server bundles.

```
1  import * as WebpackChain from 'webpack-chain'
2  import { startClientServer, startServerBuild, startClientBuild } from 'ssr-webpack'
3  import { getClientWebpack, getServerWebpack } from './config'
4
5  export function reactPlugin () {
6    return {
7      name: 'plugin-react',
8      start: async () => {
9        const serverConfigChain = new WebpackChain()
10       await startServerBuild(getServerWebpack(serverConfigChain))
11       const clientConfigChain = new WebpackChain()
12       await startClientServer(getClientWebpack(clientConfigChain))
13     },
14     build: async () => {
15       const serverConfigChain = new WebpackChain()
16       await startServerBuild(getServerWebpack(serverConfigChain))
17       const clientConfigChain = new WebpackChain()
18       await startClientBuild(getClientWebpack(clientConfigChain))
19     }
20   }
21 }
22
```

客户端插件

ssr-plugin-vue



TS index.ts

```
1  import * as WebpackChain from 'webpack-chain'
2  import { startClientServer, startServerBuild, startClientBuild } from 'ssr-webpack'
3  import { getClientWebpack, getServerWebpack } from './config'
4
5  export function vuePlugin () {
6    return {
7      name: 'plugin-vue',
8      start: async () => {
9        const serverConfigChain = new WebpackChain()
10       await startServerBuild(getServerWebpack(serverConfigChain))
11       const clientConfigChain = new WebpackChain()
12       await startClientServer(getClientWebpack(clientConfigChain))
13     },
14     build: async () => {
15       const serverConfigChain = new WebpackChain()
16       await startServerBuild(getServerWebpack(serverConfigChain))
17       const clientConfigChain = new WebpackChain()
18       await startClientBuild(getClientWebpack(clientConfigChain))
19     }
20   }
21 }
22
```

服务端插件

ssr-plugin-faas

EXPLORER

> OPEN EDITORS

▼ PLUGIN-FAAS

▼ src

TS deploy.ts

TS index.ts 1

TS start.ts

CHANGELOG.md

package.json

tsconfig.cjs.json

tsconfig.esm.json

TS index.ts ×

src > TS index.ts > ...

You, a month ago | 1 author (You)

1 import { start } from './start'

2 import { deploy } from './deploy'

3

4 export function faasPlugin () {

5 return {

6 name: 'plugin-midway-faas',

7 start,

8 deploy

9 }

10 }

11

页面渲染

PRODUCE

生产环境只依赖 core 模块
无任何中间操作性
能等于直接调用框架 API

```
import { FaaSHTTPContext, func, inject, provide,
FunctionHandler } from '@midwayjs/faas'
import { render } from 'ssr-core-react'
import { IApiService, IApiDetailService } from './interface'

interface IFaaSContext extends FaaSHTTPContext {
  apiService: IApiService
  apiDeatilservice: IApiDetailService
}

@provide()
@func('index.handler')
export class Index implements FunctionHandler {
  @inject()
  ctx: IFaaSContext

  @inject('ApiService')
  apiService: IApiService

  @inject('ApiDetailService')
  apiDeatilservice: IApiDetailService

  async handler (): Promise<String> {
    try {
      this.ctx.apiService = this.apiService
      this.ctx.apiDeatilservice = this.apiDeatilservice
      const htmlStr = await render(this.ctx)
      return htmlStr
    } catch (error) {
      const htmlStr = await render(this.ctx, {
        mode: 'csr'
      })
      return htmlStr
    }
  }
}
```

没有模版引擎

```
const Layout = (props: LayoutProps) => {
  // 注: Layout 只会在服务端被渲染, 不要在此运行客户端有关逻辑
  const { state } = useContext(window.STORE_CONTEXT)
  const { injectCss, injectScript } = props.staticList!
  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>Serverless Side Render</title>
        <script dangerouslySetInnerHTML={{ __html: "var w =
document.documentElement.clientWidth /
3.75;document.getElementsByTagName('html')[0].style['font-
size'] = w + 'px'" }} />
        { injectCss }
      </head>
      <body className={styles.body}>
        <div id='app'>{ props.children }</div>
        {
          state && <script dangerouslySetInnerHTML={{
            __html: `window.__USE_SSR__=true;
window.__INITIAL_DATA__=${serialize(state)}`
          }} />
        }
        { injectScript }
      </body>
    </html>
  )
}
```

React

```
<template>
  <!-- 注: Layout 只会在服务端被渲染, 不要在此运行客户端有关逻辑 -->
  <!-- 页面初始化数据注入内容已经过 serialize-javascript 转义 防止
xss -->
  <html>
    <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>Serverless Side Render for Vue</title>
      <!-- 初始化移动端 rem 设置, 如不需要可自行删除 -->
      <slot name="remInitial" />
      <!-- 用于通过配置插入自定义的 script 为了避免影响期望功能这块内容
不做 escape, 为了避免 xss 需要保证插入脚本代码的安全性 -->
      <slot name="customeHeadScript" />
      <slot name="cssInject" />
    </head>
    <body>
      <slot name="children" />
      <slot name="initialData" />
      <slot name="jsInject" />
    </body>
  </html>
</template>

<style lang="less">
@import './index.less';
</style>
```

Vue

It's easy to degrade csr mode

```
h('template', {
  slot: 'children'
}), [
  isCsr ? h('div', {
    // csr 只需渲染一个空的 <div id="app"> 不需要渲染具体的组件也就是
    router-view
    attrs: {
      id: 'app'
    }
  }) : h(App)
],
!isCsr && h('template', {
  slot: 'initialData'
}), [
  h('script', {
    domProps: {
      innerHTML: `window.__USE_SSR__=true; window.__INITIAL_DATA__=${serialize(store.state)}`
    }
  })
]
])
```

Vue

```
return (
  <StaticRouter>
    <Context.Provider value={{ state: fetchData }}>
      <Layout ctx={ctx} config={config} staticList={staticList}>
        {isCsr ? <></> : <Component />}
      </Layout>
    </Context.Provider>
  </StaticRouter>
)
```

React

优秀的代码分割方案

```
> ssr build
```

```
Hash: b2c6c585dafbcca9365a
```

```
Version: webpack 4.46.0
```

```
Time: 1902ms
```

```
Built at: 2021-03-03 4:59:24 |F10: PM|
```

Asset	Size	Chunks	Chunk Names
Page.server.js	16.6 KiB	0 [emitted]	Page
static/css/Page.32d5c616.css	20.8 KiB	0 [emitted] [immutable]	Page

```
Hash: 46ee69a855335595223c
```

```
Version: webpack 4.46.0
```

```
Time: 2847ms
```

```
Built at: 2021-03-03 4:59:27 |F10: PM|
```

Asset	Size	Chunks	Chunk Name
asset-manifest.json	493 bytes	[emitted]	
static/css/Page.9b4ee7a2.chunk.css	874 bytes	0 [emitted] [immutable]	Page
static/css/detail.a409b2e1.chunk.css	3.05 KiB	1 [emitted] [immutable]	detail
static/css/index.e666f74b.chunk.css	14.9 KiB	2 [emitted] [immutable]	index
static/js/Page.7a474484.chunk.js	2.91 KiB	0 [emitted] [immutable]	Page
static/js/detail.761e1db4.chunk.js	5.43 KiB	1 [emitted] [immutable]	detail
static/js/index.c3f70030.chunk.js	145 KiB	2 [emitted] [immutable]	index
static/js/runtime~Page.7a11a3c7.js	3.16 KiB	3 [emitted] [immutable]	runtime~Pa
static/js/vendor.34933e47.chunk.js	124 KiB	4 [emitted] [immutable]	vendor

优秀的代码分割方案

只对客户端 bundle 进行分块，实现成本极低



```
module.exports = [{
  layout: require('@components/layout/index.tsx').default,
  fetch: require('@pages/detail/fetch.ts').default,
  webpackChunkName: 'detail',
  path: '/detail/:id',
  component: __isBrowser__ ? require('react-loadable')({
    loader: () => import(/* webpackChunkName: "detail" */ '@pages/detail/render$id.tsx'),
    loading: function Loading () {
      return require('react').createElement('div')
    }
  }) : require('@pages/detail/render$id.tsx').default
}, {
  layout: require('@components/layout/index.tsx').default,
  fetch: require('@pages/index/fetch.ts').default,
  webpackChunkName: 'index',
  path: '/',
  component: __isBrowser__ ? require('react-loadable')({
    loader: () => import(/* webpackChunkName: "index" */ '@pages/index/render.tsx'),
    loading: function Loading () {
      return require('react').createElement('div')
    }
  }) : require('@pages/index/render.tsx').default
}]
```

优秀的代码分割方案

只对客户端 bundle 进行分块，实现成本极低



```
module.exports = [{
  layout: require('@components/layout/index.vue').default,
  App: require('@components/layout/App.vue').default,
  fetch: require('@pages/detail/fetch.ts').default,
  webpackChunkName: 'detail',
  path: '/detail/:id',
  component: __isBrowser__ ? () => import(/* webpackChunkName: "detail" */ '@pages/detail/render$id.vue') :
require('@pages/detail/render$id.vue').default
}, {
  layout: require('@components/layout/index.vue').default,
  App: require('@components/layout/App.vue').default,
  fetch: require('@pages/index/fetch.ts').default,
  webpackChunkName: 'index',
  path: '/',
  component: __isBrowser__ ? () => import(/* webpackChunkName: "index" */ '@pages/index/render.vue') :
require('@pages/index/render.vue').default
}]
```

框架对比

- 与 Serverless 高度集成
- 轻量 源代码 2400 行 vs next.js 18w 行 vs nuxt.js 2w行
- 接地气，内置 antd vant 相关配置，可直接安装使用
- 一键降级
- 没有模版引擎
- 风格统一，框架之间任意切换
- 默认示例丰富，绝非 Hello World 级别
- 高性能，没有 runInNewContext

PART 03. 未来展望

Future outlook



In the future

🚀 表示已经实现的功能

里程碑	状态
支持任意服务端框架与任意前端框架的组合使用。(Serverless/Midway/Nestjs) + (React/Vue)	🚀
最小而美的实现服务端渲染功能	🚀
针对Serverless 场景对代码包的大小的严格限制, 将生产环境的代码包大小做到极致	🚀
同时支持约定式前端路由和声明式前端路由	🚀
React 场景下 All in JSX, Vue 场景 All in template, 没有传统模版引擎, 所有部分包括 html layout 布局皆使用 JSX/Vue 来编写生成	🚀
渲染模式切换: 服务端渲染一键降级为客户端渲染	🚀
统一不同框架服务端客户端的数据获取方式, 做到高度复用	🚀
类型友好, 全面拥抱 TS	🚀
支持无缝接入 antd vant 无需修改任何配置	🚀
支持使用 less 作为 css 预处理器	🚀
实现 React/Vue SSR 场景下的 优秀代码分割方案 首屏性能做到极致	🚀
React 场景下使用 useContext + useReducer 实现极简的 跨组件通信 方案, 摒弃传统的 redux/dva 等数据管理方案	🚀
支持在阿里云 云平台 创建使用	🚀
ssr deploy 一键部署到 阿里云 平台	🚀
ssr deploy --tencent 无需修改任何配置一键部署到 腾讯云 平台	🚀
支持 vite + vue3 在 SSR 场景下的组合使用	

结合 Vite 结合 Prisma , GraphQL?

2021

YUANG

THANKS

感谢观看



<https://github.com/ykfe/ssr/>

TECHNOLOGY SHARING