服务端渲染SSR

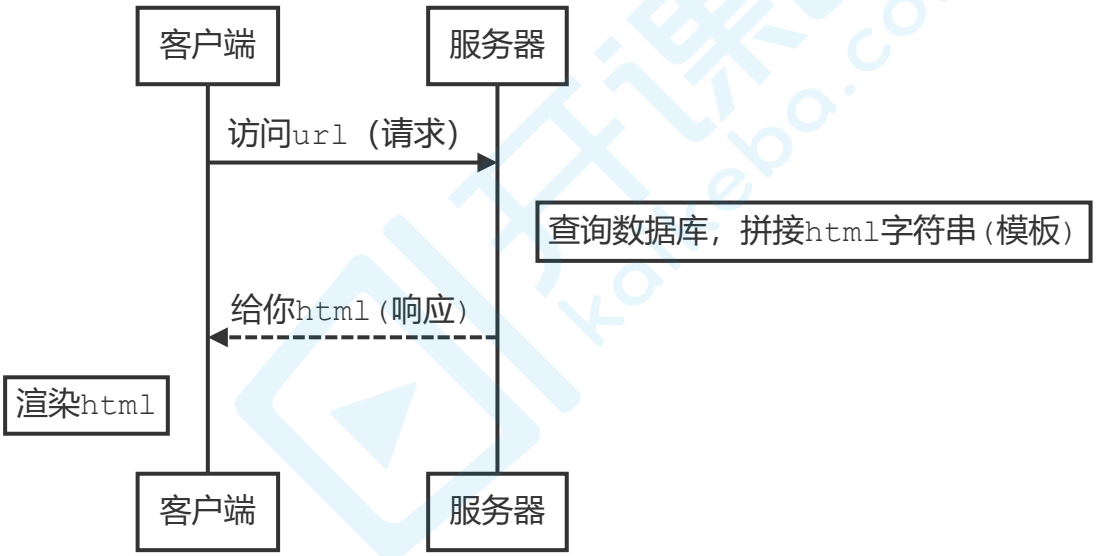
资源

1. vue ssr
2. nuxt.js

知识点

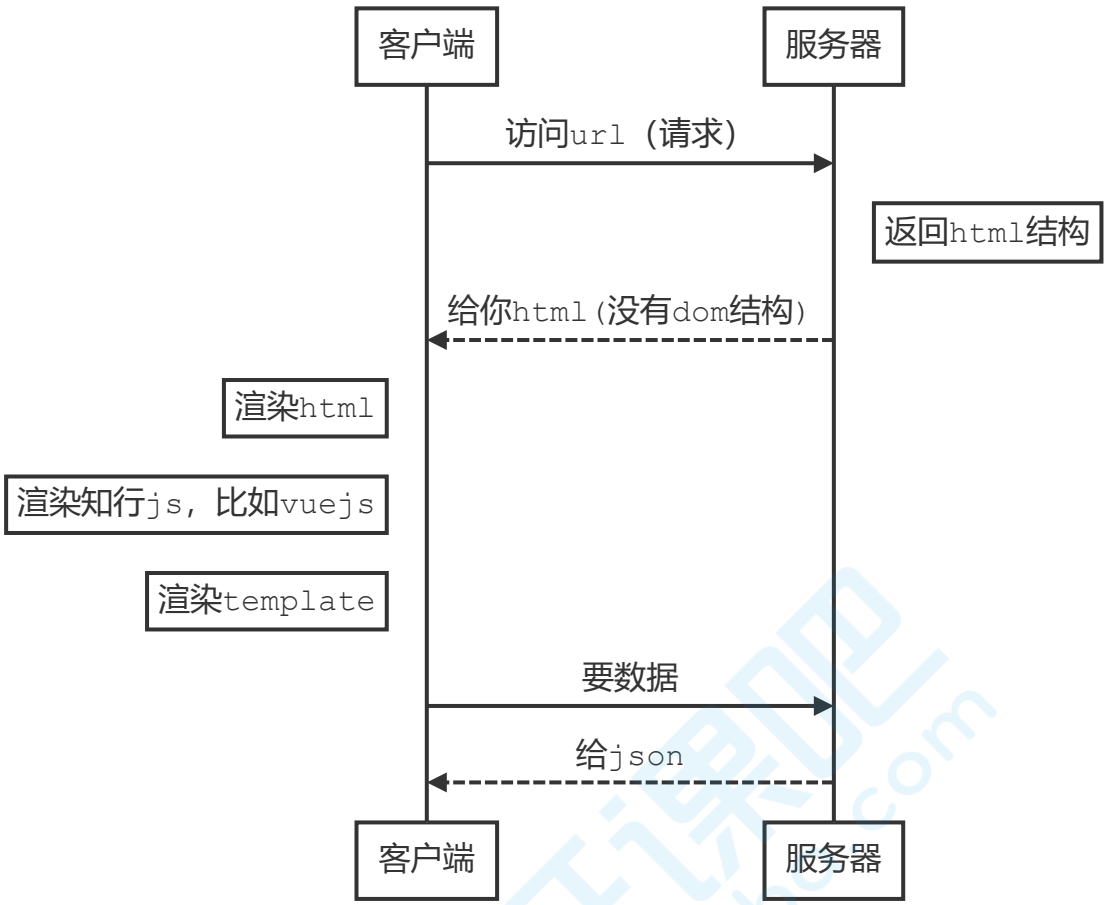
CSR VS SSR

传统的web开发



SPA时代

到了vue，react时代，单页应用优秀的用户体验，逐渐成为了主流，页面整体是JS渲染出来的，称之为客户端渲染CSR

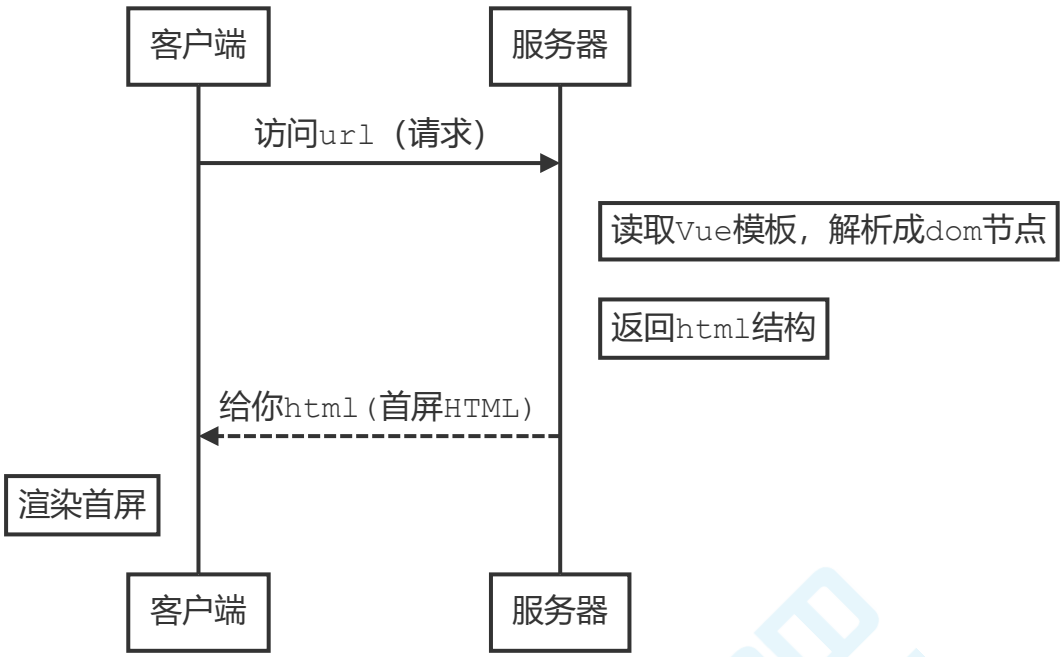


单页应用的两个问题

1. **首屏渲染等待时长**：必须得等js加载完毕，并且执行完毕，才能渲染出首屏
2. **seo不友好**：爬虫只能拿到一个div，认为页面是空的，不利于seo

SSR

为了解决这两个问题，出现了SSR解决方案，后端渲染出完整的首屏的dom结构返回，前端拿到的内容带上首屏，后续的页面操作，再用单页的路由跳转和渲染，称之为服务端渲染 (server side render)



ssr也有弊端：

1.学习难度较高

2.第三方库有时候会有问题

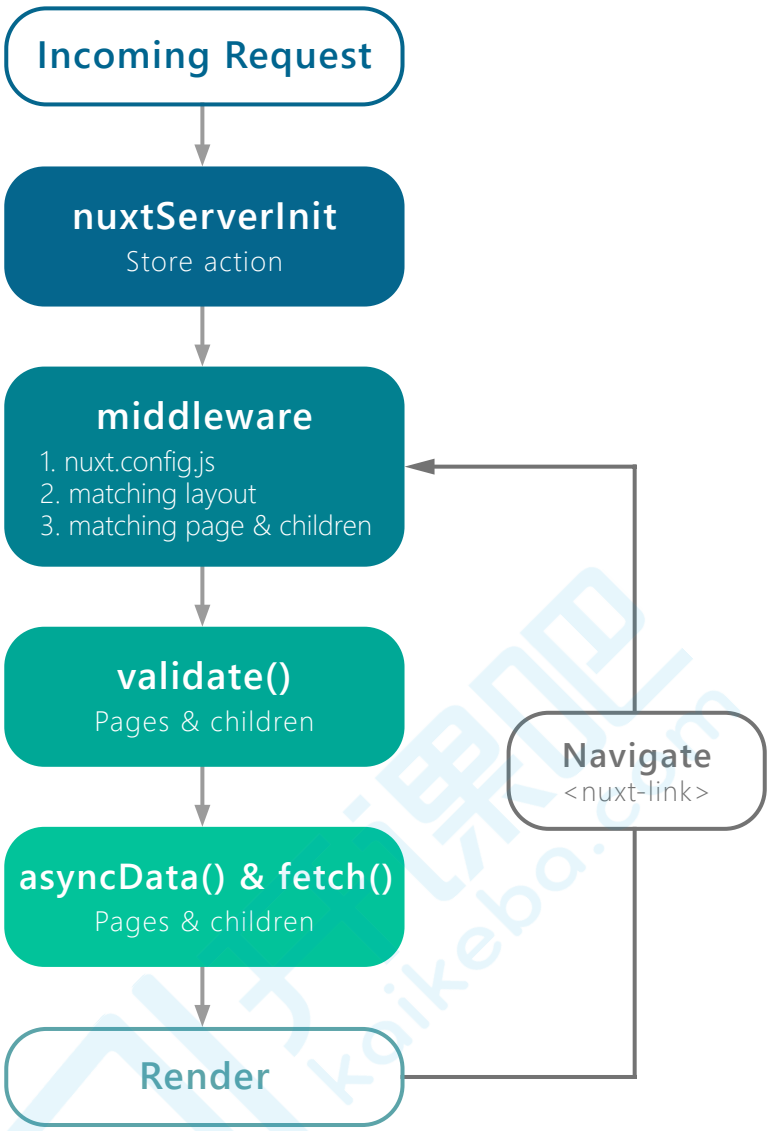
3.增加服务器压力

ssr体验：nuxt.js

Nuxt.js 是一个基于 Vue.js 的**通用应用框架**

nuxt渲染流程

一个完整的服务器请求到渲染的流程



nuxt安装

运行 create-nuxt-app

```
npx create-nuxt-app <项目名>
```

路由

路由生成

pages目录中所有 `*.vue` 文件自动生成应用的路由配置，新建：

- pages/detail.vue 详情页
- pages/cart.vue 购物车页
- pages/admin.vue 商品管理页

- pages/login.vue 登录页

> 查看.nuxt/router.js验证生成路由

导航

添加路由导航，layouts/default.vue

```
<nav>
    <nuxt-link to="/">首页</nuxt-link>
    <!--别名：n-link，NLink，NuxtLink-->
    <NLink to="/admin">管理</NLink>
    <n-link to="/cart">购物车</n-link>
</nav>
```

> 禁用预加载：`<n-link no-prefetch>page not pre-fetched</n-link>`

商品列表，index.vue

```
<template>
  <div>
    <h2>商品列表</h2>
    <ul>
      <li v-for="good in goods" :key="good.id" >
        <nuxt-link :to="`/detail/${good.id}`">
          <span>{{good.text}}</span>
          <span>￥{{good.price}}</span>
          <button @click.prevent="addCart(good)">加购物车</button>
        </nuxt-link>
      </li>
    </ul>
  </div>
</template>

<script>
export default {
  data() {
    return { goods: [
      {id:1, text:'Web全栈架构师',price:8999},
      {id:2, text:'Python全栈架构师',price:8999},
    ] }
  },
  methods: {
    addCart(){}
  }
};
</script>
```

动态路由

以下划线作为前缀的 .vue文件 或 目录会被定义为动态路由，如下面文件结构

```
pages/
--| detail/
----| _id.vue
```

> 如果detail/里面不存在index.vue，:id将被作为可选参数

嵌套路由

创建内嵌子路由，你需要添加一个 .vue 文件，同时添加一个**与该文件同名**的目录用来存放子视图组件。

构造文件结构如下：

```
pages/
--| index/
----| _id.vue
--| index.vue
```

测试代码，main.vue

```
<template>
    <ul>
      <li v-for="good in goods" :key="good.id" >
        <!--修改目标路由名称为index-id-->
        <nuxt-link :to="{name: 'index-id', params: {id: good.id}}">
          <span>{{good.text}}</span>
          <span>￥{{good.price}}</span>
          <button @click.prevent="addCart(good)">加购物车</button>
        </nuxt-link>
      </li>
    </ul>
    <!--添加路由视图-->
    <nuxt-child></nuxt-child>
</template>
```

视图

下图展示了Nuxt.js 如何为指定的路由配置数据和视图

默认布局

查看 `layouts/default.vue`

```
<template>
  <nuxt/>
</template>
```

自定义布局

创建空白布局页面 `layouts/blank.vue`，用于login.vue

```
<template>
  <div>
    <nuxt />
  </div>
</template>
```

页面 `pages/login.vue` 使用自定义布局：

```
export default {
    layout: 'blank'
}
```

自定义错误页面

创建layouts/error.vue

```
<template>
  <div class="container">
    <h1 v-if="error.statusCode === 404">页面不存在</h1>
    <h1 v-else>应用发生错误异常</h1>
    <nuxt-link to="/">首 页</nuxt-link>
  </div>
</template>

<script>
export default {
  props: ['error']
}
</script>
```

页面

页面组件就是 Vue 组件，只不过 Nuxt.js 为这些组件添加了一些特殊的配置项

给首页添加标题和meta等，index.vue

```
export default {
  head() {
    return {
      title: "课程列表",
      meta: [{ name: "description", hid: "description", content: "set page meta" }],
      link: [{ rel: "favicon", href: "favicon.ico" }],
    };
  },
};
```

异步数据获取

`asyncData` 方法使得我们可以在**设置组件数据之前异步获取或处理数据。**

范例：获取商品数据

接口准备

- 安装依赖：`npm i koa-router koa-bodyparser -S`
- 创建接口文件，server/api.js

```
const Koa = require('koa');
const app = new Koa();
const bodyparser = require("koa-bodyparser");
const router = require("koa-router")({ prefix: "/api" });

// 设置cookie加密秘钥
app.keys = ["some secret", "another secret"];

const goods = [
  { id: 1, text: "Web全栈架构师", price: 1000 },
  { id: 2, text: "Python架构师", price: 1000 }
];

router.get("/goods", ctx => {
  ctx.body = {
    ok: 1,
    goods
  };
});

router.get("/detail", ctx => {
  ctx.body = {
    ok: 1,
    data: goods.find(good => good.id == ctx.query.id)
  };
});

router.post("/login", ctx => {
```

```
  const user = ctx.request.body;
  if (user.username === "jerry" && user.password === "123") {
    // 将token存入cookie
    const token = 'a mock token';
    ctx.cookies.set('token', token);
    ctx.body = { ok: 1, token };
  } else {
    ctx.body = { ok: 0 };
  }
});

// 解析post数据并注册路由
app.use(bodyparser());
app.use(router.routes());

app.listen(8080, () => console.log('api服务已启动'))
```

整合axios

安装@nuxt/axios模块：`npm install @nuxtjs/axios -S`

配置：nuxt.config.js

```
modules: [
    '@nuxtjs/axios',
],
axios: {
    proxy: true
},
proxy: {
    "/api": "http://localhost:8080"
},
```

> 注意配置重启生效

测试代码：获取商品列表，index.vue

```
<script>
export default {
    async asyncData({ $axios, error }) {
        const {ok, goods} = await $axios.$get("/api/goods");
        if (ok) {
            return { goods };
        }
        // 错误处理
        error({ statusCode: 400, message: "数据查询失败" });
    },
}
</script>
```

测试代码：获取商品详情，/index/_id.vue

```
<template>
  <div>
    <pre v-if="goodInfo">{{goodInfo}}</pre>
  </div>
</template>

<script>
export default {
  async asyncData({ $axios, params, error }) {
    if (params.id) {
      // asyncData中不能使用this获取组件实例
      // 但是可以通过上下文获取相关数据
      const { data: goodInfo } = await $axios.$get("/api/detail", { params });

      if (goodInfo) {
        return { goodInfo };
      }
      error({ statusCode: 400, message: "商品详情查询失败" });
    } else {
      return { goodInfo: null };
    }
  }
};
</script>
```

中间件

中间件会在一个页面或一组页面渲染之前运行我们定义的函数，常用于权限控制、校验等任务。

范例代码：管理员页面保护，创建middleware/auth.js

```
export default function({ route, redirect, store }) {
  // 上下文中通过store访问vuex中的全局状态
  // 通过vuex中令牌存在与否判断是否登录
  if (!store.state.user.token) {
    redirect("/login?redirect="+route.path);
  }
}
```

注册中间件，admin.vue

```
<script>
    export default {
        middleware: ['auth']
    }
</script>
```

> 运行报错，因为不存在user模块

状态管理 vuex

应用根目录下如果存在 `store` 目录，Nuxt.js将启用vuex状态树。定义各状态树时具名导出state, mutations, getters, actions即可。

范例：用户登录及登录状态保存，创建store/user.js

```
export const state = () => ({
  token: ''
});

export const mutations = {
  init(state, token) {
    state.token = token;
  }
};

export const getters = {
  isLogin(state) {
    return !!state.token;
  }
};

export const actions = {
  login({ commit, getters }, u) {
    return this.$login(u).then(({ token }) => {
      if (token) {
        commit("SET_TOKEN", token);
      }
      return getters.isLogin;
    });
  }
};
```

插件

Nuxt.js会在运行应用之前执行插件函数，需要引入或设置Vue插件、自定义模块和第三方模块时特别有用。

范例代码：接口注入，利用插件机制将服务接口注入组件实例、store实例中，创建plugins/api-inject.js

```
export default ({ $axios }, inject) => {
  inject("login", user => {
    return $axios.$post("/api/login", user);
  });
};
```

注册插件，nuxt.config.js

```
plugins: [
    "@/plugins/api-inject"
],
```

登录页面逻辑，login.vue

```
<template>
  <div>
    <h2>用户登录</h2>
    <el-input v-model="user.username"></el-input>
    <el-input type="password" v-model="user.password"></el-input>
    <el-button @click="onLogin">登录</el-button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      user: {
        username: "",
        password: ""
      }
    };
  },
  methods: {
    onLogin() {
      this.$store.dispatch("user/login", this.user).then(ok=>{
          if (ok) {
            const redirect = this.$route.query.redirect || '/'
            this.$router.push(redirect);
          }
      });
    }
  }
};
</script>
```

范例：添加请求拦截器附加token，创建plugins/interceptor.js

```
export default function({ $axios, store }) {
  $axios.onRequest(config => {
    if (store.state.user.token) {
      config.headers.Authorization = "Bearer " + store.state.user.token;
    }
    return config;
  });
}
```

注册插件，nuxt.config.js

```
plugins: ["@/plugins/interceptor"]
```

nuxtServerInit

通过在store的根模块中定义 `nuxtServerInit` 方法，将服务端的一些数据传到客户端。

范例：登录状态初始化，store/index.js

```
export const actions = {
  nuxtServerInit({ commit }, { app }) {
    const token = app.$cookies.get("token");
    if (token) {
      console.log("nuxtServerInit: token:"+token);
      commit("user/SET_TOKEN", token);
    }
  }
};
```

> - 安装依赖模块：cookie-universal-nuxt
>
> ```
> npm i -S cookie-universal-nuxt
> ```
>
> 注册, nuxt.config.js
>
> ```
> modules: ["cookie-universal-nuxt"],
> ```

发布部署

服务端渲染应用部署

先进行编译构建，然后再启动 Nuxt 服务

```
npm run build
npm start
```

静态应用部署

Nuxt.js 可依据路由配置将应用静态化，使得我们可以将应用部署至任何一个静态站点主机服务商。

```
npm run generate
```

Vue SSR实战

新建工程

```
vue create ssr
```

安装依赖

```
npm install vue-server-renderer express --save
```

启动脚本

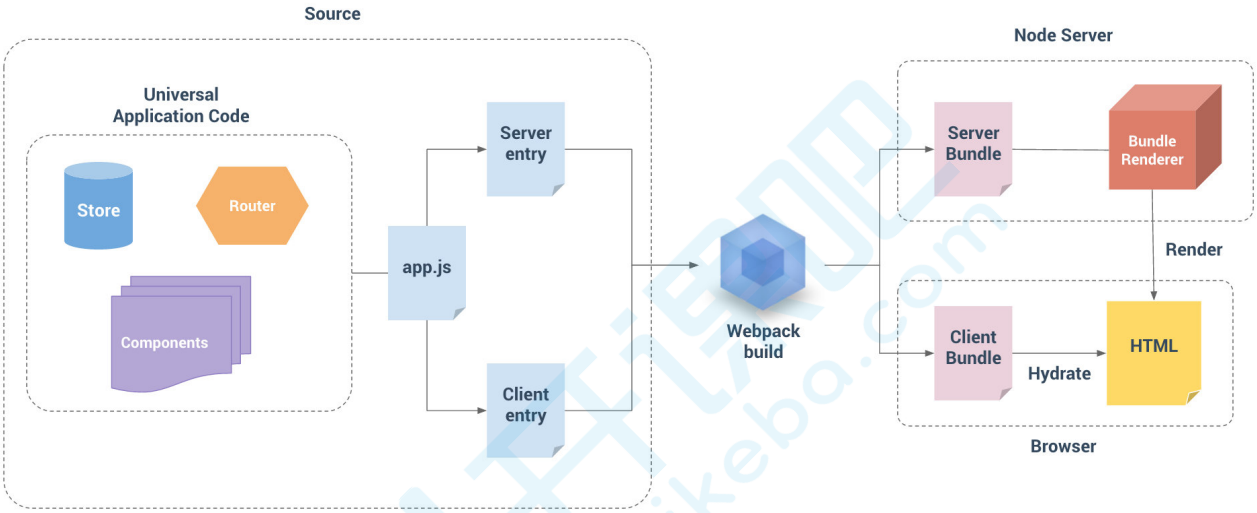
```
const express = require('express')
const Vue = require('vue')

const app = express()
const renderer = require('vue-server-renderer').createRenderer()
// 页面
const page = new Vue({
    data:{
        name:'开课吧',
        count:1
    },
    template:`
        <div >
            <h1>{{name}}</h1>
            <h1>{{count}}</h1>
        </div>
    `
})

app.get('/',async function(req,res){
    // renderToString可以将vue实例转换为html字符串
    const html = await renderer.renderToString(page)
    res.send(html)
})
```

```
app.listen(3000, ()=>{
    console.log('启动成功')
})
```

image-20190319152050276

构建步骤

webpack根据执行环境生成server bundle和client bundle



路由 Vue-router

单页应用的页面路由，都是前端控制，后端只负责提供数据

一个简单的单页应用，使用vue-router,为了方便前后端公用路由数据，我们新建router.js 对外暴露createRouter

```
npm i vue-router -s
```

```
// router.js
import Vue from 'vue'
import Router from 'vue-router'
import Index from './components/Index'
import Kkb from './components/Kkb'

Vue.use(Router)

export function createRouter () {
  return new Router({
    routes: [
```

```
        {path:"/",component:Index },
        {path:"/kkb",component:Kkb },
      // ...
    ]
  })
}
```

```
// src/components/Index.vue
<template>
    <div>
        <h1>hi {{name}}</h1>
    </div>
</template>

<script>
export default {
    data(){
        return {
            name:'首页'
        }
    }
}
</script>
```

```
// src/components/Kkb.vue
<template>

    <div>
        <h1>hi {{name}}</h1>
    </div>
</template>

<script>
export default {
    data(){
        return {
            name:'开课吧'
        }
    }
}
</script>
```

```
// src/App.vue
<template>
  <div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <ul>
      <li>
        <router-link to="/">首页</router-link>
      </li>
      <li>
```

```
        <router-link to="/kkb">开课吧</router-link>
      </li>
    </ul>
    <router-view></router-view>
  </div>
</template>

<script>
export default {
  name: 'app',
}
</script>
```

csr 和ssr统一入口

```
// src/createapp.js
import Vue from 'vue'
import App from './App.vue'
import { createRouter } from './router'

export function createApp (context) {
    const router = createRouter()
    const app = new Vue({
        router,
        context,
        render: h => h(App)
    })
    return { app, router }
}
```

csr的入口文件main.js

```
// src/main.js
import { createApp } from './createapp'

const { app, router } = createApp()
router.onReady(() => {
    app.$mount('#app')
})
```

ssr的入口文件entry-server.js

```
// src/entry-server.js
import { createApp } from './src/createapp'

export default context => {
    // 我们返回一个 Promise
    // 确保路由或组件准备就绪
    return new Promise((resolve, reject) => {
        const { app, router } = createApp(context)
```

```
        // 跳转到首屏的地址
        router.push(context.url)
        router.onReady(() => {
            resolve(app)
        }, reject)
    })
}
```

> 下面引入webpack

后端加入webpack

过一下配置和代码

```
npm install cross-env vue-server-renderer webpack-node-externals lodash.merge --save
```

具体配置

```
// vue.config.js
const VueSSRServerPlugin = require("vue-server-renderer/server-plugin");
const VueSSRClientPlugin = require("vue-server-renderer/client-plugin");
const nodeExternals = require("webpack-node-externals");
const merge = require("lodash.merge");
const TARGET_NODE = process.env.WEBPACK_TARGET === "node";
const target = TARGET_NODE ? "server" : "client";


module.exports = {
  css: {
    extract: false
  },
  configureWebpack: () => ({
    // 将 entry 指向应用程序的 server / client 文件
    entry: TARGET_NODE ?`./src/entry-${target}.js`:'./src/main.js',
    // 对 bundle renderer 提供 source map 支持
    devtool: 'source-map',
    target: TARGET_NODE ? "node" : "web",
    node: TARGET_NODE ? undefined : false,
    output: {
      libraryTarget: TARGET_NODE ? "commonjs2" : undefined
    },
    // https://webpack.js.org/configuration/externals/#function
    // https://github.com/liady/webpack-node-externals
    // 外置化应用程序依赖模块。可以使服务器构建速度更快，
    // 并生成较小的 bundle 文件。
    externals: TARGET_NODE
      ? nodeExternals({
          // 不要外置化 webpack 需要处理的依赖模块。
          // 你可以在这里添加更多的文件类型。例如，未处理 *.vue 原始文件，
          // 你还应该将修改 `global`（例如 polyfill）的依赖模块列入白名单
          whitelist: [/\.css$/]
```

```
        })
      : undefined,
    optimization: {
          splitChunks: undefined
    },
    plugins: [TARGET_NODE ? new VueSSRServerPlugin() : new VueSSRClientPlugin()]
  }),
  chainWebpack: config => {
    config.module
      .rule("vue")
      .use("vue-loader")
      .tap(options => {
        merge(options, {
          optimizeSSR: false
        });
      });
  }
};
```

服务器启动文件，server.js

```
const fs = require("fs");
const express = require('express')
const app = express()

// 开放dist目录
app.use(express.static('./dist'))

// 第 2 步：获得一个createBundleRenderer
const { createBundleRenderer } = require("vue-server-renderer");
const bundle = require("./dist/vue-ssr-server-bundle.json");
const clientManifest = require("./dist/vue-ssr-client-manifest.json");

const renderer = createBundleRenderer(bundle, {
  runInNewContext: false,
  template: fs.readFileSync("./src/index.temp.html", "utf-8"),
  clientManifest: clientManifest
});

function renderToString(context) {
  return new Promise((resolve, reject) => {
    renderer.renderToString(context, (err, html) => {
      resolve(html);
    });
  });
}

app.get('*',async (req,res)=>{
  console.log(req.url,123)
  const context = {
    title:'ssr test',
    url:req.url
  }
```

```
  const html = await renderToString(context);
  res.send(html)
})

const port = 3001;
app.listen(port, function() {
  console.log(`server started at localhost:${port}`);
});
```

宿主文件

```
// src/index.temp.html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
    <!--vue-ssr-outlet-->
  </body>
</html>
```

脚本配置

```
// package.json
  "scripts": {
    "serve": "vue-cli-service serve",
    "build:client": "vue-cli-service build",
    "build:server": "cross-env WEBPACK_TARGET=node vue-cli-service build --mode
server",
    "build": "npm run build:server && mv dist/vue-ssr-server-bundle.json bundle && npm
run build:client && mv bundle dist/vue-ssr-server-bundle.json",
    "lint": "vue-cli-service lint"
  },
```

整合Vuex

安装vuex

```
npm install --save vuex
```

store.js

```
import Vue from 'vue'
import Vuex from 'vuex'
```

```
Vue.use(Vuex)

export function createStore () {
  return new Vuex.Store({
    state: {
        count:108
    },
    mutations: {

    },
    actions: {

    }
  })
}
```

挂载store，createapp.js

```
import Vue from 'vue'
import App from './App.vue'
import { createRouter } from './router'
import { createStore } from './store'

export function createApp (context) {
    const router = createRouter()
    const store = createStore()
    const app = new Vue({
        router,
        store,
        context,
        render: h => h(App)
    })
    return { app, router }
}
```

使用

```
// src/components/Kkb.vue
<h2>num:{{$store.state.count}}</h2>
```

ssr记得先跑 npm run build