服务端渲染SSR

1. 课前准备

1. ssr概念
2. vuessr
3. nuxt.js

2.课堂主题

1. SSR 服务端渲染流程图
2. SSR优势
3. Vue SSR实战
4. vue SSR框架 nuxt.js

3.课堂目标

学会VUE服务端渲染

4.知识点

nuxt.js

> https://zh.nuxtjs.org/guide

自己折腾太麻烦拉，还好有nuxt，内置vuex vue-router，ssr最佳实践框架

新建目录 `cnpm install nuxt --save`,新增script

```
  "scripts": {
    "dev": "nuxt"
  }
```

nuxt遵循约定优于配置，我们新建pages目录，就是页面了

`mkdir pages`

新建pages/index.vue

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

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

npm run dev

nuxt约束

文件夹约束

1. assets 静态资源
2. components 组件
3. layouts 布局
4. middleware 中间件
5. store vuex
6. nuxt.config.js 个性化配置

路由

```
pages/
--| user/
-----| index.vue
-----| one.vue
--| index.vue
```

那么，Nuxt.js 自动生成的路由配置如下：

```
router: {
  routes: [
    {
      name: 'index',
      path: '/',
      component: 'pages/index.vue'
    },
    {
      name: 'user',
      path: '/user',
      component: 'pages/user/index.vue'
    },
    {
      name: 'user-one',
      path: '/user/one',
      component: 'pages/user/one.vue'
    }
  ]
}
```

异步数据获取

asyncData会在页面加载前调用，作为数据获取 支持async+ await 返回值会merge到data里

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

                <div v-for="item in data" :key="item.id">
            {{item.name}}
            </div>
    </div>
</template>
<script>

import axios from "axios";
export default {
   async asyncData() {
    const res = await axios.get("http://taro.josephxia.com/api/top");

    console.log("res", res.data);
    return { data: res.data.data };
  },
    data(){
        return {
            name:'开课吧'
```

```
        }
    }
}
</script>
```

nuxt+ vuex

store目录就是存储vuex数据的地方

```
// store/index.js
export const state = () => ({
    counter: 0
})

export const mutations = {
    increment(state) {
        state.counter++
    }
}
```

```
<template>
    <div>
        <h1>hi {{name}}</h1>
        <h1>hi {{title}}</h1>
        <h1>hi {{counter}}</h1>
        <button @click="add">添加</button> // Add.....
    </div>
</template>
<script>

const delay = ()=> new Promise((resolve,reject)=>{
    setTimeout(()=>{
        resolve({title: '异步开课吧数据'})
    },1000)
})
import axios from 'axios'
export default {
    async asyncData(){
        return await delay()
    },

    data(){
        return {
            name:'开课吧'
        }
    },
  // Add.....
    methods:{
```

```
        async add(){
            this.$store.commit('increment')
        }
    },
  // Add.....
    computed:{
        counter(){
            return this.$store.state.counter
        }
    },
}
</script>
```

Axios前后端

```
npm i axios -s
```

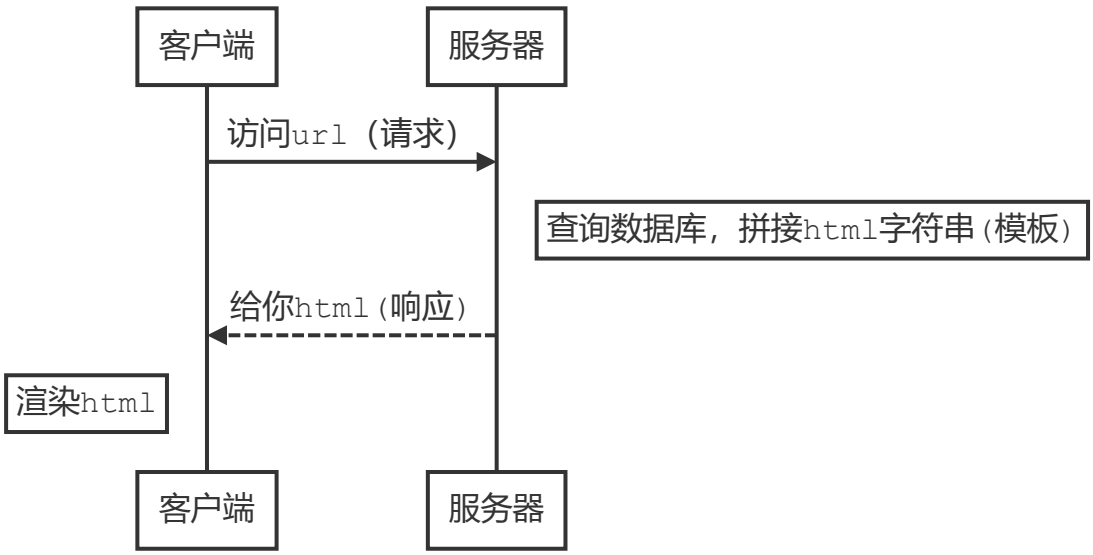
```
        // 服务器端运行
        async asyncData(){
        const res = await axios.get('http://taro.josephxia.com/api/top')
        console.log('res',res.data)
        return await delay()
    },
        // 浏览器端运行
            async add(){
            const res = await axios.get('http://taro.josephxia.com/api/top')
            console.log('res',res.data)
            this.$store.commit('increment')
        }
```

新建工程

```
vue create ssr
```

CSR(client side render) & SSR (server side render)

传统的web开发体验



express体验

```
// npm i express -s
```

```
// server.js
const express = require('express')
const app = express()

app.get('/',function(req,res){
    res.send(`
        <html>
            <div>
                <div id="app">
                    <h1>开课吧</h1>
                    <p class="demo">开课吧还不错</p>
                </div>
            </body>
        </html>
    `)
})

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

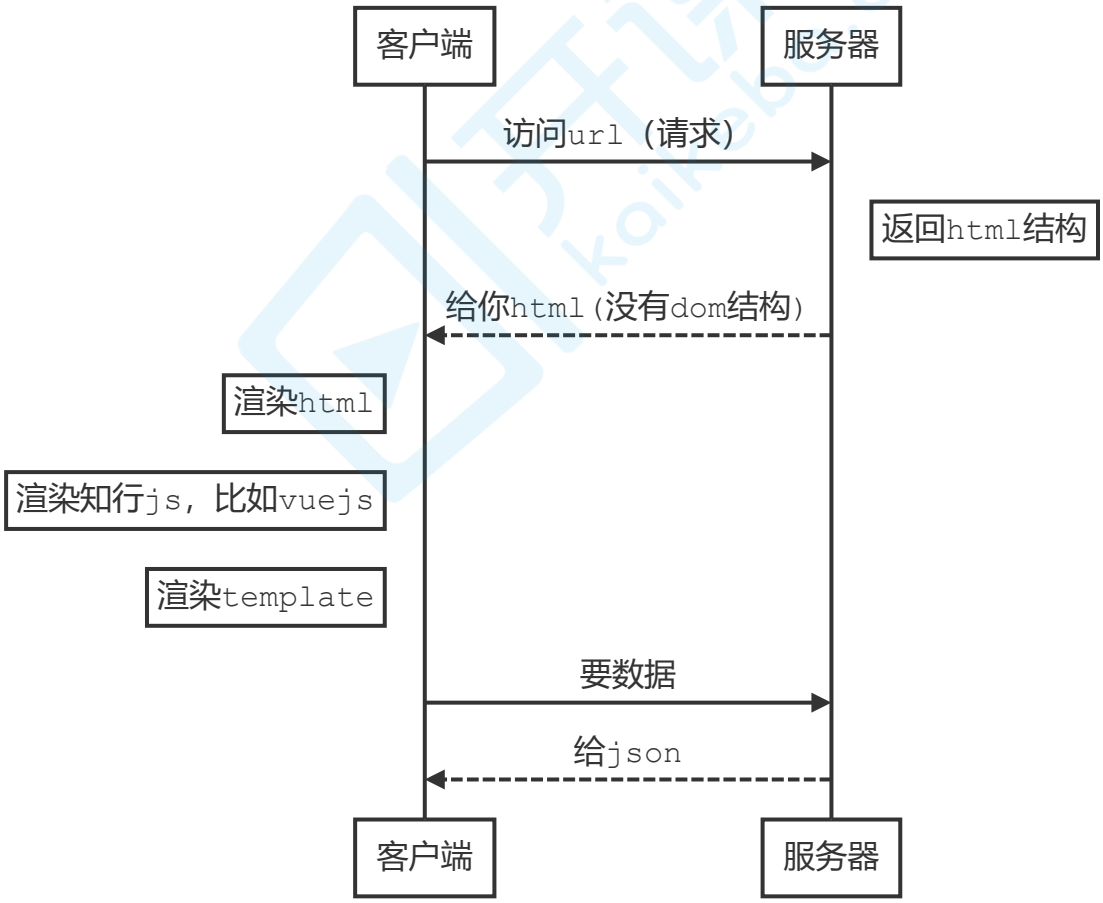
打开页面 查看源码

```
<html>
    <div>
        <div id="app">
            <h1>开课吧</h1>
            <p class="demo">开课吧还不错</p>
        </div>
    </body>
</html>
```

浏览器拿到的，就是全部的dom结构

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



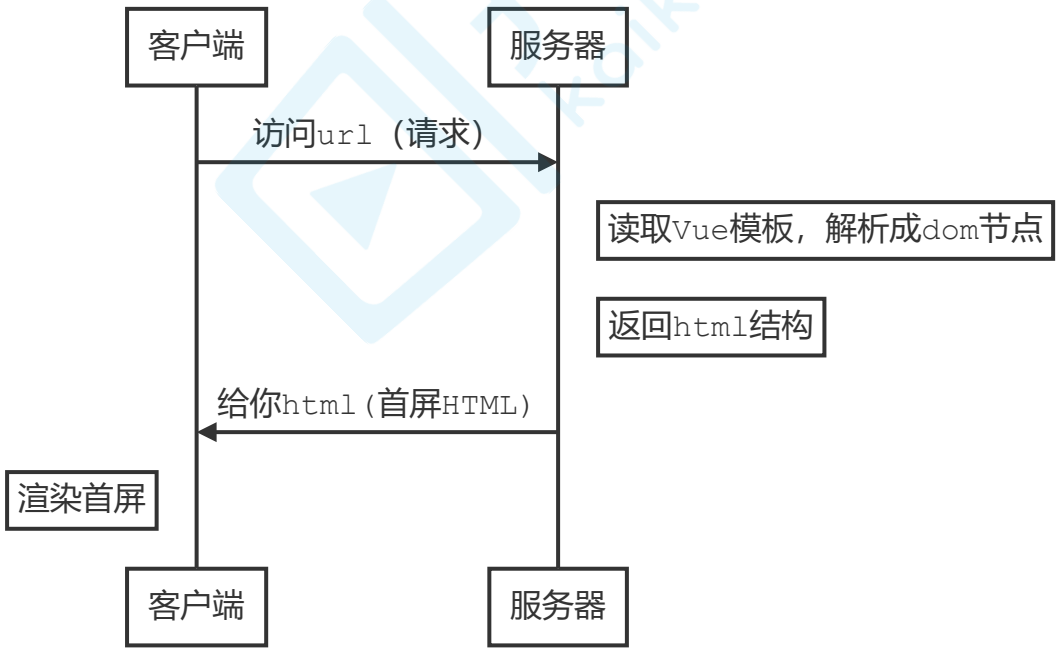
```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="/favicon.ico">
    <title>vue-ssr</title>
  <link href="/app.js" rel="preload" as="script"></head>
  <body>
    <noscript>
      <strong>We're sorry but vue-ssr doesn't work properly without JavaScript enabled. Please enable it to continue.</strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  <script type="text/javascript" src="/app.js"></script></body>
</html>
```

这里可以看到单页应用的两个缺点

1. 首屏渲染性能
    1. 必须得等js加载完毕，并且执行完毕，才能渲染出首屏
2. seo
    1. 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实战

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

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

const app = express()
const renderer = require('vue-server-renderer').createRenderer()
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){
    const html = await renderer.renderToString(page)
    res.send(html)
})

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

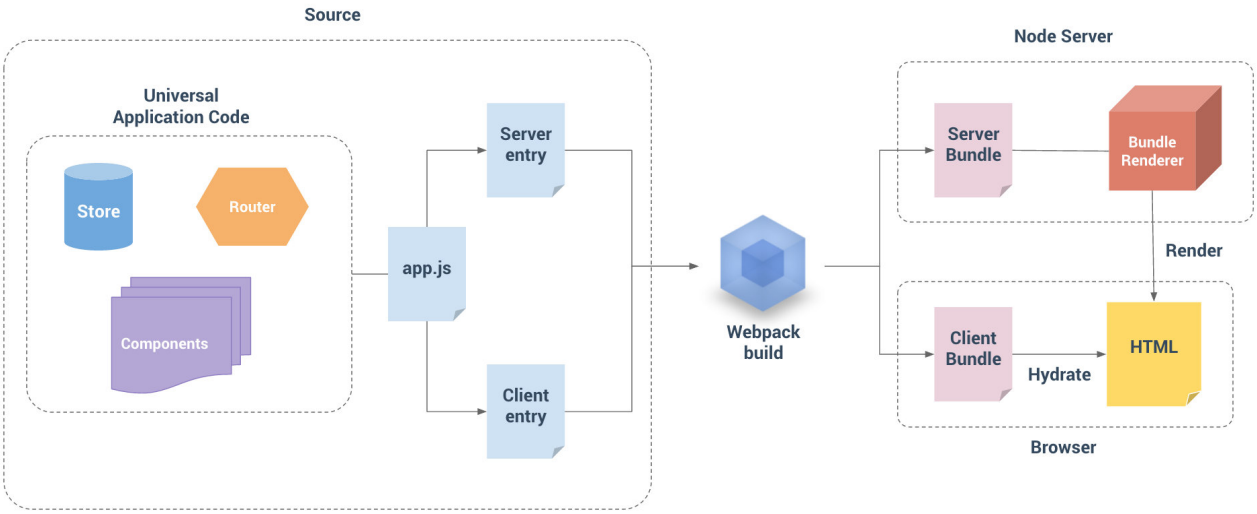
```
<div data-server-rendered="true"><h1>开课吧</h1> <h1>1</h1></div>
```

稍微复杂一些

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

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

```
            <ul >
                <li v-for="todo in todos"> {{todo}}</li>
            </ul>
        </div>
    `
})

app.get('/',async function(req,res){
    const html = await renderer.renderToString(page)
    console.log(html)
    res.send(html)
})

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

构建步骤



1. 通常前端都是vue单文件组件，用vue-lodaer构建，所以ssr环境需要webpack 怎么操作呢 下面开始

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
```

```
// /src/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>

<style>
</style>
```

csr 和ssr统一入口

```
// src/creatapp.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main.js

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

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

ssr的entry-server.js

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

```
        router.push(context.url)
        router.onReady(() => {
            resolve(app)
        }, reject)
    })
}
```

服务端渲染，我们需要能够处理加载.vue组件，所以需要webpack的支持

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
```

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

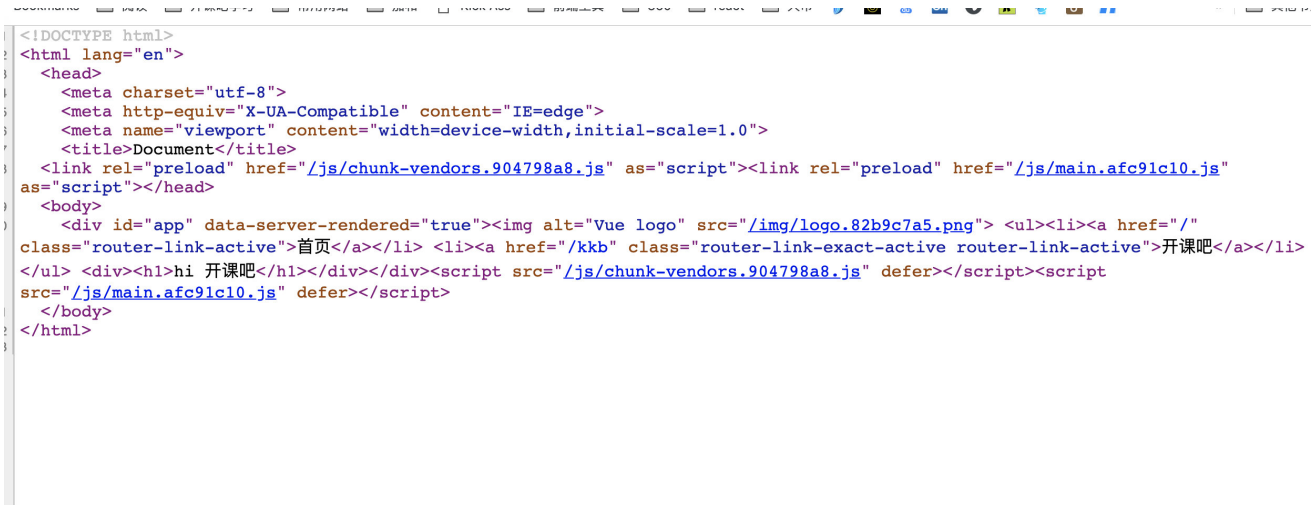
```
        })
      : undefined,
    optimization: {
          splitChunks: undefined
    },
    plugins: [TARGET_NODE ? new VueSSRServerPlugin() : new VueSSRClientPlugin()]
  }),
  chainWebpack: config => {
    config.module
      .rule("vue")
      .use("vue-loader")
      .tap(options => {
     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

```
// 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}`);
});
```

```
// 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>
```

```
// 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"
  },
```



```
<!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>
  <link rel="preload" href="/js/chunk-vendors.904798a8.js" as="script"><link rel="preload" href="/js/main.afc91c10.js"
as="script"></head>
  <body>
    <div id="app" data-server-rendered="true"><img alt="Vue logo" src="/img/logo.82b9c7a5.png"> <ul><li><a href="/"
class="router-link-active">首页</a></li> <li><a href="/kkb" class="router-link-exact-active router-link-active">开课吧</a></li>
</ul> <div><h1>hi 开课吧</h1></div></div><script src="/js/chunk-vendors.904798a8.js" defer></script><script
src="/js/main.afc91c10.js" defer></script>
  </body>
</html>
```

数据响应Vuex

store.js

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

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

Vue.use(Vuex)

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

    },
    actions: {

    }
  })
}
```

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 }
}
```

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

ssr记得先跑 npm run build

```
// 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}`);
});
```

- 首页
- 开课吧

hi 开课吧

num:108

```
<!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>
  <link rel="preload" href="/js/chunk-vendors.780b8eb1.js" as="script"><link rel="preload" href="/js/main.b71c9651.js" as="script"></head>
  <body>
    <div id="app" data-server-rendered="true"><img alt="Vue logo" src="/img/logo.82b9c7a5.png"> <ul><li><a href="/" class="router-link-active">首页</a></li> <li><a href="/kkb" class="router-link-exact-active router-link-active">开课吧</a></li></ul> <div><h1>hi 开课吧</h1> <h2>num:108</h2></div></div><script src="/js/chunk-vendors.780b8eb1.js" defer></script><script src="/js/main.b71c9651.js" defer></script>
  </body>
</html>
```

nuxt.js

> https://zh.nuxtjs.org/guide

自己折腾太麻烦拉，还好有nuxt，内置vuex vue-router，ssr最佳实践框架

新建目录 `cnpm install nuxt --save` ,新增script

```
  "scripts": {
    "dev": "nuxt"
  }
```

nuxt遵循约定优于配置，我们新建pages目录，就是页面了

`mkdir pages`

新建pages/index.vue

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

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

npm run dev

nuxt约束

文件夹约束

1. assets 静态资源
2. components 组件
3. layouts 布局
4. middleware 中间件
5. store vuex
6. nuxt.config.js 个性化配置

路由

```
pages/
--| user/
-----| index.vue
-----| one.vue
--| index.vue
```

那么，Nuxt.js 自动生成的路由配置如下：

```
router: {
  routes: [
    {
      name: 'index',
      path: '/',
      component: 'pages/index.vue'
    },
    {
      name: 'user',
      path: '/user',
      component: 'pages/user/index.vue'
    },
    {
      name: 'user-one',
      path: '/user/one',
      component: 'pages/user/one.vue'
    }
  ]
}
```

异步数据获取

asyncData会在页面加载前调用，作为数据获取 支持async+ await 返回值会merge到data里

```
<template>
    <div>
        <h1>hi {{name}}</h1>
        <h1>hi {{title}}</h1>
    </div>
</template>
<script>

const delay = ()=> new Promise((resolve,reject)=>{
    setTimeout(()=>{
        resolve({title: '异步开课吧数据'})
    },1000)
})
```

```
export default {
    async asyncData(){
        return await delay()
    },
    data(){
        return {
            name:'开课吧'
        }
    }
}
</script>
```

nuxt+ vuex

store目录就是存储vuex数据的地方

```
// store/index.js
export const state = () => ({
    counter: 0
})

export const mutations = {
    increment(state) {
        state.counter++
    }
}
```

```
<template>
    <div>
        <h1>hi {{name}}</h1>
        <h1>hi {{title}}</h1>
        <h1>hi {{counter}}</h1>
        <button @click="add">添加</button>
    </div>
</template>
<script>

const delay = ()=> new Promise((resolve,reject)=>{
    setTimeout(()=>{
        resolve({title: '异步开课吧数据'})
    },1000)
})
import axios from 'axios'
export default {
    async asyncData(){
        return await delay()
    },
```

```
    data(){
        return {
            name:'开课吧'
        }
    },
    methods:{
        async add(){
            const res = await axios.get('http://taro.josephxia.com/api/top')
            console.log('res',res.data)
            this.$store.commit('increment')
        }
    },
    computed:{
        counter(){
            return this.$store.state.counter
        }
    },
}
</script>
```

Axios前后端

```
npm i axios -s
```

```
        // 服务器端运行
        async asyncData(){
        const res = await axios.get('http://taro.josephxia.com/api/top')
        console.log('res',res.data)
        return await delay()
    },
        // 浏览器端运行
            async add(){
            const res = await axios.get('http://taro.josephxia.com/api/top')
            console.log('res',res.data)
            this.$store.commit('increment')
        }
```