服务端渲染SSR

资源

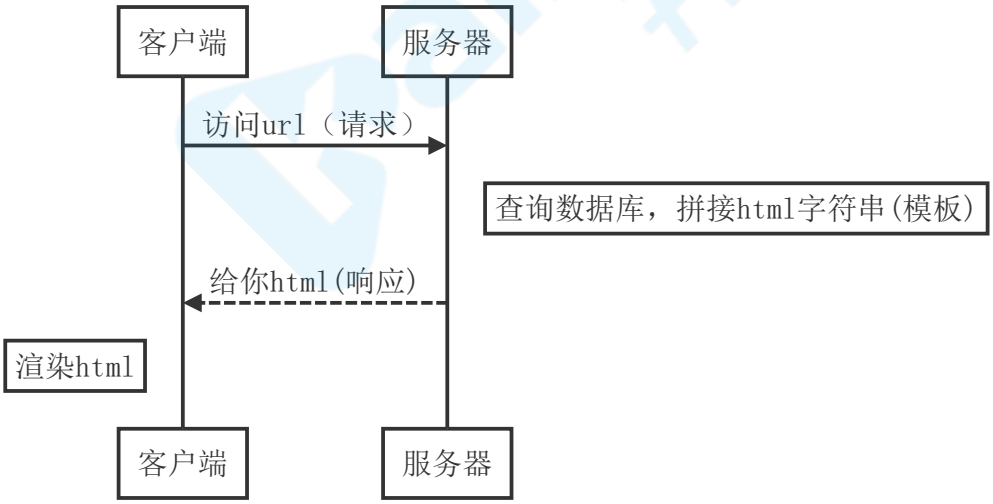
1. vue ssr
2. nuxt.js

知识点

理解ssr

CSR VS SSR

传统的web开发




```
// npm i express -s
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')
})
```

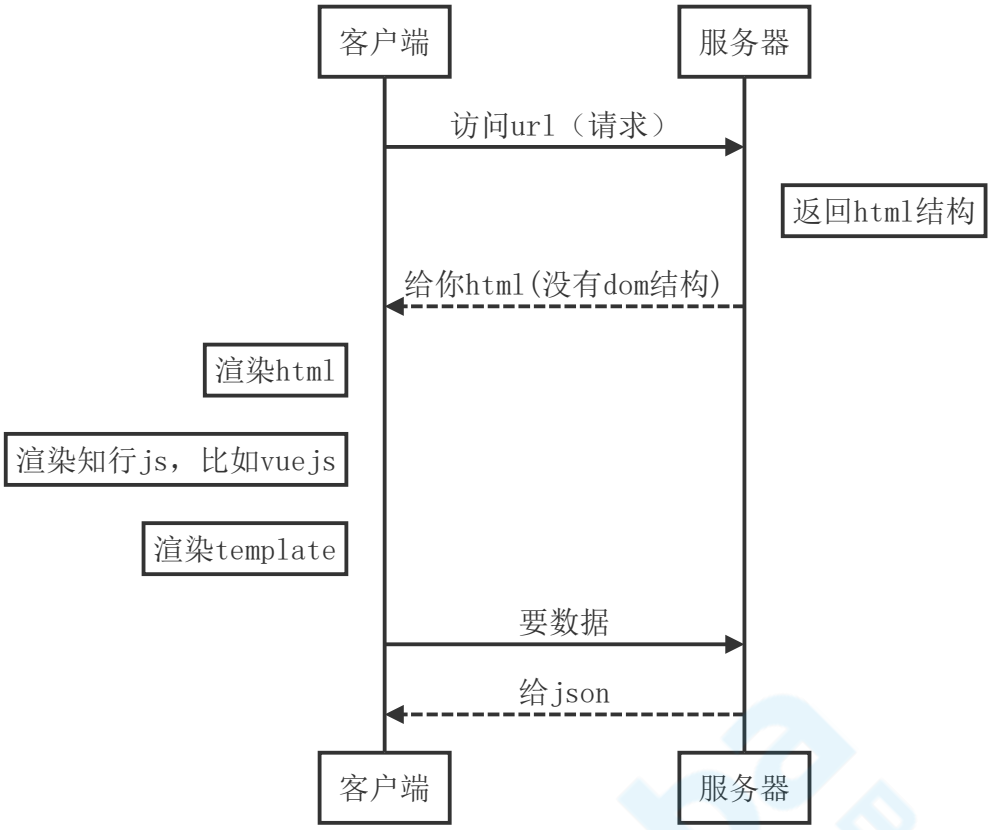
打开页面 查看源码

```
<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

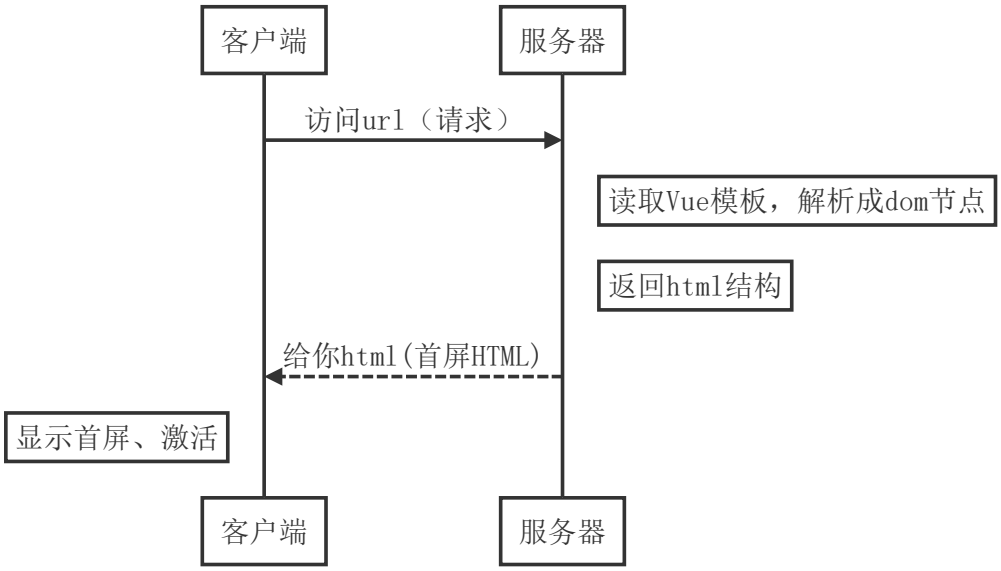
到了vue，react时代，单页应用优秀的用户体验，逐渐成为了主流，页面整体是JS渲染出来的，称之为客户端渲染CSR




```
<!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>
```

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，前端拿到的内容包括首屏及完整spa结构，应用激活后依然按照spa方式运行，这种页面渲染方式被称为服务端渲染(server side render)



Vue SSR实战

新建工程

```
vue create ssr
```

安装依赖

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

启动脚本

创建一个express服务器，将vue ssr集成进来，./server/index.js

```
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'
    },
    template:`
        <div>
            <h1>{{name}}</h1>
        </div>
    `
})

app.get('/',async function(req,res){
    // renderToString可以将vue实例转换为html字符串
    // 若未传递回调函数，则返回Promise
    try {
```

```
        const html = await renderer.renderToString(page)
        res.send(html)
    } catch (error) {
        res.status(500).send('Internal Server Error')
    }
})

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

路由

路由支持仍然使用vue-router

安装

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

配置

创建@/router/index.js

```
import Vue from "vue";
import Router from "vue-router";
// 分别创建Index.vue和Detail.vue
import Index from "@/components/Index";
import Detail from "@/components/Detail";

Vue.use(Router);

//导出工厂函数
export function createRouter() {
  return new Router({
    mode: 'history',
    routes: [
      { path: "/", component: Index },
      { path: "/detail", component: Detail }
    ]
  });
}
```

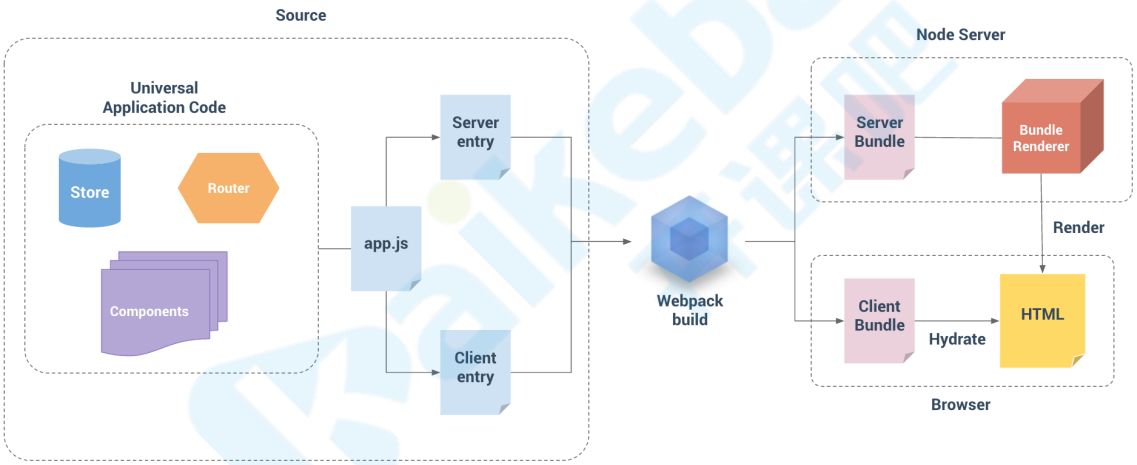
更新App.vue

```
<template>
  <div id="app">
    <nav>
      <router-link to="/">首页</router-link>
      <router-link to="/detail">详情页</router-link>
    </nav>
    <router-view></router-view>
  </div>
</template>
```

构建

对于客户端应用程序和服务器应用程序，我们都要使用 webpack 打包 - 服务器需要「服务器 bundle」然后用于服务器端渲染(SSR)，而「客户端 bundle」会发送给浏览器，用于混合静态标记。

构建流程



代码结构

```
src
├── App.vue
├── app.js # 用于创建vue实例
├── entry-client.js # 客户端入口，用于静态内容“激活”
└── entry-server.js # 服务端入口，用于首屏内容渲染
```

Vue实例创建

`app.js` 是负责创建vue实例，每次请求均会有独立的vue实例创建。创建app.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 }
}
```

服务端入口

服务端入口文件src/entry-server.js

```
import { createApp } from './app'

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)
    })
}
```

客户端入口

客户端入口只需创建vue实例并执行挂载，这一步称为激活。创建entry-client.js：

```
import { createApp } from './app'

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

webpack配置

具体配置，vue.config.js

```
const VueSSRServerPlugin = require("vue-server-renderer/server-plugin");
const VueSSRClientPlugin = require("vue-server-renderer/client-plugin");
```

```
const TARGET_NODE = process.env.WEBPACK_TARGET === "node";
const target = TARGET_NODE ? "server" : "client";

module.exports = {
  css: {
    extract: false
  },
  outputDir: './dist/'+target,
  configureWebpack: () => ({
    // 将 entry 指向应用程序的 server / client 文件
    entry: `./src/entry-${target}.js`,
    // 对 bundle renderer 提供 source map 支持
    devtool: 'source-map',
    // 这允许 webpack 以 Node 适用方式处理动态导入(dynamic import),
    // 并且还会在编译 Vue 组件时告知 `vue-loader` 输送面向服务器代码(server-oriented
code)。
    target: TARGET_NODE ? "node" : "web",
    node: TARGET_NODE ? undefined : false,
    output: {
      // 此处使用 Node 风格导出模块
      libraryTarget: TARGET_NODE ? "commonjs2" : undefined
    },
    // 这是将服务器的整个输出构建为单个 JSON 文件的插件。
    // 服务端默认文件名为 `vue-ssr-server-bundle.json`
    plugins: [TARGET_NODE ? new VueSSRServerPlugin() : new VueSSRClientPlugin()]
  })
};
```

脚本配置

安装依赖

```
npm i cross-env -D
```

定义创建脚本，package.json

```
"scripts": {
    "build:client": "vue-cli-service build",
    "build:server": "cross-env WEBPACK_TARGET=node vue-cli-service build --mode
server",
    "build": "npm run build:server && npm run build:client"
},
```

> 执行打包：npm run build

宿主文件

最后需要定义宿主文件，创建./public/index.tmpl.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>
```

服务器启动文件

修改服务器启动文件，现在需要处理所有路由，./server/index2.js

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

const resolve = dir => {
    return path.resolve(__dirname, dir)
}

app.use(express.static(resolve('../dist/client'), {index: false}))

const { createBundleRenderer } = require("vue-server-renderer");
const bundle = require(resolve("../dist/server/vue-ssr-server-bundle.json"));

const renderer = createBundleRenderer(bundle, {
  runInNewContext: false,
  template: fs.readFileSync(resolve("../public/index.tmpl.html"), "utf-8"),
  clientManifest: require(resolve("../dist/client/vue-ssr-client-
manifest.json"))
});

app.get('*',async (req,res)=>{
  console.log(req.url)
  // /admin
  const context = {
    title:'ssr test',
    url:req.url
  }
  const html = await renderer.renderToString(context);
  res.send(html)
})

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

整合Vuex

安装vuex

```
npm install -S vuex
```

store/index.js

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

Vue.use(Vuex)

export function createStore () {
  return new Vuex.Store({
    state: {
        count:108
    },
    mutations: {
        add(state){
            state.count += 1;
        }
    }
  })
}
```

挂载store，app.js

```
import { createStore } from './store'

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

使用，.src/components/Index.vue

```
<h2>num:{{$store.state.count}}</h2>
<button @click="$store.commit('add')">add</button>
```

数据预取

异步数据获取，store/index.js

```
export function createStore() {
  return new Vuex.Store({
```

```
    mutations: {
      // 加一个初始化
      init(state, count) {
        state.count = count;
      },
    },
    actions: {
      // 加一个异步请求count的action
      getCount({ commit }) {
        return new Promise(resolve => {
          setTimeout(() => {
            commit("init", Math.random() * 100);
            resolve();
          }, 1000);
        });
      },
    },
  });
}
```

组件中的数据预取逻辑，Index.vue

```
export default {
  asyncData({ store, route }) { // 约定预取逻辑编写在预取钩子asyncData中
    // 触发 action 后，返回 Promise 以便确定请求结果
    return store.dispatch("getCount");
  }
};
```

服务端数据预取，entry-server.js

```
import { createApp } from "./app";

export default context => {
  return new Promise((resolve, reject) => {
    const { app, router, store } = createApp(context);
    router.push(context.url);
    router.onReady(() => {
      // 获取匹配的路由组件数组
      const matchedComponents = router.getMatchedComponents();
      if (!matchedComponents.length) {
        return reject({ code: 404 });
      }

      // 对所有匹配的路由组件调用 `asyncData()`
      Promise.all(
        matchedComponents.map(Component => {
          if (Component.asyncData) {
            return Component.asyncData({
              store,
              route: router.currentRoute,
            });
          }
```

```
        }),
      )
        .then(() => {
          // 所有预取钩子 resolve 后，
          // store 已经填充入渲染应用所需状态
          // 将状态附加到上下文，且 `template` 选项用于 renderer 时，
          // 状态将自动序列化为 `window.__INITIAL_STATE__`，并注入 HTML。
          context.state = store.state;

          resolve(app);
        })
        .catch(reject);
    }, reject);
  });
};
```

客户端在挂载到应用程序之前，store 就应该获取到状态，entry-client.js

```
// 导出store
const { app, router, store } = createApp();

// 当使用 template 时，context.state 将作为 window.__INITIAL_STATE__ 状态自动嵌入到最终的 HTML // 在客户端挂载到应用程序之前，store 就应该获取到状态：
if (window.__INITIAL_STATE__) {
  store.replaceState(window.__INITIAL_STATE__);
}
```

客户端数据预取处理，app.js

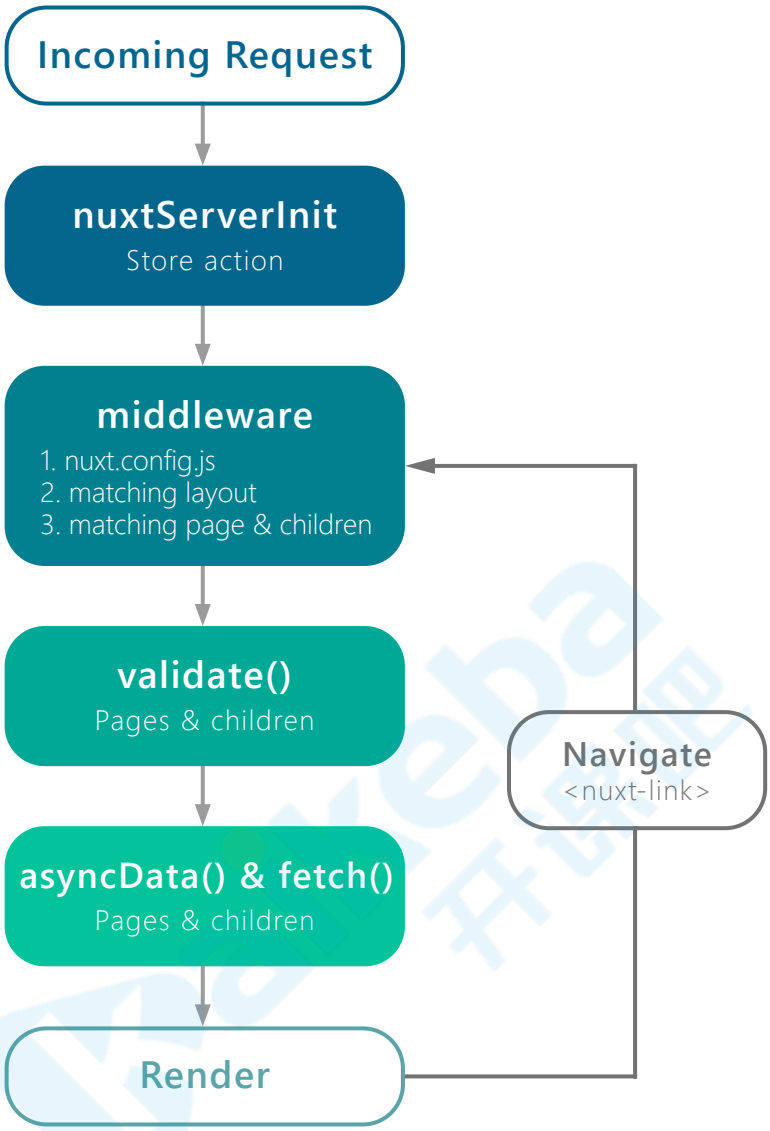
```
Vue.mixin({
  beforeMount() {
    const { asyncData } = this.$options;
    if (asyncData) {
      // 将获取数据操作分配给 promise
      // 以便在组件中，我们可以在数据准备就绪后
      // 通过运行 `this.dataPromise.then(...)` 来执行其他任务
      this.dataPromise = asyncData({
        store: this.$store,
        route: this.$route,
      });
    }
  },
});
```

SSR实战：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>
```

选项

```
PS C:\Users\yt037\Desktop\kaikeba\projects> npx create-nuxt-app nuxt-app
npx: 341 安装成功，用时 27.05 秒

create-nuxt-app v2.10.1
✨ Generating Nuxt.js project in nuxt-app
? Project name nuxt-app
? Project description My terrific Nuxt.js project
? Author name yt0379
? Choose the package manager Npm
? Choose UI framework None
? Choose custom server framework Koa
? Choose Nuxt.js modules Axios
? Choose linting tools (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Choose test framework None
? Choose rendering mode Universal (SSR)
? Choose development tools jsconfig.json (Recommended for VS Code)
- Installing packages with npm
```

运行项目：`npm run dev`

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/admin.vue 商品管理页
- pages/login.vue 登录页

> 访问http://localhost:3000/试试
>
> 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>
```

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>
        </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},
    ] }
  }
};
</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/
--| detail/
----| _id.vue
--| detail.vue
```

测试代码，detail.vue

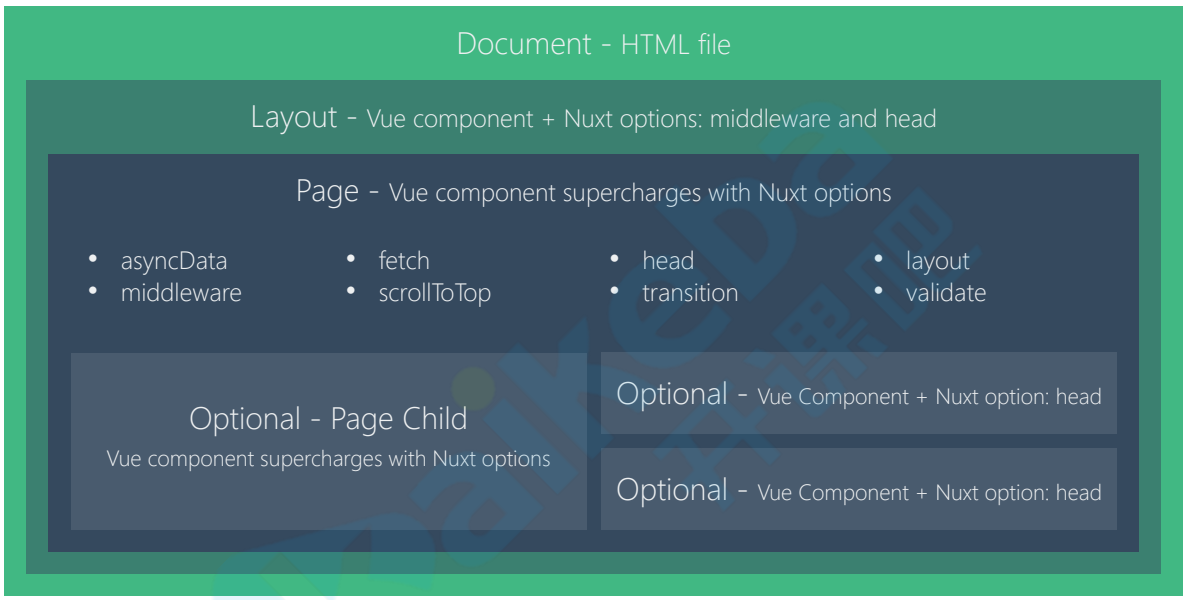
```
<template>
    <div>
        <h2>detail</h2>
        <nuxt-child></nuxt-child>
    </div>
</template>
```

配置路由

要扩展 `Nuxt.js` 创建的路由，可以通过 `router.extendRoutes` 选项配置。例如添加自定义路由:

```
// nuxt.config.js
export default {
  router: {
    extendRoutes (routes, resolve) {
      routes.push({
        name: "foo",
        path: "/foo",
        component: resolve(__dirname, "pages/custom.vue")
      });
    }
  }
}
```

页面配置



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'
}
```

配置页头

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`

> 需管理员权限

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>
```

> 全局注册：将会对所有页面起作用，nuxt.config.js
>
> ```
> router: {
>     middleware: ['auth']
> },
> ```

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.$axios.$post("/api/login", u).then(({ token }) => {
      if (token) {
        commit("init", token);
      }
      return getters.isLogin;
    });
  }
};
```

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>
```

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"
],
```

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

当我们想将服务端的一些数据传到客户端时，这个方法非常好用。

范例：登录状态初始化，store/index.js

```
export const actions = {
  // 参数1：action上下文对象，参数2：页面上下文
  nuxtServerInit({ commit }, { app }) {
    const token = app.$cookies.get("token");
    if (token) {
      console.log("nuxtServerInit: token:"+token);
      commit("user/init", 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"],
> ```
>
> - nuxtServerInit只能写在store/index.js
> - nuxtServerInit仅在服务端执行

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
```

> 生成内容在.nuxt/dist中

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
```

> 注意渲染和接口服务器都需要处于启动状态
>
> 生成内容再dist中