服务端渲染SSR

资源

1. vue ssr
2. nuxt.js

知识点

Nuxt.js实战

Nuxt.js 是一个基于 Vue.js 的**通用应用框架**。

通过对客户端/服务端基础架构的抽象组织，Nuxt.js 主要关注的是应用的 **UI渲染**。

> 结论：
>
> 1. nuxt不仅仅用于服务端渲染也可用于spa应用开发；
> 2. 利用nuxt提供的基础项目结构、路由生成、中间件、插件等特性可大幅提高开发效率
> 3. nuxt可用于网站静态化

资源

Nuxt.js官方文档

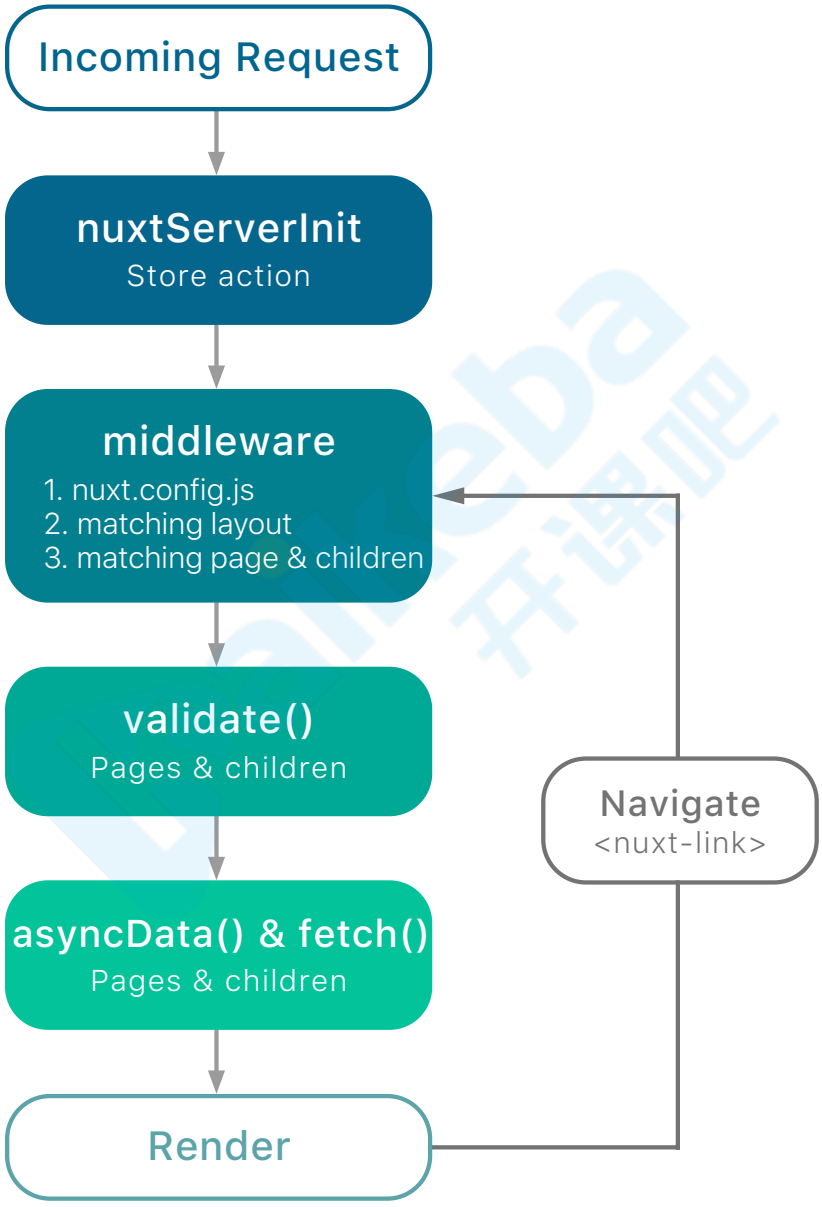
nuxt.js特性

- 代码分层
- 服务端渲染
- 强大的路由功能

- 静态文件服务
- ...

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`

目录结构

- assets：**资源目录** `assets` 用于组织未编译的静态资源如 `LESS`、`SASS` 或 `JavaScript`。
- components：**组件目录** `components` 用于组织应用的 Vue.js 组件。Nuxt.js 不会扩展增强该目录下 Vue.js 组件，即这些组件不会像页面组件那样有 `asyncData` 方法的特性。
- layouts：**布局目录** `layouts` 用于组织应用的布局组件。
- middleware：**中间件目录**用于存放应用的中间件。
- pages：**页面目录** `pages` 用于组织应用的路由及视图。Nuxt.js 框架读取该目录下所有的 `.vue` 文件并自动生成对应的路由配置。
- plugins：**插件目录** `plugins` 用于组织那些需要在 `根vue.js应用` 实例化之前需要运行的 Javascript 插件。
- static：**静态文件目录** `static` 用于存放应用的静态文件，此类文件不会被 Nuxt.js 调用 Webpack 进行构建编译处理。 服务器启动的时候，该目录下的文件会映射至应用的根路径 `/` 下。
- store：用于组织应用的 **Vuex 状态**树文件。 Nuxt.js 框架集成了 Vuex 状态树 的相关功能配置，在 `store` 目录下创建一个 `index.js` 文件可激活这些配置。
- nuxt.config.js：该文件用于**个性化配置**Nuxt应用。

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

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
```

会生成如下路由配置：

```
{
    path: "/detail/:id?",
    component: _9c9d895e,
    name: "detail-id"
}
```

> 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
```

生成的路由配置如下：

```
{
  path: '/detail',
  component: 'pages/detail.vue',
  children: [
    {path: ':id?', name: "detail-id"}
  ]
}
```

测试代码，detail.vue

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

> nuxt-child等效于router-view

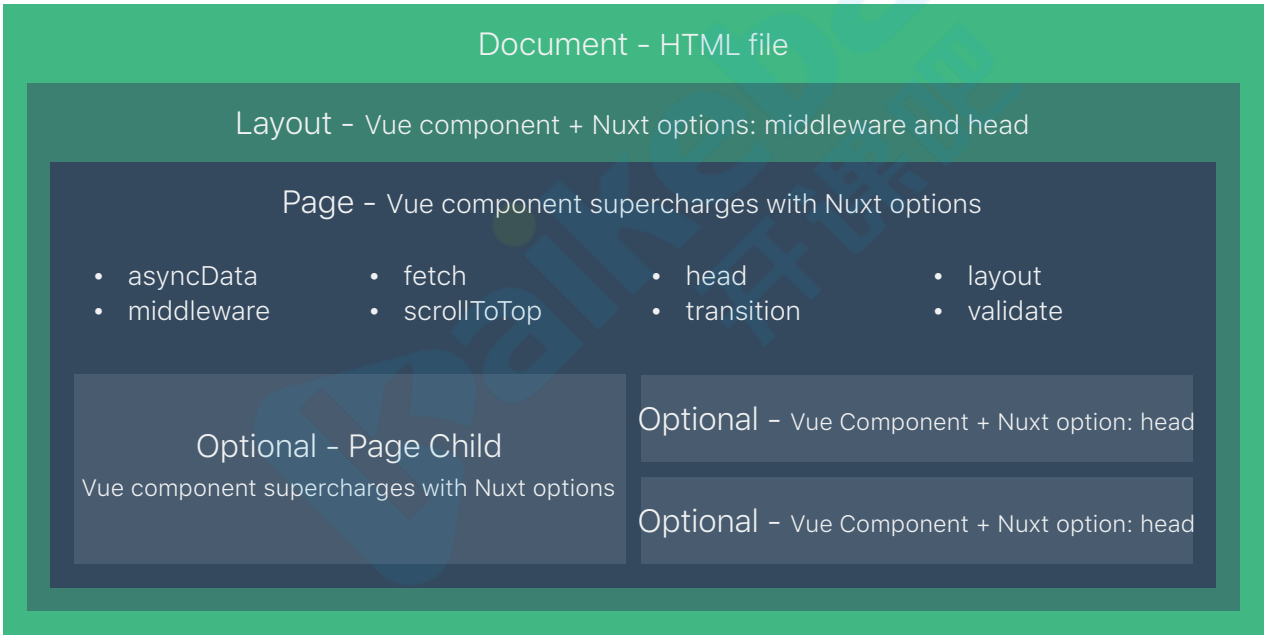
配置路由

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")
      });
    }
  }
}
```

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>
    <p>{{error}}</p>
    <nuxt-link to="/">首 页</nuxt-link>
  </div>
</template>

<script>
export default {
  props: ['error'],
  layout:'blank'
}
</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",
      // vue-meta利用hid确定要更新meta
      meta: [{ name: "description", hid: "description", content: "set page
meta" }],
      link: [{ rel: "favicon", href: "favicon.ico" }],
    };
  },
};
```

> 更多页面配置项：

属性名	描述
asyncData	最重要选项, 支持 异步数据处理，该方法的第一个参数为当前页面组件的 上下文对象。
fetch	与 `asyncData` 方法类似，用于在渲染页面之前获取数据填充应用的状态树（store）。不同的是 `fetch` 方法不会设置组件的数据。详情请参考 关于fetch方法的文档。
head	配置当前页面的 Meta 标签, 详情参考 页面头部配置API。
layout	指定当前页面使用的布局（`layouts` 根目录下的布局文件）。详情请参考 关于 布局 的文档。
loading	如果设置为`false`，则阻止页面自动调用`this.$nuxt.$loading.finish()`和`this.$nuxt.$loading.start()`,您可以手动控制它,请看例子,仅适用于在nuxt.config.js中设置`loading`的情况下。请参考API配置 `loading` 文档。
transition	指定页面切换的过渡动效, 详情请参考 页面过渡动效。
scrollToTop	布尔值，默认: `false`。 用于判定渲染页面前是否需要将当前页面滚动至顶部。这个配置用于 嵌套路由的应用场景。
validate	校验方法用于校验 动态路由的参数。
middleware	指定页面的中间件，中间件会在页面渲染之前被调用， 请参考 路由中间件。

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`
- 接口文件，server/api.js

整合axios

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

> win10有时需管理员权限启动vscode

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

通过在store的根模块中定义 `nuxtServerInit` 方法，Nuxt.js 调用它的时候会将页面的上下文对象作为第2个参数传给它。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 = {
  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