测试题

下周四17:00~下周五17:00

方式：选择题

勘误

- 发给同学们的项目中package.json里面缺少依赖库，需额外安装

```
npm i koa-bodyparser koa-router @nuxtjs/axios -S
```

- cookie获取方式有兼容性问题

  安装cookie-universal-nuxt

```
npm i --save cookie-universal-nuxt
```

  注册,nuxt.config.js

```
modules: ["cookie-universal-nuxt"],
```

  修改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);
    }
  }
};
```

- plugins/axios修改为

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

知识点补充

部署

服务端渲染应用部署

```
npm run build
npm start
```

静态应用部署

- 准备工作：
  - 将接口服务器独立出来

```
// api.js
const Koa = require("koa");
const bodyparser = require("koa-bodyparser");
const router = require("koa-router")({ prefix: "/api" });

const app = new Koa();

// 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 }
];

// /api/goods
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 };
  }
});

app.use(bodyparser());
app.use(router.routes());

app.listen(8080);


// index.js
const Koa = require("koa");
const consola = require("consola");
const { Nuxt, Builder } = require("nuxt");

const app = new Koa();

// Import and Set Nuxt.js options
let config = require("../nuxt.config.js");
config.dev = !(app.env === "production");

async function start() {
  // Instantiate nuxt.js
  const nuxt = new Nuxt(config);

  const {
    host = process.env.HOST || "127.0.0.1",
    port = process.env.PORT || 3000
  } = nuxt.options.server;

  // Build in development
  if (config.dev) {
```

```
    const builder = new Builder(nuxt);
    await builder.build();
  } else {
    await nuxt.ready();
  }

  // 页面渲染
  app.use(ctx => {
    ctx.status = 200;
    ctx.respond = false; // Bypass Koa's built-in response handling
    ctx.req.ctx = ctx; // This might be useful later on, e.g. in nuxtServerInit or with nuxt-stash
    nuxt.render(ctx.req, ctx.res);
  });

  app.listen(port, host);
  consola.ready({
    message: `Server listening on http://${host}:${port}`,
    badge: true
  });
}

start();
```

> 启动接口服务器api.js

- 代理接口，nuxt.config.js

```
axios: {
    proxy: true
},
proxy: {
    "/api": "http://localhost:8080"
},
```

- 生成应用的静态目录和文件

  配置服务器默认端口，package.json

```
"scripts": {
   "generate": "cross-env PORT=80 nuxt generate"
}
```

  执行生成命令

```
npm run generate
```

nginx配置

```
server {
    listen       80;
    server_name  localhost;

    # 静态文件服务
    location / {
        root  C:\\Users\\yt037\\Desktop\\kaikeba\\projects\\nuxt-
test\\dist;
        index index.html;
    }

    #nginx反向代理，实现接口转发
    location ^~ /api/ {
        proxy_pass http://localhost:3000;  #注意路径后边不要加/
    }
}
```

知识点

TypeScript

TypeScript是JavaScript的超集，它可编译为纯JavaScript，是一种给 JavaScript 添加特性的语言扩展。

类型注解和类型检查

```
let name = "xx"; // 类型推论
let title: string = "开课吧"; // 类型注解
name = 2;// 错误
title = 4;// 错误

//数组使用类型

let names: string[];
names = ['Tom'];//或Array<string>

let foo:any = 'xx'
foo = 3

// any类型也可用于数组
let list: any[] = [1, true, "free"];
list[1] = 100;

// 函数中使用类型注解
```

```
function greeting(person: string): string {
  return 'Hello, ' + person;
}
//void类型，常用于没有返回值的函数
function warnUser(): void { alert("This is my warning message"); }
```

函数

```
// 此处name和age是必填参数
// 如果要变为可选参数，加上?
// 可选参数在必选参数后面
function sayHello(name: string, age: number = 20, addr?: string): string {
    return '你好：' + name + ' ' + age;
}

// 重载
// 参数数量或者类型或者返回类型不同 函数名却相同
// 先声明，在实现
function info(a: { name: string }): string;
function info(a: string): object;
function info(a: { name: string } | string): any {
    if (typeof a === "object") {
        return a.name;
    } else {
        return { name: a };
    }
}
console.log(info({ name: "tom" }));
console.log(info("tom"));
```

类

```
// Class，面向对象
class Shape {
    // area: number
    // 参数属性
    constructor(protected color: string, private width: number, private
height: number) {
        // this.area = width * height
        this.color = color
    }
    // 存取器
```

```
    get area() {
        return this.width * this.height
    }

    shoutout() {
        return "I'm " + this.color + " with an area of " + this.area + "
cm squared."
    }
}

class Square extends Shape { // 继承
    constructor(color: string, side: number) {
        super(color, side, side)
        console.log(this.color)
    }
    // 覆盖
    shoutout() {
        return "我是" + this.color + " 面积是" + this.area + "平方厘米"
    }
}
const square = new Square('blue', 2)
console.log(square.shoutout())
```

接口

interface，仅定义结构，不需要实现

```
interface Person {
  firstName: string;
  lastName: string;
  sayHello(): string; // 要求实现方法
}
// 实现接口
class Greeter implements Person {
  constructor(public firstName='', public lastName=''){}
  sayHello(){
    return 'Hello, ' + this.firstName + ' ' + this.lastName;
  }
}
// 面向接口编程
function greeting(person: Person) {
  return person.sayHello();
}
// const user = {firstName: 'Jane', lastName: 'User'};
const user = new Greeter('Jane', 'User'); // 创建对象实例
console.log(user);
console.log(greeting(user));
```

泛型 Generics

Generics是指在定义函数、接口或类的时候，不预先指定具体的类型，而在使用的时候再指定类型的一种特性。

```
// 定义泛型接口
interface Result<T> {
  ok: 0 | 1;
  data: T[];
}
// 定义泛型函数
function getData<T>(): Result<T> {
  const data: any[] = [
    { id: 1, name: "类型注解", version: "2.0" },
    { id: 2, name: "编译型语言", version: "1.0" }
  ];
  return { ok: 1, data };
}

// 使用泛型
this.features = getData<Feature>().data;
```

装饰器

装饰器实际上是工厂函数，传入一个对象，输出处理后的新对象。

```
// 类装饰器
@Component
export default class Hello extends Vue {
  // 属性装饰器
  @Prop({ required: true, type: String }) private msg!: string;

  // 函数装饰器
  @Watch("features", {deep: true})
  onRouteChange(val: string, oldVal: any) {
    console.log(val, oldVal);
  }

  // 函数装饰器
  @Emit()
  private addFeature(event: any) {
    const feature = {
      name: event.target.value,
      id: this.features.length + 1,
```

```
        version: "1.0"
      };
      this.features.push(feature);
      event.target.value = feature;

      return event.target.value;
    }
  }
```

测试

测试分类

常见的开发流程里，都有测试人员，这种我们称为黑盒测试，测试人员不管内部实现机制，只看最外层的输入输出，比如你写一个加法的页面，会设计N个case，测试加法的正确性，这种代码里，我们称之为**E2E测试**。

更负责一些的 我们称之为**集成测试**，就是集合多个测试过的单元一起测试。

还有一种测试叫做白盒测试，我们针对一些内部机制的核心逻辑 使用代码 进行编写 我们称之为**单元测试**。

准备工作

推荐用使用Jest作为测试框架

新建vue项目，手动选择特性，添加Unit Testing和E2E Testing

单元测试解决方案选择：Jest

端到端测试解决方案选择：Cypress

单元测试

单元测试（unit testing），是指对软件中的最小可测试单元进行检查和验证。

新建test/unit/kaikeba.spec.js，`*.spec.js`是命名规范，写下一下代码

```
function add(num1, num2) {
    return num1 + num2
}

// 测试套件 test suite
describe('Kaikeba', () => {
    // 测试用例 test case
    it('测试add函数', () => {
        // 断言 assert
        expect(add(1, 3)).toBe(3)
        expect(add(1, 3)).toBe(4)
        expect(add(-2, 3)).toBe(1)
    })
})
```

> 这里面仅演示了toBe，更多断言API

执行单元测试

`npm run test:unit`

测试Vue组件

创建一个vue组件components/Kaikeba.vue

```
<template>
    <div>
    <span>{{ message }}</span>
    <button @click="changeMsg">点击</button>
    </div>
</template>

<script>
  export default {
    data () {
      return {
        message: 'vue-text'
      }
    },
    created () {
      this.message = '开课吧'
    },
    methods:{
        changeMsg(){
            this.message = '按钮点击'
        }
```

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

测试该组件，

```
// 导入 Vue.js 和组件，进行测试
import Vue from 'vue'
import KaikebaComp from '@/components/Kaikeba.vue'

describe('KaikebaComp', () => {
  // 检查原始组件选项
  it('由created生命周期', () => {
    expect(typeof KaikebaComp.created).toBe('function')
  })

  // 评估原始组件选项中的函数的结果
  it('初始data是vue-text', () => {
    // 检查data函数存在性
    expect(typeof KaikebaComp.data).toBe('function')
    // 检查data返回的默认值
    const defaultData = KaikebaComp.data()
    expect(defaultData.message).toBe('hello!')
  })
})
```

检查mounted之后

```
  it('mount之后测data是开课吧', () => {
    const vm = new Vue(KaikebaComp).$mount()
    expect(vm.message).toBe('开课吧')
  })
```

用户点击

和写vue没啥本质区别，只不过我们用测试的角度去写代码，vue提供了专门针对测试的`@vue/test-utils`

```
import { mount } from '@vue/test-utils'

it("按钮点击后", () => {
    const wrapper = mount(KaikebaComp);
    wrapper.find("button").trigger("click");
    // 测试数据变化
    expect(wrapper.vm.message).toBe("按钮点击");
    // 测试html渲染结果
    expect(wrapper.find("span").html()).toBe("<span>按钮点击</span>");
    // 等效的方式
    expect(wrapper.find("span").text()).toBe("按钮点击");
});
```

测试覆盖率

jest自带覆盖率，package.json里修改jest配置

```
  "jest": {
    "collectCoverage": true,
    "collectCoverageFrom": ["src/**/*.{js,vue}"],
  }
```

> 若采用独立配置，则修改jest.config.js:
>
> ```
> module.exports = {
>     ...
>     "collectCoverage": true,
>     "collectCoverageFrom": ["src/**/*.{js,vue}"]
> }
> ```

在此执行npm run test:unit

> [Vue组件单元测试cookbook]
>
> [Vue Test Utils使用指南]

E2E测试

借用浏览器的能力，站在用户测试人员的角度，输入框，点击按钮等，完全模拟用户，这个和具体的框架关系不大，完全模拟浏览器行为。

运行E2E测试

`npm run test:e2e`

修改e2e/spec/test.js

```
// https://docs.cypress.io/api/introduction/api.html

describe('端到端测试，抢测试人员的饭碗', () => {
  it('先访问一下', () => {
    cy.visit('/')
    // cy.contains('h1', 'Welcome to Your Vue.js App')
    cy.contains('span', '开课吧')

  })
})
```

测试未通过，因为没有使用Kaikeba.vue，修改App.vue

```
<div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <!-- <HelloWorld msg="Welcome to Your Vue.js App"/> -->
    <Kaikeba></Kaikeba>
</div>

import Kaikeba from './components/Kaikeba.vue'
export default {
  name: 'app',
  components: {
    HelloWorld,Kaikeba
  }
}
```

测试用户点击

```
// https://docs.cypress.io/api/introduction/api.html

describe('端到端测试，抢测试人员的饭碗', () => {
  it('先访问一下', () => {
    cy.visit('/')
    // cy.contains('h1', 'Welcome to Your Vue.js App')
    cy.contains('#message', '开课吧')

    cy.get('button').click()
    cy.contains('span', '按钮点击')

  })
})
```