1.5 工程化

1.5.1 vite

Vite是开发构建工具，开发期它利用浏览器 `native ES Module` 特性导入组织代码，生产中利用rollup作为打包工具，它有如下特点：

- 光速启动
- 热模块替换
- 按需编译

安装

```
$ npm init vite-app <project-name>
$ cd <project-name>
$ npm install
$ npm run dev
```

代码组织形式分析

关键变化是 `index.html` 中的入口文件导入方式

```
<script type="module" src="/src/main.js"></script>
```

这样main.js中就可以使用ES6 Module方式组织代码

```
import { createApp } from 'vue'
import App from './App.vue'
import './index.css'
```

浏览器会自动加载这些导入，vite会启动一个本地服务器处理不同这些加载请求，对于相对地址的导入，要根据后缀名处理文件内容并返回，对于裸模块导入要修改它的路径为相对地址并再次请求处理。

```
import { createApp } from '/@modules/vue.js'
import App from '/src/App.vue'
import '/src/index.css?import'
```

再根据模块package.json中的入口文件选项获取要加载的文件。

```
  "license": "MIT",
  "main": "index.js",
  "module": "dist/vue.runtime.esm-bundler.js",
  "name": "vue",
  "repository": {
```

> 对于开发者而言，整体没有大的变化。

范例：重构cart

1.5.2 资源加载

CSS文件导入

vite中可以直接导入 `.css` 文件，样式将影响导入的页面，最终会被打包到 `style.css` 。

> 在我们程序中，除了全局样式大部分样式都是以<style>形式存在于SFC中

CSS Module

SFC中使用CSS Module

```
<style module>
```

范例：修改Message组件样式为CSS Module形式

```
<style module>
  .message-box {
    padding: 10px 20px;
    background: #4fc08d;
    border: 1px solid #42b983;
    color: #fff
  }

  .message-box-close {
    float: right;
    cursor: pointer;
  }
</style>
```

```
<div :class="$style.messageBox">
  <h3>
    <!-- 具名插槽 -->
    <slot name="title"></slot>
    <span :class="$style.messageBoxClose"
      @click="$emit('close')">X</span>
  </h3>
```

JS中导入CSS Module：将css文件命名为 `*.module.css` 即可

```
import style from './Message.module.css'

export default {
  emits: ['close'],
  computed: {
    $style() {
      return style
    }
  },
}
```

CSS预处理器

安装对应的预处理器就可以直接在vite项目中使用。

```
<style lang="scss">
/* use scss */
</style>
```

或者在JS中导入

```
import './style.scss'
```

PostCSS

Vite自动对 `*.vue` 文件和导入的 `.css` 文件应用PostCSS配置，我们只需要安装必要的插件和添加 `postcss.config.js` 文件即可。

```
module.exports = {
  plugins: [
    require('autoprefixer'),
  ]
}
```

```
npm i postcss autoprefixer@8.1.4
```

资源URL处理

引用静态资源

我们可以在 `*.vue` 文件的template, style和纯 `.css` 文件中以相对和绝对路径方式引用静态资源。

```
<!-- 相对路径 -->
<img src="./assets/logo.png">
<!-- 绝对路径 -->
<img src="/src/assets/logo.png">
```

```
<style scoped>
#app {
  background-image: url('./assets/logo.png');
}
</style>
```

`public` 目录

`public` 目录下可以存放未在源码中引用的资源，它们会被留下且文件名不会有哈希处理。

这些文件会被原封不动拷贝到发布目录的根目录下。

```
<img src="/logo.png">
```

> 注意引用放置在 `public` 下的文件需要使用绝对路径，例如 `public/icon.png` 应该使用 `/icon.png` 引用

1.5.3 eslint

我们借助eslint规范项目代码，通过prettier做代码格式化。

首先在项目安装依赖，package.json

```
{
  "scripts": {
    "lint": "eslint \"src/**/*.{js,vue}\""
  },
  "devDependencies": {
    "@vue/eslint-config-prettier": "^6.0.0",
    "babel-eslint": "^10.1.0",
    "eslint": "^6.7.2",
    "eslint-plugin-prettier": "^3.1.3",
    "eslint-plugin-vue": "^7.0.0-0",
    "prettier": "^1.19.1"
  }
}
```

然后配置lint规则，.eslintrc.js

```
module.exports = {
  root: true,
  env: {
    node: true,
  },
  extends: ["plugin:vue/vue3-essential", "eslint:recommended", "@vue/prettier"],
  parserOptions: {
    parser: "babel-eslint",
  },
  rules: {
    "no-console": process.env.NODE_ENV === "production" ? "warn" : "off",
    "no-debugger": process.env.NODE_ENV === "production" ? "warn" : "off",
    "prettier/prettier": [
      "warn",
      {
        // singleQuote: none,
        // semi: false,
        trailingComma: "es5",
      },
    ],
  },
};
```

如果有必要还可以配置 `prettier.config.js` 修改prettier的默认格式化规则

```
module.exports = {
  printWidth: 80, // 每行代码长度（默认80）
  tabWidth: 2, // 每个tab相当于多少个空格（默认2）
  useTabs: false, // 是否使用tab进行缩进（默认false）
  singleQuote: false, // 使用单引号（默认false）
  semi: true, // 声明结尾使用分号(默认true)
  trailingComma: 'es5', // 多行使用拖尾逗号（默认none）
  bracketSpacing: true, // 对象字面量的大括号间使用空格（默认true）
  jsxBracketSameLine: false, // 多行JSX中的>放置在最后一行的结尾，而不是另起一行（默认false）
  arrowParens: "avoid", // 只有一个参数的箭头函数的参数是否带圆括号（默认avoid）
};
```

1.5.4 测试

利用jest和@vue/test-utils测试组件

安装依赖

```
"jest": "^24.0.0",
"vue-jest": "^5.0.0-alpha.3",
"babel-jest": "^26.1.0",
"@babel/preset-env": "^7.10.4",
"@vue/test-utils": "^2.0.0-beta.9"
```

配置babel.config.js

```
module.exports = {
  presets: [
    [
      "@babel/preset-env", {
        targets: {
          node: "current"
        }
      }
    ]
  ],
};
```

配置jest.config.js

```
module.exports = {
  testEnvironment: "jsdom",
  transform: {
    "^.+\\.vue$": "vue-jest",
    "^.+\\js$": "babel-jest",
  },
  moduleFileExtensions: ["vue", "js", "json", "jsx", "ts", "tsx", "node"],
  testMatch: ["**/tests/**/*.spec.js", "**/__tests__/**/*.spec.js"],
  moduleNameMapper: {
    "^main(.*)$": "<rootDir>/src$1",
  },
};
```

启动脚本

```
"test": "jest --runInBand"
```

测试代码，tests/example.spec.js

```
import HelloWorld from "main/components/HelloWorld.vue";
import { shallowMount } from "@vue/test-utils";

describe("aaa", () => {
  test("should ", () => {
    const wrapper = shallowMount(HelloWorld, {
      props: {
        msg: "hello,vue3",
      },
    });
    expect(wrapper.text()).toMatch("hello,vue3");
  });
});
```

lint配置添加jest环境

```
module.exports = {
  env: {
    jest: true
  },
}
```

将lint、test和git挂钩

```
npm i lint-staged yorkie -D
```

```
"gitHooks": {
  "pre-commit": "lint-staged",
  "pre-push": "npm run test"
},
"lint-staged": {
  "*.{js,vue}": "eslint"
},
```

> *正常情况下安装 yorkie 后会自动安装提交钩子。如果提交钩子未生效可以手动运行 `node node_modules/yorkie/bin/install.js` 来安装。 当然，你也可以运行 `node node_modules/yorkie/bin/uninstall.js` 来卸载提交钩子。*

1.5.5 typescript整合

Vite可直接导入 `.ts` 文件，在SFC中通过 `<script lang="ts">` 使用

范例：使用ts创建一个组件

```
<script lang="ts">
import { defineComponent } from 'vue'

interface Course {
  id: number;
  name: string;
}

export default defineComponent({
  setup() {
    const state = ref<Course[]>([]);
    setTimeout(() => {
      state.value.push({ id: 1, name: "全栈架构师" });
    }, 1000);
  },
});
</script>
```

ts版本指定，package.json

```
{
  "devDependencies": {
    "typescript": "^3.9.7"
  }
}
```

ts参考配置，tsconfig.json

```
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
    "moduleResolution": "node",
    "isolatedModules": true,
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "experimentalDecorators": true,
    "lib": ["dom", "esnext"]
  },
  "exclude": ["node_modules", "dist"]
}
```

1.5.6 项目配置

项目根目录创建 `vite.config.js`，可以对vite项目进行深度配置。

定义别名

导入的别名，避免出现大量相对路径，优雅且不易出错

给 `src/components` 定义别名，vite.config.js

```
const path = require("path");

module.exports = {
  alias: {
    // 路径映射必须以/开头和结尾
    "/comps/": path.resolve(__dirname, "src/components"),
  },
};
```

使用

```
import CourseAdd from "/comps/CourseAdd.vue";
import Comp from "/comps/Comp.vue";
```

代理

配置服务器代理，vite.config.js

```
export default {
  proxy: {
    '/api': {
      target: 'http://jsonplaceholder.typicode.com',
      changeOrigin: true,
      rewrite: path => path.replace(/^\/api/, '')
    }
  }
}
```

使用

```
fetch("/api/users")
  .then(response => response.json())
  .then(json => console.log(json));
```

数据mock

安装依赖

```
npm i mockjs -S
npm i vite-plugin-mock cross-env -D
```

引入插件，vite.config.js

```
plugins: [
  createMockServer({
    // close support .ts file
    supportTs: false,
  }),
],
```

设置环境变量，package.json

```
"dev": "cross-env NODE_ENV=development vite"
```

创建mock文件，mock/test.js

```
export default [
  {
    url: "/api/users",
    method: "get",
    response: req => {
      return {
        code: 0,
        data: [
          {
            name: "tom",
          },
          {
            name: "jerry",
          },
        ],
      };
    },
  },
  {
    url: "/api/post",
    method: "post",
    timeout: 2000,
    response: {
      code: 0,
      data: {
        name: "vben",
      },
    },
  },
];
```

模式和环境变量

使用模式做多环境配置，vite serve时模式默认是development，vite build时是production。

创建配置文件.env.development

```
VITE_TOKEN=this is token
```

代码中读取

```
import.meta.env.VITE_TOKEN
```

1.5.7 打包和部署

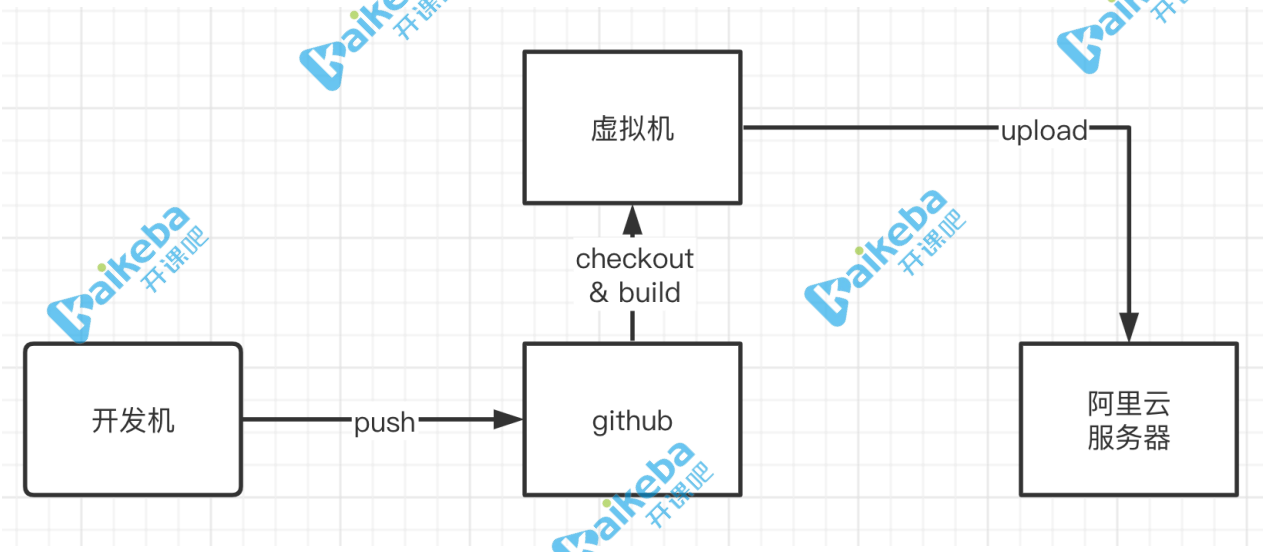
打包

使用 `npm run build` 执行打包

部署

手动上传dist中的内容到服务器，再配置好nginx当然可以，但是这一过程最好自动化处理，避免前面这些繁琐的操作。我们这里利用github actions实现ci/cd过程。

> Github Actions 让我们可以在Github仓库中直接创建自定义的软件开发生命周期工作流程。



准备工作：

阿里云linux服务器

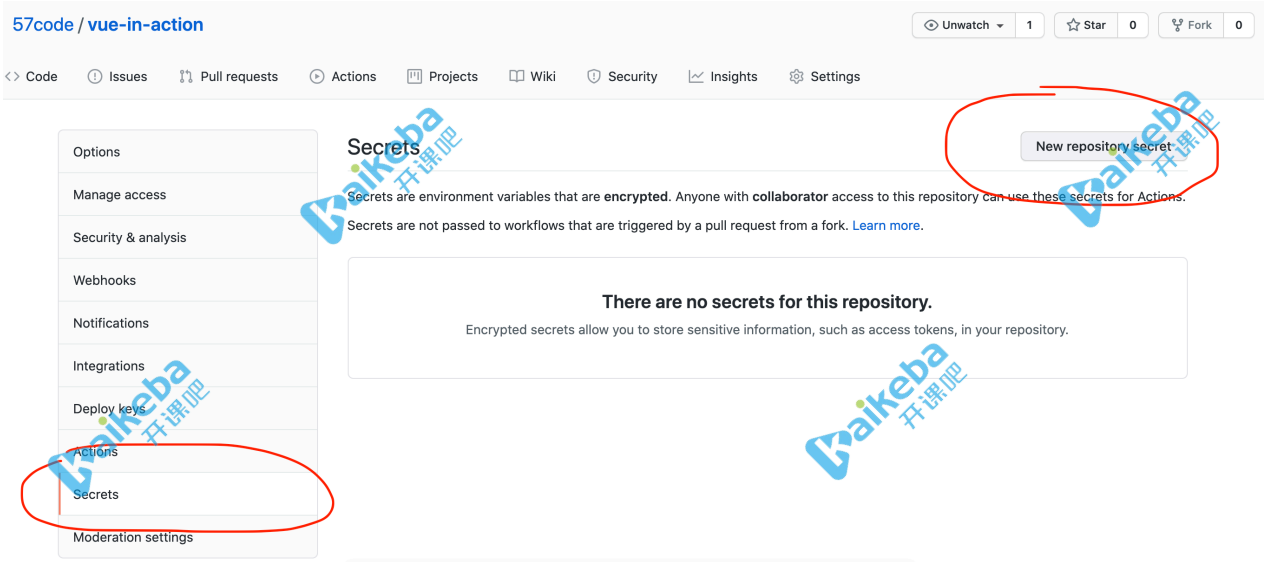
> linux操作
>
> 阿里云相关操作

第一步：配置workflow，下面的配置可以在我们push代码时自动打包我们应用并部署到阿里云服务器上，在项目根目录下创建 `.github/workflows/publish.yml`

```
name: 打包应用并上传阿里云

on:
  push:
    branches:
      - master

jobs:
  build:
    # runs-on 指定job任务运行所需要的虚拟机环境(必填字段)
    runs-on: ubuntu-latest
    steps:
      # 获取源码
      - name: 迁出代码
        # 使用action库  actions/checkout获取源码
        uses: actions/checkout@master
      # 安装Node10

      - name: 安装node.js
        # 使用action库  actions/setup-node安装node
        uses: actions/setup-node@v1
        with:
          node-version: 14.0.0

      # 安装依赖
      - name: 安装依赖
        run: npm install

      # 打包
      - name: 打包
        run: npm run build

      # 上传阿里云
      - name: 发布到阿里云
        uses: easingthemes/ssh-deploy@v2.1.1
        env:
          # 私钥
          SSH_PRIVATE_KEY: ${{ secrets.PRIVATE_KEY }}
          # scp参数
          ARGS: "-avzr --delete"
          # 源目录
          SOURCE: "dist"
          # 服务器ip: 换成你的服务器IP
```

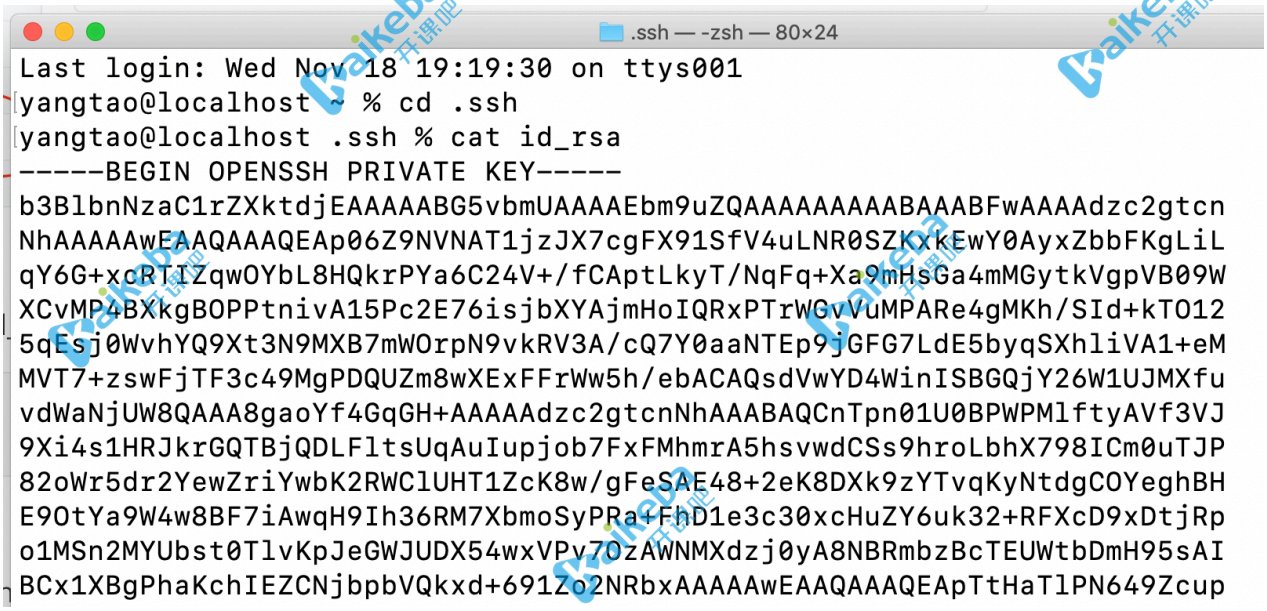
```
        REMOTE_HOST: "47.98.252.43"
        # 用户
        REMOTE_USER: "root"
        # 目标地址
        TARGET: "/root/vue-in-action"
```

第二步：在github当前项目下设置私钥选项

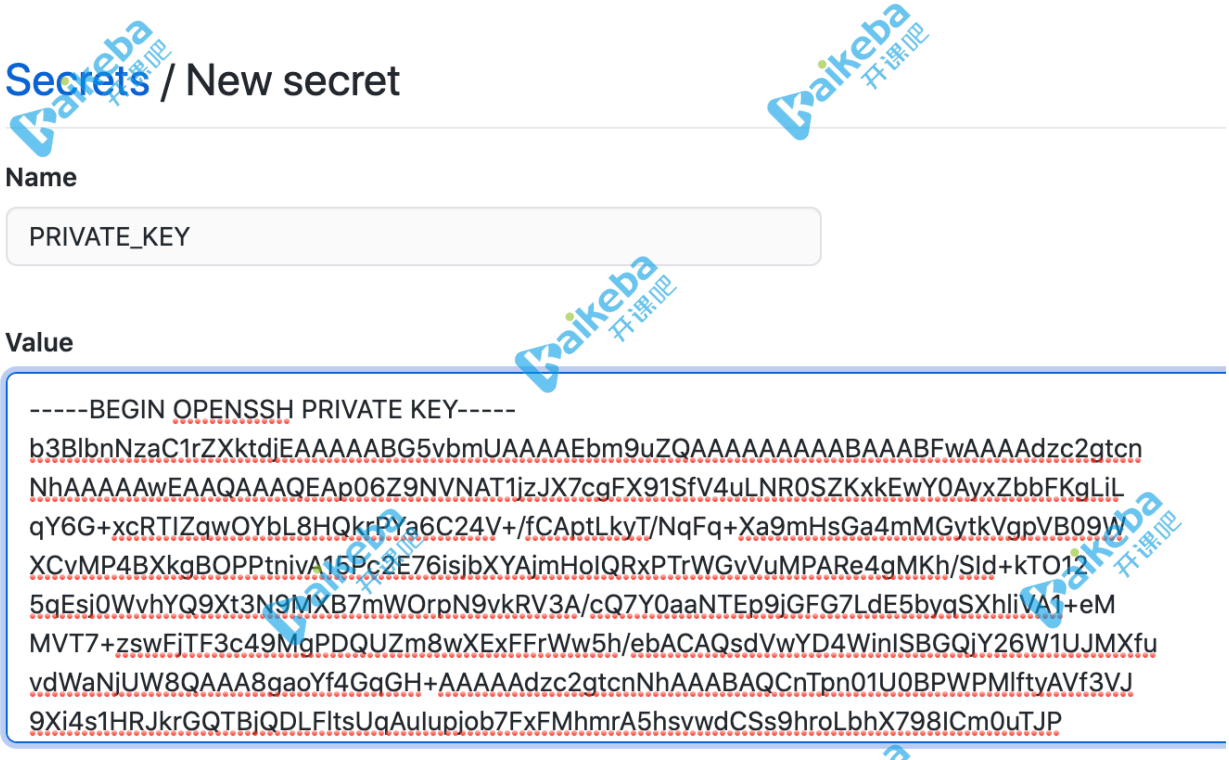


复制本地私钥，~/.ssh/id_rsa

```
# ssh秘钥生成过程自行百度
cd .ssh/
cat id_rsa
```



复制并填写到 `github-secretes`



第三步：在阿里云服务器上配置nginx

登录服务器

```
ssh root@47.98.252.43 # ip换成你的
```

配置nginx

```
cd /etc/nginx/sites-enabled/
vi vue-in-action
```

添加如下配置

```
server {
    listen 8080;
    server_name 47.98.252.43;
    location / {
        root /root/vue-in-action/dist/;
        index index.html index.htm;
    }
}
```

> 重启nginx：nginx -s reload

第四步：push代码，触发workflow

大功告成，验证上传结果

```
root@iZbp1ec6kwgaoplsfi8yqhZ:~/vue-in-action# cd dist
root@iZbp1ec6kwgaoplsfi8yqhZ:~/vue-in-action/dist# ls
01-hello.html   06-v-model.html      11-cart-composition.html  16-plugin.html
02-watch.html   07-cart.html         12-mixin.html             _assets
03-class.html   08-lifecycle.html    13-directive.html         favicon.ico
04-list.html    09-component.html    14-teleport.html          index.html
05-event.html   10-composition.html  15-render.html
root@iZbp1ec6kwgaoplsfi8yqhZ:~/vue-in-action/dist#
```

访问： 47.98.252.43:8080