小米商城总结

1 环境配置

1.1 Node 环境升降级（Linux）

```
// 下载n模块
npm i -g n
// 稳定版本
n stable
// 最新LTS版本
n lts
// 最新版本
n lates
// 某一个版本
n 10.0.0
```

1.2 Git 安装和配置

官网直接下载安装(不需要配置)

```
git --version // 查看版本
```

1.3 Git 常用命令速查表

```
master：默认开发分支                    Head：默认开发分支
origin：默认远程版本库                  Head^：Head的父提交

// 创建本版库
$ git clone <url>                      #克隆远程版本库
$ git init                             #初始化本地版本库

// 修改和提交
$ git status                           #查看状态
$ git diff                             #查看变更内容
$ git add .                            #跟踪所有改动过的文件
$ git add <file>                       #跟踪指定的文件
$ git mv <old> <new>                   #文件改名
$ git rm <file>                        #删除文件
$ git rm --cached <file>               #停止跟踪文件但不删除
$ git commit -m "commit message"       #提交所有更新过的文件
$ git commit --amend                   #修改最后一次提交

// 查看提交历史
$ git log                              #查看提交历史
$ git log -p <file>                    #查看指定文件的提交历史
$ git blame <file>                     #以列表方式查看指定文件的提交历史

// 撤销
$ git reset --hard HEAD                #撤销工作目录中所有未提交文件的修改内容
$ git checkout HEAD <file>             #撤销指定的未提交文件的修改
```

```
$ git revert <commit>                  #撤销指定的提交

// 分支与标签
$ git branch                           #显示所有本地分支
$ git checkout <branch/tag>            #切换到指定分支或标签
$ git branch <new-branch>              #创建新分支
$ git branch -d <branch>               #删除本地分支
$ git tag                              #列出所有本地分支标签
$ git tag <tagname>                    #基于最新提交创建标签
$ git tag -d <tagname>                 #删除标签

// 合并于衍合
$ git merge <branch>                   #合并指定分支到当前分支
$ git rebase <branch>                  #衍合指定分支带当前分支

// 远程操作
$ git remote -v                        #查看远程版本库信息
$ git remote show <remote>             #查看指定远程版本库信息
$ git remote add <remote> <url>        #添加远程版本库
$ git fetch <remote>                   #从远程库获取代码
$ git pull <remote> <branch>           #下载代码及快速合并
$ git push <remote> <branch>           #上传代码及快速合并
$ git push <remote> :<branch/tag-name>  #删除远程分支或标签
$ git push --tags                      #上传所有标签
```

2 跨域

2.1 CORS 跨域-服务端设置，前端直接调用

后台安装 CORS 插件，设置跨域

```
const cors = reuqire('cors');
app.use(cors({
  origin: ['url1', 'url2'],
}));
```

2.2 JSONP 跨域-前端适配，后台配合

```
npm install jsonp --save-dev  // 安装插件
```

不是 XHR 请求，只是个 js 访问，

```
https://www.imooc.com/activity/servicetime?callback=__jp0
// 传了一个callback = _jp0
// 返回时，将结果保存在_jp0里面
```

```
// 前端设置
import jsonp from "jsonp"
mounted() {
    let url = "https://www.imooc.com/activity/servicetime";
    // axios.get(url).then(() => {});
    jsonp(url, (err, res) => {
      let result = res;
      this.data = result;
    });
  },
```

2.3 接口代理

```
// vue.config.js(配置表)
module.exports = {
  devServer: {
    host: 'localhost',                    // 主机
    port: 8080,                           // 设置端口号
    proxy: {                              // 代理
      '/activity': {                      // 访问/activity时，拦截
        target: 'https://www.imooc.com',  // 目标，代理到哪里去
        changeOrigin: true,               // 是否要将主机头改为目标的url地址
        pathRewrite: {                    // 转发地址
          '/activity': '/activity'        // /activity代理到/activity
        }
      }
    }
  }
}

// 发送请求的url
let url = '/activity/servicetime';
```

3 设置

3.1 安装组件

```
npm install vue-lazyload element-ui node-sass sass-loader vue-awesome-swiper vue-
axios vue-cookie --save-dev
// node-sass安装依赖失败
npm i node-sass --sass_binary_site=https://npm.taobao.org/mirrors/node-sass/
```

3.2 引入组件

```
<!-- html -->
<template>
  <div>
    <!-- NavHeader会被简写成nav-header -->
    <nav-header></nav-header>
    <router-view></router-view>
    <nav-footer></nav-footer>
  </div>
```

```
</template>
<!-- javascript -->
<script>
import NavHeader from './../components/NavHeader'
import NavFooter from './../components/NavFooter'
export default {
  name: 'nav-home',
  components: {
    // NavHeader会被简写成nav-header
    NavHeader,
    NavFooter
  }
}
</script>
```

3.3 Storage 封装

区别：

- 存储大小：Cookie - 4K 、 Storage - 5M
- 有效期：Cookie 拥有有效期， Storage 永久存储
- Cookie 会发送到服务器端，存储在内存中， Storage 只存储在浏览器端
- 路径：Cookie 有路径限制， Storage 只存储在域名下
- API ： Cookie 没有特定的 API ， Storage 有对应的 API
  - Local Storage ：本地存储，一直存在
  - Session Storage ：内容存储，随着浏览器关闭就关闭

3.4 axios 请求

```
// get带参数
axios.get('/user', {
  params: {
    ID: 12345
  }
})
.then(res => {
  console.log(res);
})
.catch(err => {
  console.log(err)
})
.then(() => {
  // always executed
});
```

```
// post带参数
axios.post('/user', {
  firstName: 'Fred',
  lastName: 'Flintstone'
})
.then(res => {
  console.log(res)
})
.catch(err => {
  console.log(err)
});
```

```
// 复合型
axios ({
  method: 'post',
  url: '/user/12345',
  data: {
    firstName: 'Fred',
    lastName: 'flintstone'
  }
});
```

```
// 同时发送多个请求
function getUserAccount() {
  return axios.get('/user/12345');
}
function getUserPermissions() {
  return axios.get('/user/12345/permissions');
}
axios.all([getUserAccount(), getUserPermissions()])
.then(axios.spread((acct, perms) {
  // Both request are now complete
}));
```

```
const instance = axios.create({
  baseURL: 'https://some-domain.com/api/',
  timeout: 1000,
  headers: {'X-Custom-Header': 'foobar'}
});
```

3.5 环境变量

```
// 固定的
development -- 开发模式
test -- 测试模式
production -- 生产模式
```

```
// 自定义的方法
"prev": "vue-cli-service build --mode=prev"
// 创建 .env.prev 文件
NODE_ENV='prev'
```

4.6 Mock设置

> - 开发阶段，为了高效率，需要提前Mock(模拟数据)

> - 减少代码冗余、灵活插拔
> - 减少沟通、减少接口联调时间

import和require的区别：

import是预编译加载，编译的时候，文件就会加载进来，保存在内存在

request执行才加载

4. vue

4.1 模态框

```
<!-- 模态框的子组件，拒听内容使用使用具名插槽实现 -->
<slot name="body"></slot>
<!-- 父组件使用具名插槽 -->
<!-- 使用template,v-slot:name绑定 -->
<template v-slot:body>
  <p>商品添加成功！</p>
</template>
```

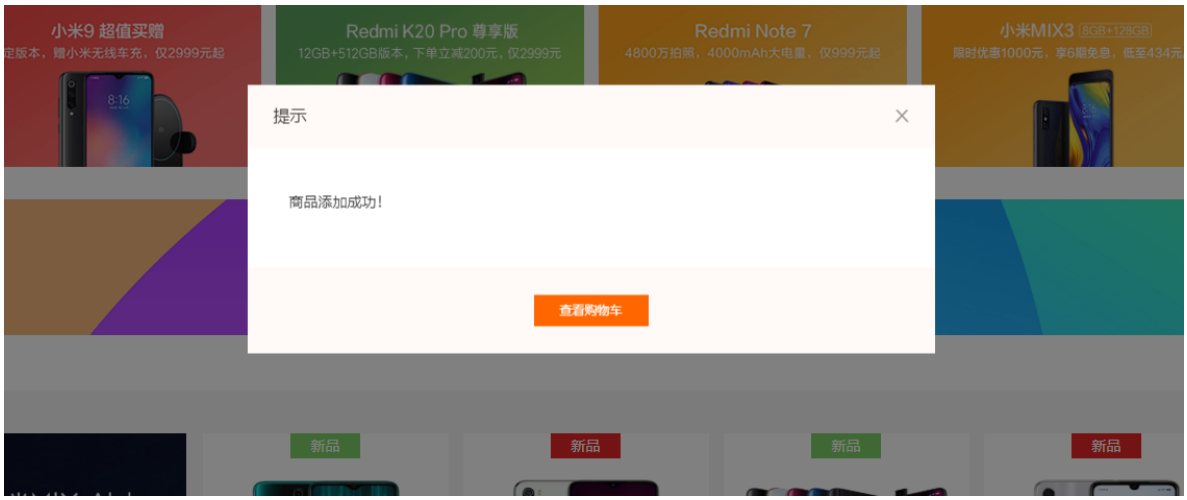
```
<!-- 模态框的显示与隐藏，在子组件中使用v-if -->
<template>
  // showModal为父组件传过来的值，用于判断子组件是否显示
  <div class="modal" v-if="showModal"></div>
</template>

<!-- 点击X关闭模态框，利用子组件向父组件传值 -->
<!-- 子组件，点击时，除法父组件的@cancel -->
<a href="javascript:;" class="icon-close" @click="$emit('cancel')"></a>
<!-- 父组件，当@cancel被触发时，直接将showModal设置为false，使得模态框隐藏起来 -->
<modal @cancel="showModal=false"></modal>
```

```
<!-- 模态框显示/隐藏的过渡动画（vue的transition动画） -->
<!-- 1、给子组件添加一层<transition>标签 -->
<!-- name用于自动生成 CSS 过渡类名,这里的name="slide"将自动拓展为 .slide-enter，.slide-enter-active等 -->
<transition name="slide"></transition>
<!-- 2、要实现动画效果的组件内部使用v-if显示、隐藏 -->
<div class="modal" v-show="showModal"></div>

<!-- 3、css样式 -->
<style lang="scss">
  // 动画效果，从-100%变为0
  // slide-enter-active必须在slide-enter之前，效果才会显示出来
  &.slide-enter-active {
    top: 0;
  }
  &.slide-enter {
    top: -100%;
  }
  &.slide-leave-active {
    top: -100%;
  }
</style>
```

效果图：



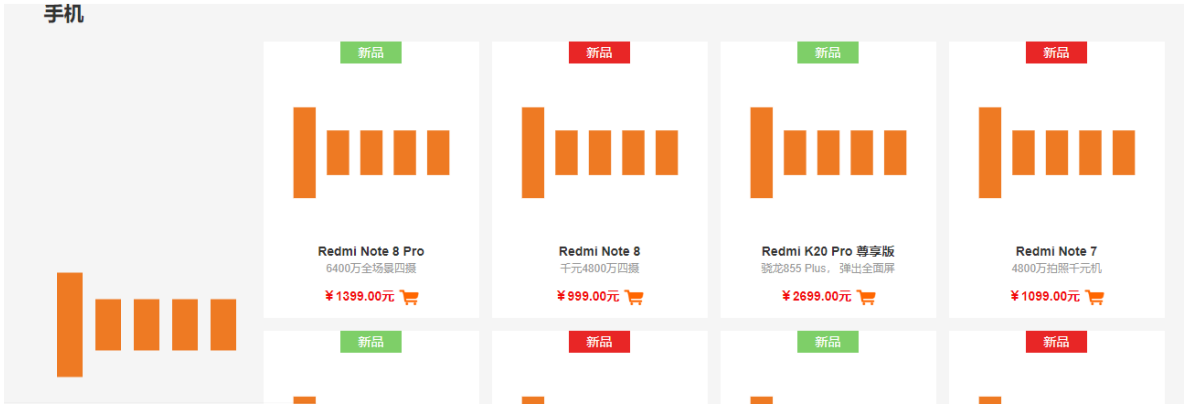
4.2 图片懒加载

```
// 安装插件
npm install vue-lazyload --save-dev
```

```
<!-- 页面中的使用 -->
<img v-lazy="变量(图片地址)" />
```

```
// main.js
import VueLazyLoad from 'vue-lazyload'  // 引入图片懒加载插件
Vue.use(VueLazyLoad, {  // 挂载到Vue实例中
  loading: '/imgs/loading-svg/loading-bars.svg'
})
```

效果图：



4.3 登录

```
<!-- 1、页面，使用v-modal双向绑定获取页面中输入的用户信息 -->
<div class="input">
  <input type="text" placeholder="请输入账号" v-model="username" />
</div>
<div class="input">
  <input type="password" placeholder="请输入密码" v-model="password" />
</div>
```

```
// 2、调用接口，检验用户信息是否正确，正确则将用户信息使用vue-cookie插件生成cookie
login() {
  let { username, password } = this;
  this.axios.post('/user/login', {
    username, password
  }).then((res) => {
    // 利用vue-cookie插件，将userId设置为cookie（1M，一个月过期）
    this.$cookie.set('userId', res.id, {expires: '1M'});
    // to-do保存用户名
    this.$router.push('/index');
  });
},
```

结果截图：

Name	Value	Domain
userId	7	localhost
JSESSIONID	9D1609EDA39C8CE05D404BC761AA8B1E	localhost

4.4 使用 vuex 完善登录功能

`Vuex` 是一个专为 `Vue.js` 应用程序开发的**状态管理模式**。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。`Vuex` 也集成到 `Vue` 的官方调试工具 `devtools extension`，提供了诸如零配置的 `time-travel` 调试、状态快照导入导出等高级调试功能

```
// store文件夹
// index.js
import Vue from 'vue';
import Vuex from 'vuex';
import mutations from './mutations';
import actions from './action';
Vue.use(Vuex);

const state = {
  username: '', // 登录用户名
  cartCount: 0, // 购物车商品数量
}

export default new Vuex.Store({
  state,
  mutations,
  actions,
});

// mutations.js
export default {
  // state状态，表示index.js中的state属性，传入username改变state.username的值
  saveUserName(state, username) {
    state.username = username;
  },
  saveCartCount(state, count) {
    state.cartCount = count;
  }
}

// action.js
```

```
export default {
  // context表示上下文
  // 使用commit触发mutations.js中的saveUserName方法，传入参数username
  saveUserName(context, username) {
    context.commit('saveUserName', username);
  },
  saveCartCount(context, count) {
    context.commit('saveCartCount', count);
  }
}
```

```
// 页面中触发action.js中的方法，并且传值过去
// 方法1
this.$store.dispatch('saveUserName', res.username);
// 方法2(适用于方法属性较多的)
import { mapActions } from 'vuex'
// 解构获得方法
...mapActions(['saveUserName', 'saveCartCount'])
// 使用解构得到的方法
this.saveUserName(res.username);
```

```
// vuex会在页面刷新后消失，所以，需要在app.vue文件中写入数据到vuex中
methods: {
  getUser() {
    this.axios.get('/user').then(res => {
      // to-do 保存在vuex里面
      if(res != undefined)
        this.$store.dispatch('saveUserName', res.username);
    })
  },
    getCartCount() {
      this.axios.get('/carts/products/sum').then(res => {
        // to-do 保存在vuex里面
        this.$store.dispatch('saveCartCount', res);
      })
    }
}
```

```
// 页面中获取vuex中是数据
import { mapState } from 'vuex'
// app.vue中请求数据存在延迟，实际得到数据会页面渲染数据慢，使用计算属性computed来解决这一问题
computed: {
   ...mapState(['username', 'cartCount'])
},
```

结果截图：

4.5 吸顶

```
// 原理：
// 1、监听页面的滚动事件
// 2、滚动到一定高度时，设置定位fixed
// 3、退出页面时，销毁事件
data() {
```

```
    return {
      isFixed: false,
    }
  },
  mounted() {
    // 绑定一个scroll事件，浏览器滚动的时候能被监听到
    window.addEventListener('scroll', this.initHeight)
  },
  methods: {
    initHeight() {
      let scrollTop = window.pageYOffset || document.documentElement.scrollTop ||
document.body.scrollTop;
      this.isFixed = scrollTop > 152 ? true : false;
    }
  },
  destroyed() {
    // 页面关闭时销毁，移除事件,防止在主页等其他不必要的页面也触发这个事件
    // 第三个参数false表示冒泡（默认是true捕获）
    window.removeEventListener('scroll', this.initHeight, false)
  }
```

```
<!-- 动态绑定类：is_fixed，当滚动到指定高度时，isFixed为true，is_fixed起作用 -->
<div class="nav-bar" :class="{'is_fixed': isFixed}"></div>
<style>
  .is_fixed {
    position: fixed;
    top: 0;
    width: 100%;
    box-shadow: 0 5px 5px $colorE;
  }
</style>
```

效果截图（吸顶前\后）：

4.6 动画效果(`animation`)

```
<!-- 页面设置 -->
<template>
  <div class="video-bg" @click="showSlide='slideDown'"></div>
  <!-- 1、v-show="showSlide控制整体是否显示" -->
  <div class="video-box" v-show="showSlide">
    <div class="overlay"></div>
    <!-- 2、样式绑定：点击图片时，showSlide="slideDown",整体显示，并且下面多出一个slideDown的
类 -->
    <div class="video" :class="showSlide">
      <!-- 点击时，触发函数closeVideo -->
```

```
      <span class="icon-close" @click="closeVideo"></span>
      <video src="/imgs/product/video.mp4" muted autoplay controls="controls">
</video>
    </div>
    </div>
</template>
```

```
// 关键样式
.video-box {
  // 过渡动画
  @keyframes slideDown {
    from {
      top: -50%;
      opacity: 0;
    }
    to {
      top: 50%;
      opacity: 1;
    }
  }
  @keyframes slideUp {
    from {
      top: 50%;
      opacity: 1;
    }
    to {
      top: -50%;
      opacity: 0;
    }
  }
  .video {
    // 过渡样式
    &.slideDown {
      animation: slideDown 0.6s linear;
    }
    &.slideUp {
      animation: slideUp 0.6s linear;
    }
  }
}
```

```
data() {
  rteurn {
    showSlide: ''
  }
},
methods: {
  closeVideo() {
    this.showSlide = "slideUp";
    // 等动画效果走完后，设置为空，视频模态框不显示
    setTimeout(() => {
      this.showSlide = "";
    }, 600);
  }
}
```

效果截图：

4.7 组件与插件

组件：是HTML的标签，放在HTML中引入，放到HTML中(例如: `Vue.use(Card);`

插件：帮我我们生成一个标签的(例如:先导入;后: `Vue.prototype.$message = Message;`)