Welcome

> just this wel

基本使用 (https://docsify.js.org/#/)

1. 全局安装docsify

npm i docsify-cli -g

2. 初始化项目

docsify init ./docs

初始化成功后，可以看到 ./docs 目录下创建的几个文件

- index.html 入口文件
- README.md 会做为主页内容渲染
- .nojekyll 用于阻止 GitHub Pages 会忽略掉下划线开头的文件

直接编辑 docs/README.md 就能更新网站内容，当然也可以写多个页面。

3. 本地预览网站

运行一个本地服务器，通过 docsify serve 可以方便的预览效果，而且提供 LiveReload 功能，可以实时的预览。默认通过 http://localhost:3000访问。

CRA(create-react-app)

1.安装

```
npx create-react-app my-app
cd my-app
npm start
```

2.使用

```
npx create-react-app my-app
```

3.开发

- 在vscode中使用装饰器tslint报错的解决方法，跟下创建jsconfig.json

```
{
  "compilerOptions": {
    "experimentalDecorators": true
  }
}
```

4.部署

- history模式下

```
loaction / {
    root root/paths;
    index index.html;
    # history 模式下，刷新页面请求的服务地址可能报404，此时需要重定向到初始化位置
    try_files $uri /index.html;
  }
```

Config

```
npm install react-app-rewired customize-cra --save-dev
```

- 修改package.json

```
// origin
"scripts": {
  "dev": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test --env=jsdom",
  "eject": "react-scripts eject"
}

// changes
"scripts": {
  "start": "react-app-rewired start",
  "build": "react-app-rewired build",
  "test": "react-app-rewired test",
  "eject": "react-app-rewired eject"
}
```

- 根目录下创建config-overrides.js

```
const {
  override,
  addWebpackAlias,
  addDecoratorsLegacy // 关于装饰器的
} = require('customize-cra')

const path = require('path')

function resolve (dir) {
  return path.join(__dirname, '.', dir)
}

module.exports = override(
  addWebpackAlias({
    '@': path.resolve(__dirname, 'src'),
    '_s': path.resolve(__dirname, 'src/store'),
    '_c': path.resolve(__dirname, 'src/components')
  }),

  addDecoratorsLegacy()
)
```

React-Router

```
npm install react-router-dom
```

- 简单实用
    - basic

```
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Link
} from 'react-router-dom'

<Router>
  <Link to="/pathName">menuName</Link>

  <Switch>
    <Route path="/pathName">
      <CompName />
    </Route>
  </Switch>
</Router>
```

    - nest

```
import {
  BrowserRouter as Router,
  Switch,
  Route,
  Link,
  useRouteMatch,
  useParams
} from 'react-router-dom'

<Router>
  <Link to="/pathName">menuName</Link>

  <Switch>
    <Route path="/pathName">
      <NestCompName />
    </Route>
  </Switch>
</Router>

function CompNestName () {
  let match = useRouteMatch()

  <Link to="/pathName">MenuName</Link>

  <Switch>
    <Route path={`${match.path}/:id`}>
      <NestChild />
    </Route>
    <Route path={match.path}></Route>
  </Switch>
}

function NestChild () {
  let {id} = useParams()
  return <div>{id}</div>
}
```

    - useRouteMatch(): get path,url,isExact,params
    - useParams(): get Route path paramsKey value of link tag to paramsValue

- 实战
    - routers

      引入路由，导出路由数组
      path为'/'需要放在Switch下最后一个Route? ?

    - container

      采用nest模式

```
<Router>
  <Switch>
    <login />
    <license />
    <layout />
      <head />
      <side /> [NavLink]
        {routerArr}
      <main /> [Switch Route Redirect]
        <Switch>
          { routerArr}
          <Redirect component="NotFound" />
        </Switch>
    <Redirect from="/" to="/Login">
  </Switch>
</Router>
```

- login & auth

  > 首次container只加载了跟下相应的路由，登陆跳转到nest下时候利用初始化initialization（constructor）来加载路由，并在componentWillUnmount删除路由

- keepAlive

  > 第三方组件 react-keep-alive (https://github.com/StructureBuilder/react-keep-alive/blob/master/README.zh-CN.md)

- 路由传递数据

  - 单纯跳转

    ```
    from: this.props.history.push(path, {key: val})
    to: this.props.location.state.key
    ```

  - 父子

    ```
    父组件：
      changeState = (e) => {
        this.setState({
          message: e
        })
      }
      <Child message="haha" callback={this.changeState}/>
    子组件：
      this.props.message
      this.props.callback(message)
    ```

  - 兄弟

```
子组件：
  this.props.message
  this.props.callback(message)
```

- 监听props变化

```
目标：数据变化后执行某些事件
```

```
componentDidUpdate (preProps, preState, snapShot) {
  consoloe.log(preProps, preState, this.props.message)
}
```

Code Splitting

```
npm install @loadable/component
```

- 应用

```
// router.js
import React from 'react'
const aa = loadable(() => import('aa.js'), {falback: <Loading />})
```

Redux & React-Redux

```
npm install react-redux
npm install redux
```

- actions

```
关于store的命令声明
```

```
export const SET_COUNT = 'SET_COUNT'

// action 创建函数
export function setCount (count) {
  return {
    type: SET_COUNT,
    count
  }
}
```

- reducers

```
接受命令进行操作，并将数据发送到store
```

```
import {SET_COUNT} from './actions'
const initialState = {
  count: 1
}

function setCount (count = 1, action) {
  switch (action.type) {
    case SET_COUNT:
      return action.count
    default:
      return count
  }
}

export default function reducers (state = initialState, action) {
  return {
    count: setCount(state.count, action)
  }
}
```

- 应用
    - App.js

      ```
      import {Provider} from 'react-redux'
      import {createStore} from 'redux'
      import Reducers from './store/reducers'
      ...
      let store = createStore(Reducers)

      <Provider store={store}>
        <Container />
      </Provider>
      ```

    - Count
      /index.js

```
import { setCount } from '../../store/actions'
import CountComponent from './Count.js'
import { connect } from 'react-redux'

const mapStateToProps = state => {
  return {
    count: state.count
  }
}

const mapDispatchToProps = dispatch => {
  return {
    setCount (count) {
      dispatch(setCount(count))
    }
  }
}
// connect 传递对象
export default connect(
  mapStateToProps,
  mapDispatchToProps
)(CountComponent)
```

/Count.js

```
// 接收容器传递过来的参数
const CountComponent = ({count, setCount}) => {
  return (
    <div className="Count-con">
      <h2>Count: {count}</h2>
      <button onClick={
        e => {
          e.preventDefault()
          setCount(++count)
        }
      }>increase</button>

      <button onClick={
        e => {
          e.preventDefault()
          setCount(--count)
        }
      }>decrease</button>
    </div>
  )
}
```

- @connect

```
引用装饰器，减少容器文件
```

```
import {connect} from 'react-redux'

@connect(state => state) // dispatch 可以直接继承，但是state需要手动写入?
class aa extends React.Component {
  ...
  this.props.count
  this.props.dispatch(actionName(value))
  ...
}
...
```

Proxy

- PlanA: edit package.json

```
"proxy": "protocol://domain:port/",
```

- PlanB: 根目录下创建setupProxy.js

```
npm install http-proxy-middleware
```

```
const { createProxyMiddleware } = require('http-proxy-middleware')

module.exports = function (app) {
  app.use(createProxyMiddleware(
    '/user', {
      target: 'http://192.168.1.99/',
      changeOrigin: true,
      pathRewrite: {
        '^/': ''
      }
    }
  ))

  app.use(createProxyMiddleware(
    '/api', {
      target: 'http://192.168.1.99/',
      changeOrigin: true,
      pathRewrite: {
        '^/': ''
      }
    }
  ))
}
```

- 应用

```
ajax.post('/user/login/', {})
ajax.get('/api/home_data/', {})
```

Sass & 行内样式

```
npm install node-sass --save-dev
```

- 应用
    - 创建sass
      文件名字.module.scss
    - 引用
      import Style from name.module.scss

```
<span className={Style.haha}>haha~</span>
```

- 行内样式

```
const styleObj = {color: 'red'}
<img style={styleObj} />
```

Vue

学习 (https://www.cnblogs.com/huangfeihong/p/9141273.html)

img引入问题

```
如果是图标，应用iconfont实现
```

- template、script、css

```
国旗和大的图片存在于static，小的图片使用import引入
```

```
/static/
```

- 打包后控制img显示问题 webpack.base.js

```
{
  test: /\.(png|jpe?g|gif|svg)(\?.*)?$/,
  loader: 'url-loader',
  options: {
    limit: 10000, // B,当size低于这个值，图片都会变成base64，即使是css引入的
    name: utils.assetsPath('img/[name].[hash:7].[ext]')
  }
}
```

- 图片太大

> 放置static中；使用压缩后的图片压缩网站 (https://tinypng.com)

引入无法npm安装的第三方插件

> 需要在index.html中根节点之后，script标签引入

e.g.

```
<script src="./static/d3Graphviz/d3.js"></script>
<script src="./static/d3Graphviz/viz.js" type="javascript/worker"></script>
<script src="./static/d3Graphviz/d3-graphviz.js"></script>
```

组件

1.父组件 =props数据=> 子组件

> 传递到组件中的数据props，最好是子组件正好能应用的

- 在父组件使用子组件的地方

```
v-bind:resName="fatherData"
:resName="fatherData"
```

- 子组件

```
props: ['resName'],
create () {
  console.log( this.resName )
}
```

2.子组件传递 =命令=> 父组件

- 子组件：this.$emit('eventName', {'para1': 'para1'}, id);
- 在父组件使用子组件的地方添加 @eventName="fatherMethod"

```
export default {
  methods: {
     fatherMethod (res, id) {
       console.log(res.para1, id)
     }
  }
}
```

3.绑定动态class,style

```
 :class="{class-name: Boolean}"
 :class="[{},{}]"
 :style="{option: data-option}"
```

4.动态组件

- 组件名称和要绑定的数据放在数组里面

```
export deafult {
  data () {
    comps: [
      {
        'componentName': 'com1',
        res: res1
      }
    ]
  }
}
```

```
<component
  :is="i.componentName"
  v-for="(i,j) in comps">
</component>
```

5.组件跳转

1. click event

```
v-on:click
@click=""
```

2. 直接跳转

```
this.$router.push({
  path: '/component-name'
})
```

3. query

> 参数path, query

```
this.$router.push({
  path:'/knowledge',
  query:{para1:'para1-val'}
})

this.$route.query.para1
```

4. params

> 参数name，params
> 目标路由刷新会忘记参数，需要用到动态路由匹配，在配置路由时候增加固定的参数

```
// routers.js

{
  path: '/knowledge/:id',
  ...
}

// 组件
this.$router.push({
  name:'knowledge',
  params:{id: '1992'}
})
this.$route.params.para2
```

插槽 slot

> 针对vue@^2.6.0

子组件

> name定义插槽名字，d等其他属性名表示传递的prop

```
<slot name="slotName" d="aa"></slot>
```

父组件

> v-slot:插槽名称 插槽作用域: props

```
<custom-comp>
  <template v-slot:slotName="props">
    {{props.d}}
  </template>
</custom-comp>
```

Router

默认显示组件

```
{
  path: '*',
  redirect: 'home'
}
```

缓存

利用route的meta组件属性设置keeplive布尔值

```
<keep-alive>
  <router-view v-if="$route.meta.keepAlive"></router-view>
</keep-alive>

<router-view v-if="!$route.meta.keepAlive">
    <!-- 这里是不被缓存的视图组件，比如 page3 -->
</router-view>
```

- router.js

```
{
  path: '/welcome',
  name: 'welcome',
  component: welcome,
  meta: {
    keepAlive: true
  }
}
```

缓存后，各个路由页面滑动的页面距离一致

```
const router = new Router{
  scrollBehavior (to, from, savedPosition) {
    return { x: 0, y: 0 }
  }
}
```

动态路由

案例

类似阿里云邮箱，菜单和邮件列表都能打开新的tab完整代码 ｜ [codepen预览(https://codepen.io/unzoa/project/editor/AMvWQg#)](https://codepen.io/unzoa/project/editor/AMvWQg#)

1. 路由的path后面必须有对应的参数, ':'为界

```
{
  name: 'Xxx',
  path: '/Xxx/:params1/:params2'
}
```

2. 组件内获取，直接使用 $route.paramsd对象

3. 多段动态路由, 夹杂静态str

```
{
  path: '/Xx/:pramas1/haha/:params2'
}
```

4. 响应路由参数的变化

> 原来的组件实例会被复用，不过，这也意味着组件的生命周期钩子不会再被调用。

- 解决方案1: 可以简单地 watch (监测变化) $route 对象

```
watch: {
  $route(to, from) {
    // 对路由变化作出响应...
  }
}
```

- 解决方案2: 使用 2.2 中引入的 beforeRouteUpdate 导航守卫

```
beforeRouteUpdate (to, from, next) {
  // react to route changes...
  // don't forget to call next()
}
```

5. 通配符

```
{
  // 会匹配所有路径
  path: '*'
}
{
  // 会匹配以 `/user-` 开头的任意路径
  path: '/user-*'
}
```

含有通配符的路由应该放在最后

当使用一个通配符时，$route.params 内会自动添加一个名为 pathMatch 参数.

```
// 给出一个路由 { path: '/user-*' }
this.$router.push('/user-admin')
this.$route.params.pathMatch // 'admin'
// 给出一个路由 { path: '*' }
this.$router.push('/non-existing')
this.$route.params.pathMatch // '/non-existing'
```

Vuex

创建store文件夹

```
idnex.js
getters.js
actions.js
mutations.js
```

main.js

```
import store from './vuex'
new vue({
    ...
    store
    ...
})
```

index.js

```
import Vue from 'vue'
import Vuex from 'vuex'
import * as getters from './getters'
import * as actions from './actions'
import * as mutations from './mutations'

Vue.use(Vuex)

const state={
    count:0
}

const store = new Vuex.Store({
    state,
    getters,
    actions,
    mutations
})

export default store
```

getters.js

```
计算属性computed中调用
```

```
export const count = state => state.count
```

mutations.js

```
tate只能在这里被编辑
```

```
/**
 * @param  {[type]} state [store的state]
 * @param  {[type]} value [组件传递]
 */
export const increment = (state, value) => { state.count += value }
export const decrement = state => { state.count-- }
```

actions.js

> 请求接口数据在action中进行

```
import ajax from '../ajax.js'

export const increment = ({ commit }, value) =>{
    ajax.get(apiName, {
        value
    }).then(res => {
        if (res.status === 200) {
            commit('increment', res.num)
        }
    })
}
```

应用

```
import {mapGetters, mapActions} from 'vuex'
computed:{
    ...mapGetters(['state-name'])
},
methods: {
    ...mapActions(['mutation-type'])
}
```

直接应用方法

```
// getters
this.$store.state.obj

// mutations
this.$store.commit('mutationFunction', params)

// actions
this.$store.dispatch('actionsFunction', params)
```

请求的应用

axios

```
npm install axios --save-dev
```

api.js

> 导出一个json对象

```
export default {
  'login': 'login/', // 登陆接口 post
  'gene_analysis': 'api/uploadsample/' // 文件上传
}
```

axios请求

```
import Axios from 'axios'

export default {
  get (Interface, requestData = {}) {
    return new Promise(resolve => {
      Axios.get(Interface, {
        params: requestData,
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded'
        }
      })
        .then(res => {
          resolve(res)
        }).catch(res => {
          resolve(res)
        })
    })
  },

  blob (Interface, requestData = {}) {
    let configOri = {
      params: requestData,
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded;charset=utf-8'
      },
      responseType: 'blob'
    }
    return new Promise(resolve => {
      Axios.get(Interface, configOri)
        .then(r => {
          resolve(window.URL.createObjectURL(new Blob([r])))
        }).catch(res => {
          resolve(res)
        })
    })
  },

  post (Interface, requestData = {}) {
    return new Promise(resolve => {
```

```
    Axios.post(Interface, requestData, {
      transformRequest: [function (requestData) { // 转换数据格式，有待测试传送文件的方式时候同样可行。
        let ret = ''
        for (let it in requestData) {
          ret += encodeURIComponent(it) + '=' + encodeURIComponent(requestData[it]) + '&'
        }
        ret = ret.slice(0, ret.length - 1)
        return ret
      }],
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      }
    })
      .then(res => {
        resolve(res)
      }).catch(res => {
        resolve(res)
      })
  })
},

upload (Interface, formData, config) {
  return new Promise(resolve => {
    Axios.post(Interface, formData, config)
      .then(res => {
        resolve(res)
      }).catch(res => {
        resolve(res)
      })
  })
}
}
```

axios拦截

```
import Axios from 'axios'
import api from '@/config/common/api.js'
import router from '@/config/common/router'

function TOKEN () {
  return localStorage['token']
}

Axios.interceptors.request.use( // 请求拦截器----------
  config => {
    let mock = false

    if (config.method === 'post') {
      if (Object.prototype.toString.call(config.data) === '[object FormData]') {
        config.data.append('token', TOKEN())
      } else {
        config.data.token = TOKEN()
```

```
    }

    !config.data.mock || (mock = true)
  } else if (config.method === 'get') {
    config.params = {
      ...config.params,
      token: TOKEN()
    }
    !config.params.mock || (mock = true)
  }

  // 判断mockjs拦截
  mock || (config.url = api[config.url])

  return config
 },
 error => {
  return Promise.reject(error)
 }
)

Axios.interceptors.response.use( // 响应拦截器----------
 response => {
  let data = response.data || response

  if (typeof data === 'string' && response.headers['content-disposition']) { // 判断是流文件
    // 获取流文件的名称
    let streamFileName = response.headers['content-disposition'].split(';')[1].split('=')[1].replace(/"/g, '')
    localStorage.setItem('streamFileName', streamFileName)
  }

  // 401 跳转到login
  if (data.status === 401) {
    // 清理信息存储
    ...

    router.push({path: '/Login'}).catch(err => (err))
  }

  return data
 },
 error => {
  // return error // 返回请求失败信息
  // Message.error('服务连接失败...')
  // return Promise.reject(error)
 }
)
```

跨域

config/index.js配置dev:

```
{
  ...
  proxyTable:{
    '/api': {
      target: 'http://192.168.1.26',
      changeOrigin: true
    },
    '/user': {
      target: 'http://192.168.1.26',
      changeOrigin: true
    }
  }
  ...
}
```

vue-resource

```
npm install vue-resource --save-dev
```

```
// main.js
import VueResource from 'vue-resource'
Vue.use(VueResource)
Vue.http.options.emulateJSON = true;

// use in component
this.$http.post(url,{}).then((res)=>{})
```

打包

打包app.js vendor文件过大问题解决方案

针对vue-cli2

- Step 1. webpack.prod.conf.js

```
new HtmlWebpackPlugin({
  ...
  isProduction: process.env.NODE_ENV === 'production' // index.html判断条件，是否加入external的模块
})
```

- Step 2. index.html

```
<!-- <% if (htmlWebpackPlugin.options.isProduction) { %> -->
  <script src="/static/vendor/vue.min.js"></script>
  <script src="/static/vendor/vue-router.min.js"></script>
  <script src="/static/vendor/vuex.min.js"></script>
  <link rel="stylesheet" href="/static/vendor/element-ui/theme-chalk/index.css" >
  <script src="/static/vendor/element-ui/index.js"></script>
<!-- <% } %> -->
```

- Step 3. webpack.base.conf.js

```
// 区分开发和打包模式
let externals = {}
if (process.env.NODE_ENV === 'production') {
  externals = {
    'vue': 'Vue',
    'vue-router': 'VueRouter',
    'vuex': 'Vuex',
    'element-ui': 'ELEMENT',
  }
}

module.exports = {
  ...
  externals: externals,
  ...
}
```

- Step 4. 修改element-ui的样式

```
if (process.env.NODE_ENV !== 'production') { // 打包时候不引入，利用index.html下elementui
  require('element-ui/lib/theme-chalk/index.css')
}
```

项目中引入Electron

> This project was generated with [electron-vue](https://github.com/SimulatedGREG/electron-vue)@[7c4e3e9](https://github.com/SimulatedGREG/electron-vue/tree/7c4e3e90a772bd4c27d2dd4790f61f09bae0fcef) using [vue-cli](https://github.com/vuejs/vue-cli). Documentation about the original structure can be found [here](https://simulatedgreg.gitbooks.io/electron-vue/content/index.html)

1. 安装模块

```
npm install electron@^2.0.18 --save-dev
npm install electron-packager@^12.2.0 --save-dev
```

2.调用electron-quick-start

> 把electron-quick-start项目中的main.js搬到vue的build文件中，并改个名字，这是最终dist下的文件electron.js，与build下差异就在与入口文件的地址。

```
// and load the index.html of the app.
// 意思就是加载首页文件，这里绑定打包后的dist/index.html
mainWindow.loadURL(url.format({
  pathname: path.join(__dirname, '../dist/index.html'),
  protocol: 'file:',
  slashes: true
}))
```

3.编辑package.json

> 在package.json文件中增加两条指令，electron-dev和electron-build，这是最终的文件package.json

```
 "scripts": {
   "dev": "node build/dev-server.js",
   "start": "npm run dev",
   "build": "node build/build.js",
   "lint": "eslint --ext .js,.vue src",
   "electron_dev": "npm run build && electron build/electron.js",
   "electron_build": "electron-packager ./dist/ --platform=win32 --arch=x64 --icon=./src/assets/favicon.ico --
overwrite"
 }
```

3.1 关于electron-packager的配置

> electron-packager sourcedir appname –platform=platform –arch=arch --icon=icon [optional flags…]

- sourcedir： 资源路径，在本例中既是 ./dist/
- appname： 打包出的App名称
- platform： 平台名称（windows是win32） (mac是darwin) （linux）
- arch： 版本，本例为x64
- icon： App的图标，格式ico，相对路径, 尺寸256 x 256

4.打包出/dist,并配置electron

```
npm run build
```

1. 在dist文件夹下添加electron-quick-start下的的main.js改名electron.js和package.json
2. package.json里面的main属性，指向从electron.js

```
{
  "name": "RFID",
  "version": "1.0.0",
  "description": "A minimal Electron application",
  "main": "electron.js",// 注意此处
  "scripts": {
    "start": "electron ."
  },
}
```

3. electron.js 更改app的入口文件路径

```
// electron.js
mainWindow.loadURL(url.format({
    pathname: path.join(__dirname, 'index.html'), // here
    protocol: 'file:',
    slashes: true
}))
```

5.electron_build

```
npm run electron_build
```

其他

1. 配置窗口

```
function createWindow () {
  // Create the browser window.
  Menu.setApplicationMenu(null)
  mainWindow = new BrowserWindow({
    width: 900,
    height: 630,
    minWidth: 900,
    minHeight: 630,

    resizable: false,
    fullscreen: false,
    fullscreenable: false,

    frame: false
  })
  ...
```

2. 事件,配置electron.js

```
  const ipc = electron.ipcMain
  ipc.on('customEventName', function () {
    // electron api useage
  })
```

3. 使用

- index.html 由于electron中引用问题，开发时暴露会出错

```
<% if (htmlWebpackPlugin.options.isProduction) { %>
 <script>
  window.electron = require('electron')
 </script>
<% } %>
```

- Head.vue中应用

```
const {ipcRenderer: ipc} = window.electron ll {
 ipcRenderer: {
  send () {
   console.log('development mode.')
  }
 }
}

// 执行想要的动作
ipc.send('customEventName')
```

4. static问题

```
打包后[.exe]找不到static下文件
因为resorce/app 需要引用的static路径错误，未找到正统解决办法，这里只有一个字体文件，故手动修改
```

5. inno打包

- 下载inno
- 安装并运行
- 执行
    - File -> New
    - page: application information
        - Apllication name: 安装时候程序的名称
        - [next]
    - page: application folder [next]
    - page: application files
        - browser: find .exe
        - add folder: find full files folder
        - [next]
    - page: application shortcuts [next]
    - page: application documentation [next]
    - page: setup install mode [next]
    - page: setup languages
        - choose what u want

        - [next]
    - page: compiler settings
        - custom compiler output folder
        - compiler output base file name
        - custom setup icon file
        - [next]
    - page: Inno setup preprocessor [next]
    - [finish]

6. 跨域问题

> 前端解决跨域，本项目a页面创建 iframe(目标，携带参数) 目标执行api请求后得到token，创建iframe(本项目b页，携带token等参数)，b页执行存储，a、b同源，所以a定时获取即可。

当目标https证书在谷歌显示不安全情况下，需要对electron设置忽略证书安全性：

```
const {app} = require('electron')
app.commandLine.appendSwitch('ignore-certificate-errors') // 针对不安全证书问题


...
 new BrowserWindow({
   webPreferences: {
     webSecurity: false // 可以针对 not allowed to load local resource
   }
 })
...
```

创建自己的npm包

1.安装一个纯净的带有babel,eslint的vue-cli

2.设置库的构建，在project.json中配置

```
"scripts": {
 "serve": "vue-cli-service serve",
 "build": "vue-cli-service build",
 "build-bundle": "vue-cli-service build --target lib --name unzoa-ui ./src/components/index.js",
 "lint": "vue-cli-service lint"
}
```

build-bundle命令中, --name 接指定要库的名字

3.后面接库的入口文件，使其在倒入时候自动注册vue组件

```
import Vue from 'vue'
import TopBar from './TopBar.vue'
const Components = {
  TopBar
}

Object.keys(Components).forEach(name => {
  Vue.component(name, Components[name])
})

export default Components
```

4.然后配置package.json中，增加main属性，当我们引入该组件库时，默认加载main的路径文件

```
"main": "./dist/unzoa-ui.common.js"
```

5.配置package.json中files属性，配置发布到npm上的路径

```
"files": [
  "dist/*",
  "public/*",
  "src/*",
  "*.json",
  "*.js"
]
```

6.执行npm run build-bundle

7.发布

此时应该有一个npm账号，没有就去npm创建一个

- 执行npm login, 输入用户名，密码
- 执行npm whoami 验证
- 执行npm publish --access public, 记得把package.json中private改为false, 以设置为公开库

8.测试应用

```
npm install unzoa-ui -D
```

```
import 'unzoa-ui'
```

> 组件库编写时候，就直接注册了，所以此时可以直接应用了 <TopBar />

9. 组件库更新

组件库是有版本号的，每次更新需要先更新版本号

- 查看版本 npm view unzoa-ui versions
- 更新版本号 原1.0.0

    - npm version []
        - patch 增加补丁 -> 1.0.1
        - minor 这个是小修小改 -> 1.1.0
        - major 这个是大改咯 -> 2.0.0

- npm publish
- npm view unzoa-ui versions

> ['0.1.0', '0.1.1']

10.删除包

取消发布包可能并不像想象得那么容易，这种操作是受到诸多限制的，撤销发布的包被认为是一种不好的行为（试想一下你撤销了发布的包[假设它已经在社区内有了一定程度的影响]，这对那些已经深度使用并依赖你发布的包的团队是件多么崩溃的事情！）

首先如果就是想要删除当前的这个版本，执行命令npm unpublish unzoa-ui，去官网查看发现已经没有这个包了，如果权限不够加上 --force

- 根据规范，只有在发包的24小时内才允许撤销发布的包（ unpublish is only allowed with versions published in the last 24 hours），需要我们发邮件给官方来删除
- 即使你撤销了发布的包，发包的时候也不能再和被撤销的包的名称和版本重复了（即不能名称相同，版本相同，因为这两者构成的唯一标识已经被“占用”了）
- 例如我在撤销包后尝试再发布同一名称+同一版本的包

小程序

点击事件

```
bindtap="tap" data-pp="pp"

tap (e) {
  let data = e.currentTarget.dataset
  let pp = data.pp
}
```

组件 Components

> 根目录下创建Components文件夹

```
|-Components
|-|-aa
|-|-|-aa.js
|-|-|-aa.json
|-|-|-aa.wxml
|-|-|-aa.wxss
```

aa.js

- options => multipleSlots: true // 在组件定义时的选项中启用多slot支持

```
<slot name="haha"></slot>

<view slot="haha"></view>
```

- properties => 类似于Vue的props，这里属性需要表明数据类型
- data => 组件内参数
- methods => funcs

页面利用组件

需要在页面json中添加配置信息

```
{
  "usingComponents": {
    "aa": "/components/aa/aa"
  }
}
```

数据问题

- 子组件请求公共数据，其他组件数据同步变化

  > 不能利用app.js作为类似vuex, 函数可请求，但是数据不会变化，也不会监听

- 父组件传递的数据可以同步到子组件中

注意事项

- 当设置app.json中设置了tabBar时，页面内再去控制跳转是不起作用的

Wex5

> Java 部署

Step 1. 部署包root/mysql/my.ini 修改数据库端口-#port=... => port = xxxx
（为防止和已有的数据库乱掉）

Step 2. x5-窗口-首选项-数据源-修改端口

Step 3. 打包

```
  ip-服务器+8080
  weburl- /myitem
```

Step 4. 部署发布包

```
UI- Native下myitem/www/ 复制到部署包apache-tomcat/webapps/ ,将部署包apache-tomcat/webapps/app-template/WEB-INF文件夹复制到项目下
BaasServer- x5root/runtime/BaasServer拷贝到部署包 runtime下
数据库配置文件- x5root/apache-tomcat/conf/context.xml 拷贝到部署包同位
数据库驱动文件- x5root/apache-tomcat/lib/ mysql-connector-java-5.1.36-bin.jar拷贝到部署包同位
```

Step 5. IIS部署

```
域名或者ip+端口 指向
```

Mock

> 单独编辑mock.js，并引用即可

```
import Mock from 'mockjs'

function getHappy (config) {
  return { happy: 'm really happy!' }
}

Mock.mock(/getHappy[\s\S]*?/, 'get', getHappy) // 正则匹配api名称携带参数
```

Js 导出 docx

模块文件

GSAP

Installation:

```
npm i gsap -D
# version ^3.4.2
```

- Use in vue:资料 (https://blog.usejournal.com/vue-js-gsap-animations-26fc6b1c3c5a)
- main use:

```
import gsap from 'gsap'

// gsap(target, duration, vars)
gsap.to('.logo', 1, {x: number})
```

- use plugins:

```
import gsap from 'gsap'
import MotionPathPlugin from 'gsap/MotionPathPlugin'

gsap.to(
  '.logo',
  1,
  {
    motionPath: {
      path: '.path',
      aligin: '.path',
      alignOrigin: [0.5, 0.5], // logo的重心沿着path移动
      autoRotate: true // logo的横向方向和path水平
    },
    repeat: -1, // 重复动画
    ease: 'power1' // 动画的速度方案
  }
)
```

屡试不爽组件

颜色类

1. javascript-color-gradient (https://www.npmjs.com/package/javascript-color-gradient)

   渐变颜色

```
const colorGradient = new Gradient();

const color1 = "#3F2CAF";
const color2 = "#e9446a";
const color3 = "#edc988";
const color4 = "#607D8B";

colorGradient.setMidpoint(20);

colorGradient.setGradient(color1, color2, color3, color4);

// outputs ["#4e4ab9", "#5d68c4", "#6d86ce", "#7ca4d9", "#8bc2e3", ...] 20个
```

2. rgb-hex (https://www.npmjs.com/package/rgb-hex)

```
const rgbHex = require('rgb-hex');

rgbHex(65, 131, 196);
//=> '4183c4'
```

声音类

1. howler (https://www.npmjs.com/package/howler)

```
var sound = new Howl({
  src: ['sound.webm', 'sound.mp3']
});

sound.play();
```

数据类

1. dayjs (https://www.npmjs.com/package/dayjs)

```
dayjs(val).format('YYYY/MM/DD HH:mm:ss')
```

服务类

1. http-server (https://www.npmjs.com/package/http-server)

```
npm install --global http-server

# 启动
http-server

# 或者更改端口
http-server -p 9099
```

样式类

组件类

CSS

1. 优先级

!important>>行内>>页内>>link

2. 响应式布局

- meta 介绍 (http://caibaojian.com/mobile-meta.html)
- media query
- 单位
    - html {font-size:百分数;} 百分数=自定义基数/浏览器默认字体
    - rem就给html设置font-size，用em就给body设置font-size
    - 百分比 %
    - 文字缩进：2em 就是两个文字
- img
    - 没必要同时设置宽、高，因为需要保持图片的原始比例

- layout
    - 清除float，可在父级直接添加overflow: hidden; 不用添加多余的html元素去clear: both;
    - display: inline-block; 块级元素边行内元素
    - display: flex; 介绍 (https://www.cnblogs.com/qingchunshiguang/p/8011103.html)
    - display: grid; 介绍 (https://developer.mozilla.org/zh-CN/docs/Web/CSS/CSS_Grid_Layout) l 详细 (https://www.jianshu.com/p/41c038baf994)

3. position

值	描述
static	默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right 或者 z-index 声明）。
relative	生成相对定位的元素，相对于其正常位置进行定位。因此，"left:20" 会向元素的 LEFT 位置添加 20 像素。
absolute	生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位。 元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。
fixed	生成固定定位的元素，相对于浏览器窗口进行定位。元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。
sticky	粘性定位，该定位基于用户滚动的位置。 它的行为就像 position:relative; 而当页面滚动超出目标区域时，它的表现就像 position:fixed;，它会固定在目标位置。 注意: Internet Explorer, Edge 15 及更早 IE 版本不支持 sticky 定位。 Safari 需要使用 -webkit- prefix (查看以下实例)。
inherit	规定应该从父元素继承 position 属性的值。

4. 动画属性

名称	属性
transform	translate、 rotate、 scale、 skew
transition	property duration timing-function delay
animation	custom-name duration timing-function delay teration-count direction:alternate @keyframes custom-name{from{}to{} or 0%{}100%{}}

timing-function	解释
linear	规定以相同速度开始至结束的过渡效果（等于 cubic-bezier(0,0,1,1)）
ease	规定慢速开始，然后变快，然后慢速结束的过渡效果（cubic-bezier(0.25,0.1,0.25,1)）
ease-in	规定以慢速开始的过渡效果（等于 cubic-bezier(0.42,0,1,1)）
ease-out	规定以慢速结束的过渡效果（等于 cubic-bezier(0,0,0.58,1)）
ease-in-out	规定以慢速开始和结束的过渡效果（等于 cubic-bezier(0.42,0,0.58,1)）
cubic-bezier(n,n,n,n)	在 cubic-bezier 函数中定义自己的值。可能的值是 0 至 1 之间的数值

4.1. 动画停留

```
/*动画*/
.act-in{
    animation: act-in .5s ease;
    animation-fill-mode: forwards;
}
@keyframes act-in {
    0%{left: -50%;}
    100%{left: 0;}
}
.act-out{
    animation: act-out .5s ease;
    animation-fill-mode: forwards;
}
@keyframes act-out {
    0%{left: 0;}
    100%{left: -50%;}
}
/*动画结束*/
.actor-dancer {
    position: relative;
    left: -50%;
}
```

5. 改变placeholder颜色

```
::-webkit-input-placeholder { /* WebKit, Blink, Edge */
    color: #fff!important;
}
:-moz-placeholder { /* Mozilla Firefox 4 to 18 */
    color: #fff!important;
    opacity: 1;
}
::-moz-placeholder { /* Mozilla Firefox 19+ */
    color: #fff!important;
    opacity: 1;
}
:-ms-input-placeholder { /* Internet Explorer 10-11 */
    color: #fff!important;
}
```

6. 加省略号

```
<p class='h_news_title'>bala</p>
```

```
.h_news_title {
    width: 55%;

    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
}
```

7. 滚动条

```
::-webkit-scrollbar {
    width: 0px;
    height: 1px;
}
::-webkit-scrollbar-thumb {
    border-radius: 5px;
    -webkit-box-shadow: inset 0 0 5px rgba(0, 0, 0, 0.2);
    background: rgba(0, 0, 0, 0.2);
}

/*定义滚动条高宽及背景 高宽分别对应横竖滚动条的尺寸*/
::-webkit-scrollbar {
    width: 16px;
    height: 16px;
    background-color: #F5F5F5;
}

/*定义滚动条轨道 内阴影+圆角*/
::-webkit-scrollbar-track {
    -webkit-box-shadow: inset 0 0 6px rgba(0,0,0,0.3);
    border-radius: 10px;
    background-color: #F5F5F5;
}

/*定义滑块 内阴影+圆角*/
::-webkit-scrollbar-thumb {
    border-radius: 10px;
    -webkit-box-shadow: inset 0 0 6px rgba(0,0,0,.3);
    background-color: #555;
}
```

8. 毛玻璃

```
.frosted-glass {
  -webkit-filter: blur(5px);
  -moz-filter: blur(5px);
  -ms-filter: blur(5px);
  -o-filter: blur(5px);
  filter: blur(5px);
  filter: progid:DXImageTransform.Microsoft.Blur(PixelRadius=4, MakeShadow=false);
}
```

9. 移动端

- 解决ios下样式不一样 -webkit-appearence:none;
- 解决ios移动端滑动动画问题 -webkit-overflow-scrolling:touch

10. padding 增加实际容器高度问题

A：写padding之前，增加box-sizing: border-box;

JavaScript

技巧

1. && ll 替换if

```
let a = true
let b = !a

if (a) {}
// 相当于
a && ...

if (!b) {}
// 相当于
b ll ...
```

注意 在vue中template中v-bind:style中不能使用

基本

1. 关于Dom操作

- 关于节点

```
// 创建新节点
creatElement()
creatTextNode()

// 操作节点
appendChild()
removeChild()
replaceChild()
insertBefore()

// 查找
getElementsByTagName()
getElementsByClassName()
getElementsByName()
getElementById()
querySeletor()

// 可在页面上改变元素内容
document.body.contentEditable='true'

// 获取可视宽高
document.body.offsetWidth
document.body.offsetHeight

// 获取元素的实际
document.querrySelect('.sth').getBoundingClientRect()

// 获取浏览器计算后的元素style
// **得到字符串**
window.getComputedStyle(ele, null).getPropertyValue(key)

// 获取网页可视高度
// https://www.cnblogs.com/ckmouse/archive/2012/01/30/2332070.html
document.documentElement.clientHeight

// 获取页面是否缩放
window.visualViewport.scale // 默认是 1
```

- 应用

```
<ul class="test-ul">
  <li>111</li>
    <li>112</li>
    <li>113</li>
    <li>114</li>
    <li>115</li>
</ul>

<script>
    var lis = document.querySelector('.test-ul li')
    console.log(lis) // <li>111</li>
    var lis2 = document.getElementsByClassName('test-ul')[0].children
    console.log(lis2) // li li li li li
    var lis3 = document.getElementsByTagName('li')
    console.log(lis3) // li li li li li
</script>
```

- jQuery:

  ```
  cdn https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js
  ```

```
$(document).ready(function)

// 当 DOM（文档对象模型） 已经加载，并且页面（包括图像）已经完全呈现时，会发生 ready 事件
// onload 事件会在页面或图像加载完成后立即发生

append
text
html
empty
remove

find('')
children('')
parentsUntil('')
eq()
siblings('')
```

2.Function

```
1. 函数声明 Function Declaration
function func1 () {}
2. 函数表达式 Function Expression
var func2 = function () {}
3. 构造函数
new Function('a', 'b', 'return a + b');
4. 箭头函数
(params) => {}
```

3.Ajax

```
var xhr = new XMLHttpRquest();
xhr.open('get',url,true);
xhr.send();
xhr.onreadystatechange = function(){
    if(xhr.readyState === 4 && xhr.status == 200){
        // do sth
    }
}
```

4.Promise

```
var pro = new Promise((resolve,reject)=>{
    if(true){
        resolve(data)
    }else{
        reject(msg)
    }
})
pro.then((data)=>{

}).catch((msg)=>{

})
```

5.Array & Object

- Array

```
// 直接修改arr
push
pop
shift // 把数组的第一个元素从其中删除
unshit // 向数组的开头添加一个或更多元素
reverse
sort arr.sort((a, b) => { return a - b })
splice(index,count,new1,new2) // 替换原数组项目从index开始count个位置

// 新数组
concat // 连接数组
join // 数组变为字符串
slice(start,end)
set 去重 Array.from(new Set(arr))

arr.forEach( ()=>{} ) // 返回字符串，数组元素
arr.map( ()=>{ return } ) // 加工原数组
filter(function(item){ return type item == 'number'}) // 返回过滤后的数组,满足条件的留下

a.filter(ea=>b.every(eb=>eb!==ea)) // 数组差集 a包含b
```

- Object

```
// keys & values
Object.keys(obj) // 返回一个obj的keys的数组
Object.values(obj) // 返回一个obj的values的数组
Object.entries(obj) // 返回[ [key, value], [key, value]]
function (target) {
    return Object.keys(obj).map(i => {
        return [i, obj[i]]
    })
}

// 融合
Object.assign(obj, ...sources) // 返回融合后的对象
// es5 浏览器兼容
Object.assign = Object.assign ||
function (target) {
    for (var i = 1; i < arguments.length; i++) {
        var source = arguments[i]
        for (var key in source) {
            if (Object.prototype.hasOwnProperty.call(source, key)) {
                target[key] = source[key]
            }
        }
    }
    return target
}

// 删除某个值
delete obj.key1 // obj中删除了key1

// 拷贝
// 1、利用JSON（推荐，JS语言自支持，不需要依赖其他工具）
let newObj = JSON.parse(JSON.stringify(oldObj))

// 2、Lodash（推荐，在某些集成了Loadsh的开发环境代码显得更简洁）
let newObj = _.cloneDeep(oldObj)

// 3、ES6的对象拓展运算符：...（有坑，慎重使用，参考[关于ES6的拓展运算符进行深拷贝]
(https://blog.csdn.net/zomixi/article/details/84064255)）
let newObj = {...oldObj}

// 4、ES6的对象拓展运算符：Object.assign()（有坑，同上）
let newObj = Object.assign({}, oldObj)
```

6.JSON

```
JSON.parse() 解析一个JSON字符串，构造由字符串描述的JavaScript值或对象
JSON.stringify()
```

7.模块化

```
commonJs
   服务器端模块规范，一个文件是一个模块，例如Node.js
   同步加载，程序在所有模块都加载完毕后才执行
amd
cmd
```

8.call & apply & bind

> 每个函数都包含，并且是非继承下来的
> 执行函数person，函数内this指向thisObject, 携带的参数arg

```
function person (name = 'bob', sex = 'dragon') {
   console.log(name + ' is ' + sex + ' and ' + this.look)
}
```

- call 主构造函数call(目标构造函数)，即：目标构造函数继承了主构造的属性

```
// 携带的参数arg需要枚举
person.call(thisObject, arg1, arg2)
```

- apply

```
// 参数需要是数组
person.apply(thisObject, argArray)
```

- bind 函数继承对象的属性

```
const a = {
   x: 1
}

function b () {
   console.log(this.x)
}

b.bind(a)()
```

CANVAS

1. 解决加载图片模糊问题

- 获取 canvas

```
let canvasCon = document.querySelector('.canvas-con')
let canvas = document.getElementById('canvas-id')
context = canvas.getContext('2d')
```

- 获取设备的 ratio 的方法

```
let getPixelRatio = (context) => {
  let backingStore = context.backingStorePixelRatio ||
    context.webkitBackingStorePixelRatio ||
    context.mozBackingStorePixelRatio ||
    context.msBackingStorePixelRatio ||
    context.oBackingStorePixelRatio ||
    context.backingStorePixelRatio || 1
  return (window.devicePixelRatio || 1) / backingStore
}
```

- 可以使得 canvas 携带 ratio

```
canvas.ratio = getPixelRatio(canvas)
```

- 设置 canvas 实际宽高

```
canvas.height = canvasCon.offsetHeight * canvas.ratio
canvas.width = canvasCon.offsetWidth * .6 * canvas.ratio
```

- 设置 canvas 样式宽高

```
canvas.style.height = canvasCon.offsetHeight + 'px'
canvas.style.width = canvasCon.offsetWidth * .6 + 'px'
```

- 将 canvas 缩小一半，相当于图像扩大二倍

```
context.scale(canvas.ratio, canvas.ratio)
```

SVG

> SVG 文件必须使用 .svg 后缀来保存

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">

<circle cx="100" cy="50" r="40" stroke="black"
stroke-width="2" fill="red"/>

</svg>
```

代码解释：

1. 第一行包含了 XML 声明。请注意 standalone 属性！该属性规定此 SVG 文件是否是“独立的”，或含有对外部文件的引用。standalone="no" 意味着 SVG 文档会引用一个外部文件 - 在这里，是 DTD 文件。

2. 第二和第三行引用了这个外部的SVGDTD。该DTD位于
[http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd]
(http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd)。该DTD位于W3C，含有所有允许的SVG 元素。

3. SVG 代码以 <svg> 元素开始，包括开启标签 <svg> 和关闭标签 </svg> 。这是根元素。width 和 height 属性可设置此 SVG 文档的宽度和高度。version 属性可定义所使用的 SVG 版本，xmlns 属性可定义 SVG 命名空间。

4. SVG 的 <circle> 用来创建一个圆。cx 和 cy 属性定义圆中心的 x 和 y 坐标。如果忽略这两个属性，那么圆点会被设置为 (0, 0)。r 属性定义圆的半径。

5. stroke 和 stroke-width 属性控制如何显示形状的轮廓。我们把圆的轮廓设置为 2px 宽，黑边框。
fill 属性设置形状内的颜色。我们把填充颜色设置为红色。

6. 关闭标签的作用是关闭 SVG 元素和文档本身。

鉴权

1.HTTP Basic Authentication：用的比较少，平常FTP登录是用的这种方式吧？感觉可以用在内部网系统。

2.session-cookie：这个在老的系统见得多，只适用于web系统。以前用java servlet写服务端时候，都会自动维护session，会在cookie写一个JSESSIONID的值。

3.Token：现在主流都是用这个，适用于app鉴权，微信开发平台access token也是差不多这种思路。

4.OAuth：这个是趋势吧，现在想要推广自己的应用都先接入微信 QQ等登录，降低用户使用门槛。特别是微信渠道的手游，都是接入了微信开发授权登录。

Event Loop

- 宏任务

```
// 整体代码、setTimeout、setInterval、promise.then
```

- 微任务

```
// promise
```

区别：

- 微任务是元素
- 宏任务是一个集合，可以包着多个元素

执行：
元素立即执行，即 “微任务”立即执行，宏任务排后。

注意：

1. promise的执行函数注意微任务，then后的异步属于宏任务

入口entry

> webpack 采用模块化思想，搜有文件和配置都是一个个模块，同时联系在一起。可以简单的理解为一颗树状结构，那么对开始就是一个根（入口文件）entry
> webpack在执行构建的时候
> 1.找到入口文件
> 2.根据入口开始，寻找、遍历、递归解析出所有入口依赖的模块

```
module.export = {
  entry: {
    app: './src/main.js'
  },
  // 还有output，module等其他配置
}
```

- 静态entry

类型	例子	含义
string	'./app/entry'	可以是相对路径
array	['./app/entry', './app/entry2']	?
object	{a: './app/entryA', b: './app/entryA'}	配置对歌入口，每个入口都生成一个chunk

- 动态entry

```
// 同步函数
entry () {
  return {
    a: '',
    b: ''
  }
}

// 异步函数
entry () => {
  return new Promise(resolve => {
    resolve({
      a: '',
      b: ''
    })
  })
}
```

output

> 类型：只能是一个对象，包含输出配置

module

> 用于配置处理模块的规则：
> 1.配置loader
> 2.配置noParse
> 3.配置parse

1. 配置loader

- use:
  多个loader时，执行顺序是 （左<=右）
  ['style-loader', 'css-loader', 'sass-loader']执行'sass-loader' => 'css-loader' => 'style-loader'

- enforce:
  可以用enforce强制某个loader的执行在最前面(post)还是最后面(pre)

- cacheDirectory:
  传给babel-loader参数，用于缓存babel的编译结果，加快编译的速度

```
module: {
  rules: [
    {
      test: /\.js$/,
      inlude: path.resolve(__dirname, 'src'),

      // use 可以是普通字符串数组，也可以是对象数组
      use: ['babel-loader?cacheDirectory'],
      use: [
        {
          loader: 'babel-loader',
          options: {
            cacheDirectory: true
          },
          enforce: 'post'
        }
      ]
    },

    {
      test: /\.scss$/,
      use: ['style-loader', 'css-loader', 'sass-loader'],
      exclude: path.resolve(__dirname, 'node_modules')
    },

    {
      // 对非文件文件采用file-loader 加载
      test: /\.(gif|png|jpe?g|eot|woff|ttf|svg|pdf)$/,
      use: ['file-loader']
    },

    // 配置更多的loader
  ]
}
```

2. 配置noParse
   noParse 让webpack忽略不采用的模块化的文件，不进行编译处理，例如：JQuery没有采用模块化标注，webpack解析耗时无意义

```
module: {
  rules: [],

  noParse: /jquery|lodash/,
  noParse: content => {
    return /jquery/.test(content)
  }
}
```

注意
被忽略文件不应包含 import、require、define 等模块化语句，不然会导致在构建出代码中包含无法在浏览器环境下执行模块化语句

3. 配置parse

webpack以模块化的js文件为入口，所以内置了对模块化js的解析，支持AMO，CommonJS, SystemJS, ES6。parse属性可以更细粒度配置哪些模块需要解析。
同noParse配置项区别于，parse可以精确到语法层面，noParse只能控制哪些文件不被解析。

```
module: {
 rules: [
  {
   test: /\.js$/,
   use: ['babel-loader'],
   parse: [
    amd; false, // 禁用AMD
    commonjs: false,
    system: false,
    harmony: false
   ]
  }
 ]
}
```

注意

> parse和noParse是同级属性，可以嵌套到rules，表示针对某个loader应用该属性。

resolve

> 配置webpcak如何寻找模块中对应的文件，例如：通过import导入的模块，resolve告诉webpcak去解析

1. alias

> 配置路径别名

```
resolve: {
  alias: {
    'components': './src/components',
    'react
: '/path/to/react.js',
    '@': './src'
  }
}
```

2. extentions

> 配置文件后缀
> 在引用文件时候，例如vue文件 import Ha from '@/view/Ha' Ha.vue 在路径中可以不用写'.vue'文件后缀

```
resolve: {
  extentions: ['.js', '.json', '.vue']
}
```

webpack会一次找对应路径下该名称的文件 Ha.js => Ha.json => Ha.vue

3. modules

> 配置webpack哪些目录查找第三方模块，默认只会查找node_modules。

```
resolve: {
  modules: ['./src/components/', 'node_modules']
}
```

这样就可以更简单的书写了 import Ha from 'Ha' 和从node_modules中引入一样

4. descriptionFiles

> 配置描述第三方模块的文件名称 默认package.json

5. enforceExtention

> 配置后缀名是否必须加上

plugin

> plugins 包含很多webpcak自己的插件，也有其他开源的插件，致力于解决loader之外的相关构建的事
> plugins 是一个数组，可以传入多个插件

```
const CommonsChunkPplugin = require('webpack/lib/optimize/CommonsChunk{lugin')

modules: {
  plugins: [
    new CommonsChunkPlugin({
      name: 'common',
      chunks: ['a', 'b']
    }),

    // 也可以配置其他插件
  ]
}
```

devServer

> 主要用于本地的开发，配置本地服务的各种特性。

- hot [boolean] 热更新
- inline [boolean] 实时刷新
- contentBase [boolean] 本地服务的文件根目录
- host [string] 域名
- port [number] 端口
- allowedHosts: [] // 请求域名的范围
- https [boolean] https服务
- compress [boolean] Gzip压缩
- open [boolean] 服务开启后，自动打开浏览器
- devtool: 'source-map' // 配置webpack是否生成sorce map方便调试
- watch [boolean] 监听文件修改后自动编译

注意

> webpack 中那些最易混淆的 5 个知识点 (https://juejin.im/post/5cede821f265da1bbd4b5630#heading-4)

1. module, chunk, bundle的区别

其实就是同一份逻辑代码在不同转换场景下的三个名字：

- 直接写出来的是module
- webpack处理时候的是chunk
- 最后在浏览器生成的是bundle

2. filename, chunkFilename

- filename 是在entry中的，打包后输出的文件的名称
- chunkFilename 是未在entry，却需要被打包的文件的名称

入门配置

1. 全局安装

```
npm i -g webpack webpack-cli
```

2. 项目安装

```
npm init
npm i -D webpack
```

3. 创建目录

```
|- src
|-|- main.js
|- package.json
|- webpack.config.js
```

webpack.config.js

```
const path = require('path')

module.export = {
  entry: './src/main.js',

  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  }
}
```

单页面应用

1. 认识单页面应用

通过入口文件main.js打包成bundle.js，入口文件也是必然要引用在html文件中。
如何创建html文件？怎么创建build以后的html文件？以及html文件如何引入js文件？

[html-webpack-plugin (https://www.webpackjs.com/plugins/html-webpack-plugin/)](https://www.webpackjs.com/plugins/html-webpack-plugin/)

实现功能:
1.不用手动创建html文件，build之后自动生成一个index.html
2.生成的index.html不需要手动引入js文件，会自动创建script标签，并引入生成的bundle.js

2. 创建单页面应用

Step 1. 安装插件

```
npm i html-webpack-plugin -D
```

Step 2. 配置插件

```
const HtmlWebpackPlugin = require('html-webpack-plugin')

plugins: [
  new HtmlWebpackPlugin({
    filename: 'index.html', // 编译后生成的html文件名称
    template: './index.html' // 以哪个文件作为模版，在dist下生成新的html文件
  })
]
```

接入Babel

1. 认识babel

> babel是一个js解释器，可以将es6转化为es5;
> 可以通过插件机制扩展

在babel执行编译过程中，会读取根下 .babelrc的配置

```
{
  "presets": [
    ["env", {
      "modules": false,
      "targets": {
        "browsers": ["> 1%", "last 2 versions", "not ie <= 8"]
      }
    }],
    "stage-2",
    "react"
  ],
  "plugins": [
    "transform-vue-jsx",
    "transform-runtime"
  ]
}
```

- presets: 告诉babel要转换哪些特性，例如‘react’
- plugins: 告诉babel利用哪些插件去转换

2. 接入Babel

- webpack.config.js

```
modules: [
  {
    rules: /\.js$/,
    use: ['babel-loader']
  }
]
```

- 创建 .babelrc

```
{
  "presets": ["env"],
  "plugins": []
}
```

- 安装插件

```
# webpack接入babel必需模块
npm i -D babel-core babel-loader

# 根据需求，安装不同的presets 或者 plugins
npm i -D babel-preset-env
```

接入scss

> scss 即为了我们书写css更方便且更有效率，出现了less，sass，scss等css的预处理器，那么，在最后部署上线的时候，依然要转成css，才可以在浏览器端运行。

Step 1. 安装依赖

```
# webpack loader
npm i -D scss-loader css-loader style-loader

# sass-loader 依赖 node-sass
npm i -D node-sass
```

Step 2. 在modules中配置

```
module: {
  rules: [
    {
      test: /\.scss$/,
      use: ['style-loader', 'css-loader', 'scss-loader']
    }
  ]
}
```

说明:

1. 当有多个loader的时候，编译顺序是从后往前
2. sass-loader: 首先通过sass-loader将scss代码转换成css代码，再交给css-loader处理
3. css-loader 会找出 css 代码中 import 和 url ()这样的导入语句，告诉 Webpack 依赖这些资源 。 同时支持 CSS Modules、压缩 css 等功能 。处理完后再将结果交给 style-loader处理。
4. style-loader会将 css代码转换成字符串后，注入 JavaScript代码中，通过 JavaScript 向 DOM 增加样式。

以上是配置wepbakc接入scss的基本配置， 除此之外，我们还可以加入优化配置：

1. 压缩css
2. css与js代码分离：ExtractTextPlugin

接入vue

Step 1. 安装

```
npm i -D vue

#构建所需的依赖
npm i -D vue-loader css-loader vue-template-compiler
```

说明：

1. vue-loader:解析和转换.vue文件，提取出其中的逻辑代码 script、样式代 码 style及 HTML模板 template，再分别将它们交给对应的 Loader去处理。
2. css-loader:加载由非vue-loader提取出的css代码。
3. vue-template-compiler: 将vue-loader提取出的HTML模板编译成对应的可执行的JavaScript代码

Step 2. 配置loader
webpack.config.js

```
const VueLoaderPlugin = require('vue-loader/lib/')

module: {
  rules: [
    {
      test: /\.vue$/,
      use: ['vue-loader']
    }
  ]
},
plugins: [
  new VueLoaderPlugin()
]
```

Step 3. 引入vue文件

```
//此时就和vue-cli提供的模版main.js一样去引入vue和vue相关文件
import Vue from 'vue'
import App from './app.vue'
new Vue({
    el: '#app',
    render: h => h(App)
})
```

javascript与css处理

1. 分离javascript与css

> 打包entry入口文件的时候，文件中所依赖的css等样式模块也会包含js中.

目标：分离javascript和css，分别存放在不同的文件中。

Step 1. 安装依赖
Webpack 4.x已经不再支持extract-text-webpack-plugin，推荐使用mini-css-webpack-plugin， 如果想继续使用该插件，请 参考该文档 (https://blog.csdn.net/qq_38526769/article/details/82427800)

此处，我们依然用该插件来实现js与css的代码分离：

```
npm i extract-text-webpack-plugin@next --save-dev
```

注意： 后面的@next必须加上，webpack4.x只支持该版本
或者可以用一个新插件： mini-css-extract-plugin

Step 2. 增加webpack配置

webpack.config.js

```
const ExtractTextPlugin = require ('extract-text-webpack-plugin')

module.exports = {
  module:{
    rules:[
      {
        test: /\.css$/,
        use: ExtractTextPlugin.extract({
          fallback: "style-loader",
          use: "css-loader"
        })
      }
    ]
  },
  plugins: [
    new ExtractTextPlugin({
      filename: path.resolve(__dirname, '/style/bundle.css') //注意可以指定分离出来的css文件的指定目录
    })
  ]
};
```

2. 压缩css

Step 1. 安装依赖

```
npm i optimize-css-assets-webpack-plugin --save-dev
```

Step 2. 添加配置

```
const OptimizeCssAssetsPlugin = require('optimize-css-assets-webpack-plugin')
module.export = {
  plugins: [
    //只要配置了该插件，即可对分离出来的css进行压缩
    new OptimizeCssAssetsPlugin()
  ]
}
```

3. 压缩javascript

> 很多教程使用的是UglifyJsPlugin，不过是webpack4.x之前的版本可以使用，webpack4.x已经不支持使用，移除webpack.optimize.UglifyJsPlugin 压缩配置了, 推荐使用optimization.minimize 属性替代。

```
module.exports = {
  optimization: {
    minimize: true // webpack内置属性，默认为true，build之后的js文件压缩
  }
}
```

4. 提取公共javascript

问题 如果每个页面的代码都将这些公共的部分包含进去，则会造成以下问题：

1. 相同的资源被重复加载，浪费用户的流量和服务器的成本。
2. 每个页面需要加载的资源太大，导致网页首屏加载缓慢， 影响用户体验。

开发需求 一些公共的工具函数所在的common.js文件，这个公共文件中的工具函数会被多个页面同时使用，其实，还有一个base.js也就是最基础的文件，例如，我们vue开发过程中的vue.js

所以，此处所说的公共文件包含以下两部分：

1. common.js （手动创建）每次都可能更新
2. base.js （引入的第三方依赖：如vue.js，lodash，jquery等），基本不会动基本库

webpack提取 webpack4.x提取公共代码推荐使用：webpack内置的SplitChunksPlugin插件，4.x之前所使用的CommomsChunksPlugin已被淘汰

实例演示：

Step 1. 我们创建了两个入口文件：main.js 和 index.js， 还有一个公共的common.js文件，同时再安装一个lodash第三方库，然后两个入口文件中，分别引入common.js和lodash;

```
// mian.js和index.js
import './assets/js/common';
import 'lodash';
```

Step 2. 配置多入口文件

```
module.export = {
  entry: {
    main: './src/main.js',
    index: './src/index.js'
  },
  output: {
    filename: '[name].bundle.js',
    path: path.resolve(__dirname, 'dist')
  },
  optimization: {
    splitChunks: {
      chunks: "initial",
      minSize: 0 // webpack 默认30000 3k的文件分离
    }
  }
}
```

检查 js & css

检查 JavaScript

ESLint 最常用的 JavaScript 检查工具是 ESlint (eslint.org)，它不仅内置了大量的常用检查规则，还可以通过插件机制做到灵活扩展。

Step 1.

- 安装

```
npm i eslint --save-dev
```

- 创建.eslintrc

```
{
  //从 eslint :recommended 中继承所有检查规则
  "extends" : "eslint:recommended",
  // 再自定义一些规则
  ”rules”:{
    //需要在每行结尾加 ;
    "semi":["error", "always"] , //需要使用""包裹字符串
    "quotes" : [ "error", "double"]
    }
}
```

- 运行命令

```
// 检查fileName.js文件中存在的错误
eslint fileName.js
```

Step 2. webpack配置

- 安装

```
npm i -D eslint-loader
```

- 添加loader

```
module: {
 rules: [
  {
    test: /\.js$/,
    use: [
     {
       loader:'eslint-loader',
       options: {
         formatter: require('eslint-frinedly-formatter') // 默认的错误提示
       },
       enforce: 'pre', // 编译前检查
       exclude: 'node_modules',
       include: [__dirname + '/src']
     }
    ]
  }
 ]
}
```

检查 CSS

> stylelint (stylelint.io)是目前最成熟的 css 检查工具，在内置了大量检查规则的 同时，也提供了插件机制让用户自定义扩展 。 stylelint 基于 PostCSS，能检查任何 PostCSS 能解析的代码，例如 scss、 Less 等 。

- 安装

```
npm i -D stylelint
```

- 创建.stylelintrc文件

```
{
 //继承 stylelint-config-standard 中所有的检查规则
 "extends":"stylelint-config-standard",
 // 再自定义检查规则
 "rules": {
   "at-rule-empty-line-before": null
 }
}
```

- 执行命令

```
styleLint 'filename.css'
```

搭建本地环境

1. 搭建本地开发环境webpack-dev-server

```
npm i -D webpack-dev-server
```

2. 配置package.json

```
scripts: {
  "dev": "webpack-dev-server --inline --progress --config webpack.dev.config.js"
}
```

说明：此处我们新建一个测试环境的webpack配置文件，用于区分正式环境和测试环境

3. 配置devServer

```
const {merge}  = require('webpack-merge')
const baseWebpackConfig = require('./webpack.config.js')

module.exports = merge(baseWebpackConfig, {
  mode: 'development',
  devServer: {
    port: 9090,
    contentBase: './dist'
  },
  plugins: []
})
```

- 实时刷新

  ```
  刷新整个页面
  ```

```
devServer: {
  inline: true // 实时刷新
}
```

- 热更新

  ```
  刷新部分页面
  ```

```
devServer: {
  hot: true // 热替换
},
plugins; [
  new webpack.HotModuleReplacementPlugin()
]
```

多页面应用

1. 认识多页面应用

首先说明一下单页面和多页面的区别：

单页面：只有一个html文件，页面之前的跳转通过路由机制去控制，平时我们使用vue-cli等脚手架自动生成的模版都是单页面，路由通过vue-router去控制跳转。
多页面：多个html文件，页面之间的跳转是通过浏览器原生的机制去控制，比如在没有vue，react等框架之前，基本都是多页面的开发。

2. 创建多页面应用

方式1：创建多个入口文件，同时结合 html-webpack-plugin 创建多个html文件，实现多页面应用

方式2: 使用web-webpack-plugin 插件

Gulp

> 参考 (https://css-tricks.com/gulp-for-beginners/)

1.安装

```
# 全局安装gulp
 npm install gulp -g
```

```
npm init
npm install gulp --save-dev
npm install gulp-sass --save-dev
```

2.命令

```
#   编译指定路径下的sass文件为css文件
gulp sass

# 监听指定路径下[.scss]文件变化,并立刻输出最新的css文件
gulp watch
```

3.gulpfile.js

```
var gulp = require('gulp');
// 引入依赖包
var sass = require('gulp-sass');
gulp.task('sass', function(){
   //sass()方法用于转换sass到css
  return gulp.src('/app/scss/styles.scss')
    .pipe(sass()) // Converts Sass to CSS with gulp-sass
    .pipe(gulp.dest('app/css'))
});

//Watching Sass files for changes
gulp.task('watch', function(){
  // gulp.watch('app/scss/**/*.scss', ['sass']);
  gulp.watch('app/sass/*.scss', gulp.series('sass')); // gulp 4.0
  // Other watchers
})
```

页面滚动，动态高亮Menu

需求说明：
UI分为两部分

Menu 文本内容（一整个dom块）
菜单1 内容模块1
菜单2 内容模块2
菜单3 内容模块3

Menu中每一项都对应着文本内容的一个模块，需要实现两个功能：

1. 点击menu 跳转到对应模块
2. 滚动文本内容，阅读的文本模块需要在menu中高亮对应的条目

原理：主要针对文本内容滚动，菜单动态高亮问题

> 确定正在阅读文本块：需要文本内容阅读块在距离屏幕最上端开始，在文本块的最下端移动到屏幕最上端结束。

1. 利用dom的 getBoundingClientRect() 得到模块的top，bottom值判断正在阅读文本块
2. let num = 文本块距离屏幕上边缘多少时属于正在阅读
3. 判断条件 top <= num && bottom > 0, 获取这个模块的‘标记’
4. 进而通过‘标记’，对菜单高亮。

操作：

```
// 滚动到目标文本块
menuClickEvent () {}

// 监听页面滚动
window.addEventListener('scroll', () => {
  // 根据正在阅读文本块，找到并高亮菜单
  highlightTargetMenu()
})
```

Proxy

Plan 1.

A.html

> A添加iframe后定时检测结果

- iframe src=B.html?paramsB

  > 接受url上参数，并请求token
  > - iframe src=C.html?paramsA
  > > 接受参数存储

注意：A、C同源, 与B不同源

gitbook

> 学习 (http://www.chengweiyang.cn/gitbook/index.html)

```
npm install gitbook-cli -g

# 初始化目录文件
gitbook init

# 生成静态网页并运行服务器
gitbook serve

# 生成静态网页
gitbook build
```

1.summry 折叠

```
# 安装折叠插件
npm i gitbook-plugin-toggle-chapters -D
```

- sunmary 同级目录增加 book.json文件

- book.json

```
{
  "plugins": ["toggle-chapters"]
}
```

2.语言显示

- book.json

```
"language": "zh-hans"
```

3.export PDF

```
sudo -s ln -s /Applications/calibre.app/Contents/MacOS/ebook-convert /usr/local/bin
```

报错

- gitbook serve 报错
    - Q

```
Error: Couldn't locate plugins "toggle-chapters, splitter, anchor-navigation-ex, prism, copy-code-button, alerts, theme-comscore", Run 'gitbook install' to install plugins from registry.

Error: ENOENT: no such file or directory, stat 'D:\workspace\core-solution-docs\_book\gitbook\gitbook-plugin-fontsettings\fontsettings.js'

Error: ENOENT: no such file or directory, stat 'D:\workspace\core-solution-docs\_book\gitbook\gitbook-plugin-livereload\plugin.js'

Error: ENOENT: no such file or directory, stat 'D:\workspace\core-solution-docs\_book\gitbook\gitbook-plugin-alerts\plugin.js'

Error: ENOENT: no such file or directory, stat 'D:\workspace\core-solution-docs\_book\gitbook\gitbook-plugin-livereload\plugin.js'

Error: ENOENT: no such file or directory, stat 'D:\workspace\core-solution-docs\_book\gitbook\gitbook-plugin-search\lunr.min.js'
```

* A
> 修改.gitbook\versions\3.2.3\lib\output\website\copyPluginAssets.js 搜索关键字 'confirm' 将值改为 false

```js
 return fs.copyDir(
   assetFolder,
   assetOutputFolder,
   {
     deleteFirst: false,
     overwrite: true,
     confirm: false
   }
 );
```

清理图标

```
defaults write com.apple.dock ResetLaunchPad -bool true; killall Dock
```

MacOS上vue-cli项目发热严重，node占用cpu大

1. 现象

在MacOS 上启动vue-cli的项目，电脑6，7位置过热（烫手）

2. 分析

项目是从仓库中拖下来的，直接npm install，运行...
mac开始旋转小风扇微波加热什么东西...

▼

没啥特别的配置，它在干啥?
打开活动监视器，node占用cpu过高！嗯...那就是它了

▼

为什么node会占用高呢？查了一圈是fsevents的问题

fsevents

> Native access to MacOS FSEvents in Node.js
> The FSEvents API in MacOS allows applications to register for notifications of changes to a given directory tree. It is a very fast and lightweight alternative to kqueue.
> This is a low-level library. For a cross-platform file watching module that uses fsevents, check out Chokidar.

所以，我的fsevents出了啥问题？？？

▼

难道是node_modules中的它...应该不是bug，那差不多就是版本问题了。

package-lock.json

> package-lock.json is automatically generated for any operations where npm modifies either the node_modules tree, or package.json. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.

3. 检测解决方案

删除package-lock.json
选择性的删除（我是直接删除了）node_modules

```
npm install
npm run dev
```

此时监视器中node依然很高
项目启动完成了，emm...
成功了

4. 结论

项目安装时候fsevents安装应该不符合我的电脑（版本），删除老版本的重新安装就可以了。

x-code

’npm i 模块‘的时候报警告

```
No receipt for 'com.apple.pkg.CLTools_Executables' found at '/'.

No receipt for 'com.apple.pkg.DeveloperToolsCLILeo' found at '/'.

No receipt for 'com.apple.pkg.DeveloperToolsCLI' found at '/'.

gyp: No Xcode or CLT version detected!
gyp ERR! configure error
gyp ERR! stack Error: `gyp` failed with exit code: 1
```

解决办法：

```
# 删除已经安装的CommandLineTools
sudo rm -rf $(xcode-select -p)

# 重新安装，此时git可能会自动提示安装
sudo xcode-select --install

# 同意条款选择同意，会提示系统更新一并同意
```

重装电脑

1. command line tools
2. code-select --install
3. xcode-select: error: command line tools are already installed, use "Software Update" to install updates
4. rm -rf /Library/Developer/CommandLineTools
5. xcode-select --install
6. Homebrew
7. /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
8. https://www.bbsmax.com/A/D854L3GQ5E/ [HELP]
9. Nvm git 下载
10. Node 8.9.4
11. github 链接
12. 关闭spotlight
13. sudo launchctl unload -w /System/Library/LaunchDaemons/com.apple.metadata.mds.plist
14. sudo mdutil -i off

Start

1. 安装

下载 (https://dev.mysql.com/downloads/mysql/)傻瓜式安装即可!

2. 配置环境变量

首先找到配置文件
/etc/profile
在后面输入export路径
export PATH=$PATH: /usr/local/mysql/bin

3. 执行source 命令

```
source /etc/profile
```

4. zsh中每次启动都需要重新执行步骤3的解决

在用户下找到
.zshrc
同样在最后一行添加
export PATH=$PATH: /usr/local/mysql/bin

最后执行步骤3
结束!

首次使用

- mac上需要在'系统偏好设置'中MySQL中start MySQL Server;
- 第一次使用需要自定义密码;

```
cd /usr/local/mysql/bin/

sudo su
# 输入mac的管理员密码

# 禁止mysql验证功能
./mysqld_safe --skip-grant-tables &
# 回车后mysql会自动重启（偏好设置中mysql的状态会变成running）

./mysql
FLUSH PRIVILEGES;
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('你的新密码');
```

从mac -> centos

1. 导出sql文件

```
# cd 要导出到的目录
mysqldump -u root -p 要导出的数据库名>名字随意.sql
```

2. 目标主机导入sql文件

```
mysql -u root -p
 show databases;
create database 数据库名;
 use 数据库名;
 source 将sql文件拖入终端;
```

数据库相关

```
# 目前拥有的数据库
show databases;

# 创建数据库
create database 数据库名;

# 切换了要操作的数据库
use 数据库名

# 展示本数据库下的数据库表
show tables;
```

数据表 相关

```
# 创建数据表
create table `表名`
 (`字段名` 字段类型, `字段名` 字段类型);

#      设置主键
create table personTable
 (personID int, name varchar(20), gender varchar(10), primary key(personID));

#      设置字段值唯一
create table personTable
 (
   personID int,
   name varchar(20) unique,  # unique 表示这个字段下的值，唯一
   gender varchar(10),
   primary key(personID)
 );

# 展示table下的编码
show create table 表名;

desc 数据表;

# 展示数据表下字段描述
show columns from 数据表;

# 更改table字符集
alter table 表名 convert to character set utf8;

# 删除数据表
drop table 表名;
```

查询

```
# 统计表行数
select count(*) from 表名;


# 查询全部
select * from 表名;
select 字段名 from 表名;
```

增加字段

```
# 增加字段
alter tabel 数据表 add 字段名 字段类型;

# 一次添加多个列(字段)
ALTER TABLE table_name
  ADD func varchar(50),
  ADD gene varchar(50),
  ADD genedetail varchar(50);
```

更新字段

```
# 修改字段名
alter table 数据表
  change 前字段名 后字段名 后类型;

# 修改字段类型
alter table 表名
  modify 字段名称 修改后字段类型;

# 更改某字段值唯一性
alter table 表名 add unique(字段);

# 修改mysql主键（id)的值为自增
alter table 表名
  modify id int auto_increment primary key;
```

删除字段

```
# 删除字段
alter table 数据表 drop 字段名;
```

插入数据

```
# 插入新数据
insert into weibo
  (id, text, originImg)
  values(0,?,?)
```

更新某一行

```
# 更新数据
update students set
  stu_name = "zhangsan",
  stu_gender = "m"
  where stu_id = 5;

# 删除某个字段中的某个字符
update 表名 set
  字段名 = trim(
    both ',' from replace
    (
      concat(',', 字段名, ','),
      ',要删除的字符,',
      ','
    )
  )

#
# 假如表名为user，表如下

# id    devid
# 1    1,2,3,12,13,14
# 要删除表user的devid字段中的1的值，注意12,13,13是不能删除。

update user set devid = trim(both ',' from replace(concat(',', devid, ','), ',1,', ',')) where id = '1';

# 拼接某字段和目标字符
update user_type set
  user_id = concat(user_id, ',${user_id}')
  where id = ${id};
```

删除行

```
# 删除数据
delete from 数据表 where age = 23;

# 模糊删除
delete from 表名 where 字段 in (值1, 值2, 值...)
```

Mac 在设置中启动mysql失效

1. 关闭mysql服务器
   sudo /usr/local/mysql/support-files/mysql.server stop
2. 进入目录
   cd /usr/local/mysql/bin

3. 获取权限
   sudo su

4. 重启服务器
   ./mysqld_safe --skip-grant-tables &

5. control + D退出编辑

6. 配置短命令
   alias mysql=/usr/local/mysql/bin/mysql

7. 进入mysql命令模式
   mysql

创建一个项目

初始化命令
npm init
npm init -yes

npm_install

> package.json 和 package-lock.json 的应用

要点

1. install [moduleName@version]，uninstall [moduleName] 会同时操作俩个文件的安装和卸载
2. 在有package-lock.json时候，进行npm install命令，会读取这个文件，但速度相对于没有它时，执行package.json会慢一些

解读

> 假设我们创建了一个新项目，它将使用express。 在运行npm init之后，在撰写本项目时，最新的express版本是4.15.4。 (默认情况下，npm 将安装最新版本)

> 因此在package.json中,"express":"^ 4.15.4"被添加作为依赖项。 假设明天，express的维护者会发布一个 bug 修复，所以最新版本变成了4.15.5。 然后，如果有人想要为我的项目做贡献，他们会克隆它，然后运行 npm install, 因为4.15.5是一个更高版本的主要版本，这是为他们安装的。 我们都有express依赖，但我们有两个不同的版本。 理论上，它们应该还是兼容的，但是也许这个 bug 会影响我们正在使用的功能，而我们的应用程序在使用Express版本4.15.4与4.15.5进行比较时会产生不同的结果.

> 而package-lock.json的作用就是用来保证我们的应用程序依赖之间的关系是一致的, 兼容的.

> 当不存在package-lock.json文件时，使用npm install时，会自动生成这个文件。当存在这个文件时，使用npm install安装，会安装package-lock.json里指定版本的插件，而且相比没有package-lock.json文件时，安装速度会快很多。因为package-lock.json文件里已经存在插件的版本、下载地址、整个node_modules的结构等信息。

> 当存在package-lock.json文件时，每次npm install安装就会安装package-lock.json里对应插件的版本。这样同一份package-lock.json文件，大家安装的插件版本一致。

> 如果某个插件版本变更。又不想删除package-lock.json文件，重新生成。方法是：npm install plugin@version，及重新安装这个插件，并指定插件的版本，这样，package.json和package-lock.json会自动更新。当然，也可以直接修改package-lock.json文件，这样npm install时，也会安装修改后的版本。但是如果只修改package.json，不修改package-lock.json，npm install还是会安装package-lock.json里的插件版本

Start

- 链接远程服务器

```
ssh root@ip
# 输入密码
```

重启服务器

阿里云执行重启成功

- 启动nginx /usr/sbin/nginx
- 启动mysql systemctl start mysqld
- 启动微博项目 npm run get && npm run serve

centos7 安装mysql5.7

[参考 (https://blog.csdn.net/wohiusdashi/article/details/89358071)](https://blog.csdn.net/wohiusdashi/article/details/89358071)

1. 安装YUM Repo

```
# 下载
wget https://dev.mysql.com/get/mysql57-community-release-el7-9.noarch.rpm


# 安装
rpm -ivh mysql57-community-release-el7-9.noarch.rpm
# 校验
# 执行完成后会在/etc/yum.repos.d/目录下生成两个repo文件mysql-community.repo mysql-community-source.repo
```

2. 使用yum命令即可完成安装

注意：必须进入到 /etc/yum.repos.d/目录后再执行以下脚本

```
# 安装
yum install mysql-server
# 一路y，傻瓜安装

# 启动
systemctl start mysqld

# 获取安装时的临时密码
grep 'temporary password' /var/log/mysqld.log // O11RdxUQ5M=D
# 2020-11-02T08:56:09.728544Z 1 [Note] A temporary password is generated for root@localhost: fO*Ti+oeL6)c

# 登录验证
mysql -u root -p;
Enter password: fO*Ti+oeL6)c;
```

3. 登录成功

```
# 首次使用修改密码
set password for 'root'@'localhost' = '密码';
```

4. 修改mysql存储emoji

```
# /etc 下创建my.cnf, 并修改内容

[client]
default-character-set = utf8mb4

[mysql]
default-character-set = utf8mb4

[mysqld]
character-set-client-handshake = FALSE
character-set-server = utf8mb4
collation-server = utf8mb4_unicode_ci
init_connect='SET NAMES utf8mb4'

# 重启
systemctl restart mysqld

# 查看编辑结果
SHOW VARIABLES WHERE Variable_name LIKE 'character\_set\_%' OR Variable_name LIKE 'collation%';
```

Nginx

参考 (https://www.cnblogs.com/jeffhong99/p/11362361.html)

```
# 安装nginx
yum -y install nginx

# 安装成功后nginx的几个默认目录

# 输入命令:
whereis nginx
# 输出
# 执行目录：/usr/sbin/nginx
# 模块所在目录：/usr/lib64/nginx/modules
# 配置所在目录：/etc/nginx/
# 默认站点目录：/usr/share/nginx/html

# 主要配置文件：/etc/nginx/nginx.conf
# 指向：/etc/nginx/conf.d/default.conf


# 启动
/usr/sbin/nginx

# 重启
cd /usr/sbin
./nginx -s reload

# 停止
cd /usr/sbin
./nginx -s stop
```

安装node

```
# 安装 wget
yum install -y wget

# 查找node的版本，复制下载链接
wget https://nodejs.org/dist/v14.15.1/node-v14.15.1-linux-x64.tar.xz

# 解压
xz -d node-v14.15.1-linux-x64.tar.xz
tar -xf node-v14.15.1-linux-x64.tar

# 重命名
mv node-v14.15.1-linux-x64 node

ln -s node-v14.15.1-linux-x64/bin/node /usr/bin/node
ln -s node-v14.15.1-linux-x64/bin/npm /usr/bin/npm
ln -s node-v14.15.1-linux-x64/bin/npm /usr/bin/npx

vim ~/.bash_profile

# 在PATH=$PATH:$HOME/bin 后面增加 :~/node/bin

# 测试node
node -v
# 输出版本号 v14.15.1, 成功
```

同步文件

- Server上文件位置

```
l-root
l-l-project
l-l-l-docs
```

- 导入静态项目

Step 1. 进入本地路径

```
# 打包文件
tar -zcvf folder.tar.gz folder
scp filename.format root@ip:/serverFolderIWant
scp -r ./docs root@ip:/serverFolderIWant
```

Step 2. 服务上

```
# 查看上传的文件
cd project && ls

# 解压文件
tar -xvf fileName

# 删除文件
rm fileName.format

# 删除文件夹
rm -rf floder
```

关于scp

```
# 把本地的source.txt文件拷贝到192.168.0.10机器上的/home/work目录下
scp /home/work/source.txt work@192.168.0.10:/home/work/

# 把192.168.0.10机器上的source.txt文件拷贝到本地的/home/work目录下
scp work@192.168.0.10:/home/work/source.txt /home/work/

# 把192.168.0.10机器上的source.txt文件拷贝到192.168.0.11机器的/home/work目录下
scp work@192.168.0.10:/home/work/source.txt work@192.168.0.11:/home/work/

# 拷贝文件夹，加-r参数
scp -r /home/work/sourcedir work@192.168.0.10:/home/work/
```

vim使用

```
# 打开文件
vim file

# 开始编辑
i

# 保存文件
# 保存并推出
:wq

# 仅推出
:q!

# 保存文件，不退出
:w
```

Storage

查看服务器存储使用情况

```
df -lh
# 输出
# /dev/vda1        40G  4.0G   34G   11% /
# devtmpfs        911M     0  911M    0% /dev
# tmpfs           920M     0  920M    0% /dev/shm
# tmpfs           920M  420K  920M    1% /run
# tmpfs           920M     0  920M    0% /sys/fs/cgroup
# tmpfs           184M     0  184M    0% /run/user/0
```

Nginx

打开配置文件

- Mac上打开配置

```
open /usr/local/etc/nginx/
# nginx.conf
```

- centos 配置

```
cd /etc/nginx
ls
vim nginx.conf

# 重启
/usr/sbin/nginx -s reload
```

- Windows

直接找到nginx文件夹，打开 nginx.conf

配置

```
...

listen       port;
server_name  localhost;

# 配置项目路径
# server_name:listen/ 相当于 root/paths
# 访问：server_name:listen

loaction / {
  root root/paths;
  index index.html;
  # history 模式下，刷新页面请求的服务地址可能报404，此时需要重定向到初始化位置
  try_files $uri /index.html;
}

# 配置代理服务
# 代理地址：proxy_pass, 结尾处带斜杠，访问指向的地址会删除前缀；反之，不删除。
# 访问：server_name:listen/前缀/sth/
# proxy_pass/ 指向 proxy_pass/sth/
# proxy_pass  指向 proxy_pass/前缀/sth/

location /api {
  proxy_pass http://server:port;
}
location /user {
  proxy_pass http://server:port;
}

...
```

接口匹配

```
ajax.get('/api/haha/') // react history模式
// 或者
ajax.get('api/haha/') // vue hash模式
```

Help

```
start nginx
nginx -s reload
nginx -s quit
```

记录版本号

> 在windows上实现，只记录提交者机器上的版本好

- 建立两个js文件，ver-tmp.js, ver.js，格式需要完全一样

```
// ver-tmp.js
const ver = $WCREV$
export default ver

// ver.js，程序内引用
const ver = 596
export default ver
```

- 执行模版

```
subwcrev . ver-tmpl.js ver.js
```

- 程序应用

```
# 建立.bat在windows下可运行
cd .\src\assets\js
subwcrev . ver-tmpl.js ver.js
cd ../../../
npm run build

# package.json
"win-build": "build.bat"

# build for production
npm run win-build
```

sublime

Preferences.sublime-setting --- User

```
{
  "auto_complete_triggers":
  [
    {
      "characters": "abcdefghijklmnopqrstuvwxyz< :.",
      "selector": "text.wxml"
    }
  ],
  "color_scheme": "Packages/Material Theme/schemes/Material-Theme.tmTheme",
  "expand_tabs_on_save": true,
  "file_exclude_patterns":
  [
    "swiper.css"
  ],
  "folder_exclude_patterns":
  [
    ".svn",
    ".git",
    ".hg",
    "CVS",
    "node_modules",
    "dist",
    "vendor",
    "map"
  ],
  "font_size": 13,
  "highlight_line": true,
  "ignored_packages":
  [
    "Vintage"
  ],
  "line_padding_bottom": 1,
  "line_padding_top": 1,
  "margin": 8,
  "tab_size": 2,
  "theme": "Default.sublime-theme",
  "translate_tabs_to_spaces": true,
  "trim_trailing_white_space_on_save": true
}
```


List Packages

- A File icon
- Babel
- BrackerHighlighter
- Color Higlighter
- DocBlocker
- Dracula Color Scheme
- Emmet
- GitGutter
- JavaScript & NodeJS Snippets

- JavaScript Completions
- LESS
- Node Completions
- Nodejs
- ReactJS
- SASS
- Subime wxapp
- SublimeLinter

```
{
  "styles": [
    {
      "icon": "bookmark",
      "mark_style": "fill"
    }
  ],
  "linters": {
    "eslint": {
      "selector": "text.html.vue, source.js - meta.attribute-with-value"
    }
  }
}
```

- SublimeLinter-eslint
- Terminal
- Vue Syntax Highlighter

VS Code

setting.json

```
{
  "editor.tabSize": 2,
  "editor.minimap.enabled": true,
  "files.trimTrailingWhitespace": true,
  "grunt.autoDetect": "off",
  "gulp.autoDetect": "off",
  "jake.autoDetect": "off",
  "php.suggest.basic": false,
  "editor.scrollBeyondLastLine": false,
  "workbench.activityBar.visible": true,
  "window.zoomLevel": 0,
  "breadcrumbs.enabled": true,
  "explorer.confirmDelete": false,
  "workbench.iconTheme": "material-icon-theme",
  "editor.fontSize": 14,
  "terminal.integrated.rendererType": "dom",
  "workbench.colorTheme": "Community Material Theme High Contrast",
  "emmet.includeLanguages": {
    "javascript": "javascriptreact"
  },
```

```
"explorer.confirmDragAndDrop": false,
"javascript.updateImportsOnFileMove.enabled": "always",
"git.enableSmartCommit": true,
"git.confirmSync": false,
"git.autofetch": true,
"background.useDefault": false,
"background.style": {
  "content": "''",
  "pointer-events": "none",
  "position": "absolute",
  "z-index": "99999",
  "height": "100%",
  "background-position": "0% 0%",
  "background-size": "cover",
  "background-repeat": "no-repeat",
  "opacity": 0.2
},
"background.customImages": [
  "/Users/unzoa/Documents/.."
],
"search.followSymlinks": false,
"files.exclude": {
  "**/.git": true,
  "**/.svn": true,
  "**/.hg": true,
  "**/CVS": true,
  "**/.DS_Store": true,
  "**/tmp": true,
  "**/node_modules": true,
  "**/bower_components": true,
  "**/dist": true
},
"files.watcherExclude": {
  "**/.git/objects/**": true,
  "**/.git/subtree-cache/**": true,
  "**/node_modules/**": true,
  "**/tmp/**": true,
  "**/bower_components/**": true,
  "**/dist/**": true
}
}
```

Packages List

- Amethyst Themes
- Dracula Official
- Github Sharp Themes
- Markdown Preview Enhanced
- Material Icon Theme
- Vetur
- VSC Netease Music

Git

初始化

```
git config --global user.name "unzoa"
git config --global user.email "unzoa@xxx.com"

# 创建文件夹unoza
git init
```

和github关联

```
# 删除原来的秘要

# 生成ssh秘a钥
ssh-keygen -t rsa -C "unzoa@xxx.com"

# 测试ssh
ssh -T git.github.com

# 提交修改过的项目
git add .
git commit -a -m 'commit'
git pull
git push

# 删除仓库文件 or 文件夹
git rm -r --cached folder/filename
git rm -r --cached folder
git commit -m'xxx'
git push
```

解决 warning：LF will replace CRLF

```
# windows中的换行符为 CRLF， 而在linux下的换行符为LF，所以在执行add . 时出现提示，解决办法：

# 删除.git
rm -rf .git

# 禁用自动转换
git config --global core.autocrlf false

# 然后重新执行：
git init
git add .
```

key 挂掉了

```
# 如果之前用过需要清理原来的rsa，执行命令：
mkdir key_backup
cp id_rsa* key_backup
rm id_rsa*

ssh-keygen -t rsa -C name@xx.com

# 复制id_rsa.pub到github ssh key

ssh -T git@github.com
```

svn

- 从服务器上拉下来项目svn checkout (site)
- 更新项目 svn update
- 提交修改 svn commit -m'haha'
- 提交单独的文件 svn commit path -m 'haha'
- 添加文件 svn add path 然后svn commit
- 删除文件 svn delete path 然后svn commit

zsh

1. oh-my-zsh

> 安装oh-my-zsh (https://gitee.com/mirrors/oh-my-zsh#manual-installation)，手动模式即可。

2. 更换主题

> 主题：agnoster地址 (https://github.com/agnoster/agnoster-zsh-theme)

根据README.md步骤执行：

1.安装Powerline-patched font (https://github.com/powerline/fonts)，

- 采用git clone模式，本机随意位置都可
- 执行包内文件 install.sh

2.安装solarized主题 (https://ethanschoonover.com/solarized/)

- 下载文件包
- 打开终端，[command + ,] 在描述文件tab下
- 将下载的包包内/osx-terminal.app-colors-solarized/下文件直接拖动到终端配置的左侧栏目

3.主题文件路径 /你的用户名字/.oh-my-zsh/themes/agnoster.zsh-theme

- 隐藏主机名 (https://github.com/agnoster/agnoster-zsh-theme/issues/39#issuecomment-307338817)

加密狗

购买品牌

0.0覆置文本 Http:/T₴nV4PcSpyBsl₴转移至　ta0寶　【飞天诚信ePass3000usbkey加密锁数字签名身份认证智能卡登陆空锁】

- 1，加密狗自带程序

  类似ukey的驱动
  下载

- 2，加密狗服务文件

  自建的连接ukey的服务，服务地址 http://127.0.0.1:1234/serial
  下载

使用

- 安装1文件
- 2文件使用方式

  ```
  # 终端
  shell:startup
  ```

  将2文件放在这里，双击打开

- 插入加密狗
- 电脑左下角弹出
  - key已插入
- 浏览器输入http://127.0.0.1:1234/serial
  - 一串数字 [获取成功]
  - 烫烫烫烫烫烫 [获取失败]