袋鼠云前端开发手册

1.命名规范

1.1 JavaScript

1.1.3 常量

- **命名方式：**全部大写
- **命名格式：**使用大写字母和下划线来组合命名，下划线用以分割单词

示例：

```
const MAX_COUNT = 10;

const URL = 'https://www.dtstack.com/';
```

1.1.3 变量

- **命名方式：**小驼峰
- 命名格式：
  - 列表：<名称>+List
  - 状态：<名称>+Status
  - 数组：<名称>+Array
  - 配置：<名称>+Config

示例：

```
const articleList = [{
 name:'JavaScript高级程序设计',
 author:'Nicholas C.Zakas'
},{
 name:'深入浅出Node.js',
 author:'朴灵',
},{
 name:'Css世界',
 author:'张鑫旭',
}];

const loadingStatus = true;

const sexTypeArray = ['男','女','中性'];

const webpackConfig = {};
```

1.1.3 函数

- **命名方式**：小驼峰
- 命名格式：
    - <动词>+<名词>
    - <动词>+<名词>+<介词>+<名词>
    - <动词>+<介词>+<名词>
    - handle +<名词>+<动词>（只针对事件响应函数）

注：这里的名词可以是单独的<名词>也可以是<形容词+名词>

常用动词列表：

动词	含义	返回值
jump	跳转到某个页面	无返回值
can	判断是否可执行	布尔值。true-可以；false：不可以
has	判断是否含有某个元素	布尔值。true-含有；false：不含有
is	判断是否为某个值	布尔值。true-是该值；false：不是该值
get	获取某个值	有返回值
set	设置某个值（功能逻辑）	无返回值或者返回是否设置成功
load	加载某些数据	无返回值或者返回是否加载成功
add	新增某个值	无返回值或者返回是否新增成功
update	更新某个值（业务逻辑）	无返回值或者返回是否更新成功
delete	删除某个值	无返回值或者返回是否删除成功
handle	处理事件响应	无返回值

示例：

```
//根据id查找对应的人
function getPersonById(id){
 if(id==='007'){
  return '隔壁老王';
 }
}

//加载老师列表
function loadTeacherList(){
 return ['苍老师','波多老师','吉泽老师']
}

//跳转到某个页面
function jumpToSpecifiedUrl(url){
 window.location.href=url;
}


//响应组件添加
function handleComponentAdd(){
 ....
}
```

1.1.4 组件和类名

- 命名方式：大驼峰

```
function ToolTip(props){
 const {msg} = props;
 return (<div className="comp-tooltip">{msg}</div>)
}

import React from 'react';
class QueryTable extends React.PureComponent{
 ...
}
```

1.1.5 api方法名

- 命名方式：小驼峰
- 命名格式：
    - <动词>+<名词>
    - <动词>+<名词>+<介词>+<名词>
    - <动词>+<介词>+<名词>

常用动词列表：

动词	含义
query	查询
add	新增
update	更新或者编辑

delete	删除
match	模糊匹配
send	发送
get	获取
login	登录
register	注册

```
queryUserListByFilter  //根据过滤条件查询用户列表
addUser     //新增用户
updateUser //编辑或者更新用户
deleteUser  //删除用户
matchUserByName //根据名字模糊匹配用户
sendMessageToUser //发送消息给用户
getImageCode  //获取图片验证码
```

1.2 Css

1.2.1 id和类

以BEM为基础，以NEC的模块分类

1.3 其他

1.3.1 文件及文件名

- 命名方式：小驼峰

1.3.2 vue标签名

- 命名方式：单词之间以横杠”-“分割

示例：

```
<template>
 <ko-btn></ko-btn>
 <ko-table></ko-table>
</template>
```

1.3.3 路由

- 命名方式：单词之间以横杠”-“分割

示例：

```
/dtstack/user-center

/dtstack/product-introduction
```

2. 注释规范

2.1 单行注释

JavaScript

```
// 单行注释
dosomthing()
```

Css

```
/*单行注释*/
.box{}
```

2.2 多行注释

```
/**
* 第一行注释
* 第二行注释
* ...
* 第n行注释
*/
```

2.3 函数注释

注释名	语法	含义
@param	@param {<类型>} <参数名> <描述信息>	描述参数的信息
@return	@return {<类型>}	返回值的类型
@author	@author <作者> <日期, YYYY-MM-DD> <时间, HH:mm>	谁在何时创建了这个函数
@update	@update <更新者> <日期, YYYY-MM-DD> <时间, HH:mm>	谁在何时更新了这个函数

示例：

1. 简略版

```
/**
* @param {string} name 姓名
* @param {number} age 年龄
* @return {object}
*/
function createPerson(name,age){
 var person = new Object();
 person.name = name;
 person.age = age;
 return person;
}
```

2. 完整版

```
/**
* @param {string} name 姓名
* @param {number} age 年龄
* @return {object}
* @author 隔壁老王 2018-10-26 13:36
* @update 长卿 2018-10-28 10:26
*/
function createPerson(name,age){
 var person = new Object();
 person.name = name;
 person.age = age;
 return person;
}
```

2.4 文件注释

```
/**
* @description: <文件的描述>
* @author: Created by <作者名称> on <日期> <时间，精确到分钟>
*/
//--------------------------------------------------------
/**
* @description: 销售类型关联表
* @author: Created by 长卿 on 2018-10-26 13:36
*/
```

3. 组件开发

3.1 React

3.1.1 页面

每个页面都要具备以下这种结构：

- index.js 页面的入口文件（必需）
- style.scss 页面样式（必需）
- components 该页面特有的区块
- reducer.js 存放redux的reducer
- actions.js 存放redux的action
- constant.js存放该页面所需要的一些常量

示例：

index.js（标准版）

```
//npm包优先导入，项目内文件次之
import React from 'react';
//组件名大驼峰
class Index extends React.PureComponent{
 state={
  msg:'Hello World'
 }
 //加载页面主数据
 loadMainData(isClear){
  if(isClear){
   //清除一些过滤条件
  }
  //请求http数据
 }
 componentDidMount(){
  this.loadMainData(true)
 }
 render(){
  const {msg} = this.state;
  //必须要有一个【page-<组件名>】的类名
  return (<div className="page-index">{msg}</div>)
 }
}

export default Index;
```

index.js（redux版）

```
//npm包优先导入，项目内文件次之
import React from 'react';
import {connect} from 'react-redux';

@connect(
 mapStateToProps,//必需的
  mapDispatchToProps,
 mergeProps,
 options
)
//组件名大驼峰
export default class Index extends React.PureComponent{
 state={
  msg:'Hello World'
 }
 //加载页面主数据
 loadMainData(isClear){
  if(isClear){
   //清除一些过滤条件
  }
  //请求http数据
 }
 componentDidMount(){
  this.loadMainData(true)
 }
 render(){
  const {msg} = this.state;
  //必须要有一个【page-<组件名>】的类名
  return (<div className="page-index">{msg}</div>)
 }
}
```

3.2.2 区块

每个区块都应该具有以下结构

- index.js(必需)
- style.scss(必需)

示例：

index.js（有状态）

```
//npm包优先导入，项目内文件次之
import React from 'react';
import PropTypes from 'prop-types';
//组件名大驼峰
export default class Index extends React.PureComponent{
  static propTypes = {
        msg: PropTypes.string
    }
 static defaultProps = {
     msg: 'Hello World'
   }
 render(){
  const {msg} = this.props;
  //必须要有一个【comp-<组件名>】的类名
  return (<div className="comp-index">{msg}</div>)
 }
}
```

index.js（无状态状态）

```
//npm包优先导入，项目内文件次之
import React from 'react';
import PropTypes from 'prop-types';
//组件名大驼峰
export default function Index(props){
 const {msg='Hello World'} = props;
 return (
  //必须要有一个【comp-<组件名>】的类名
  <div className="comp-index">{msg}</div>
 )
}
```

3.2 Vue

3.2.1 页面

每个页面都要具备以下这种结构：

- index.vue 页面的入口文件 （必需）
- components 该页面特有的区块(必需)
- vuex.js 该页面的vuex
- constant.js存放该页面所需要的一些常量

示例：

index.vue

```
<template>
 /*必须要有一个【page-<组件名>】的类名*/
 <div class="page-index">
  <p>{{msg}}</p>
  <ko-button>点击</ko-button>
 <div>
</template>
<script>
 //npm包优先导入，项目内文件次之
 import {Button} from 'ko-ui';
 export default {
  data(){
   return {
    msg:'Hello World'
   }
  },
  components:{
   'ko-button':Button //单词之间以横杠”-“分割
  }
  methods:{
   //加载页面主数据
   loadMainData(isClear){
    if(isClear){
     //清除一些过滤条件
    }
    //请求http数据
   }
  },
  created(){
   this.loadMainData();
  }
 }
</script>
<style>
 .page-index{
  color:red;
 }
</style>
```

3.2.2 区块

每个区块都应该具有以下结构

- index.vue（必需）

示例：

index.vue

```
<template>
 /*必须要有一个【comp-<组件名>】的类名*/
 <div class="comp-index">
  {{msg}}
 <div>
</template>
<script>
 export default {
  props:{
   msg: String,
   default: 'Hello World'
  }
    }
</script>
<style>
 .comp-index{
  color:red;
 }
</style>
```

4. API管理

url.js

```
export default {
  getUserData:{
    method:'get',
    url:'/mock/userData.json'
  },
  getNavData:{
    method:'get',
    url:'/mock/navData.json'
  },
  getUserList:{
    method:'get',
    url:'/mock/userList.json'
  }
}
```

接口地址url采用基于对象的维护方式，所有接口地址对应改造成一个个对象，该对象拥有两个属性：

- method（请求方法，这个并不是对应http协议中的方法，而是自己封装的http请求库的方法）
- url（接口请求地址）

index.js

```
import _ from 'lodash';
import API_URL from './url';
import http from '@/utils/http';

function mapUrlObjToFuncObj(urlObj){
  const API = {};
  _.keys(urlObj).forEach((key)=>{
    const item = urlObj[key]
    API[key]=function(params){
      return http[item.method](item.url,params)
    }
  });
  return API;
}
function mapUrlObjToStrObj(urlObj){
  const Url = {};
  _.keys(urlObj).forEach((key)=>{
    const item = urlObj[key]
    Url[key]=item.url
  });
  return Url;
}

export const API = mapUrlObjToFuncObj(API_URL);
export const URL = mapUrlObjToStrObj(API_URL);
```

这里有两个方法：

- mapUrlObjToFuncObj（将url对象做一个基于http库请求方法封装的映射）
- mapUrlObjToStrObj（将url对象做一个接口地址的映射）

最终向外暴露的API和URL，对应的值为：

```
import http from '@/utils/http';
const API = {
 getUserData: function(params){
  return http.get('/mock/userData.json',params)
 },
 getNavData: function(params){
  return http.get('/mock/navData.json',params)

 },
 getUserList: function(params){
  return http.get('/mock/userList.json',params)
 }
}
const URL = {
 getUserData: '/mock/userData.json',
 getNavData: '/mock/navData.json',
 getUserList: '/mock/userList.json'
}
```

5. 工具库

附录1：常用中英文对照表

页面结构

中文	英文
容器	container
头部	header
主体	main
内容	content
尾部	footer
导航	nav
子导航	subnav
菜单	menu
子菜单	submenu
侧边栏	sidebar
左中右	left center right

功能

中文	英文
标志	logo
广告	banner
登陆	login
注册	regsiter
搜索	search
功能区	shop
标题	title
状态	status
按钮	btn
列表	list
滚动	scroll
标签页	tab
提示信息	msg
小技巧	tips
当前的	current
图标	icon
注释	note

指南	guild
服务	service
热点	hot
新闻	news
下载	download
投票	vote
合作伙伴	partner
链接	link
版权	copyright