---

link: null
title: 珠峰架构师成长计划
description: config\proxy.ts
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=78 sentences=144, words=1462

1.前端项目最佳实践

类别 选择 框架

JS 语言

CSS 语言

JS 编译

模块打包

单元测试

路由

数据流

代码风格

JS 压缩

CSS 压缩

请求库

UI

国际化

hooks 库

静态文档

微前端

图表库

- React 框架
- TypeScript 语言
- Less+CSS Modules
- Eslint+Prettier+固定配置
- 固定数据流方案 dva
- 固定 babel 插件
- Jest+Enzyme
- 框架版本不允许锁定，^前缀必须有
- 主要依赖不允许自定义依赖版本
- 不仅是框架功能，还有 UI 界面
- 路由、布局、菜单、导航、面包屑、权限、请求、埋点、错误处理
- 只管写 Page 页面就可以了
- 给 node.js 使用，比如 webpack、babel 相关配置，静态路由配置
- 给浏览器用、比如渲染逻辑、动态修改路由、获取用户信息
- 国际化
- 数据流
- MOCK
- 目录结构
- 404
- 权限策略
- Service
- 配置文件
- 通过最佳实践减少不必要的选择的差异
- 通过插件和插件集的架构方式，满足不同场景的业务
- 通过资产市场和场景市场着力解决 70%的开发者问题
- 通过对垂直场景采取强约束的方式，进一步提升研发效率
- 不给选择、配置化、约定化

2.Ant Design Pro

```
yarn create umi
```

- 我们已经为你生成了一个完整的开发框架，提供了涵盖中后台开发的各类功能和坑位，下面是整个项目的目录结构。

```
├─config # umi 配置，包含路由，构建等配置
├─mock   # 本地模拟数据
├─public
│  └─icons
├─src
│  ├─components # 业务通用组件
│  │  ├─Footer
│  │  ├─HeaderDropdown
│  │  ├─HeaderSearch
│  │  ├─NoticeIcon
│  │  └─RightContent
│  ├─e2e       # 集成测试用例
│  ├─locales   # 国际化资源
│  │  ├─en-US
│  │  ├─id-ID
│  │  ├─pt-BR
│  │  ├─zh-CN
│  │  └─zh-TW
│  ├─pages     # 业务页面入口和常用模板
│  │  ├─ListTableList
│  │  │  └─components
│  │  └─user
│  │      └─login
│  ├─services # 后台接口服务
│  └─utils    # 工具库
└─tests       # 测试工具
```

```
npm install
npm start:dev
```

```
git init
git add -A
git commit -m"1.init"
```

config\proxy.ts

```
export default {
  dev: {
    '/api/': {
+     target: 'http://localhost:4000/',
      changeOrigin: true,
      pathRewrite: { '^': '' }
    }
  }
};
```

src\app.tsx

```
import React from 'react';
import { BasicLayoutProps, Settings as LayoutSettings, PageLoading } from '@ant-design/pro-layout';
import { notification } from 'antd';
import { history, RequestConfig } from 'umi';
import RightContent from '@/components/RightContent';
import Footer from '@/components/Footer';
import { ResponseError } from 'umi-request';
import { queryCurrent } from './services/user';
import defaultSettings from '../config/defaultSettings';

export const initialStateConfig = {
  loading: ,
};

export async function getInitialState(): Promise Promise;
}> {
  const fetchUserInfo = async () => {
    try {
      const currentUser = await queryCurrent();
      return currentUser;
    } catch (error) {
      history.push('/user/login');
    }
    return undefined;
  };
  // 如果是登录页面，不执行
  if (history.location.pathname !== '/user/login') {
    const currentUser = await fetchUserInfo();
    return {
      fetchUserInfo,
      currentUser,
      settings: defaultSettings,
    };
  }
  return {
    fetchUserInfo,
    settings: defaultSettings,
  };
}

export const layout = ({
  initialState,
}: {
  initialState: { settings?: LayoutSettings; currentUser?: API.CurrentUser };
}): BasicLayoutProps => {
  return {
    rightContentRender: () => ,
    disableContentMargin: false,
    footerRender: () => ,
    onPageChange: () => {
      const { currentUser } = initialState;
      const { location } = history;
      // 如果没有登录，重定向到 login
      if (!currentUser && location.pathname !== '/user/login') {
        history.push('/user/login');
      }
    },
```

```
    menuHeaderRender: undefined,
    ...initialState?.settings,
  };
};

const codeMessage = {
  200: '服务器成功返回请求的数据。',
  201: '新建或修改数据成功。',
  202: '一个请求已经进入后台排队（异步任务）。',
  204: '删除数据成功。',
  400: '发出的请求有错误，服务器没有进行新建或修改数据的操作。',
  401: '用户没有权限（令牌、用户名、密码错误）。',
  403: '用户得到授权，但是访问是被禁止的。',
  404: '发出的请求针对的是不存在的记录，服务器没有进行操作。',
  405: '请求方法不被允许。',
  406: '请求的格式不可得。',
  410: '请求的资源被永久删除，且不会再得到的。',
  422: '当创建一个对象时，发生一个验证错误。',
  500: '服务器发生错误，请检查服务器。',
  502: '网关错误。',
  503: '服务不可用，服务器暂时过载或维护。',
  504: '网关超时。',
};

/**
 * 异常处理程序
 */
const errorHandler = (error: ResponseError) => {
  const { response } = error;
  if (response && response.status) {
    const errorText = codeMessage[response.status] || response.statusText;
    const { status, url } = response;

    notification.error({
      message: `请求错误 ${status}: ${url}`,
      description: errorText,
    });
  }

  if (!response) {
    notification.error({
      description: '您的网络发生异常，无法连接服务器',
      message: '网络异常',
    });
  }
  throw error;
};

export const request: RequestConfig = {
   errorHandler,
+  headers:{
+    Authorization:`Bearer ${localStorage.getItem('token')}`
+  }
};
```

src\services\API.d.ts

```
declare namespace API {
  export interface CurrentUser {
    avatar?: string;
   username?: string;
    title?: string;
    group?: string;
    signature?: string;
    tags?: {
      key: string;
      label: string;
    }[];
    userid?: string;
    access?: 'user' | 'guest' | 'admin';
    unreadCount?: number;
  }

  export interface LoginStateType {
    status?: 'ok' | 'error';
    type?: string;
+   token?:string;
  }

  export interface NoticeIconData {
    id: string;
    key: string;
    avatar: string;
    title: string;
    datetime: string;
    type: string;
    read?: boolean;
    description: string;
    clickClose?: boolean;
    extra: any;
    status: string;
  }
}
```

src\pages\user\login\index.tsx

```
import {
  AlipayCircleOutlined,
  LockTwoTone,
  MailTwoTone,
  MobileTwoTone,
  TaobaoCircleOutlined,
  UserOutlined,
  WeiboCircleOutlined,
```

```
} from '@ant-design/icons';
import { Alert, Space, message, Tabs } from 'antd';
import React, { useState } from 'react';
import ProForm, { ProFormCaptcha, ProFormCheckbox, ProFormText } from '@ant-design/pro-form';
import { useIntl, Link, history, FormattedMessage, SelectLang } from 'umi';
import Footer from '@/components/Footer';
import { fakeAccountLogin, getFakeCaptcha, LoginParamsType } from '@/services/login';

import styles from './index.less';

const LoginMessage: React.FC = ({ content }) => (

);

/**
 * 此方法会跳转到 redirect 参数所在的位置
 */
const goto = () => {
  const { query } = history.location;
  const { redirect } = query as { redirect: string };
  window.location.href = redirect || '/';
};

const Login: React.FC = () => {
  const [submitting, setSubmitting] = useState(false);
  const [userLoginState, setUserLoginState] = useState({});
  const [type, setType] = useState('account');
  const intl = useIntl();

  const handleSubmit = async (values: LoginParamsType) => {
    setSubmitting(true);
    try {
      // 登录
      const msg = await fakeAccountLogin({ ...values, type });
+      if (msg.status === 'ok' && msg.token) {
+        localStorage.setItem('token',msg.token);
        message.success('登录成功！');
        goto();
        return;
      }
      // 如果失败去设置用户错误信息
      setUserLoginState(msg);
    } catch (error) {
      message.error('登录失败，请重试！');
    }
    setSubmitting(false);
  };
  const { status, type: loginType } = userLoginState;

  return (

      {SelectLang && }

              Ant Design

          Ant Design 是西湖区最具影响力的 Web 设计规范

           dom.pop(),
              submitButtonProps: {
                loading: submitting,
                size: 'large',
                style: {
                  width: '100%',
                },
              },
            }}
            onFinish={async (values) => {
              handleSubmit(values);
            }}
          >

            {status

            )}
            {type
              <>
                ,
                  }}
                  placeholder={intl.formatMessage({
                    id: 'pages.login.username.placeholder',
                    defaultMessage: '用户名: admin or user',
                  })}
                  rules={[
                    {
                      required: true,
                      message: (

                      ),
                    },
                  ]}
                />
                ,
                  }}
                  placeholder={intl.formatMessage({
                    id: 'pages.login.password.placeholder',
                    defaultMessage: '密码: ant.design',
                  })}
                  rules={[
                    {
                      required: true,
                      message: (

                      )
                    },
```

```
                ]}
              />
            </>
          )}

  );
};

export default Login;
```

3.后端

```
cnpm i express body-parser  jwt-simple cors express-session connect-mongo mongoose axios -S
```

```
/api/register
{"name":"admin","password":"123456","autoLogin":true,"type":"account"}
/api/login/account
{"username":"admin","password":"123456"}
```

```
let express = require("express");
let bodyParser = require("body-parser");
let jwt = require('jwt-simple');
let cors = require("cors");
let Models = require('./model');
let session = require("express-session");
let MongoStore = require('connect-mongo')(session);
let config = require('./config');
let app = express();
app.use(
    cors({
        origin: config.origin,
        credentials: true,
        allowedHeaders: "Content-Type,Authorization",
        methods: "GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS"
    })
);
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(
    session({
        secret: config.secret,
        resave: false,
        saveUninitialized: true,
        store: new MongoStore({
            url: config.dbUrl,
            mongoOptions: {
                useNewUrlParser: true,
                useUnifiedTopology: true
            }
        })
    })
);
app.get('/', async (req, res) => {
    res.json({ code: 0, data: `hello` });
});

app.post('/api/register', async (req, res) => {
    let user = req.body;
    let hash = require('crypto').createHash('md5').update(user.email).digest('hex');
    user.avatar = `https://secure.gravatar.com/avatar/${hash}?s=48`;
    user = await Models.UserModel.create(user);
    res.send({ status: 'ok', currentAuthority: 'user' });
});
app.post('/api/login/account', async (req, res) => {
    let user = req.body;
    let query = {};
    if (user.type == 'account') {
        query.name = user.username;
        query.password = user.password;
    }
    let dbUser = await Models.UserModel.findOne(query);
    if (dbUser) {
        dbUser.userid = dbUser._id;
        let token = jwt.encode(dbUser, config.secret);
        return res.send({ status: 'ok', token, type: user.type, currentAuthority: dbUser.currentAuthority });
    } else {
        return res.send({
            status: 'error',
            type: user.type,
            currentAuthority: 'guest'
        });
    }
});

app.get('/api/currentUser', async (req, res) => {
    let authorization = req.headers['authorization'];
    if (authorization) {
        try {
            let user = jwt.decode(authorization.split(' ')[1], config.secret);
            res.json(user);
        } catch (err) {
            res.status(401).send({});
        }
    } else {
        res.status(401).send({});
    }
});
app.get('/api/login/outLogin', async (req, res) => {
    res.send({ data: {}, success: true });
});
app.listen(4000, () => {
    console.log('服务器在4000端口启动!');
});
```

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
let config = require('./config');
const conn = mongoose.createConnection(config.dbUrl, { useNewUrlParser: true, useUnifiedTopology: true });
const UserModel = conn.model('User', new Schema({
    userid: { type: String },
    email: { type: String },
    name: { type: String },
    password: { type: String, required: true },
    avatar: { type: String, required: true },
    currentAuthority: { type: String, required: true,default:'user' }
}));

module.exports = {
    UserModel
}
```

```
module.exports = {
    secret: 'pro',
    dbUrl: "mongodb://localhost:27017/pro",
    origin: ["http://localhost:8000"]
}
```

```
  "scripts": {
    "start": "nodemon app.js"
  }
```