

项目交付二部-前端组 张腾伟

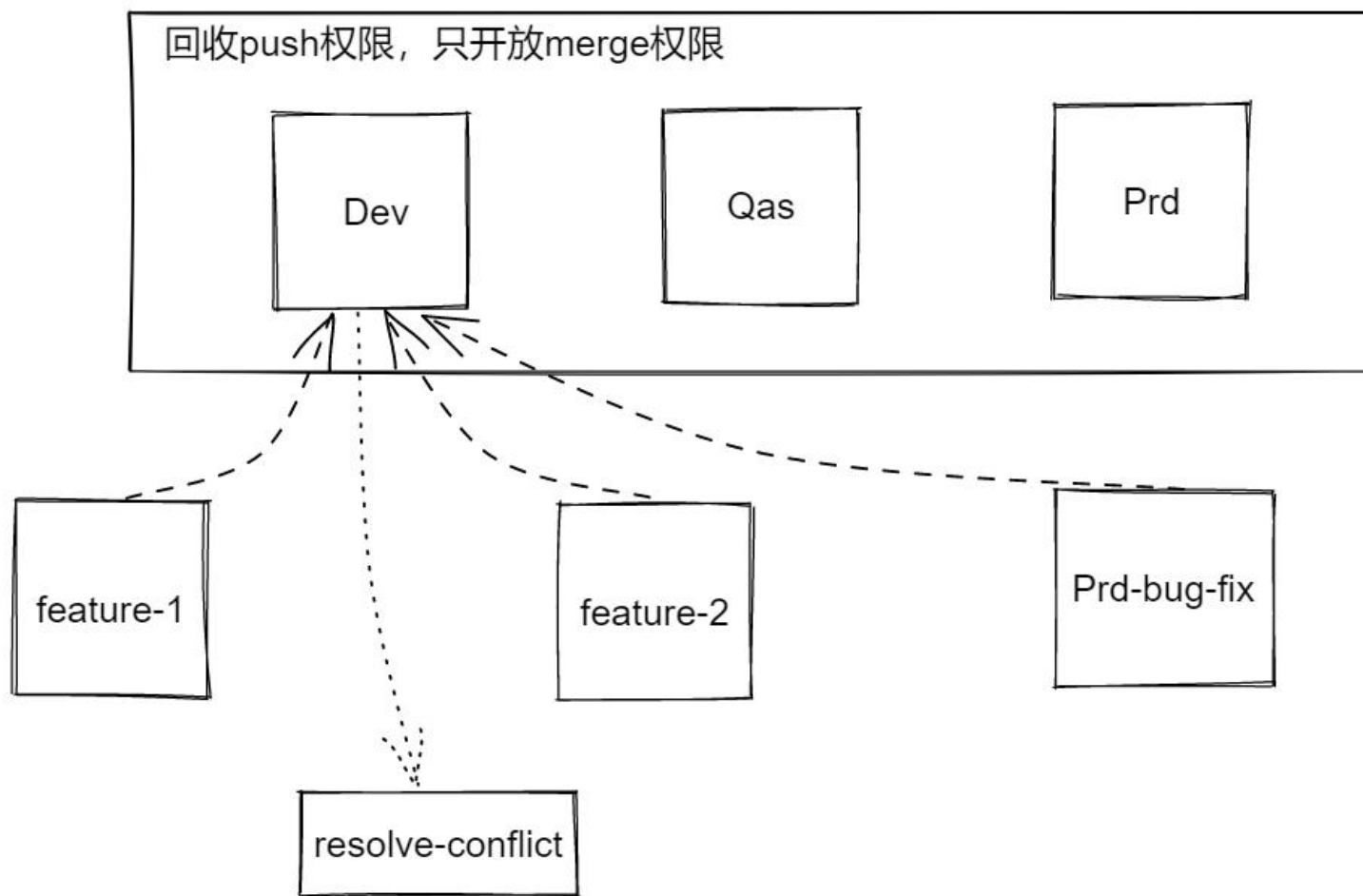
eBest Mobile 2023

Prepared for internal use only
仅供内部查阅

前端的一些分享

1. 前端Git分支管理规范
2. uni-app工程转为vue-cli项目
3. 小程序&pc端项目上的优化

1. Git分支管理规范



Dev、Qas分支：只做发布验证，不做开发。feature1、feature2...featureN等分支只承担常规&需求开发任务。feature1任务开发好之后在gitlab后台发起MR操作，指定本次修改要merge到哪个分支上，同时指定某个人进行code review，code review通过之后代码merge到具体分支，发布好之后通知测试去验证。

A同学在gitlab后台创建MR给B同学审核，将feature1与Dev分支merge产生冲突时解决方案：

步骤1：本地切换到Dev分支执行git checkout -b resolve-conflict

git merge feature1，这个时候在vscode里就能看到冲突的文件，解决完冲突之后

git add .

git commit -m ""

git push origin resolve-conflict//将本地分支推送到远程

步骤2：进入gitlab再次创建MR，源分支选择resolve-conflict目标分支选择Dev，让B同学审核，B同学审核通过之后当前改动进入到Dev分支。

注意在步骤1解决冲突时可能会修改代码，即在resolve-conflict分支去解决冲突时不是二选一，而是多个提交者的综合，这个时候切换到Dev分支

查看冲突的文件的提交记录，执行git log --all --full-history -- src/components/HelloWorld.vue可以查看到


HelloWorld.vue都在哪些分支被提交过，

假如本次在feature1和Prd-bug-fix上被修改过。将解决冲突时修改的文件假如为HelloWorld.vue文件，切换到feature1分支，

执行git checkout Dev src/components/HelloWorld.vue 只覆盖当前分支的HelloWorld.vue文件。Prd-bug-fix也同理只覆盖该文件。

这样被修改的冲突的文件就都修改到相关分支上了。切换到feature1分支，执行git branch -D feature-resolve-conflict删除本地解决冲突的临时分支，

执行git push origin --delete feature-resolve-conflict删除远程上解决冲突的临时分支。

 **GitLab**

Projects ▾ Groups ▾ Activity Milestones Snippets

+ ▾ This project Search 🔍 📄 🔔 1 📧 99+

S SFEP_Portal

🏠 Project

📁 Repository

📄 Issues 0

🔗 Merge Requests 2

🔧 CI / CD

⚙️ Operations

📖 Wiki

✂️ Snippets

⚙️ Settings

China Bu SH Group > Unilever > SFEP_Portal > Merge Requests

Open 2 Merged 421 Closed 83 All 506



Edit merge requests **New merge request**

🔍 Search or filter results... Created date ▾

Feature shipto
!506 · opened 5 hours ago by jintao.chu 🇻 Dev updated 2 hours ago

fix: activity
!503 · opened 21 hours ago by huihong.huang 🇻 Dev updated 1 hour ago

New Merge Request

Source branch	Target branch
<div>ChinaSH/Unilever/SFEP_Portal ▾</div> <div>feature-20230617 ▾</div>	<div>ChinaSH/Unilever/SFEP_Portal ▾</div> <div>Dev ▾</div>
<div> fix: 报错提示 huihong.huang authored 1 hour ago</div> <div>0db9dd49 📄</div>	<div> Merge branch 'feature-target' into 'Dev' ... 张腾伟 authored 9 hours ago</div> <div>b1c95d25 📄</div>
Compare branches and continue	

New Merge Request

From **feature-20230617** into **Dev**

[Change branches](#)

Title

Start the title with **WIP:** to prevent a **Work In Progress** merge request from being merged before it's ready.

Add [description templates](#) to help your contributors communicate effectively!

Description

Write Preview

Write a comment or drag your files here...

Markdown and [quick actions](#) are supported

[Attach a file](#)

Assignee

Milestone

Labels

Source branch

Target branch [Change branches](#)

☐ Remove source branch when merge request is accepted.


☐ Squash commits when merge request is accepted. [About this feature](#)

[Submit merge request](#)

[Cancel](#)

Commits 399 [Changes](#) 96+

12 Jun, 2023 1 commit

 fix: 报错提示
huihong.huang authored 1 hour ago

0db9dd49 

2. uni-app工程转为vue-cli项目

1.为什么要转:

- ① 转为vue-cli项目之后可以通过工程化的手段加入一些自己的逻辑, 比如执行构建之前自动修改AppID和域名, 减少手工修改的错误。
- ② 可以安装webpack插件, 来分析代码中的循环依赖问题
- ③ 能够根据环境的不同注入不同的变量, 例如uat和prd的blob地址是不一样的, 小程序用到的图片等静态资源都放到blob, 可以根据不同的环境去加载对应blob里面的资源
- ④ 能够自定义打包输出目录名
- ⑤ 可以用vscode或者其他ide开发小程序项目, HBuilderX不好用
- ⑥ 在执行对应环境的build命令之前, 加入自己的脚本控制, 比如可以控制只能在dev分支build dev环境的小程序, 只能在prd分支build prd环境的小程序, 防止发行prd时没有切换分支导致将没有验证过的代码发到线上
- ⑦ UL项目出现HBuilderX工具升级之后造成企微不能登录的生产环境的问题, 因为升级之后HBuilderX把wx.qq.login的qq抹掉了造成小程序在企微侧不能登录。如果用vue-cli项目创建项目则可以固定依赖, 避免因为升级插件造成bug

2.通过HBuilderX可视化创建的工程跟vue-cli创建的工程的区别:

- ① cli创建的项目，是传统的node项目结构。工程代码在src目录下，编译器在项目下，编译结果在dist目录下
- ② HBuilderX可视化创建的项目，是一种免node开发概念。工程代码在项目目录下，编译器在HBuilderX安装目录下而不是项目下，编译结果在项目的unpackage目录下。
- ③ HBuilderX可视化创建、运行、发布项目，底层调用的也是npm的run、build等命令。只是编译器不在项目下，而是在HBuilderX的目录下。

3.怎么转:

- ① 全局安装vue-cli，如果已经安装过可以忽略该步骤npm install -g @vue/cli
 - ② 创建一个空的vue工程vue create -p dcloudio/uni-preset-vue my-project，把通过HBuilderX创建的工程中除unpackage、node_modules（如果有）目录以外的代码挪至src目录下
 - ③ 将项目中的依赖包汇总到cli项目的package.json中，重新安装依赖
- 具体参考: <https://uniapp.dcloud.net.cn/quickstart-cli.html>

转为vue-cli项目的缺点：

- ① 每个项目都有一套编译器node_module，占用较多磁盘空间，如果用HBuilderX创建项目，则所有项目共用一套编译器，节省磁盘空间
- ② 通过HBuilderX创建的项目，编译器会随着HBuilderX工具的升级而自动升级，用vue-cli创建的项目则要手动升级，相对来说比较繁琐

转为vue-cli项目中遇到的问题:

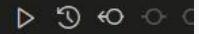
- ① HBuilderX工程使用的是node-sass来编译scss, node-sass依赖python环境, 本地还得安装python, 比较麻烦且node-sass非常容易安装失败, 对本地node版本还有要求, 用起来很繁琐, 所以替换为sass和sass-loader来解析scss
- ② 改为sass, sass-loader之后, 项目中/v-deep/语法报错。解决方法: 全局替换为::v-deep, 因为sass不支持/v-deep/
- ③ 现有项目里既有commonjs语法又有esmodule语法, 所以需要在babel.config.js里增加plugins.push('@babel/plugin-transform-modules-commonjs')

NodeJS	Supported node-sass version
Node 17	7.0+
Node 16	6.0+
Node 15	5.0+, <7.0
Node 14	4.14+
Node 13	4.13+, <5.0
Node 12	4.12+
Node 11	4.10+, <5.0
Node 10	4.9+, <6.0
Node 8	4.5.3+, <5.0
Node <8	<5.0

加入的一些脚本:

```
"scripts": {  
  "run:mp-weixin:dev": "cross-env BUILD_TYPE=dev NODE_ENV=development node build/modifyEnvProduction.js && cross-env NODE_ENV=development node build/modifyEnvProduction.js",  
  "run:mp-weixin:uat": "cross-env BUILD_TYPE=uat NODE_ENV=development node build/modifyEnvProduction.js && cross-env NODE_ENV=development node build/modifyEnvProduction.js",  
  "run:mp-weixin:prd": "cross-env BUILD_TYPE=prd NODE_ENV=development node build/modifyEnvProduction.js && cross-env NODE_ENV=development node build/modifyEnvProduction.js",  
  "build:mp-weixin:dev": "cross-env BUILD_TYPE=dev NODE_ENV=production node build/modifyEnvProduction.js && cross-env NODE_ENV=production node build/modifyEnvProduction.js",  
  "build:mp-weixin:uat": "cross-env BUILD_TYPE=uat NODE_ENV=production node build/modifyEnvProduction.js && cross-env NODE_ENV=production node build/modifyEnvProduction.js",  
  "build:mp-weixin:prd": "cross-env BUILD_TYPE=prd NODE_ENV=production node build/modifyEnvProduction.js && cross-env NODE_ENV=production node build/modifyEnvProduction.js",  
  "serve": "npm run dev:h5"
```

JS modifyEnvProduction.js X JS envConfig.js



build > JS modifyEnvProduction.js > modifyAppID > <function> > fs.readFile() callback

chaofan.zhang, 3个月前 | 1 author (chaofan.zhang)

```
1  const fs = require("fs");
2  const path = require("path");
3  const os = require("os");
4  const { exec } = require("child_process");
5
6  const envConfig = require("../envConfig");
7  const rootPath = path.resolve(__dirname, ".."); //项目根目录路径
8
9  const NODE_ENV_DEVELOPMENT = "development";
10 const NODE_ENV_PRODUCTION = "production";
11 const BUILD_TYPE_DEV = "dev";
12 const BUILD_TYPE_UAT = "uat";
13 const BUILD_TYPE_PRD = "prd";
14 const ENV_PRODUCTION_FILE_NAME = ".env.production";
15 const ENV_DEVELOPMENT_FILE_NAME = ".env.development";
16
17 const NODE_ENV_LIST = [NODE_ENV_DEVELOPMENT, NODE_ENV_PRODUCTION]; //NODE_ENV只支持development和production(development对应运行, production
18 const BUILD_TYPE_LIST = [BUILD_TYPE_DEV, BUILD_TYPE_UAT, BUILD_TYPE_PRD]; //BUILD_TYPE只支持dev、uat、prd。运行或者发行到哪个环境(根据BUILD
19
20 async function run() {
21   try {
22     //await checkBranch();
23     await clearEnv();
24     await writeEnv();
25     await modifyAppID();
26   } catch (e) {
27     console.error(`发生错误error=${e}`);
28     throw e;
29   }
30 }
```



```
JS modifyEnvProduction.js × JS envConfig.js
build > JS modifyEnvProduction.js > modifyAppID > <function> > fs.readFile() callback
chaofan.zhang, 3个月前 | 1 author (chaofan.zhang)
1  const fs = require("fs");
2  const path = require("path");
3  const os = require("os");
4  const { exec } = require("child_process");
5
6  const envConfig = require("../envConfig");
7  const rootPath = path.resolve(__dirname, ".."); //项目根目录路径
8
9  const NODE_ENV_DEVELOPMENT = "development";
10 const NODE_ENV_PRODUCTION = "production";
11 const BUILD_TYPE_DEV = "dev";
12 const BUILD_TYPE_UAT = "uat";
13 const BUILD_TYPE_PRD = "prd";
14 const ENV_PRODUCTION_FILE_NAME = ".env.production";
15 const ENV_DEVELOPMENT_FILE_NAME = ".env.development";
16
17 const NODE_ENV_LIST = [NODE_ENV_DEVELOPMENT, NODE_ENV_PRODUCTION]; //NODE_ENV只支持development和production(development对应运行, production
18 const BUILD_TYPE_LIST = [BUILD_TYPE_DEV, BUILD_TYPE_UAT, BUILD_TYPE_PRD]; //BUILD_TYPE只支持dev、uat、prd。运行或者发行到哪个环境(根据BUILD
19
20 async function run() {
21   try {
22     //await checkBranch();
23     await clearEnv();
24     await writeEnv();
25     await modifyAppID();
26   } catch (e) {
27     console.error(`发生错误error=${e}`);
28     throw e;
29   }
30 }
```

3.小程序&pc端项目上的优化

小程序项目上的优化:

随着项目的迭代, 模块越来越多, 功能越来越复杂, 就比较容易容易出现循环依赖的问题。对于简单的A模块依赖B模块, B又依赖A这种是比较容易发现的, 但实际情况是A依赖B, B依赖C, C依赖D, 最后D又依赖A, 这种间接的循环依赖就难以发现, 造成的影响就是对象上属性为undefined

举例:

// main.js

```
const bar = require('./bar.js')
console.log('当前是main.js内:', bar) // {}
module.exports = '在main.js内'
```

// bar.js

```
const main = require('./main.js')
console.log('当前是bar.js内:', main)
module.exports = 'bar.js内'
```

// 执行 node ./main.js , 输出:

当前是bar.js内: {} // 解析: 执行到bar.js内时, main.js还没有执行完, 就没有东西导出, 会默认导出空对象

当前是main.js内: bar.js内

在main.js中引用bar.js暴露出来的方法时就会失败

3.小程序&pc端项目上的优化

解决方法:

npm install circular-dependency-plugin -D

vue.config.js中的配置如右图

运行打包时该插件就会分析出项目中存在的循环依赖情况

```
1 // webpack.config.js
2 const CircularDependencyPlugin = require('circular-dependency-plugin')
3
4 module.exports = {
5   entry: './src/index',
6   plugins: [
7     new CircularDependencyPlugin({
8       // 排除检测符合正则的文件
9       exclude: /a\.js|node_modules/,
10      // 将符合正则的文件包含在内
11      include: /dir/,
12      // 向 webpack 输出错误而不是警告
13      failOnError: true,
14      // 允许包含异步导入的循环
15      // 举例: via import(/* webpackMode: "weak" */ './file.js')
16      allowAsyncCycles: false,
17      // 设置当前的工作目录以显示模块路径
18      cwd: process.cwd(),
19    })
20   ]
21 }
```



```
> webpack
```

```
assets by status 533 bytes [cached] 1 asset
```

```
./demo12/commonjs/index.js 193 bytes [built] [code generated]
```

```
./demo12/commonjs/a.js 283 bytes [built] [code generated]
```

```
./demo12/commonjs/b.js 283 bytes [built] [code generated]
```

```
WARNING in configuration
```

```
The 'mode' option has not been set, webpack will fallback to 'production' for this value.
```

```
Set 'mode' option to 'development' or 'production' to enable defaults for each environment.
```

```
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/
```

```
ERROR in Circular dependency detected:
```

```
demo12/commonjs/a.js -> demo12/commonjs/b.js -> demo12/commonjs/a.js
```

```
ERROR in Circular dependency detected:
```

```
demo12/commonjs/b.js -> demo12/commonjs/a.js -> demo12/commonjs/b.js
```

PC项目上的优化:

1: useless-files-webpack-plugin

vue.config.js配置如右图。运行npm run build之后，会在项目根目录下生成一个unused-files.json文件，保存着无用的文件列表，根据列表中提供的路径，确认不需要了手动删除文件即可。

2: 升级webpack版本，由webpack4.X升级为webpack5，原先webpack4随着项目代码的增多，热更新速度越来越慢，仅改动一个字符，就要等将近1分钟才能更新到页面上，大大影响开发效率，升级为webpack5之后，热更新只要10s左右

```
const UnusedFilesWebpackPlugin = require('useless-files-webpack-plugin')

plugins: [
  new UselessFile({
    root: './src', // 项目目录
    output: './unused-files.json', // 输出文件列表
    clean: false // 删除文件，
  })
]
```