代码编写规范

前端代码规范

1. 命名规范

1.1 文件和目录命名

- 小程序目录和文件名使用驼峰法命名方式。

```
pages/
  home/
    home.wxml
    home.wxss
    home.js
    home.json
```

1.2 变量和函数命名

- 使用驼峰命名法（camelCase）。
- 变量名和函数名要有描述性，避免使用单个字符或无意义的命名。

```
// 不好的命名var a = 10;

// 好的命名var itemCount = 10;
```

1.3 组件命名

- 使用大驼峰命名法（PascalCase）。
- 组件名应该具有描述性，反映组件的功能。

```
// 不好的命名var myComponent = { /* ... */ };

// 好的命名var MyCustomComponent = { /* ... */ };
```

2. 编码风格

2.1 缩进

- 使用两个空格进行缩进。

2.2 空格

- 在运算符前后添加空格，提高代码可读性。

```
// 不好的写法var sum=a+b;

// 好的写法var sum = a + b;
```

2.3 换行

- 在函数参数较多时，适当换行，保持代码的清晰可读性。

```
// 不好的写法function exampleFunc(arg1, arg2, arg3, arg4, arg5) {// ...
}

// 好的写法function exampleFunc(
  arg1,
  arg2,
  arg3,
  arg4,
  arg5
) {// ...
}
```

3. 注释

3.1 行内注释

- 使用行内注释解释关键步骤或复杂代码块。

```
var total = price * quantity; // 计算总价
```

3.2 多行注释

- 使用多行注释对代码块进行整体描述。

- /*
  这是一个示例函数，用于演示多行注释的使用
  参数：
  param1: 参数1的说明
  param2: 参数2的说明
  */function exampleFunction(param1, param2) {// ...
  }

4. 其他规范

4.1 小程序 API 调用

- 将小程序 API 调用集中放在页面或组件的生命周期函数中，以提高代码的清晰度。

```
Page({onLoad() {// 页面加载时调用的 API
  },onShow() {// 页面显示时调用的 API
  },// 其他生命周期函数...
});
```

4.2 页面函数顺序

- 将页面构造相关函数放在前面
- 例如4.1中部分，onLoad,onShow等函数优先定义。

后端代码规范

1. 项目结构

1.1 项目目录结构

- 使用 Django 推荐的项目目录结构，确保清晰的组织和模块化。

```
plaintextCopy code
myproject/
  ├── myapp/
  │   ├── migrations/
  │   ├── templates/
  │   ├──
__init__
.py
  │   ├── admin.py
  │   ├── apps.py
```

```
11  │   ├── models.py
12  │   ├── views.py
13  │   └── ...
14  ├── myproject/
15  │   ├──
16  __init__
17  .py
18  │   ├── settings.py
19  │   ├── urls.py
20  │   └── wsgi.py
21  └── manage.py
```

2. 数据库模型

2.1 模型命名

- 使用单数形式为模型命名，采用驼峰命名法。

2.2 字段命名

- 使用小写字母和下划线分隔的方式（snake_case）为字段命名。

```
1  pythonCopy code
2  class UserProfile(models.Model):
3      first_name = models.CharField(max_length=50)
4      last_name = models.CharField(max_length=50)
```

2.3 CharField 和 TextField

- 对于较长文本字段，优先使用 `TextField` 。

```
1  pythonCopy code
2  class BlogPost(models.Model):
3      title = models.CharField(max_length=255)
4      content = models.TextField()
```

2.4 索引

- 在需要进行查询的字段上添加索引，以提高检索性能。

```
1  pythonCopy code
2  class UserProfile(models.Model):
```

```
3     email = models.EmailField(unique=True, db_index=True)
```

3. 视图

3.1 视图函数

- 使用有意义的函数名，明确函数的作用。

```
1 def user_profile(request, user_id):...
```

3.2 使用 Django Rest Framework

- 如果项目需要提供 RESTful API，优先使用 Django Rest Framework。

4. URL 设计

4.1 URL 命名

- 使用有意义的 URL 命名，遵循 RESTful 风格。

```
1 pythonCopy code
2 path('users/', views.user_list, name='user-list')
```

5. 查询和过滤

5.1 使用 QuerySet 方法

- 使用 Django 提供的 QuerySet 方法，避免直接使用原始 SQL 查询。

```
1 pythonCopy code
2 users = UserProfile.objects.filter(is_active=True)
```

6. 注释和文档

6.1 注释

- 提供清晰的注释，解释代码的目的和特殊处理。

6.2 文档

- 使用文档字符串为模块、类和函数提供说明文档。

git分支操作与工作流

git分支操作

请在本地项目根目录下进行：

- 分支查看

git branch --list

其中绿色的是当前分支

- 创建新分支

> git checkout -b [branch_name]

- 删除分支

> git branch -D [branch_name]

- 切换当前本地分支

> git checkout [branch_name]

- 拉取远程主分支(请在main分支下进行)

> git pull --prune

- 拉取远程其他分支并在本地创建对应的新分支

> git checkout -f [remote_branch_name]

git工作流

在开发之前，请先**将远程main分支pull到本地！**

在进行新的修改之前，请**建立新分支！**

工作流

进行新分支的创建，并在新分支上进行修改。

git add [进行过修改的文件]

请将所有需要提交的文件使用git add添加到本地暂存区。

git commit -m "[commit_name]"

如果远程已经存在将要提交的commit所在的分支，请使用：

git push origin [branch_name]

否则，请使用：

git push -u origin [branch_name]

然后请在github.com上查看pr并填写相关信息，申请合并。

commit与分支命名规范

分支命名：

使用[分支类型]_[函数名]进行命名。

分支类型如下：

- feature：新功能
- bugfix：bug修复
- rewrite：重构
- test：测试

commit提交

commit进行提交时遵循以下命名规范：

[分支名]_[commit状态]

commit状态指当前commit结束后代码所处状态，包括：

- unfinished：尚未完成，主体功能仍有欠缺
- uncomfirmed：代码基本实现，正确性尚未验证
- untested：正确性部分验证，测试尚未实现
- tested：进行了部分测试且正确
- unfixed：bug尚未修复
- fixed：进行测试，原先已经修复
- backed：发生了回退情况

其中，前4中状态针对功能开发与重构，后三种针对bug修复。

提交说明依然会根据情况进行修改。

还请大家按说明提交（可能会比较繁琐）

尽量避免一个分支涉及多个函数的修改，如果不慎涉及，请在说明中写出，并将分支命名为[]*[]+*

如果涉及路由修改，请使用[][urls]