# 编程规范

## 1.命名

### 1.1.变量命名

代码中代码严禁拼音和英文混合的方式，更不允许直接用纯中文标识。

布尔类型的变量不要加 is 前缀。例如 isDeleted 变量，RPC 框架在反向解析的时候以为属性是 delect

### 1.2.接口命名

接口类中的方法和属性不要添加任何修饰符号，保持代码的间接性。

### 1.3.函数命名

函数命名采用大小写结合，突出其功能的形式，如 getQusestion。

### 1.4.注意点

命名中尽可能的少出现数组以及单字母的变量，尽可能用其英文释义来进行命名，增加可读性。

## 2.前端规范

### 2.1.框架

前端使用 react 框架&antd 组件，进行编程，保证命名一致的情况下，需要尽可能提高代码的复用率，减少重复代码的出现

### 2.2.语法

（1）用两个空格来代替制表符（tab） -- 这是唯一能保证在所有环境下获得一致展现的方法。

（2）为选择器分组时，将单独的选择器单独放在一行。

（3）为了代码的易读性，在每个声明块的左花括号前添加一个空格。

（4）声明块的右花括号应当单独成行。

（5）每条声明语句之后应该插入一个空格。

（6）为了获得更准确的错误报告，每条声明都应该独占一行。

（7）所有声明语句都应当以分号结尾。最后一条声明语句后面的分号是可选的，但是，如果省略这个分号，你的代码可能更易出错。

（8）对于以逗号分隔的属性值，每个逗号后面都应该插入一个空格（例如 ，box-shadow）。

（9）不要在 rgb()、rgba()、hsl()、hsla() 或 rect() 值的内部的逗号后面插入空格。这样利于从多个属性值（既加逗号也加空格）中区分多个颜色值（只加逗号，不加空格）。

（10）对于属性值或颜色参数，省略小于 1 的小数前面的 0 （例如，.5 代替 0.5；-.5px 代替 -0.5px）。

（11）十六进制值应该全部小写，例如，#fff。在扫描文档时，小写字符易于分辨，因为他们的形式更易于区分。

（12）尽量使用简写形式的十六进制值，例如，用 #fff 代替 #ffffff。

（13）为选择器中的属性添加双引号，例如，input[type="text"]。只有在某些情况下是可选的，但是，为了代码的一致性，建议都加上双引号。

（14）避免为 0 值指定单位，例如，用 margin: 0; 代替 margin: 0px;。

### 2.3. 注释

（1）代码是由人编写并维护的。请确保你的代码能够自描述、注释良好并且易于他人理解。好的代码注释能够传达上下文关系和代码目的。不要简单地重申组件或 class 名称。

（2）对于较长的注释，务必书写完整的句子；对于一般性注解，可以书写简洁的短语

### 2.4. 代码组织

（1）以组件为单位组织代码段。

（2）制定一致的注释规范。

（3）使用一致的空白符将代码分隔成块，这样利于扫描较大的文档。

（4）如果使用了多个 CSS 文件，将其按照组件而非页面的形式分拆，因为页面会被重组，而组件只会被移动。

### 2.5. 示例

```
handlePost = () =>{
    if (this.state.header === '' || this.state.content === '' || this.state.theme === [])
        return;
    let url = apiUrl + '/postQuestion';
    let postJson = {
        header: this.state.header,
        content: draftjs(this.state.content),
        userId: parseInt(localStorage.getItem('userId')),
        theme: this.state.theme,
        tags: this.state.tag
    }
    console.log(postJson)

    let callback = data => {
        console.log(data);
        if (data.status === 0) {
            message.info("发帖成功")
            this.setState({
                header: '',
                content:'',
                theme: [],
                tag: []
            })
        }
        if (data.status === 1) {
            message.error(data.msg);
        }
        if (data.status === 2) {
            message.error('登录失效');
            history.replace("/SignInPage")
        }
    };
    postRequest(url, postJson, callback)
        .then((data) => {
            callback(data);
        })
        .catch((error)=>{
            console.log(error);
        });
};
```

## 3. 后端代码规范

会对后端的代码规范进行介绍，主要存在 （1）entity repository dao service controller 分层，（2）接口与实现分离（3）控制反转等

### 3.1. 基础规范

（1）排版：缩进必须用 space，不能使用 tab 键，可以在 eclipse 或其他开发工具配置一个 tab 用 4 个 space 代替；单行字符数不超过 120 个；开发工具建议使用 intellij Idea，工具稳定性好，智能化，内存消耗稳定；可以使用工具自动格式化功能。

（2）类名：大驼峰式命名，即单词首字母大写，如：UserService;抽象类必须用 abstract 开头；接口名不加前缀；接口的实现类必须加上 Impl，如 UserServiceImpl。

（3）注释规范：功能方法接口名称必须有注释； 复杂代码逻辑必须有注释； 代码注释不超过一行使用’//’，超过一行使用‘/* */’ 。

（4）代码提交规范：原则上完成一个完整功能并自测无异常后，方可 checkin 代码，必须保证无编译报错；提交代码必须写注释，能够完整描述本次提交变更的内容

### 3.2.分层规范

按照 entity repository dao service controller 分层，用传统的 spring boot 来进行实现，保证层与层之间的接口一致，实现可改，避免代码的大量无效修改。

### 3.3.接口设计规范

（1） 一切基于接口开发，提供业务逻辑服务必须提供接口

（2） 接口方法参数超过 3 个，需要转换为 dto 属性，对象入参

（3） 对外提供服务接口 JSON 返回结构[状态,错误代码,错误描述,数据]

（4） Module 内部服务接口类名以 Service 结尾，接口方法细粒度;对外提供接口类名以 Facade 结尾，接口方法定义粗粒度，禁止跨域调用 DAO。

（5） Service 实现内部细粒度业务逻辑，Service 不能跨域调用 Service。

（6） Façade 层负责对外提供粗粒度功能方法，采用模板设计模式方法，实现仅包括：1. 入参校验；2. 入参转换、清洗；3. 粗粒度业务功能流程控制（具体实现下沉到 Service 层，超过 5 个步骤应考虑下沉到 Service）；4.出参数据转换。

（7） Facade 调用：不同发布单元接口（Façade）调用，封装成内部 Service 方式，并实现入参出参记录，耗时统计等，系统内部只与内部 Service 交互；系统内不同待拆分模块之前接口调用 Façade。

（8） 尽量使用缓存：配置类数据使用内存（一级）缓存；业务类数据使用 redis（二级）缓存

（9） 服务必须设计为无状态服务，即，请求可以在任何实例完成处理

（10）服务设计为基于 https 短连接服务

（11）服务设计可并发执行、幂等性

（12）禁止使用 Map 作为入参出参，会增加接口复杂度，容易造成序列化、反序列化失败

（13）接口入参出参禁止删除字段、修改参数字段名，

（14）接口可以新增入参，出参；如果接口逻辑不兼容修改，必须使用版本号，并通知下游修改

（15）接口默认使用 dubbo 协议，单次请求入参、出参不能超过 100k，否则必须更换 hessian 协议

### 3.4.数据库设计规范

（1） 禁止删除字段

（2） 禁止更新字段名称，类型,减少长度；可以修改增加字段长度或备注

（3） 数据库脚本支持提前上线，否则会影响热发布或者灰度发布

（4） 每个领域实体表必须有一个领域主键驱动，便于信息查询

（5） 查询结果按需读取，防止因字段过大造成数据传输开销

（6） SQL 命名准确，功能要单一，不能包含多种含义功能，维护会更复杂

### 3.5.示例

```
@Override
public Msg postQuestion(JSONObject object){
    if((!object.containsKey("header"))
            || (!object.containsKey("content"))
            || (!object.containsKey("userId"))
            || (!object.containsKey("theme"))
            )
        return MsgUtil.makeMsg(MsgCode.PARAMETER_ERROR);
    int userId = -1;
    List<String> tagList = new ArrayList<>();
    String header = "";
    String content = "";
    String theme = "";
    try {
        userId = (Integer) (object.get("userId"));
    }catch (Exception e) {
        return MsgUtil.makeMsg(MsgCode.ERROR,"userId必须是个整型常量 :(");
    }
    try {
        header = (String) (object.get("header"));
    }catch (Exception e) {
        return MsgUtil.makeMsg(MsgCode.ERROR,"header必须是个字符串 :(");
    }
    try {
        content = (String) (object.get("content"));
    }catch (Exception e) {
        return MsgUtil.makeMsg(MsgCode.ERROR,"content必须是个字符串 :(");
    }
    try {
        theme = (String) (object.get("theme"));
```

## 4.开发原则

4.1.所有团队的程序模块都要以通过 Service Interface 方式将其数据与功能开放出来。

4.2.团队间的程序模块的信息通信，都要通过这些接口。

4.3.除此之外没有其它的通信方式。其他形式一概不允许：不能使用直接链结程序、不能直接读取其他团队的数据库、不能使用共享内存模式、不能使用别人模块的后门等等，唯一允许的通信方式只能是能过调用 Service Interface。

4.4.任何技术都可以使用。比如：HTTP、Corba、Pubsub、自定义的网络协议等等，都可以。

4.5.所有的 Service Interface，毫无例外，都必须从骨子里到表面上设计成能对外界开放的。也就是说，团队必须做好规划与设计。