文档4：编程规范

修订历史记录

编写日期	版本	说明	作者
3.25	1.0	前后端编码的规范制定	黄彦铭、杨严

1.编程一致性要求

1.1 代码风格

1.1.1 使用统一的缩进风格，建议使用四个空格进行缩进。

1.1.2 命名规范请参考第2条。

1.1.3 注释请使用中文，并且应该清晰、简洁明了，用以解释代码的意图、原理或设计思路，方便他人阅读和维护。

1.2 便于维护

1.2.1 代码中尽可能不要使用硬编码，而是使用常量或配置文件中的变量来代替，方便后期修改和维护。

1.2.2 不要使用过多的注释来解释代码的含义，而应该尽可能使用清晰的命名和结构化的代码来让代码自身就能够表达其含义。

2.命名规范

2.1 类名和变量名

2.1.1 类名应该采用首字母大写的驼峰式命名规则，例如：PersonInfo。

2.1.2 变量名应该采用首字母小写的驼峰式命名规则，例如：personInfo。

2.2 表名和字段名

2.2.1 表名应该采用小写的下划线命名规则，例如：person_info。

2.2.2 字段名应该采用小写的下划线命名规则，例如：person_name。

3.程序框架

3.1 技术选型

3.1.1 微信小程序前端

3.1.2 SpringBoot后端采用SpringMVC作为编程框架，使用MybatisPlus作为关系对象映射，使用MySQL作为关系型数据库。

3.2 编程样例

3.2.1 前端编程样例

框架的视图层由 WXML 与 WXSS 编写，由组件来进行展示。
将逻辑层的数据反映成视图，同时将视图层的事件发送给逻辑层。
WXML(WeiXin Markup language) 用于描述页面的结构。
WXS(WeiXin Script) 是小程序的一套脚本语言，结合 WXML，可以构建出页面的结构。
WXSS(WeiXin Style Sheet) 用于描述页面的样式。
组件(Component)是视图的基本组成单元。

WXML

（WeiXin Markup Language）是框架设计的一套标签语言，结合基础组件、事件系统，可以构建出页面的结构。

```
<!--pages/discovery/discovery.wxml-->
<view class="yy_page display_flex">
  <search-icon></search-icon>
  <text>pages/discovery/discovery.wxml</text>
</view>
```

WXSS

WXSS (WeiXin Style Sheets)是一套样式语言，用于描述 WXML 的组件样式。

WXSS 用来决定 WXML 的组件应该怎么显示。

为了适应广大的前端开发者，WXSS 具有 CSS 大部分特性。同时为了更适合开发微信小程序，WXSS 对 CSS 进行了扩充以及修改。

```
/** app.wxss **/
@import "common.wxss";
```

```
.middle-p
{
    padding:15px;
}
```

WXS

WXS（WeiXin Script）是小程序的一套脚本语言，结合 `WXML` ，可以构建出页面的结构。

1. WXS 不依赖于运行时的基础库版本，可以在所有版本的小程序中运行。
2. WXS 与 JavaScript 是不同的语言，有自己的语法，并不和 JavaScript 一致。
3. WXS 的运行环境和其他 JavaScript 代码是隔离的，WXS 中不能调用其他 JavaScript 文件中定义的函数，也不能调用小程序提供的API。
4. WXS 函数不能作为组件的事件回调。
5. 由于运行环境的差异，在 iOS 设备上小程序内的 WXS 会比 JavaScript 代码快 2 ~ 20 倍。在 android 设备上二者运行效率无差异。

```
<!--wxml-->
<!-- 下面的 getMax 函数，接受一个数组，且返回数组中最大的元素的值 -->
<wxs module="m1">
var getMax = function(array) {
  var max = undefined;
  for (var i = 0; i < array.length; ++i) {
    max = max === undefined ?
      array[i] :
      (max >= array[i] ? max : array[i]);
  }
  return max;
}

module.exports.getMax = getMax;
</wxs><!-- 调用 wxs 里面的 getMax 函数，参数为 page.js 里面的 array -->
<view> {{m1.getMax(array)}} </view>
```

3.2.2 后端编程样例

后端使用Springboot编程时，需要遵循以下统一规范：

(1)每个接口需要有Entity，Mapper，DTO，Service(ServiceImpl)，Controller，按照从左到右的顺序依次往上层传递。

(2)Entity为实体类，从数据库中取出的数据都必须先放在Entity中。DTO是数据传输对象，负责Service和Controller之间的数据传输，Entity必须要在到达Service层之前就转换为DTO。Entity转换为DTO是在Mapper层实现的，需要在Mapper接口中写一个toDTO方法，这个方法会由Mybatis自动进行实现，不需要手动实现。

(3)每个Service接口可能有一个或多个ServiceImpl类进行实现。例如，我们可能会定义一个OrderService 接口来管理订单，然后有两个不同的实现类：OnlineOrderServiceImpl 和OfflineOrderServiceImpl，分别用于处理线上订单和线下订单。在这种情况下，两个实现类都实现了OrderService 接口中的方法，但是实现的逻辑可能是不同的。

以下是利用Springboot编写一个接口的样例：

Student实体

```
public class Student {
    private Long id;
    private String name;
    private Integer age;
    private String gender;

    // getter和setter省略
}
```

StudentMapper

```
@Mapper
public interface StudentMapper {
    StudentDTO toDTO(Student student);    //Mapper中要写toDTO函数,这个toDTO方法系统

    List<Student> getAllStudents();
    Student getStudentById(Long id);
    void addStudent(Student student);
    void updateStudent(Student student);
    void deleteStudentById(Long id);
}
```

StudentDTO

```
public class StudentDTO {
    private Long id;
    private String name;
    private Integer age;
```

```
    private String gender;

    // getter和setter省略
}
```

StudentService

```
@Service
public class StudentService {
    private final StudentMapper studentMapper;

    public StudentService(StudentMapper studentMapper) {
        this.studentMapper = studentMapper;
    }

    public List<StudentDTO> getAllStudents() {
        return studentMapper.getAllStudents().toDTO();
    }

    public StudentDTO getStudentById(Long id) {
        return studentMapper.getStudentById(id).toDTO();
    }

    public void addStudent(StudentDTO studentDTO) {
        Student student = new Student();
        student.setName(studentDTO.getName());
        student.setAge(studentDTO.getAge());
        student.setGender(studentDTO.getGender());
        studentMapper.addStudent(student);
    }

    public void updateStudent(StudentDTO studentDTO) {
        Student student = new Student();
        student.setId(studentDTO.getId());
        student.setName(studentDTO.getName());
        student.setAge(studentDTO.getAge());
        student.setGender(studentDTO.getGender());
        studentMapper.updateStudent(student);
    }

    public void deleteStudentById(Long id) {
        studentMapper.deleteStudentById(id);
    }
}
```

StudentController

```
@RestController
@RequestMapping("/students")
public class StudentController {
    private final StudentService studentService;

    public StudentController(StudentService studentService) {
        this.studentService = studentService;
    }

    @GetMappingpublic Result<List<StudentDTO>> getAllStudents() {
        List<StudentDTO> students = studentService.getAllStudents();
        return new Result<>(true, students);
    }

    @GetMapping("/{id}")public Result<StudentDTO> getStudentById(@PathVariable L
        StudentDTO student = studentService.getStudentById(id);
        return new Result<>(true, student);
    }

    @PostMappingpublic Result<Void> addStudent(@RequestBody StudentDTO studentDT
        studentService.addStudent(studentDTO);
        return new Result<>(true);
    }

    @PutMappingpublic Result<Void> updateStudent(@RequestBody StudentDTO student
        studentService.updateStudent(studentDTO);
        return new Result<>(true);
    }

    @DeleteMapping("/{id}")public Result<Void> deleteStudentById(@PathVariable L
        studentService.deleteStudentById(id);
        return new Result<>(true);
    }
}
```

4.包规范

4.1 前端包规范

(1)components：组件，被页面调用

(2)pages：页面文件夹，页面的所有代码文件的名字必须和本文件夹名字相同

(3)icon:静态图片

(4)util:通用的js工具函数

(5)app.js:项目的全局

(6)app.json:全局配置文件

(7)app.wxss:全局样式文件

(8)project.config.json:项目的配置文件

(7)sitemap.json:微信索引配置文件

4.2 Springboot后端包规范

(1) 项目基本包：com.tp.backend.{模块名}

(2) 配置文件：pom.xml与application.yml

(3) config：配置类

(4) common：公共类，定义常量类，通用组件

(5) entity: 数据库相关的实体类 ，有得使用pojo

(6) model:数据模型类(参数模型，数据传输模型等) ，有的使用vo

(7) controller:控制层接口

(8) service: 服务层

(9) mapper：数据库访问层

(10) util： 工具类

5.开发环境规范

- (1) 开发环境：JDK 17
- (2) 开发工具：IntelliJ IDEA、微信开发者工具
- (3) 构建工具：Maven 3
- (4) 代码管理工具：github
- (5) 分支管理：开发时需要在各自的dev分支上进行，部署时需要在master分支上合并
- (6) 静态代码分析工具：SonarQube 9.9

6.版本分支管理规范

6.1使用工具

前端和后端代码仓库都使用Github进行代码版本控制管理，并遵循统一的管理规范

6.2 开发主干

命名：master

内容：可以部署到生产环境中的，正常运作的代码，是其他开发分支的基本底盘。

修改：

一般地，只允许其他branch将代码合入，不允许向master分支直接提交代码。

特殊地，涉及项目的基础框架、环境配置时，可以直接向master提交代码。

通常是branch阶段性开发完成后，branch合并到master中。

master内容更新后，其他开发分支应当主动地将master合并到自己分支中。

6.3 开发分支

命名：dev-XXX，XXX建议为个人姓名缩写

内容：在master主干的基础上，针对某些模块进行的进一步具体开发

修改：

分支负责人可自由修改本开发分支。

开发分支取得阶段性成果，确认可正常运行后，开发分支向master主干合并，并修缮代码合并的问题，确保合并后代码正常运行才能push到master。

以上是本次的编程规范文档，开发人员需要遵守以上的规范，以确保代码的可读性、可维护性和可扩展性。