Java开发规范

1 编码规范

1.1 源文件

（1）文件名

源码文件名由顶级 class 的类名，加上 .java 后缀组成。

（2）文件编码

源码文件使用 UTF-8 编码。

（3）文件结构

源码文件按照先后顺序，由以下几部分组成：

- License 或 copyright 声明信息
- `package` 语句
- `import` 语句
- 顶级类或接口声明

（4）类和接口声明

类和接口个组成部分声明按照以下顺序：

- 类或接口的文档注释（/*.....*/）
- class 或 interface 语句
- 类或接口的实现注释（/*.....*/）
- 静态变量，顺序按照 public > protected > package level > private
- 实例变量，顺序同上
- 构造方法
- 方法，按功能分组

1.2 格式

（1）基本规范

- 缩进使用 4 个空格，禁止使用 Tab
- 一行只有一个语句
- 一行只声明一个变量
- 单行长度不超过 120，`package` 和 `import` 语句不受长度限制
- 换行符使用 Unix 格式：`\n`
- `long` 和 `float` 类型声明使用大写 `L` 和 `F`
- 声明数组使用 Java 风格，禁止使用 C 风格，示例：

```
错误：String args[]
正确：String[] args
```

（2）换行

当一行语句长度超过 120，使用以下规则断开：

- 逗号后面断开
- 操作符前面断开
- 换行缩进使用 8 个空格

（3）语句块

语句块指类、方法以及`if/else/for/do/while/switch/try/catch/finally`语句之后的代码块，语句块总是使用 `{ }` 符号，使用规则：

- `{` 位于起始行结尾
- `}` 换行并与起始行对齐
- 被括语句使用一个缩进单位
- 语句块只有单行语句也必须使用 `{ }`

（4）空格

- 所有保留关键字前后使用一个空格
- 语句块的 `{` 前面使用一个空格
- 一元操作符不加空格
- 二元或三元操作符使用一个空格，除了 `.`
- `,`、`:`、`;`、`)` 后面使用一个空格
- 强制转型后面使用一个空格

（5）空行

空行将不同的代码段分隔开，用以提高可读性：

- 源文件和类的各组成部分之间都是用一个空行
- 文件结尾以一个空行结束
- 类的第一个成员前面和最后一个成员后面使用一个空行
- 方法之间使用一个空行
- 语句块前后各使用一个空行
- `/*……*/` 和 `//` 注释之前使用一个空行
- （可选）成员变量之间使用一个空行做逻辑分组
- （可选）一个方法内两个逻辑段之间使用一个空行

1.3 命名

（1）包

全部使用小写字符并且前缀总是一个顶级域名，禁止使用下划线。

示例：`com.sun.eng`、`com.apple.quicktime.v2`、`cn.xhd`

（2）类或接口

使用大写开头的驼峰式命名：**UpperCamelCase**

（3）方法

使用小写开头的驼峰式命名：**lowerCamelCase**

（4）变量

使用小写开头的驼峰式命名：**lowerCamelCase**

（5）常量

全部使用大写字符，单词之间用下划线隔开：**CONSTANCE_CASE**

（6）其它

- 类名使用名词或名词短语，方法名使用动词开头
- 禁止使用拼音或中文
- 缩写只能使用通用的熟知的缩写字符，且全部使用大写字符，示例：`HTML`、`API`、`DTO`
- 抽象类名使用 `Abstract` 开头
- 异常类名使用 `Exception` 结尾
- 测试类名使用 `Test` 结尾，示例：`HashTest`
- 测试方法名使用 `test` 开头，格式：`test<methodUnderTest>_<state>`，示例：`testPop_emptyStack()`
- 布尔类型变量名使用形容词，禁止开头使用 `is`

1.4 注释

（1）Javadoc

- Java 独有的，由 `/**……*/` 界定，可通过 Javadoc 工具生成 HTML 文档。
- `public` 类必须使用 Javadoc
- `public` 和 `protected` 成员变量和成员方法必须使用 Javadoc
- 例外：方法名显而易见时，可以不使用 Javadoc，示例：`getFoo`、`setFoo`
- 例外：Override 或 Overload 方法有时可不使用 Javadoc

（2）其它

- 使用中文
- `//` 注释后面使用一个空格，示例：`// 这是一条注释`
- 特殊注释标记，待办事项：`// TODO xxx`
- 特殊注释标记，错误，不能工作：`// FIXME xxx`
- 谨慎注释掉代码，无用则删除，代码仓库可查历史记录

1.5 编程实践

（1）访问控制

保持类成员变量和成员方法的可见性最小。

（2）静态成员的访问

避免用一个对象访问此类的静态变量和方法，用类名替代。

```
Foo aFoo = …;
Foo.aStaticMethod(); // OK
aFoo.aStaticMethod(); // 避免
```

（3）`@Override`

所有重写的方法，必须使用 @Override 注解。

（4）`@Deprecated`

避免使用有 `@Deprecated` 注解的方法。

（5）常量调用 `equals()`

对象和常量 `equals` 时，将常量放在前面，避免空指针异常，示例：`"test".equals(aString)`。

（6）避免 clone()

避免重写 `clone()`，谨慎使用 `clone()`。

（7）避免 `finalize()`

避免重写 `finalize`。

（8）异常捕获

避免忽略catch的异常处理。

（9）格式化代码

提交代码之前在本地 IDE 中执行格式检查和整理 `import` 语句。

1.6 Lombok实践

（1）数据对象

数据容器类型的对象，在类上加 `@Data` 注解，包括接口层传输对象，领域模型层对象

（2）值对象

领域模型层如需使用值对象，在类上加 `@Value` 注解。

（3）日志

所有 Java 工程日志统一使用 Slf4j 作为日志接口，底层日志框架统一使用 logback，需要打日志的类加 `@Slf4j` 注解

（4）IDEA 设置

安装 Lombok 插件之后，需要在项目上进行以下设置：

1. 打开Preferences -> Build, Execution, Deployment -> Compiler -> Annotation Processors
2. 勾选“Enable annotation processing”
3. 选择“Obtain processors from project classpath”

1.7 service 层标准方法命名（rpc 和 controller 层也适用）

后台管理 CRUD 系统操作方法，请使用以下统一的命名形式：

方法	说明	示例
`List<T> list<名词复数>(<查询条件参数>)`	返回资源列表	`List<User> listUsers(String name, Date createTime)`
`T get<名词单数>(Long id)`	根据 id 返回单个资源	`User getUser(Long id)`
`void create<名词>(T entity)`	创建资源	`void createAppVersion(AppVersion version)`
`void update<名词>(T entity)`	更新资源	`void updateUser(User user)`
`void delete<名词>(Long id)`	根据 id 删除资源	`void deleteUser(Long id)`
`void batchCreate<资源复数>(List<T> entities)`	批量创建资源	`void batchCreateUsers(List<User> users)`
`void batchUpdate<资源复数>(List<T> entities)`	批量更新资源	`void batchUpdateUsers(List<User> users)`
`void batchDelete<资源复数>(List<Long> ids)`	批量删除资源	`void batchDeleteUsers(List<Long> ids)`

开发前台系统操作方法，请参考以下命名建议：

方法	说明	示例
`List<T> get<业务条件><名词复数>(<条件>)`	根据业务条件获取资源列表	`Category getEnabledCategories()`
`T get<业务条件><名次单数>(<条件>)`	根据业务条件获取单个资源	`AppVersion getLatestAppVersion()`
写操作	使用通用业务语言，避免使用技术语言	下单： 使用 `void placeOrder()` 不要 `addOrder` 或者 `createOrder`

1.8 repository 层标准方法命名

数据访问层方法，请使用以下统一的命名形式：

方法	说明	示例
`void save(T entity)`	保存单个资源	
`void saveAll(List<T> entities)`	批量保存资源	
`T findById(Long id)`	根据 id 查找资源	
`List<T> findAll()`	无条件查找全部资源	
`Page<T> findAllBy(<查询条件参数>)`	后台管理列表查找资源，通常带分页	`List<User> findAllBy(String name, String role)`
`List<T> | T findBy<条件>(<查询条件参数>)`	根据少量条件查找资源（3或以内）	`List<User> findByTag(String tag)`
`List<T> | T find<标签>By<条件>(<查询条件参数>)`	根据少量条件查找特定业务场景资源	`AppVersion findLatestByType(OperatingSystem os)`
`List<T> findByIds(List<Long> ids)`	根据 id 批量查找资源	
`boolean existsBy<条件>(<查询条件参数>)`	根据条件判断资源是否存在	`boolean existsByUserId(Long userId)`
`long countBy<条件>(<查询条件参数>)`	根据条件计数	`long countByCategoryId(Long categoryId)`
`void deleteById(Long id)`	根据 id 删除资源	
`void deleteByIds(List<Long> ids)`	根据 id 批量删除资源	
`void deleteBy<条件>(<查询条件参数>)`	根据条件删除资源	`void deleteByCategoryId(Long categoryId)`

1.9 MyBatis mapper 层标准方法定义

mapper文件中的方法，请使用一下统一的命名形式：

方法	说明	示例
`selectById`	根据 id 查找数据	
`selectByIds`	根据 id 批量查找数据	
`selectAll`	无条件查找全部数据	
`selectAllBy`	后台管理列表查找数据	
`selectBy<条件>`	根据少量条件查找数据（3 或以内）	`selectByName`
`select<标签>By<条件>`	根据少量条件查找特定业务场景资源	`findLatestByType`
`existsBy<条件>`	根据少量条件判断是否存在	`existsById`
`insert`	插入单个数据	
`insertAll`	批量插入数据	
`update`	更新单个数据	
`updateBy<条件>`	根据条件更新数据	`updateByEnabled`
`deleteById`	根据 id 删除数据	
`deleteByIds`	根据 id 批量删除数据	
`countBy<条件>`	根据条件计数	`countByCategoryId`

1.10 标准字段命名

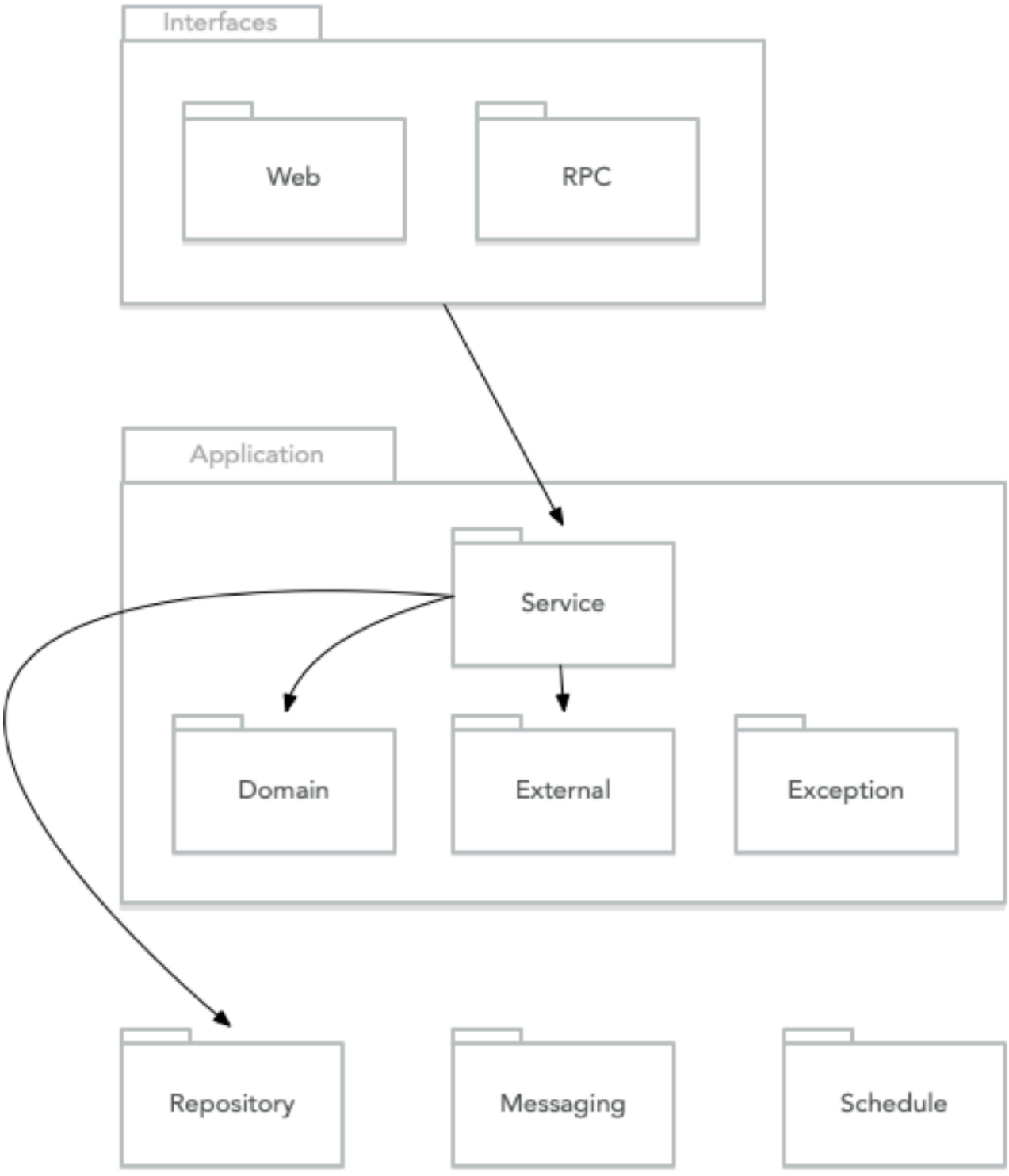
定义了常用的一些数据模型属性名称，开发时应优先考虑这些属性命名，以提高命名风格的一致性。可用于 domain 模型和数据库字段。

字段	说明
name	通用名称，特定场景应使用特定字段，比如以下的username，display_name等
username	用户名，通常用作登录
display_name	显示名称，可表示昵称
title	标题，比较正式的名称
description	文本描述
deleted	删除标志
parent	对于有层级关系的资源表示上级
children	对于有层级关系的资源表示下级集合

create_time	创建时间
update_time	更新时间
expire_time	到期时间
start_time	某时间段的开始时间
end_time	某时间段的结束时间
creator	创建者
editor	编辑者
enabled	启用/禁用状态
display_order	显示序号，用于前台显示顺序控制，数字从大到小排序，默认0
status	状态
count	条目的计数
link	链接或跳转
note	备注或类似概念
review/comment	评论
text	通用文本内容
image	通用图片
url	表示网址的字段可使用url后缀
code	业务含义的唯一编码
from/until	从...到...，表示时间范围
category	通用分类
read/unread	已读/未读
province	行政省
city	行政市
district	行政区
region	通用地区
phone_number	手机号码
postal_address	通讯地址

2 工程规范

2.1 应用分层



（1）接口层

直接面向客户端或外部系统的调用，包盒类使用以下命名规则：

包名	类名	示例
interfaces		
interfaces.web.v1	<业务>Controller	UserController
interfaces.web.v1.request	<POST请求方法>DTO	PlaceOrderDTO
interfaces.web.v1.response	<数据>DTO	UserDTO
interfaces.rpc.v1	Grpc<业务>Service	GrpcUserService

（2）应用层

定义系统操作，处理核心业务逻辑，包和类使用以下命名规则：

包名	类名	示例
application		
application.service	<业务>Service	UserService
application.service.impl	<业务>ServiceImpl	UserServiceImpl
domain	<领域模型名>	Student
external	<外部系统>Facade	EASFacade
exception	<异常名>Exception	EntityNotFoundException

（3）其它层

涵盖了基础设施、底层技术服务和工具类，包括数据访问层、MQ 消息层、定时调度、缓存、存储等。

包名	类名	示例
repository	<数据模型>Repository	UserRepository
repository.impl	<数据模型>RepositoryImpl	UserRepositoryImpl
messaging	<业务>Consumer | <业务>Producer	EASConsumer
schedule	<任务名>Job	PushNotificationJob
cache	CacheManager	
storage	StorageManager	
util	<工具名>Utils	DateTimeUtils

若有以上表格之外的层，则其中的类命名优先考虑使用 `Manager` 作为后缀。

2.2 GAV

（1）GroupID

格式：cn.xhd.<业务线>.<子业务线>，全部小写字符，最多4级，示例：com.google.errorprone

（2）ArtifactID

格式：<产品名>-<模块名>，全部小写字符，grpc-java

- 接口层命名可使用 api 或 admin 为后缀，示例：newchannel-api，newchannel-admin
- 服务层命名统一使用 service 为后缀，示例：user-service
- 非业务通用服务使用 server 为后缀，示例：discovery-server，gateway-server

（3）Version

格式：<主版本号>.<次版本号>.<修订号>

- 主版本号：产品方向改变，或大规模API不兼容，或架构不兼容
- 次版本号：保持向下兼容的前提下，新增主要功能特性
- 修订号：保持向下兼容的前提下，进行问题修正，或新增次要功能特性

未生产发布前使用格式说明：

a. 测试环境下可使用 RC 版本号，示例：1.0.0-rc.1

b. 开发版本统一使用 SNAPSHOT 后缀，示例：1.0.0-SNAPSHOT

（4）参考地址

https://semver.org/lang/zh-CN/

2.3 源代码管理

（1）版本管理

代码仓库统一使用 Git。

本地全局性设置中文名和公司邮箱：

```
git config --global user.name <姓名全称>
```

```
git config --global user.email <公司邮箱>
```

（2）提交文件

提交文件范围只应包括：

- 程序代码
- 配置文件
- 构建脚本
- README.md
- CHANGELOG.md

禁止提交以下文件：

- IDE 工程文件

- Windows 或 Mac 系统文件
- Maven 或 Gradle 构建结果
- 应用日志
- 各种临时文件

为了屏蔽禁止文件，所有工程单独使用 `.gitignore` 文件。

（3）代码检查

为了确保提交到 Git 仓库的代码符合编码规范，本地开发环境可以使用以下工具进行辅助检查：

- IDE 中执行"代码格式化"和"整理 Imports"
- 使用 Gradle 执行代码检查：`gradle check`

3 数据库规范

3.1 字符集

（1）数据库服务器字符集统一使用 utf8mb4

（2）客户端连接数据库服务器使用 utf8

3.2 命名

（1）库名、表名、字段名

全部使用小写字符，单词之间用下划线隔开：lower_case

（2）表名和字段名避免

表名和字段名避免使用统一前缀

示例：避免 tbl，*t*，yy_

（3）表名单数形式

表名使用名词的单数形式

示例：使用 student，避免students

（4）索引名

所有索引名以固定前缀加字段名命名：idx_

示例：idx_type

（5）关联主键

关联主键字段命名：<表名>_id，示例：user_id

（6）避免缩写

避免使用非通用缩写

示例：避免使用 uid，pid，oid等

（7）布尔字段

布尔字段命名使用形容词，禁止使用 is_ 前缀

3.3 最佳实践

（1）数据库只负责存储索引，程序中 SQL 复杂度尽量降到最低

（2）禁止使用存储过程和视图

（3）禁止使用外键，程序控制约束

（4）表和字段必须加注释，枚举类型必须把所有值列出来

（5）选择合适的字段类型，节省存储空间

（6）所有字段必须设置默认值，避免 NULL

（7）布尔字段使用类型 tinyint(1)，值为 0 或 1

（8）枚举类型使用无符号 tinyint(1)，避免使用 varchar

（9）程序中 SQL 语句所有关键字必须使用大写字符

（10）所有表的 create_time 和 update_time 字段必须加索引

（11）所有表的关联 id 字段必须加索引

3.4 通用业务字段

（1）逻辑删除字段

所有表使用逻辑删除字段，命名 deleted，布尔类型

（2）创建时间字段

所有表统一加创建时间字段，命名 create_time，类型 timestamp，默认值 CURRENT_TIMESTAMP

（3）最后修改时间字段

所有表统一加最后修改时间字段，命名 update_time，类型 timestamp，默认值 CURRENT_TIMESTAMP，并且设置 ON UPDATE CURRENT_TIMESTAMP

（4）创建者字段

所有后台管理的表统一加创建者字段，命名 creator，关联后台操作人 id

（5）修改者字段

所有后台管理的表统一加修改者字段，命名 editor，关联后台操作人 id

（6）排序字段

业务上需要后台设置排序的表使用统一排序字段，命名 display_order，类型无符号int，默认值0，数值大优先级高

（7）日期时间

业务相关的日期时间字段使用datetime，命名优先参考以下形式：

- arrival_date (time)

- start_date (time)
- end_date (time)
- 有效期：valid_from 和 valid_until

4 RESTful规范

4.1 命名

（1）URI

全部采用小写字符，单词之间使用下划线隔开，不能以 `/` 结尾：snake_case

示例：`/v1/animal_types`

（2）属性名

全部采用小写字符，单词之间使用下划线隔开：snake_case

示例：animal_type_id

（3）属性值

所有属性值必须具有人类可读性

日期使用 ISO 8601 标准格式，不包含时区，默认为 UTC+8，格式：`yyyy-MM-dd'T'hh:mm:ss`

枚举值使用英文字符不要使用数字

（4）资源

每个地址表示一种资源，资源也可以是一种服务，URI 可体现资源的层级关系，资源命名使用名词复数形式

示例：`/v1/zoos`，`/v1/zoos/1/employees`，`/v1/transactions`

4.2 认证

服务端接口基于 OAuth2 做认证保护，使用 JWT 作为 token 格式，所有资源端接口都被保护起来。

客户端访问资源端接口，使用 Authorization 请求头，并且使用 Bearer 方式传递访问 token：Authorization: Bearer 。

示例：

GET /resources HTTP/1.1

Host: server.example.com

Authorization: Bearer mF_9.B5f-4.1JqM

参考地址：

https://oauth.net/2/

https://jwt.io

4.3 请求

（1）HTTP方法

- GET：获取资源，不改变资源状态
- POST：修改资源，改变资源状态

（2）请求体

所有请求体数据使用 JSON 格式，请求 Content-Type：application/json。

4.4 响应

（1）响应体

所有响应体数据使用 JSON 格式，响应 Content-Type：application/json。

（2）统一响应格式：

```
{
    "code": 200
    "data": {} / []
}
```

（3）状态码

请求成功 HTTP 状态码：

200 OK： 请求成功

（4）分页响应格式

分页请求结果使用统一的响应数据格式：

```
{
    "code": 200
    "pagination": { // 只有电梯式分页时返回
        "current": 1,
        "per_page": 20,
        "total": 100,
        "pages": 2000
    },
    "data": [ … ]
}
```

4.5 过滤

（1）查询条件

一般的过滤条件使用查询字符串形式的参数

示例：`/v1/users?type=1&age=16`

（2）分页

- 电梯式分页参数统一使用 page 和 per_page 的查询字符串形式的参数，page 表示第几页（从1开始），per_page 表示每页几条数据

示例：`/v1/users?page=1&per_page=20`

- 游标式分页参数统一使用 cursor 和 count 的查询字符串形式的参数，cursor 表示游标位置（exclusive 模式），count 表示抓取的条目数量

示例：`/v1/articles?cursor=2015-01-01T15:20:30&count=10`

（3）快捷方式

经常使用的特定的复杂条件查询可以使用标签化的快捷方式 URI

示例：

`/trades?state=closed&sort=created,desc`

可替换为

`/trades/recently_closed`

4.6 错误处理

（1）HTTP状态码

请求失败状态码：

- 400 Bad Request：请求参数错误
- 401 Unauthorized：未进行身份认证
- 403 Forbidden：无权限访问
- 404 Not Found：资源未找到，主要指URI中的对应的资源找不到
- 415 Unsupported Media Type：请求格式不支持
- 429 Too Many Requests：请求太频繁，用于限流
- 500 Internal Server Error：服务器内部各种异常和错误
- 40XX：自定义状态码

（2）统一响应格式

使用以下 JSON 格式：

```
{
    "code": 400,
    "message": "Invalid value ‘-1’",
    "errors": []
}
```

4.7 版本

随着系统演进，API 可能需要同时支持多版本，版本号定义以如下形式体现在 URI 中：v

示例：`GET /v1/users/1`

4.8 标准请求定义

标准请求定义了常用的一些请求形式，包括基本增删改查和一般形式，开发时应优先考虑这些定义形式，以提高 REST 接口风格的一致性。

方法	URI	请求体	说明	示例
GET	//	无	获取资源列表	GET /v1/articles
GET	///:id	无	获取单个资源	GET /v1/reviews/1234
POST	//	JSON	创建资源	POST /v1/articles
POST	///:id	JSON	更新资源	POST /v1/articles/1234
POST	///:id/delete	无	删除资源	POST /v1/articles/1234/delete
GET	//?query	无	过滤参数以query形式提供	GET /v1/users?gender=1
GET	///:tag	无	过滤参数以标签形式提供	GET /v1/trades/recently_closed
POST	///:action	JSON	通过业务方法改变资源状态	POST /v1/ads/1234/enable

4.9 RESTful参考案例

参考案例：

Github API v3：https://developer.github.com/v3/

Enchant REST API：http://dev.enchant.com/api/v1

5 gRPC规范

5.1 proto文件

一个 proto 文件定义一个 `service`，文件命名风格：lower_snake_case.proto。

单个文件内不同组成部分按以下顺序排列：

- `syntax`、`package`、`import`、`option` 语句
- `service` 定义
- 请求和响应消息定义
- 资源消息定义
- 枚举定义

5.2 包

包命名使用全小写字符，使用 `.` 分隔。

包名格式：newchannel.<服务名>，示例：`newchannel.enterprise.service` Java 包名格式：cn.xhd.newchannel.<服务名>，示例：`cn.xhd.newchannel.enterprise.service`

5.3 服务

proto 文件中定义的 `service`。

命名风格：**UpperCamelCase**

命名格式：<资源>Service，示例：`service UserService { }`

5.4 方法

服务中定义的 rpc 方法。

命名风格：**UpperCamelCase**

命名格式：<动词><资源>，示例：`ListBooks`，`CreateBook`

5.5 消息

rpc 方法传递的消息，包括请求消息和响应消息。

命名风格：**UpperCamelCase**

请求命名格式：<方法>Request

响应命名格式：<方法>Response

以下响应消息例外：

- 删除或更新方法返回空消息，可使用 `google.protobuf.Empty`
- 返回单个资源类型，例如：Get 方法可返回一个 User 资源类型

5.6 枚举

枚举类命名风格：**UpperCamelCase**

枚举值命名风格：**CAPITALIZED_NAMES_WITH_UNDERSCORES**

5.7 字段

命名风格：**lower_case_underscore_separated_names**

5.8 标准方法定义

标准方法定义了常用的一些方法形式，包括常用的增删改查和一般方法，开发时应优先考虑这些定义形式，以提高方法风格的一致性。

动词	名词	方法名称	请求消息	响应消息
List	Book	`ListBooks`	`ListBooksRequest`	`ListBooksResponse`
Search	Book	`SearchBooks`	`SearchBooksRequest`	`SearchBooksResponse`
Get	Book	`GetBook`	`GetBookRequest`	`Book`
Create	Book	`CreateBook`	`Book`	`google.protobuf.Empty`
BatchCreate	Book	`BatchCreateBook`	`BatchCreateBookRequest`	`google.protobuf.Empty`
Update	Book	`UpdateBook`	`Book`	`google.protobuf.Empty`
Delete	Book	`DeleteBook`	`DeleteBookRequest`	`google.protobuf.Empty`
Batch	Book	`BatchDeleteBook`	`BatchDeleteBookRequest`	`google.protobuf.Empty`
Rename	Book	`RenameBook`	`RenameBookRequest`	`RenameBookResponse`