

Java 开发规范

编 写	Tom			编写时间	2015 年 7 月 20 日
适用对象	适用于所有参与 java 开发工作的人员。				
修订历史					
版本	章节	类型	日期	作者	说明
V1.0	ALL	C	2015-07-20	Tom	创建第一版本。
V1.1	6.4	A	2015-12-10	Tom	新增 url 命名空间说明。

说明：创建 (C)、修改 (U)、删除 (D)、增加 (A)

目录

一、编写目的.....	1
二、术语解释.....	1
三、总体要求.....	1
四、开发前准备.....	1
五、目录和文件组织结构.....	2
1、根目录.....	2
2、源文件夹.....	3
3、配置文件.....	5
4、测试用例.....	6
5、静态 web 页面文件结构.....	7
6、jar 包库.....	8
六、命名规范.....	9
1、包 (Package) 命名原则.....	9
2、类 (Class) 命名原则.....	10
3、接口 (Interfaces) 命名原则.....	10
4、方法 (Methods) 命名原则.....	10
5、变量 (Variables) 命名原则.....	12
6、常量 (Constants) 命名原则.....	12
7、HTML 页面内容命名原则.....	12
七、编码规范.....	12
1、Java 源文件 (Java Source Files).....	12
2、开头注释 (Beginning Comment).....	13
3、包和引入语句 (Package and Import Statements).....	13
4、类和接口声明 (Class and Interface Declarations).....	13
5、缩进排版 (Indentation).....	14
6、注释 (Comments).....	14
7、声明 (Declarations).....	17
八、代码示例.....	18
1、实体类 (Entity)	18
2、数据访问层 (Dao)	19
3、业务逻辑接口层 (Service Interface)	20
4、业务逻辑实现层 (Service Implementation)	21
4、测试用例 (Test).....	23
5、请求响应、输出层 (Action).....	23
九、附：统一状态代码常量表.....	25

一、编写目的

开发规范对于程序员而言尤为重要，有以下几个原因：

1. 一个软件的生命周期中，80%的花费在于维护。几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护，制定开发规范，有利于减少维护成本。
2. 开发规范可以改善软件的可读性，可以让程序员尽快而彻底地理解代码。
3. 编写开发规范，为以后对项目扩展提供很好的参考依据。

二、术语解释

开发服务器：开发人员用来调试程序的服务器。

测试服务器：测试人员跟业务人员用来测试开发人员的实现是否符合业务需求的服务器。

引导服务器：业务部署到生产服务器前用来测试业务的服务器，跟生产服务器数据一致。

生产服务器：正式面向实际用户提供服务的服务器，通常以集群的方式呈现。

三、总体要求















1. 开发人员只在本机、开发服务器、测试服务器调试开发，不得在引导服务器、生产服务器进行开发和调试工作。
2. 开发人员工作站使用统一开发工具 STS、Tomcat7、JDK1.7、SecureCRT5.13，采用 UTF-8 编码。
3. 服务器统一使用 CentOS 6 操作系统，统一安装 Tomcat7、JDK1.7、redis2.8。
4. java 包名一律以“com.dsmo2o”开头。
5. 每个类、页面、JS 脚本、方法都必须有详细注释。要求写出模块的名称、功能、输入、输出参数介绍、创建、修改时间。

四、开发前准备

- 1、操作系统：window7 x64 及以上版本
- 2、Maven 客户端下载地址：<http://maven.apache.org/download.cgi>
- 3、Redis 下载地址：<https://github.com/MSOpenTech/redis/archive/2.8.zip>
- 4、开发工具(STS)下载地址：<http://spring.io/tools/sts/>
- 5、MySQL 下载地址：<http://dev.mysql.com/downloads/mysql/>

五、目录和文件组织结构

1、根目录

- ▼  dsmo2o-demo [dsmo2o-demo]
 - >  src/main/java
 - >  src/main/resources
 -  src/test/java
 -  src/test/resources
 - >  JRE System Library [JavaSE-1.6]
 - >  Maven Dependencies
 - >  documents
 - ▼  src
 - ▼  main
 - >  webapp
 -  test
 - >  target
 -  pom.xml 29 15-12-16 下午3:06 yongde

























src/main/java	放置 java 源代码。
src/main/resources	放置项目所需的所有配置文件。
src/test/java	放置测试用例。
src/test/resources	放置测试环境下所有的配置文件
src/main/webapp	放置 html 页面、js、css 等静态资源。
pom.xml	配置所有依赖的 jar 包，及开发和部署环境。

2、源文件夹

com.dsmo2o	
com.dsmo2o.common	公共包，放置系统级的公共资源，在此包下增加类时，最好先检查一下是否存在类似的功能，以免重复添加。
com.dsmo2o.common.config	放置公共配置读取类，类名以 Config 结尾。
com.dsmo2o.common.constants	放置公共常量，类名以 Constants 结尾。
com.dsmo2o.common.interceptors	放置系统级的拦截器，类名以 Interceptor 结尾。
com.dsmo2o.common.listeners	放置系统级的监听器，类名以 Listener 结尾。
com.dsmo2o.common.servlets	放置系统级的 Servlet，类名以 Servlet 结尾。
com.dsmo2o.common.utils	放置公共操作类，类名以 Util 结尾。
com.dsmo2o.core	核心包，该包下的类原则上都是用来被继承的。此包下的所有类开发者一般不要去修改。
com.dsmo2o.core.dao	放置用于被继承的基础 Dao。分库分表策略也放置在此包中。
com.dsmo2o.core.model	放置用于被继承的基础 Model。
com.dsmo2o.core.mvc	放置 MVC 模块中基础的 Action 或 Vo。
com.dsmo2o.core.mvc.action	放置基础的用于被继承。
com.dsmo2o.core.service	放置用于被继承的基础逻辑处理 Service。
com.dsmo2o.demo	子系统名称，此包下实现系统的具体功能。
com.dsmo2o.demo.dao	数据访问层，返回值为 Page、int、实体类、或者 List<Map>这四种格式，类名以 Dao 结尾。
com.dsmo2o.demo.model	数据载体，即放置实体类，类名一般与数据库表名保

	保持一致，采用驼峰命名法。
com.dsmo2o.demo.mvc	MVC 模式中的请求响应模块，即
com.dsmo2o.demo.mvc.action	放置 SpringMVC 中的 Controller，主要负责接收请求参数和输出响应结果。
com.dsmo2o.demo.mvc.action.open	放置开放 API，即只接收来自 /open/*.json 的请求，此类请求需要提供有效的令牌才可以访问。
com.dsmo2o.demo.mvc.action.web	放置提供给 web 网页调用的接口，即只接受来自 /web/*.json 的请求，此类请求不加任何非业务性限制。
com.dsmo2o.demo.vo	放置 vo，如果参数个数较多时，可将参数列表封装成 vo 整体传输。
com.dsmo2o.demo.service	业务逻辑层，处理所有的业务逻辑，返回值为 ResultMsg，内部可按功能再划分子模块，类名以 Service 结尾。
com.dsmo2o.demo.service.*	业务逻辑层其他子模块。
com.dsmo2o.demo.plugin.*	其他基础服务模块，在此包下任意扩展。

3、配置文件

- ▼  src/main/resources
 - ▼  i18n
 -  en.properties 2 16-4-22 下午3:15 yongde
 -  zh_CN.properties 2 16-4-22 下午3:15 yongde
 -  zh_TW.properties 2 16-4-22 下午3:15 yongde
 - ▼  local
 -  db.properties 2 16-4-22 下午3:15 yongde
 -  log4j-cfg.xml 13 16-4-25 下午5:33 yongde
 - ▼  product
 -  db.properties 2 16-4-22 下午3:15 yongde
 -  log4j-cfg.xml 13 16-4-25 下午5:33 yongde
 - ▼  test
 -  db.properties 24 16-4-26 上午11:02 yongde
 -  log4j-cfg.xml 12 16-4-25 下午5:32 yongde
 -  application-aop.xml 22 16-4-26 上午10:52 yongde
 -  application-beans.xml 261 16-5-5 下午3:19 yongde
 -  application-common.xml 266 16-5-5 下午3:37 yongde
 -  application-context.xml 2 16-4-22 下午3:15 yongde
 -  application-db.xml 22 16-4-26 上午10:52 yongde
 -  application-web.xml 264 16-5-5 下午3:34 yongde
 -  auth-rule.properties 317 16-5-6 下午4:32 yongde
 -  mail.properties 263 16-5-5 下午3:34 yongde
 -  system.properties 381 16-5-10 下午6:59 yongde
 -  webdefault.xml 2 16-4-22 下午3:15 yongde

local/*	放置本地运行环境下的配置，如果所有环境配置一致，则此目录下不放文件。
product/*	放置线上运行环境下的配置，如果所有环境配置一致，则此目录下不放文件。
test/*	放置测试运行环境下的配置，如果所有环境配置一致，则此目录下不放文件。
application-aop.xml	配置所有的切面，如自动事务管理、代理监听等。
application-beans.xml	配置所有的单例对象或功能 bean。
application-common.xml	公共配置，如家在配置文件等。
application-context.xml	Spring 启动入口，确定配置文件加载顺序。
application-db.xml	配置所有和数据库操作相关的操作，如数据源配置。
application-web.xml	配置所有和 web 有关的操作，如配置拦截器，配置文件的上传与下载等。
log4j-cfg.xml	Log4j 相关的配置，配置日志输出优先级，日志输入格式，日志文件存放位置。
db.properties	配置数据库连接参数，如连接字符串、驱动包名、用户名密码、连接池参数等。
auth-rule.properties	权限过滤参数配置。
system.properties	配置系统全局参数。
mail.properties	发送邮件的配置参数。

4、测试用例

```

src/test/java
├── com.dsomo2o.demo.test
│   ├── dao
│   └── service

```

com.dsomo2o	测试用例下所有的类均以 Test 结尾，方法名均以 test 开头。
com.dsomo2o.demo.test	admin 模块下的测试用例，一个子系统均创建一个子包。
com.dsomo2o.demo.test.dao	在 src 中，一个 Dao 对应这里一个 Dao，在最后加 Test，一个方法对应这里一个方法，在方法前加 test。
com.dsomo2o.demo.test.service	在 src 中，一个 Service 对应这里一个 Service，在最后加 Test，方法同样一一对应，在方法前加 test。

5、静态 web 页面文件结构

- ▼ webapp
 - > company
 - > css
 - > images
 - > js
 - > product
 - > settings
 - ▼ static
 - > css
 - > fonts
 - > images
 - ▼ js
 - > lib
 - browser.support.js 303 16-5-6 下午12:02 yongde
 - common.tools.js 56 16-4-27 下午3:38 yongde
 - domain.js 10 16-4-25 下午5:09 yongde
 - handlebars.helper.js 77 16-4-28 下午12:02 guorun
 - jquery.pager.js 77 16-4-28 下午12:02 guorun
 - util.js 196 16-5-4 下午12:13 yongde
 - > WEB-INF

webapp	静态文件根目录。
webapp/static	公共的静态资源依赖包。
webapp/static/css	公共的 css。
webapp/static/css/fonts	公共的字体。
webapp/static/images	公共的图片文件。
webapp/static/js	公共的静态 js 文件。
webapp/static/js/lib	公共的 Js 类库。
WebRoot/css	主站使用的 css 文件。
WebRoot/images	主站使用的图片文件。
WebRoot/js	主站使用的 js 文件。

6、jar 包库

目录位置：WebRoot/WEB-INF/lib

Jar 包引用原则：需要才加，在项目中没有用到的不加。

aopalliance-1.0.jar	Spring AOP 依赖包
aspectjweaver-1.6.2.jar	处理事务和 AOP 所需包
cglib-nodep-3.1.jar	Spring 中自动代理所需的依赖包
commons-collections-3.2.1.jar	对标准的 java Collection 的扩展
commons-fileupload-1.3.jar	HTTP 文件上传处理
commons-io-2.2.jar	针对 java.io.InputStream 和 Reader 进行了扩展
commons-lang-2.4.jar	对 java.lang.* 的扩展
commons-logging-1.2.jar	日志处理扩展包
commons-beanutils-1.8.3.jar	Apache 提供的对象操作工具包

core-common-1.0.jar	自有框架的公共依赖包
core-common-encrypt-1.0.jar	自有框架加密处理包
core-common-jdbc-1.0.jar	自有框架 jdbc 扩展包
core-common-redis-1.0.jar	自有框架 redis 操作类包
core-common-utils-1.0.jar	自有框架常用操作工具类
druid-1.0.9.jar	阿里巴巴开源的程序性能检测包，可检测 sql 执行效率等
fastjson-1.1.32.jar	阿里巴巴开源的 JSON 处理包
junit-4.8.1.jar	用于单元测试
log4j-1.2.14.jar	提供日志输出功能
mysql-connector-java-5.1.14.jar	MySQL 数据库连接驱动包
ojdbc14.jar	Oracle 数据库连接驱动包
oraclejdbc.jar	Oracle 数据 sql 操作依赖包
persistence-api-1.0.jar	Java 持久化注解依赖包
poi-3.6.jar	操作 Excel、Word、PPT 所需的依赖包
rt.jar	Java 基础包，有些 IDE 没有自动引入。
spring-aop-3.0.5.RELEASE.jar	Spring AOP 核心 API
spring-asm-3.0.5.RELEASE.jar	Spring 做持久化操作通讯时所依赖的包
spring-beans-3.0.5.RELEASE.jar	Spring 对象实例化扩展包
spring-context-3.0.5.RELEASE.jar	Spring IOC 容器支持包
spring-context-support-3.0.5.RELEASE.jar	Spring IOC 容器扩展包
spring-core-3.0.5.RELEASE.jar	Spring 核心包，使用 Spring 必须引入
spring-expression-3.0.5.RELEASE.jar	Spring AOP 表达式解析包
spring-jdbc-3.0.5.RELEASE.jar	Spring JDBC 依赖包
spring-test-3.0.5.RELEASE.jar	Spring 测试用例依赖包
spring-tx-3.0.5.RELEASE.jar	Spring 事务处理支持包
spring-web-3.0.5.RELEASE.jar	Spring 对 J2EE 的扩展包
spring-webmvc-3.0.5.RELEASE.jar	Spring MVC 支持包

六、命名规范

1、包 (Package) 命名原则

包名的前缀以项目一级域名逆序排列，包名应该以 com.dsno2o 开头。包名的后续部分接下来以项目简拼、项目简拼、分层模块、功能模块名称命名。所有字母必须小写命名。

1、Action：都必须放在 com.dsno2o.子项目名.mvc.action 目录下。

例：「电能服务网」子系统的包名为例 “com.dsno2o.demo.mvc.action”；

2、Service：com.dsno2o.demo.service.模块名称

3、Dao：com.dsno2o.demo.dao

4、model: com.dsmo2o.model

5、vo: com.dsmo2o.demo.vo

2、类 (Class) 命名原则

类名是个一名词，采用大驼峰式命名法。每个单词的首字母大写。尽量使你的类名简洁而富于描述。使用完整单词，避免缩写词(除非该缩写词被更广泛使用)。

1、Action: 功能名称+Action

2、Service: 实体类名+Service

3、Dao: 实体类名+Dao

3、接口 (Interfaces) 命名原则

接口命名均以 I 字母开头，其后跟类命名原则相同，采用大驼峰式命名法，即首字母大写，其后单词的首字母大写。

4、方法 (Methods) 命名原则

方法命名，采用小驼峰式命名法(首字母小写，其后单词的首字母大写)。另对通用性方法做以下规定：

Dao 方法命名：

查询：select+功能描述。

删除：delete+功能描述。

更新：update+功能描述。

新增：insert+功能描述。

批量操作：在其后加上 ForBatch，例如批量插入用户 insertUserForBatch。

Service 方法命名：

查询：get。

分页查询：getPage

不分页查询：getList

无条件查询：getListAll

根据特定条件查询： 查询命名+By+条件描述。例如根据 Id 获取结果 getById。

(伪) 删除：remove

批量(伪) 删除：removeForBatch

恢复(伪) 删除：recovery

无条件全部(伪) 删除：removeAll

根据特定条件（伪）删除：（伪）删除命名+By+条件描述。例如根据 id（伪）删除对象 removeById

彻底（真）删除：realDel

批量彻底（真）删除：realDelForBatch

无条件全部彻底（真）删除：realDelAll

根据特定条件彻底（真）删除：（真）删除命名+By+条件描述。例如根据 id（真）删除对象 realDelById

新增：add

批量新增：addForBatch

修改：modify

批量修改：modifyForBatch

根据特定条件修改：修改命名+By+条件描述。根据 id 修改对象 modifyById

登录：login

Action 方法命名：

原则上，CRUD 操作和 Service 命名规则保持一致，以下对特定方法命名进行规定：

登录：login，对应 Url：login.json

登出：logout，对应 Url：logout.json

注册：signin，对应 Url：signin.json

检查唯一性：check+功能名，例如检查用户是否存在 checkUserExist，对应 Url：checkUserExist.json

导入：import+功能名，例如导入用户 importUser，对应 Url：importUser.json

导出：export+功能名，例如导出用户 exportUser，对应 Url：exportUser.file

上传：upload+功能名，例如上传视频 uploadVideo，对应 Url：uploadVide.json

下载：download+功能名，例如下载图片 downloadImg，对应 Url：downloadImg.file

页面跳转：to+页面名称，例如进入到欢迎页 toWelcome，对应 Url：toWelcome.page

输出 JSON 数据的方法：get+功能名，例如获得用户列表 getUserList，对应 Url：getUserList.json

Action URL 命名

①一般来说，action 请求 url 中要带有参数，那么如果 url 中带有参数，我们做以下规定：

如：删除一个用户，用户 ID 为必选参数，规定 url 为：/web/member/remove/用户 id.json

即将必选参数直接写入到 url 路径中。

如：分页查询，url 可规定为：/web/member/getForPage.json?pageNo=1&pageSize=20

因为页码和每页条数一般都有默认值。

②action Url 命名空间分为 system、open、admin、web、mobile 五个级别：

system:该命名空间下只提供系统级接口，例如登录、登出、i18n 多语言切换等操作

open:开放给第三方的 jsonp 接口，需要携带授权码方可访问。

admin:给后台管理服务提供的接口，需要授权的 IP 地址才能访问。

web:给网页提供的的数据展示接口，只要满足业务权限即可访问。

mobile:给手机端提供的的数据接口，同样只需要满足业务权限即可访问。

如：在每个子系统下访问/system/logout.json 即可退出登录。

如：/web/app/getAll.json，一看便知是给网页提供的的数据接口。

5、变量 (Variables) 命名原则

采用小驼峰式命名法，除了变量名外，所有实例，包括类，类常量，均采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写。变量名不应以下划线或美元符号开头，尽管这在语法上是允许的。变量名应简短且富于描述。变量名的选用应该易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。临时变量通常被取名为 i, j, k, m 和 n，它们一般用于整型；c, d, e，它们一般用于字符型。

6、常量 (Constants) 命名原则

常量的声明，应该全部大写，单词间用下划线隔开。

7、HTML 页面内容命名原则

采用全小写命名法，即字母全部小写，每个单词之间用_隔开。例如：

1、列表页：功能名_list.html

2、管理页：功能名_manager.html

3、Form 表单 ID 命名原则：frm-功能名

4、HTML 页面编码使用 UTF-8

5、其他诸如图片、按钮等命名规则请参看《HTML 前端开发规则》。

七、编码规范

1、Java 源文件 (Java Source Files)

每个 Java 源文件都包含一个单一的公共类或接口。若私有类和接口与一个公共类相关联，可以将它们和公共类放入同一个源文件。公共类必须是这个文件中的第一个类或接口。

Java 源文件还遵循以下规则：

1、开头注释（参见“开头注释”）

- 2、包和引入语句（参见“包和引入语句”）
- 3、类和接口声明（参见“类和接口声明”）

2、开头注释(Beginning Comment)

所有的源文件都应该在开头有一个注释，其中列出类名、版本信息、日期和版权声明，其格式如下：

```
/**
 * @(#) $CurrentFile
 * 版权声明 XXXXXX 公司, 版权所有 违者必究
 *
 * <br> Copyright: Copyright (c) 2015
 * <br> Company: 咕泡学院
 * <br> @author 文件创建者姓名（电子邮箱地址）
 * <br> 2015-07-05
 * <br> @version 1.0
 */
```

3、包和引入语句 (Package and Import Statements)

在多数 Java 源文件中，第一个非注释行是包语句。在它之后可以跟引入语句。示例：

```
package java.awt;

import java.awt.peer.CanvasPeer;
```

4、类和接口声明 (Class and Interface Declarations)

序号	类/接口声明的各部分	注解
1	类/接口文档注释 (/**... */)	该注释中所需包含的信息，参见“文档注释”
2	类或接口的声明	
3	类/接口实现的注释 (/*... */) 如果有必要的话	该注释应包含任何有关整个类或接口的信息，而这些信息又不适合作为类/接口文档注释。
4	类的(静态)变量	首先是类的公共变量，随后是保护变量，再后是包一级别的变量(没有访问修饰符，access modifier)，最后是私有变量。
5	实例变量	首先是公共级别的，随后是保护级别的，再后是包一级别的(没有访问修饰符)，最后是私有级别的。
6	构造器	

7	方法	这些方法应该按功能，而非作用域或访问权限，分组。
---	----	--------------------------

5、缩进排版(Indentation)

4 个空格常被作为缩进排版的一个单位。缩进的确切解释并未详细指定(空格 vs. 制表符)。一个制表符等于 8 个空格(而非 4 个)。

5.1 行长度(Line Length)

尽量避免一行的长度超过 80 个字符，因为很多终端和工具不能很好处理之。

注意：用于文档中的例子应该使用更短的行长，长度一般不超过 60 个字符。

5.2 换行(Wrapping Lines)

当一个表达式无法容纳在一行内时，可以依据如下一般规则断开之：

- 1、在一个逗号后面断开
- 2、在一个操作符前面断开
- 3、宁可选择较高级别 (higher-level) 的断开，而非较低级别 (lower-level) 的断开
- 4、新的一行应该与上一行同一级别表达式的开头处对齐
- 5、如果以上规则导致你的代码混乱或者使你的代码都堆挤在右边，那就代之以缩进 8 个空格。

以下是断开方法调用的示例：

```
someMethod(longExpression1, longExpression2, longExpression3,
            longExpression4, longExpression5);
```

以下是断开算术表达式的示例：

```
longName1 = longName2 * (longName3 + longName4 - longName5) + 4 * longName6;
```

6、注释(Comments)

Java 程序有两类注释：实现注释(implementation comments)和文档注释(document comments)。

实现注释：是那些在使用 `/*...*/` 和 `//` 界定的注释。

文档注释(被称为“doc comments”)：是 Java 独有的，并由 `/**...*/` 界定。

文档注释可以通过 javadoc 工具转换成 HTML 文件。

实现注释的格式 (Implementation Comment Format)

程序可以有 4 种实现注释的风格：块 (block)、单行 (single-line)、尾端 (trailing) 和行末 (end-of-line)。

6.1 块注释(Block Comments)

块注释通常用于提供对文件，方法，数据结构和算法的描述。块注释被置于每个文件的开始处以及每个方法之前。它们也可以被用于其他地方，比如方法内部。在功能和方法内部的块注释应该和它们所描述的代码具有一样的缩进格式。块注释之首应该有一个空行，用于把块注释和代码分割开来，**比如：**

```
/*
 *这里是具体注释内容.
 */
```

6.2 单行注释(Single-Line Comments)

短注释可以显示在一行内，并与其后的代码具有一样的缩进层级。如果一个注释不能在一行内写完，就该采用块注释。单行注释之前应该有一个空行。

示例：

```
if (condition) {
    /*这里是单行注释. */
    ...
}
```

6.3 尾端注释(Trailing Comments)

极短的注释可以与它们所要描述的代码位于同一行，但是应该有足够的空白来分开代码和注释。若有多个短注释出现于大段代码中，它们应该具有相同的缩进。

示例：

```
if (a == 2) {
    return true;
} else {
    return isPrime(a);
}
/*尾端注释 1*/
/*尾端注释 2*/
```

6.4 行末注释(End-Of-Line Comments)

注释界定符“//”，可以注释掉整行或者一行中的一部分。它一般不用于连续多行的注释文本；然而，它可以用来注释掉连续多行的代码段。

示例：

```

if (foo > 1) {
    //可以是注释掉的代码段.
    ...
} else {

}

return false;
//这里是行末注释.

```

6.5 文档注释(Documentation Comments)

文档注释描述 Java 的类、接口、构造器,方法,以及字段 (field)。每个文档注释都会被置于注释定界符 `/**...*/` 之中，一个注释对应一个类、接口或成员。该注释应位于声明之前。

示例:

```

/**
 * <pre>类说明</pre>
 * <b>功能描述: </b>
 * 具体功能描述内容,可以为空。
 * 注意事项:
 * 具体注意事项,可以为空。
 */
public class Example { ...

```

注意: 顶层的类和接口是不缩进的，而其成员是缩进的。描述类和接口的文档注释的第一行 (`/**`) 不需缩进；随后的文档注释每行都缩进 1 格 (使星号纵向对齐)。成员，包括构造函数在内，其文档注释的第一行缩进 4 格，随后每行都缩进 5 格。

6.6 维护代码注释

增加维护注释主要是为了开发者在维护代码以后，可以很清楚地了解当时改了哪些地方，为什么这样改。

新增代码注释：

每一条维护记录必须在源代码中加上注释

```

//— 操作类型( Modify 、 Remove 、 Add ) by 操作人 @ 日期(yyyy-MM-dd HH:mm:dd) Begin —
//Remove 和 Modify 的代码直接注释即可
//您操作的内容
//— 操作类型( Modify 、 Remove 、 Add ) by 操作人 @ 日期(yyyy-MM-dd HH:mm:dd) End —

```

例如:

```

//— Remove by Tanyongde @2015-07-06 14:03:00 Begin —

```

```
//ModelMap mmap = new ModelMap();
//— Remove by Tanyongde @2015-07-06 14:03:00 End —

//— Modify by Tanyongde @2015-07-06 14:06:11 Begin —
//System.out.println("");
System.out.println("—————");
//— Modify by Tanyongde @2015-07-06 14:06:11 End —

//— Add by Tanyongde @2015-07-06 14:06:11 Begin —
System.out.println("");
//— Add by Tanyongde @2015-07-06 14:06:11 End —
```

7、声明(Declarations)

7.1 每行声明变量的数量 (Number PeLine)

推荐一行一个声明，因为这样以利于写注释。示例：

```
int level; //变量描述 1
int size;  //变量描述 2
```

7.2 类和接口的声明 (Class and Interface Declarations)

当编写类和接口时，应该遵守以下格式规则：

- 1、在方法名与其参数列表之前的左括号“(”间不要有空格。
- 2、左大括号“{”位于声明语句同行的末尾
- 3、右大括号“}”另起一行，与相应的声明语句对齐，除非是一个空语句，“}”应紧跟在“{”之后

```
class Sample extends Object {
    int ivar1;
    int ivar2;
    Sample(int i, int j) {
        int var1 = i;
        int var2 = j;
    }
    int emptyMethod() {}
    ...
}
```

- 4、方法与方法之间以空行分隔
- 5、if 语句

if 语句总是用“{”和“}”括起来，无论单行还是多行，避免使用如下容易引起错误的格式：

```
if (condition) //注意尽量避免使用此格式!
    statement;
```

应该使用的格式：

```
if (condition){ //尽量使用此格式!
    statement;
}
```

八、代码示例

这里以用户登录相关的业务处理为例：

用户实体类为 `com.dsmo2o.model.Member`；

数据访问Dao 为：`com.dsmo2o.passport.dao.MemberDao`；

业务逻辑处理接口为：`com.dsmo2o.passport.service.IMemberService`；

业务逻辑处理实现为：`com.dsmo2o.passport.service.impl.MemberService`；

测试用例类类为：`com.dsmo2o.passport.service.test.MemberServiceTest`；

1、实体类 (Entity)

```
package com.dsmo2o.model;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
/**
 * 用户实体示例
 * @author Tanyongde
 */
@Entity
@Table(name="t_member")
public class Member implements Serializable {
    private long mid;           //用户 ID
    private String realName;    //用户姓名
    private String nickName;    //昵称
```

```

private String loginName;    //登录名
private String password;    //登录密码
private String mnum;        //编号
private String idNum;       //身份证号
private long createTime;    //创建时间
private int mtype;          //用户类型:老师 1, 学生 2
private String lastLoginIp; //最后登录 IP
private long lastLoginTime; //最后登录时间
private String lastLoginAddr; //最后登录地址
private int loginCount;     //登录次数
private int rid;            //角色 ID
private int fromId;         //所属单位 ID
private int state;          //状态: 未激活-1, 已激活 0, 已禁用 1, 已删除 2
@Id
public long getMid() { return mid; }
public void setMid(long mid) { this.mid = mid; }
//生成 get、set 方法 getter & setter (略)
}

```

2、数据访问层 (Dao)

```

package com.dsmo2o.passport.dao;
import java.util.*;
import javax.annotation.Resource;
import javax.core.common.*;
import javax.sql.DataSource;
import org.springframework.stereotype.Repository;
import com.dsmo2o.model.Member;
/** Dao 操作示例
 * @author Tanyongde
 */
@Repository
public class MemberDao extends BaseDaoSupport<Member, Long>{
    /**
     * 配置动态数据源
     * @param dataSource
     */
    @Resource(name="dynamicDataSource")
    public void setDataSource(DataSource dataSource){
        super.setDataSourceReadOnly(dataSource);
        super.setDataSourceWrite(dataSource);
    }
    /** 设置主键列名 */
}

```

```

@Override
protected String getPKColumn() { return "mid"; }
/**
 * 用户登录时查询
 * @param name
 * @param password
 * @param type
 * @return
 */
public Member selectForLogin(String loginName,String password,int mtype){
    QueryRule queryRule = QueryRule.getInstance();
    queryRule.andEqual("loginName", loginName);
    queryRule.andEqual("password", password);
    queryRule.andEqual("mtype", mtype);
    return super.findUnique(queryRule);
}
}

```

3、业务逻辑接口层 (Service Interface)

```

package com.dsmo2o.passport.service;
import java.io.InputStream;
import javax.core.common.ResultMsg;
/**
 * 用户操作 Service 示例接口
 * @author Tanyongde
 */
public interface IMemberService {
    /**
     * 用户登录
     * @param loginName 登录名
     * @param password 登录密码
     * @return
     */
    public ResultMsg<?> getForLogin(String loginName,String password);
}

```

4、业务逻辑实现层 (Service Implementation)

```

package com.dsmo2o.passport.service.impl;

```



```

import java.io.InputStream;
import java.util.*;
import javax.core.common.*;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.alibaba.fastjson.JSONObject;
import com.dsmo2o.common.constants.SystemConstant;
import com.dsmo2o.passport.dao.MemberDao;
import com.dsmo2o.model.Member;
import com.dsmo2o.passport.service.facade.IMemberService;
import com.dsmo2o.passport.vo.MemberVo;

/**
 * 实现用户相关的业务逻辑示例
 * @author Tanyongde
 */
@Service
public class MemberService implements IMemberService {

    @Autowired private MemberDao memberDao;

    /**
     * 密码加密存储
     */
    private final String PASS_ENC = "&^9&%&$-1348253";

    /**
     * 用户登录
     */
    public ResultMsg<?> getForLogin(String loginName, String password) {
        if ("".equals(loginName) || "".equals(password)) {
            //没有填写用户名和密码
            return new ResultMsg<Object>(SystemConstant.RESULT_PARAM_ERROR,
                "没有填写用户名和密码");
        }

        //根据用户名获取账号类型
        Member member = memberDao.selectByLoginName(loginName);

        password = MD5.calcMD5(password + PASS_ENC);
        if (null == member) {
            //用户不存在

```

```

        return new ResultMsg<Object>(SystemConstant.RESULT_PARAM_ERROR,
            "用户不存在");
    }else if(!password.equals(member.getPassword())){
        return new ResultMsg<Object>(SystemConstant.RESULT_PARAM_ERROR,
            "密码不正确");
    }else{
        if(member.getState() == -1){
            //用户未激活
            return new ResultMsg<Object>(SystemConstant.RESULT_PARAM_ERROR,
                "账号未激活,请查收邮件激活后再登录");
        } else if(member.getState() == 1) {
            //用户被禁用
            return new ResultMsg<Object>(SystemConstant.RESULT_PARAM_ERROR,
                "账号不允许使用");
        } else if(member.getState() == 2) {
            //用户被删除
            return new ResultMsg<Object>(SystemConstant.RESULT_PARAM_ERROR,
                "账号不存在");
        }

        MemberVo vo = new MemberVo();
        DataUtils.copySimpleObject(member, vo);
        return new ResultMsg<MemberVo>(SystemConstant.RESULT_STATUS_SUCCESS,
            "登录成功",vo);
    }
}
}

```

4、测试用例 (Test)

```

package com.dsmo2o.passport.service.test;
import javax.core.common.ResultMsg;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.*;
import com.alibaba.fastjson.JSON;
import com.dsmo2o.passport.service.IMemberService;

@ContextConfiguration(locations = {"classpath*:application-context.xml"})
@RunWith(SpringJUnit4ClassRunner.class)
public class MemberServiceTest {

```

```

@Autowired private IMemberService memberService;
//用户登录
@Test
@Ignore
public void testGetForLogin() {
    String loginName="tyd_java@163.com";
    String password="222222";
    ResultMsg<?> result = memberService.getForLogin(loginName, password);
    System.out.println(JSON.toJSONString(result));
}
}

```

5、请求响应、输出层 (Action)

```

package com.dsmo2o.passport.mvc.action.web;

import java.io.IOException;
import javax.core.common.*;
import javax.servlet.http.*;

import org.springframework.*;
import com.alibaba.fastjson.*;

import com.dsmo2o.common.constants.SystemConstant;
import com.dsmo2o.core.mvc.action.BaseAction;
import com.dsmo2o.passport.service.IMemberService;

/**
 * Web Action 层实现示例
 * 在这一层，只接收来自 Web 请求的参数和输入响应结果
 * 至于逻辑如何处理则直接调用来自 Service 层的逻辑处理方法
 *
 *
 * 只拦截来自 xxx/web/xxx.json 的请求
 * @author tanyongde
 *
 */
@Controller
@RequestMapping("/demo/web/")
public class DemoWebAction extends BaseAction {

    @Autowired private IMemberService memberService;

```

```

/**
 * 用户登录
 * @param request
 * @param response
 * @param uname
 * @param upass
 * @return
 */
@RequestMapping("login.json")
public ModelAndView login(HttpServletRequest request
    ,HttpServletResponse response
    ,@RequestParam(value="loginName",defaultValue="") String loginName
    ,@RequestParam(value="password",defaultValue="") String password
    ){
    String ip = super.getIpAddr(request);
    ResultMsg<?> msg = memberService.getForLogin(loginName, password,ip);
    if(msg.getStatus() == SystemConstant.RESULT_STATUS_SUCCESS){
        request.getSession().setAttribute(SystemConstant.SYSTEM_LOGIN_FLAG,
msg.getData());
    }
    return callBackForJsonp(request, JSON.toJSONString(msg));
}
}

```

更多详细操作请从 SVN 下载 passport 源代码。

九、附：统一状态代码常量表

常量名	数据类型	常量值	说明
ENABLE	int	1	已生效、已激活、已删除等，表示肯定。
DISABLE	int	0	未生效、已激活、已删除等，表示否定。
RESULT_PARAM_ERROR	int	9	请求参数错误。
RESULT_STATUS_SUCCESS	int	1	返回结果正常。
RESULT_STATUS_ERROR	int	0	返回结果有错误，服务器内部异常。
RESULT_STATUS_TIMEOUT	int	3	请求超时。
RESULT_STATUS_UNALLOW	int	4	不允许访问，没有权限。
RESULT_STATUS_UNLOGIN	int	99	未登录。