**数据中心-编码规范**

V1.0.0

常官清 @ Enlink 数据中心

**目录**

[1. 概述 3](#_Toc528663004)

[1.1. 目的 3](#_Toc528663005)

[1.2. 适用范围 3](#_Toc528663006)

[2. 文件 4](#_Toc528663007)

[2.1. 文件注释 4](#_Toc528663008)

[2.2. 文件包含 4](#_Toc528663009)

[2.3. 文件空行 4](#_Toc528663010)

[3. 代码 5](#_Toc528663011)

[3.1. 空格 5](#_Toc528663012)

[3.2. 缩进 5](#_Toc528663013)

[3.3. 换行（拆行） 6](#_Toc528663014)

[3.4. 命名 7](#_Toc528663015)

[3.5. 注释 8](#_Toc528663016)

[3.5.1. 类注释 8](#_Toc528663017)

[3.5.2. 函数/方法注释 9](#_Toc528663018)

[3.5.3. 常量注释 10](#_Toc528663019)

[3.5.4. 细节注释 10](#_Toc528663020)

# 概述

本文档描述数据中心部门的编码规范，所有本部门研发人员均需按照本规范进行编码。

## 目的

本文档主要针对的是Java和JS（Javascript、NodeJS、Typescript等）语言，但是产品或项目中使用的其他语言，也要适当的遵循本文档所约定的内容，达到规范要求的目的。

本文档所制定的编码规范，旨在达到如下目的：

* 风格统一
* 可读性强
* 注释完整
* 易于维护

本文档后续章节将对编码规范作详细的描述。对于文字部分未提及的规范，请参考文档所给出的代码示例。对于本文档未涉及的部分，编码时请参照上述“规范目的”。

## 适用范围

本文档适用于数据中心部门所有研发人员。

# 文件

所有文本文件（包括代码、脚本、README等），均使用无BOM头的UTF-8格式保存。使用Unix/Linux格式的换行符，即只使用“\n”表示换行。

## 文件注释

所有的代码文件（包括脚本）必需包含文件注释，文件注释的位置必需在文件的开头部分。注释的内容包括如下三个部分：

* 文件名及文件说明；
* 文件作者（研发人员）；
* 版权声明。

文件注释的示例如表2-1所示：

表格 2-1 文件注释

|  |
| --- |
| /\* 文件名: User.java  \*  \* 作者: 张立丹 (zhangld@enlink.cn)  \* 描述: 本文实现用户管理及用户限制控制功能.  \* 包括用户的增加/删除/修改/权限检查等功能.  \*  \* Copyright @2018 Enlink, All Rights Reserved.  / \* |

其中，文件注释必需使用 /\* ... \*/ 进行块注释，除非编码用的语言中无该格式的注释，否则不能使用其他格式的注释。

## 文件包含

代码文件中需要包含（或引用）其他文件时，文件包含的位置必需紧跟文件注释之后。除非特殊情况，不允许出现在文件的其他位置。如表2-2所示：

表格 2-2 文件包含

|  |
| --- |
| / \*  \* <文件注释>  / \*  import User;  import Mysql;  import NewBinTree; // 优化后的二叉树实现 |

其中，对于不常用、缩写等库的引用，需要增加必要的注释。

## 文件空行

所有文本文件的结尾，必需保留一行“空行”。

# 代码

本小节描述研发编码时，源代码（Java、JS等）的编码风格。

## 空格

代码中需在关键字、操作符等左右保留必要的空格，若无特殊情况，一般为1个空格，具体规范如下：

* 二元、三元操作符的两侧保留空格；
* 一元运算符与操作对象之间不需要空格；
* 代码块左大括号“{”前保留空格；
* if/else/for/while/function/switch/do/try/catch/finally等关键字后保留空格；

代码中使用空格的示例如表3-1所示：

表格 3-1 代码中使用空格

|  |
| --- |
| // 运算符示例  var a = !arr.length;  a++;  a = b + c;  // 代码块、关键字示例  if (condition) {  // do something …  }  // for 循环示例  for (int i = 0; i < 10; i++) {  // do something …  } |

**注意**，if语句中的条件表达式与左、右两侧的括号“(”和“)”之间不要出现空格。对于其他未列出的语句，例如while (condition) { ... }、do { ... } while (condition) 等语句同样需要遵守本规则。

源代码中，单独的空行、每行代码的结尾等均不能出现多余的空格。

## 缩进

编写代码时，所有的“代码块（被 {} 大括号包围）”需使用缩进，且比父块多一个缩进级别。使用4个空格或一个TAB字符作为一个缩进层级，每个源文件中使用的缩进字符必需统一，即全部使用空格进行缩进或者全部使用TAB进行缩进，禁止混用。

一般而言，为了减少源码文件的字节数，以一个TAB字符作为缩进层级，但此规则并非强制约定 。

代码缩进示例如表3-1所示：

表格 3-1 代码缩进（swich...case语句）

|  |
| --- |
| // Example 1  switch (variable) {  case '1':  // do something ...  break;  case '2':  // do something ...  break;  default:  // do something ...  }  // Example 2  function func1(arg1, arg2, arg3) {  if (condition) {  while (condition) {  // do something ...  }  } else {  for (int cc = 0; cc < 10; cc ++) {  if (cc % 2 == 0) {  // do something ...  } else {  // do something ...  }  }  }  } |

## 换行（拆行）

为了便于阅读，对于过长（一般超过100字符）的代码行需要进行换行。对于较短的代码行，一般不建议换行。代码需要拆行时，在行尾使用行连接符“\”（如果有的话）。

对于换行（拆行）的规范可规则如下：

* 两个独立的语句之间必需换行，禁一行中包含两个语句；
* 运算符处于换行时，运算符必须在新行行首;
* 函数（或对象）声明和调用、for语句等场景中，不允许在“,”或“:”前换行;
* 不同的逻辑语句使用空行隔开;
* 对于if…else…、try…catch…finally等语句，在“}”后不需要换行。

代码示例如表3-3所示：

表格 3-3 换行（拆行）代码示例

|  |
| --- |
| // 代码示例1  int sum = num1 + num2 + num3 + num4; // 不换行  int value = student1.baseInfo.age + student2.baseInfo.age + student3.baseInfo.age  + student4.baseInfo.age; // 换行  // 代码示例2  if (condition1 && condition2) { // 不换行  // Do something ...  }  // 换行  if (( condition1 && condition2)  || (condition3 || condition4)  || (conidtion5 && condition6)  ) {  // Do something ...  }  // 代码示例3  var obj = { a: 1, b: 2, c: 3 }; // 不换行  var obj = { // 换行  elementName1: value % 100,  elementName2: prevVar.elementValue ? "yes" : "no",  ...  };  // 代码示例4 (函数定义与此类似)  function\_call(arg1, arg2, arg3); // 不换行  function\_call( // 换行  aVeryVeryLongArgument,  anotherVeryLongArgument,  anotherVeryLongArgument2,  callback  );  // 代码示例 5  if (condition) {  // do something ...  } else { // 包括前、后大括号在一行内；  // do something ...  }  try {  // do something ...  } catch (ex) { // 包括前、后大括号在一行内；  // do something ...  } |

## 命名

编码时，对于变量、函数、参数等名称命名时，统一使用“砣峰”命名法，除了第一个单词之外，名称中的每个单词需首字母大写。

对于命名的规范如下所示：

* 使用有意义的单词进行命名；
* 尽量避免使用简写，如使用简写时需要对简写增加必要的注释；
* 常量、枚举属性等全部大写，并使用下划线“\_”分割；
* 语言相关的命名规则 ，参考各自的一般约定：  
  例如，Java类首字母大写；Go允许外部访问时首字母大写，不允许则首字母小写。

代码示例如表3-4所示：

表格 3-4 命名代码示例

|  |
| --- |
| // 1) 代码示例  final int PAGE\_SIZE = 20;  // 2) 代码示例  public Class CodeStyle {  private String fontName;  private int fontSize;  public void getFontSize() {  return this.fontSize;  }  }  // 3) 代码示例  public enum ResultCode {  OK,  CREATED,  NO\_CONTENT;  } |

## 注释

注释是对代码的辅助解释，帮助其他人更好的阅读代码，另外注释也起到分隔代码的功能。注释应该使代码更加清晰易懂，应简洁明了，避免添加不必要的注释。

对于一般的getter和setter方法不需要添加注释，类、接口、构造函数、方法、全局变量等必须添加注释，字段属性可选择性添加简单注释。对于可以很明显就能从代码中获取到代码含义的部分，则不需要添加额外的注释。

### 类注释

类的注释采用多行（/\* ... \*/）注释格式，注释中需对类的功能作一些必要的介绍。一般情况下，一个文件中只包含一个类，或者包含多个类时，所有类均由一个作者实现，作者信息一般不出现在类的注释中。代码示例如表3-5所示：

表格 3-5 类注释代码示例

|  |
| --- |
| /\*\*  \* CodeStyle - 代码样式类.  \*  \* CodeStyle类从配置文件中读取代码样式配置, 使用者创建该类时,指定  \* 需要加载的配置文件路径(绝对路径), 然后通过对应的getter方法获取  \* 样式的具体属性值.  \*/  public Class CodeStyle {  private String fontName;  private int fontSize;  /\* \*  \* CodeStyle() - 构造函数  \*  \* 加载指定 @path 文件中的样式配置信息, 如果配置中无对应的配置属性,  \* 或者属性值配置不合法, 则使用默认的属性值.  /\*  public CodeStyle(String path) {  // Do something ...  }  public int getFontSize() {  return this.fontSize;  }  } |

### 函数/方法注释

函数（或方法）的注释必须包含函数说明、参数及说明、返回值或异常处理等。其中参数应包含参数名、类型、说明、注释标识，返回值应包含返回值的类型和描述；方法描述与参数设置之间应有空行。代码示例如表3-6所示：

表格 3-6 函数（方法）注释代码示例

|  |
| --- |
| public Class CodeStyle {  /\*\*  \* updateCodeStyle() - 更新字体大小及样式.  \*  \* @param {int} size 字体大小;  \* @param {String} fontName 字体样式名.  \*  \* @desc 函数将指定的 @size和 @fontName 更新到代码样式对象的属性中。  \* 更新之前检测各属性值的合法性。  \*  \* @return {CodeStyle} 返回代码样式对象本身(this).  \* @throws NullPointerException 空指针异常  \*/  public CodeStyle updateStyle(int size, String fontName) throws NullPointerException {  // Do something ...  return this;  } |

### 常量注释

代码中的所有常量必须添加注释，采用单行（//）格式。示例代码如表3-7所示：

表格 3-7 常量注释代码示例

|  |
| --- |
| // 分页请求时，用于限制每次请求的记录个数。  public final int PAGE\_SIZE = 20;  // 缓冲区最大值.  // 接收用户请求时, 用于创建缓冲区保存用户的请求数据.  public final int MAX\_BUFSIZE = 4 \* 1024; |

其中，对于某些常量而言，需要增加必要的注释来说明变量的作用。

该规则同样适用于变量的注释。

### 细节注释

对于内部实现、不容易理解的逻辑说明、摘要信息等，需要编写细节注释。细节注释应遵循单行注释的格式。

尽量通过规范的命名来避免不必要的注释，示例代码如表3-8所示：

表格 3-8 细节注释代码示例

|  |
| --- |
| // 错误的代码示例  // 二分插入  int a = 0; // 低位  int b = list.size() -1; // 高位  int c = 0; // 当前位置  // 二分插入  int lowPosition = 0;  int highPosition = list.size() – 1;  int middlePosition = 0; |