**编码规范说明文档**

1. 目的

（1）为了统一小组成员在软件开发设计过程的编程规范。

（2）使小组开发人员能很方便的理解每个目录，变量，控件，类，方法的意义。

（3）为了保证编写出的程序都符合相同的规范，保证一致性、统一性而建立的程序编码规范。

（4）编码规范和约定必须能明显改善代码可读性，并有助于代码管理、分类范围适用于所有基于 NodeJS 的软件开发工作。

2. 范围

本规范适用于本小组全体人员，作用于软件项目开发的代码编写阶段和后期维护阶段。

3. 排版格式

程序排版格式虽然不是十分严格的规范要求，但整个项目都服从统一的编程风格，这样使所有人在阅读和理解代码时更加容易。格式虽然不会影响程序的功能，但会影响可读性。程序的排版格式追求清晰、美观，是程序风格的重要构成因素。可以把程序的排版格式比喻为“书法”。好的“书法”可让人对程序一目了然，看得兴致勃勃。差的程序“书法”如螃蟹爬行，让人看得索然无味，更增加了项目维护着的维护成本。

3.1. 函数的声明与定义

（1）函数名和左圆括号间没有空格；圆括号与参数间没有空格。

（2）左大括号总是与参数列表在同一行；右大括号总是单独位于函数最后一行。

（3）函数的内容总与左括号保持一个制表符的缩进。

（4）参数间的逗号总加一个空格。

（5）函数的大小一般不要超过50行，函数越小，代码越容易维护。

（6）函数声明前应加上注释，注明该函数的作用，如果该函数有比较多的参数，还应该加上参数含义和返回值的注释。

3.2. 空行

空行起着分隔程序段落的作用。空行得体（不过多也不过少）将使程序的布局更加清晰。空行不会浪费内存，虽然打印含有空行的程序是会多消耗一些纸张，但是值得。所以不要舍不得用空行。

（1）在每个类声明之后、每个函数定义结束之后都要加空行。

（2）在一个函数体内，逻揖上密切相关的语句之间不加空行，其它地方应加空行分隔。

3.3. 代码行

（1）一行代码只做一件事情，如只定义一个变量，或只写一条语句。这样的代码容易阅读，并且方便于写注释。

（2）if、for、while、do等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加{}。这样可以防止书写失误。

（3）在定义变量的同时必须初始化该变量。如果变量的引用处和其定义处相隔比较远，变量的初始化很容易被忘记。如果引用了未被初始化的变量，可能会导致程序错误。

3.4. 代码行内的空格

（1）‘，’之后要留空格，如 Function(x, y, z)。如果 ; 不是一行的结束符号，其后要留空格，如 for (initialization; condition; update)。

（2）赋值操作符、比较操作符、算术操作符、逻辑操作符、位域操作符，如“=”、“+=” “>=”、“<=”、“+”、“\*”、“%”、“&&”、“||”、“<<”,“^”等二元操作符的前后应当加空格。

（3）一元操作符如“!”、“~”、“++”、“–”、“&”（地址运算符）等前后不加空格。

（4）像“［］”、“.”、“->”这类操作符前后不加空格。

3.5. 单双引号的使用

由于双引号在别的场景下使用较多，在 Node 中使用字符串时尽量使用单引号，这样无需转义，如：

var html = '<a href="http://cnodejs.org">CNode</a>';

而在 JSON 中，严格的规范是要求字符串用双引号，内容中出现双引号时，需要转义。

3.6. 逗号

逗号用于变量声明的分隔或是元素的分隔。如果逗号不在行结尾，前面需要一个空格。此外，逗号不允许出现在行首，比如：

var foo = "hello",

bar = "world";

// 或是

var hello = { foo: "hello", bar: "world" };

// 或是

var world = ["hello", "world"];

3.7. 分号

给表达式结尾添加分号。尽管 JavaScript 编译器会自动给行尾添加分号，但还是会带来一些误解。由于自动添加分号可能带来未预期的结果，所以添加上分号有助于避免误会。

3.8. 对齐

（1）程序的分界符‘{’和函数名在同一行，‘}’应独占一行并且和函数名在同一列，同时与引用它们的语句左对齐。

（2）{ }之内的代码块在‘{’右边一个制表符左对齐。

3.9. 长行拆分

（1）代码行最大长度宜控制在70至80个字符以内。代码行不要过长，否则眼睛看不过来，也不便于打印。

（2）长表达式要在低优先级操作符处拆分成新行，操作符放在新行之首（以便突出操作符）。拆分出的新行要进行适当的缩进，使排版整齐，语句可读。

3.10. 被注释的代码

在代码中经常残留一下被注释的代码，如果这段代码还有价值，必须对该段代码加上被注释的原因，或者不需要有的就直接删除。

3.11. 注释

合理的注释，对保证代码可读性至为重要。当然，注释的确很重要，但最好的代码本身就是文档，类名和变量名意义明确要比通过注释解释模糊的命名要好的多。

（1）注释是对代码的“提示”，而不是文档。程序中的注释不可喧宾夺主，注释太多了会让人眼花缭乱。注释的花样要少。

（2）如果代码本来就是清楚的，则不必加注释。否则多此一举，令人厌烦。单行注释用//，多行注释用/\* \*/。

（3）边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

（4）注释应当准确、易懂，防止注释有二义性。错误的注释不但无益反而有害。

（5）尽量避免在注释中使用缩写，特别是不常用缩写。

（6）注释的位置应与被描述的代码相邻，可以放在代码的上方或右方，不可放在下方。

4. 命名规则

命名规则是软件编程中重要的规则之一，规范命名风格会使代码风格保持一致，更容易被小组中其他成员理解。好的命名可以令代码赏心悦目，带来愉悦的阅读享受，令代码具有良好的可维护性。总体原则是所有命名必须做到见名知意。命令的主要范畴有变量、常量、方法、类、文件、包等。

（1）标识符应当直观且可以拼读，可见名知意，不必进行“解码”。标识符最好采用英文单词或其组合，便于记忆和阅读。切忌使用汉语拼音来命名。程序中的英文单词一般不会太复杂，用词应当准确。例如不要把 currentValue 写成 nowValue。

（2）命名规则要求符合匈牙利命名法。

（3）所有的变量，函数，类的命名，若需要多个单词时，每个单词直接连写，不要用下划线(“\_”)或横线（“-”）分开。如：deviceInfo，remoteCamera。

（4）函数的名字应当使用“动词”或者“动词＋名词”（动宾词组）。

（5）变量的名字应当使用“名词”或者“形容词＋名词”。

（6）对于变量命名，禁止取单个字符（如i、j、k…），建议除了要有具体含义外，还能表明其变量类型、数据类型等，但i、j、k作局部循环变量是允许的。

（7）项目的名字都使用“名词”，首字母以大写开头。

（8）用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。

（9）避免名字中出现数字编号，如 Value1，Value2 等，除非逻辑上的确需要编号。这是为了防止程序员偷懒，不肯为命名动脑筋而导致产生无意义的名字（因为用数字编号最省事）。

（10）程序中不要出现仅靠大小写区分的相似的标识符。

（11）程序中不要出现标识符完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法错误，但会使人误解。

（12）命名风格保持一致，自己特有的命名风格，要自始至终保持一致，不可来回变化。项目组或产品组对命名有统一的规定，个人服从团队。说明：个人的命名风格，在符合所在项目组或产品组的命名规则的前提下，才可使用。（即命名规则中没有规定到的地方才可有个人命名风格）。

（13）命名中若使用特殊约定或缩写，则要有注释说明。应该在源文件的开始之处，对文件中所使用的缩写或约定，特别是特殊的缩写进行必要的注释说明。

5. 编码规则

5.1. 错误检查规则

（1）编程中要考虑函数的各种执行情况，尽可能处理所有流程情况。

（2）检查所有的系统调用的错误信息，除非要忽略错误。

（3）将函数分两类：一类为与屏幕的显示无关， 另一类与屏幕的显示有关。对于与屏幕显示无关的函数，函数通过返回值来报告错误。对于与屏幕显示有关的函数，函数要负责向用户发出警告，并进行错误处理。

（4）错误处理代码一般放在函数末尾。

（5）对于通用的错误处理，可建立通用的错误处理函数，处理常见的通用的错误。

5.2. 大括号规则

将左大括号放置在关键词的同一行右边。如：

for (int i = 0; i < 10; ++i) {

// do something

}

5.3. 缩进规则

采用 2 个空格缩进，而不是 tab 缩进。 空格在编辑器中与字符是等宽的，而 tab 可能因编辑器的设置不同。2 个空格会让代码看起来更紧凑、明快。

5.4. 变量声明规则

永远用 var 声明变量，不加 var 时会将其变成全局变量，这样可能会意外污染上下文，或是被意外污染。 在 ECMAScript 5 的 strict 模式下，未声明的变量将会直接抛出 ReferenceError 异常。需要说明的是，每行声明都应该带上 var，而不是只有一个 var，示例代码如下：

var assert = require("assert");

var fork = require("child\_process").fork;

var net = require("net");

var EventEmitter = require("events").EventEmitter;

5.5. 小括号规则

不要把小括号和关键词（if 、while等）紧贴在一起，要用空格隔开它们。

不要把小括号和函数名紧贴在一起。

除非必要，不要在Return返回语句中使用小括号。因为关键字不是函数，如果小括号紧贴着函数名和关键字，二者很容易被看成是一体的。

5.6. 其他规格

（1）If Else规则

如果有用到 else if 语句的话，通常最好有一个 else 块以用于处理未处理到的其他情况。可以的话放一个记录信息注释在 else 处，即使在 else 没有任何的动作。if 和循环的嵌套最多允许 4 层。

（2）比较规则

总是将恒量放在等号/不等号的左边。一个原因是假如你在等式中漏了一个等号，语法检查器会为你报错。第二个原因是你能立刻找到数值而不是在你的表达式的末端找到它。

（3）对齐规则

变量的申明和初始化都应对齐。

（4）单语句规则

除非这些语句有很密切的联系，否则每行只写一个语句。

（5）单一功能规则

原则上，一个程序单元（函数、例程、方法）只完成一项功能。

（6）简单功能规则

原则上，一个程序单元的代码应该限制在一页内（25~30行）。

（7）明确条件规则

不要采用缺省值测试非零值。例如：使用“if ( 0 != f( ) )”而不用“if ( f( ) )”。

（8）选用FALSE规则

大部分函数在错误时返回FALSE、0或NO之类的值，但在正确时返回值就不定了（不能用一个固定的TRUE、1或YES来代表），因此检测一个布尔值时应该用 FALSE、0、NO之类的不等式来代替。例如：使用“if ( FALSE != f( ) )” 而不用“if (TRUE == f( ) )”。

（9）独立赋值规则

嵌入式赋值不利于理解程序，同时可能回造成意想不到的副作用，应尽量编写独立的赋值语句。例如：使用“a = b + c ; e = a + d;”而不用“e = ( a = b + c ) + d ”。

（10）定义常量规则

对于代码中引用的常量（尤其是数字），应该define成一个大写的名字，在代码中引用名字而不直接引用值。

（11）模块化规则

某一功能，如果重复实现一遍以上，即应考虑模块化，将它写成通用函数。并向小组成员发布。同时要尽可能利用其它人的现成模块。

（12）交流规则

共享别人的工作成果，向别人提供自己的工作成果。在具体任务开发中，如果有其它的编码规则，则在相应的软件开发计划中予 以明确定义。

6. 编程准则

6.1. 变量使用

（1）不允许随意定义全局变量。

（2）一个变量只能有一个用途；变量的用途必须和变量的名称保持一致。

（3）所有变量都必须在类和函数最前面定义，并分类排列。

6.2. 数据库操作

（1）查找数据库表或视图时，只能取出确实需要的那些字段。

（2）使用无关联子查询，而不要使用关联子查询。

（3）清楚明白地使用列名，而不能使用列的序号。

（4）用事务保证数据的完整性。

6.3. 对象使用

尽可能晚地创建对象，并且尽可能早地释放它。

6.4. 模块设计原则

（1）不允许随意定义公用的函数和类。

（2）函数功能单一，不允许一个函数实现两个及两个以上的功能。

（3）不能在函数内部使用全局变量，如要使用全局变量，应转化为局部变量。

（4）函数与函数之间只允许存在包含关系，而不允许存在交叉关系。即两者之间只存在单方向的调用与被调用，不存在双向的调用与被调用。

6.5. 结构化要求

（1）禁止出现两条等价的支路。

（2）避免使用GOTO语句

（3）用 IF 语句来强调只执行两组语句中的一组。禁止 ELSE GOTO 和 ELSE RETURN。

（4）用 CASE 实现多路分支

（5）避免从循环引出多个出口。

（6）函数只有一个出口。

（7）不使用条件赋值语句。

（8）避免不必要的分支。

（9）不要轻易用条件分支去替换逻辑表达式。

7. 代码包规范

本项目中，每个任务在完成一个稳定的版本后，都应打包并且归档。

7.1. 代码包的版本号

项目中，代码包的版本号由圆点隔开的两个数字组成，第一个数字表示发行号，第二个数字表示该版的修改号。具体用法如下：

（1）当代码包初版时，版本号为 V1.00；

（2）当代码包被局部修改或bug修正时，发行号不变，修改号第二个数字增1。例如，初版代码包作了第一次修订，则版本号为 V1.01；

（3）当代码包在原有的基础上增加部分功能，发行号不变，修改号第一个数字增1，例如对V1.12版的基础上增加部分功能，则新版本号为 V1.20；

（4）当代码包有重要修改或局部修订累积较多导致代码包发生全局变化时，发行号增1。例如，在 V1.15 版的基础上作了一次全面修改，则新版本号为 V2.00。

7.2. 代码包的标识

本项目所产生的代码包都有唯一、特定的编码，其构成为：S-项目标识-代码包类型-版本号/序号。

（1）S：本项目的标识，表明本项目是“XXXX”。

（2）项目标识：简要标识本项目，此标识适用于整个项目的文档。

（3）代码包类型：取自以下表的两位字母编码。

（4）版本号：本代码包的版本号。

（5）序号：四位数字编码，指明该代码包在项目代码库的总序号。

8. 数据库命名规范

使用本系统遵循以下命名规范：

（1）表命名：用一个或三个以下英文单词组成，单词首字母大写，如：DepartmentUsers；

（2）表主键名称为：表名+ID，如Document表的主键名为：DocumentID

（3）存储过程命名：表名+方法,如：p\_my\_NewsAdd,p\_my\_NewsUpdate;

（4）视图命名：View\_表名，如：ViewNews;

（5）Status为表中状态的列名，默认值为0，在表中删除操作将会改变Status的值而不真实删除该记录；

（6）Checkintime为记录添加时间列，默认值为系统时间；

（7）表、存储过程、视图等对象的所有都为dbo，不要使用数据库用户名，这样会影响数据库用户的更改。

数据库服务器命名规范，控件名缩写+控件作用单词的全称,如：btnNext，txtPassword(密码文本框)，txtRPassword(密码确认文本框)，单词的第一个字母必须大写如果有多个单词，则为控件缩写+控件作用的第一个单词+第二个单词的全称，每个单词名的第一个字母必须大写。