|  |  |
| --- | --- |
|  |  |

全空间信息系统平台

软件编码规范

《全空间信息系统与智能设施管理》项目软件组

二○一六年十一月四日

文档版本记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 作者 | 日期 | 版本 | 纪要 |
|  | 张政 | 2016年11月22日 | V1.0.0 | 软件编码规范C++版和java版初稿 |
|  | 张亚军 | 2016年12月6日 | V2.0.0 | 合稿修订成一个版本 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[目录 3](#_Toc468968117)

[1 引言 5](#_Toc468968118)

[1.1 编写目的 5](#_Toc468968119)

[1.2 参考资料 5](#_Toc468968120)

[2 文件规范 5](#_Toc468968121)

[3 命名规范 5](#_Toc468968122)

[3.1 命名空间 5](#_Toc468968123)

[3.1.1 总体原则 5](#_Toc468968124)

[3.1.2 约束规范 5](#_Toc468968125)

[3.2 类的命名 6](#_Toc468968126)

[3.3 数据类型 7](#_Toc468968127)

[3.4 函数命名 7](#_Toc468968128)

[3.4.1 成员函数 7](#_Toc468968129)

[3.4.2 全局函数 7](#_Toc468968130)

[3.5 变量命名 7](#_Toc468968131)

[3.5.1 成员变量 7](#_Toc468968132)

[3.5.2 局部变量 8](#_Toc468968133)

[3.5.3 全局变量 9](#_Toc468968134)

[3.5.4 函数参数 9](#_Toc468968135)

[3.6 常量命名 9](#_Toc468968136)

[3.7 宏定义命名 9](#_Toc468968137)

[3.8 编译命名 9](#_Toc468968138)

[3.8.1 可执行文件 9](#_Toc468968139)

[3.8.2 动态库文件 10](#_Toc468968140)

[3.8.3 静态库文件 10](#_Toc468968141)

[3.8.4 Java库文件 10](#_Toc468968142)

[4 注释规范 10](#_Toc468968143)

[4.1 总体原则 10](#_Toc468968144)

[4.2 注释约束 10](#_Toc468968145)

[5 代码版式 11](#_Toc468968146)

[5.1 总体原则 11](#_Toc468968147)

[5.2 版式约束 11](#_Toc468968148)

# 引言

## 编写目的

为了能够使全空间信息系统平台及应用程序的结构和编码风格标准化，便于阅读和理解，促进开发出易于维护、高质量的软件原型，我们设计并编写了该软件编码规范。

## 参考资料

（1）C++编码规范

（2）Java编码规范

# 文件规范

（1）所有源代码文件都采用UTF-8编码格式；

（2）文件名应与类名严格保持一致（包括大小写）；

（3）文件包含可以使用带路径包含方式（使用“/”来分隔文件路径），但绝不允许用绝对路径；

（4）源文件必须包含：“版权信息”、“版本更新信息”的内容；

# 命名规范

## 命名空间

### 总体原则

（1）命名空间由英文单词或短语组成，不要使用汉语拼音，不要使用下划线“\_”分隔单词或短语，所有字母都用小写；

（2）命名空间的结构组织及命名应尽量与源文件所在目录一致，且大小写一致；

（3）java包的命名也参照上述两条。

### 约束规范

（1）全空间信息系统源代码根目录为onegis，即根命名空间为onegis；

（2）子系统的命名空间必须在根命名空间onegis下；

（3）模块的命名空间必须在子系统命名空间下；

（4）子系统或模块中带界面部分的根命名空间一定为ui；

（4）命名空间在起名时，必须查阅命名空间对照表。

命名空间对照表（样例模板）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 子系统 | 子系统说明 | 模块 | 模块说明 |
| 1 | psde | 全空间数据访问引擎 | core | 核心逻辑结构 |
| … | … |
| 2 | psdf | 全空间信息系统开发框架 | core | C++核心模块扩展框架库 |
| ui | 界面扩展框架库 |
| … | … |
| 3 | … | … | … | … |

## 类的命名

（1）类名应为英文名词或名词短语，不要使用汉语拼音；

（2）对类名中所包含的每一个单词或缩写，以大写开头，其余部分小写；

（3）可实例化的类采用非A和非I的大写字母开头（例如：C++语言采用一般采用大写C开头）；

（4）抽象类采用大写字母A开头；

（5）接口类（纯虚类）采用大写字母I开头。

注意：类头宏定义采用命名空间+“\_”+类名的方式，字母都大写，例如：

#ifndef ONEGIS\_PSDE\_C\_OBJECT

#define ONEGIS\_PSDE\_C\_OBJECT

namespace onegis{

namespace psde{

class C\_Object

{

};

}// namespace psde

}// namespace onegis

#endif

## 数据类型

（1）结构体命名：使用大写S开头的形式，如S\_OType；

（2）枚举类型命名：使用大写E开头的形式，如E\_FieldType；

（3）共同体类型命名：使用大写U开头的形式，如U\_Data。

## 函数命名

### 成员函数

（1）可以采用Pascal的代码风格：没有前缀，首字母大写，函数名中所包含的每个短语的首字母大写，其余小写，如：OpenFile()、CreateWorkArea()、Close()；也可以采用近年来用的越来越多的Camel风格：首字母小写，函数名中所包含的每个短语的首字母大写，其余小写，如openFile()、createWorkArea()、close()。但选定了一种风格后，在同一模块中应严格保持命名风格的一致性。

（2）不要用下划线“\_”分隔单词，方法名应使用动词或动词短语，尽量不要使用汉语拼音命名。

（3）对于获取或者设置类属性（成员变量）的方法，应由Get/get或者Set/set开头，如：GetHeight()/getHeight()、SetHeight()/setHeight()。

（4）如果方法是测试类的某个布尔属性，则方法名应为IsXXX()或者isXXX()，如bool isVisible()。

### 全局函数

以G\_为前缀，后跟全局函数名，如：

G\_CreateLayerObj()、G\_GetMainControllerObjectPtr()。

注：除了系统级和模块级等少数对象工厂函数外，尽量少使用全局函数。

## 变量命名

### 成员变量

（1）用m\_为前缀，紧跟变量数据类型缩写（小写字母），后面每个单词词组以大写字母开头，其余部分小写，不要用下划线“\_”分隔单词或词组，尽量不要使用汉语拼音，如：

int m\_iFeatureCount;

bool m\_bIsOpen;

char\* m\_szFileName;

ILayer\* m\_pLayer;

（2）布尔类型的变量，可以冠以Is，或者以able结束，如：

bool m\_bIsNull;

bool m\_bResizable;

（3）指针类型的变量，应冠以p，后面跟大写字母，如：

ILayer\* m\_pLayer;

（4）变量命名中常用的数据类型缩写表（参考匈牙利命名法）

|  |  |  |
| --- | --- | --- |
| **缩写** | **数据类型** | **举例** |
| b | 布尔bool | bool m\_bIsOpen |
| c | 单个字符char | char m\_cCharactor |
| str | 字符串string类型 | string m\_strFileName |
| sz | char类型字符数组 | char m\_szLayerName[40] |
| i | 整型int | int m\_iFeatureCount |
| s | 短整型short | short m\_sValue |
| f | 单精度浮点float | float m\_fValue |
| d | 双精度浮点double | double m\_dValue |
| p | 指针。如果指向自定义类对象，则只用p；如果指向上述某个基本类型，则再加基本类型缩写作为后缀 | ILayer\* m\_pLayer  char\* m\_pszName |
| u | 无符号类型，后跟其类型缩写 | unsigned char m\_ucType |
| a | 数组，后跟其它类型 | int m\_aiXCoords[10] |

### 局部变量

（1）允许在函数的任何位置声明变量，我们提倡在尽可能小的作用域中声明变量，离第一次使用越近越好，且要求在声明变量时将其初始化。这可以使代码易于阅读，易于定位变量的位置、变量类型和初始值。

（2）局部变量的命名全部使用小写，单词或短语间用下划线“\_”分隔，从而便于识别变量的作用范围，与函数参数明确区分。

（3）如果通过名称能够明确得知数据类型，则不用加上数据类型缩写作为前缀；如果通过名称不能明确得知数据类型，则建议加上数据类型缩写作为前缀，并用下划线“\_”与变量名分开，如：

int i, j, feature\_count;

int i\_field\_value;

double d\_dem\_value;

（4）尽量不要使用单个字母作为变量名，如p，r。对于简单的for循环的计数变量，可以用i、j等单字母变量；但对于有意义的计数变量，应用具有意义的名词，如field\_index。

### 全局变量

（1）用g\_作为前缀，其余同类的成员变量命名规则，如：

int g\_iMaxFeatureCount;

（2）不要使用class类型的全局变量（包括STL的string、vector等），因为它们的初始化顺序有可能导致构造出现问题。內建类型和由內建类型构成的没有构造函数的结构体可以使用，如果一定要使用class类型的全局变量，请使用单件模式（singleton pattern）。

（3）对于全局的字符串常量，使用C风格的字符串，而不要使用STL的字符串，如：

const char g\_szAppName[] = “OneGIS”;

### 函数参数

除了没有m\_前缀之外，与类成员变量的命名方式相同，如：

SetFieldHeight(ILayer\* pLayer, int iFieldHeight)

## 常量命名

（1）全部大写，使用“\_”来分隔不同的部分，如：

const int MAX\_PROCESS\_COUNT = 64；

（2）对于需要作为API的一部分提供给二次开发用户的常量，应以ONEGIS\_作为前缀，如：

const int ONEGIS\_ZERO = 10e-6；

## 宏定义命名

宏定义以ONEGIS\_作为前缀，全部大写，用下划线“\_”将各个单词或短语分隔开，形如：

#define ONEGIS\_MAX\_FILENAME\_LEN 1000；

## 编译命名

### 可执行文件

对于Windows平台而言，编译成的可执行文件命名应形如“onegis\_\*.exe”。

对于Linux平台而言，编译成的可执行文件命名应形如“onegis\_\*”。

### 动态库文件

Windows平台上，编译生成的动态库文件名应形如“onegis\_\*.dll”。

Linux平台上，编译生成的动态库文件名应形如“onegis\_\*.so”。

### 静态库文件

Windows平台上，编译生成的静态库文件名应形如“onegis\_\*.lib”。

Linux平台上，编译生成的静态库文件名应形如“onegis\_\*.a”。

### Java库文件

对于java语言，编译成执行或库文件名应形如“onegis\_\*.jar”。

# 注释规范

## 总体原则

（1）源程序有效注释量必须在20%以上。注释的原则是有助于对程序的阅读理解，在该加的地方加注释，注释不宜太多也不能太少，注释语言必须准确、易懂、简介。

（2）边写代码边注释，修改代码的同时修改相应的注释，以保证注释与代码的一致性，没用的注释要删除。

（3）注释的内容要含义准确，防止注释二义性（错误的注释不但无益反而有害）。

（4）对代码块的注释，注释语句放在代码块之上，用“//”进行注释，如果注释语句本身超过一行，则用“/\* \*/”进行注释。

## 注释约束

（1）靠近原则。尽量将注释写在代码前面，如：

// increase index

index++;

（2）API原则。对于需要作为API的类型、常量、方法等，应提供文档性注释，以方便抽取编程参考手册，如：

/\*\*

\* @brief 判断当前对象是否支持给定接口。

\*

\* 判断当前对象是否支持给定接口。

\*

\* @param interfaceId 接口的ID

\*

\* @return 若当前对象支持给定接口，则返回1，否则返回0。

\*/

int interfaceSupported ( int interfaceId );

# 代码版式

## 总体原则

（1）一行代码只做一件事情，如定义一个变量，或只写一条语句。

（2）程序块要采用缩进风格编写，缩进的空格数为4个，使用space字符（说明：对于由开发工具自动生成的代码可以允许不一致）。

## 版式约束

（1）组合语句采用形如以下的版式

int i;

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

{

statement1;

if(j < 200)

{

statement2;

statement3;

}

else

{

statement4;

statement5;

} // end if

} // end for

注意：“if”、“for”、“while”、“do”、“try”、“catch”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“{}”，这样可以防止书写和修改代码时出现失误。

（2）使用一致的折行版式，例如：

enum EColor

{

red,

orange,

yellow,

green,

violet

} ;

（3）多个参数方法，可采用如下版式

int add ( int iArg1,

char\* pArg2,

double dArg3 );

（4）空格版式约束

在适当的地方加空格，以增强代码的清晰性和可读性。在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符（如->）后不应加空格。