**中科红旗（北京）信息科技有限公司**



|  |
| --- |
| **代码编写规范** |

|  |  |
| --- | --- |
| **文件编号：** | GF-CX08-01 |

|  |  |  |  |
| --- | --- | --- | --- |
| **拟制：** | 刘华先 | **日期：** | 2020.5.28 |
|  |  |  |  |
| **审核：** | 董金彩 | **日期：** | 2020.5.28 |
|  |  |  |  |
| **批准：** | 刘偊赜 | **日期：** | 2020.5.28 |

修订记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **版本** | **修订说明** | **修订者** | **修订日期** |
| 1 | 1.0 | 创建此文件 | 刘华先 | 2020.5.28 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 目的

在软件开发过程中，研发人员应遵循代码编写规范，保证代码的规范性、可读性和可维护性，为产品的稳定性和可靠性提供保障，也使得编码过程变得工程化和和轻松高效。

# 适用范围

适用于本公司软件项目开发过程中代码编写的过程。同时作为项目代码评审的依据之一。

# C/C++代码规范

## 关于单元测试

1. **测试代码的位置**

测试代码应与src目录并列，并单独建立一个名为tests的目录。

tests目录应至少包活两个文件：一个test.h和一个main.cpp。其中test.h文件至少包括以下内容：

#include <QApplication>

#include <QtGui>

#include <QtTest/QtTest>

using namespace QTest;

1. **测试类与被测类的关系**

为访问被测类的私有成员，故设置测试类为被测类的友元，故被测类的结构为：

class Xxx : public

{

public:

friend class XxxTest;

Xxx(); // constructor

~Xxx();// destructor

......

};

被测类应作为测试类的私有成员变量，故测试类的结构为：

class XxxTest : public QObject

{

Q\_OBJECT

public:

XxxTest();

~XxxTest();

private slots:

void initTestCase();

void cleanupTestCase();

void init();

void cleanup();

void function1Test();

void function2Test();

......

private:

class Xxx \*Xxx\_p;

};

1. **测试代码的命名规范**

测试代码的命名规范应遵循《C/C++代码规范》。

一个类的测试类的定义应在一个.h文件中，测试类的实现在一个.cpp文件中。

假设被测函数为function，测试函数命名为functionTest；

假设被测类为Xxx，测试类命名为XxxTest，相应的头文件为XxxTest.h,相应的cpp文件应为XxxTest.cpp。

1. **main函数示例**

其中main.cpp如下所示：

#include "test.h"

#include "xxx.h"

int main(int argc, char \*argv[])

{

QApplication app(argc, argv);

XxxTest XxxTest\_t;

Qtest::qExec(&XxxTest\_t);

YyyTest YyyTest\_t;

QTest::qExec(&YyyTest\_t);

......

return 0;

}

1. **测试覆盖率**

要求测试覆盖类的所有成员函数，包括公有、保护和私有，但是纯虚函数和通过继承得到的函数不需要测试。

1. **执行测试的时机**

代码在提交到svn前必须执行单元测试；在编包时必须执行单元测试。

1. **对源代码的修改**

见b点，friend class XxxTest。（b、测试类与被测类的关系）

## 关于代码中声明部分的说明

1. **define守卫**

为避免多次包含，所有头文件都应该有#define宏。其格式为\_<PROJECT>\_<PATH>\_<FILE> \_H\_

1. **头文件依赖**

为了减少不必要的编译依赖，在头文件中应尽量使用前置声明(forward declaration)来代替#include。

1. **函数参数顺序**

在定义函数时，参数基本顺序的原则是：输入参数，输入输出参数，输出参数。

1. **局部变量使用**

在函数中使用局部变量时，应在最靠近其第一次使用的地方声明，使其作用域越窄越好。例外情况是，对于复杂类，不要在循环内部定义，以避免不必要的多次构造和析构。

1. **全局变量的使用**

类类型的全局变量是不允许的，内置类型允许。对于类类型，可以使用单例模式。

1. **显式构造函数**

为避免不希望的转型，参数的构造函数应使用explicit关键字。

1. **多继承**

多继承一般是不允许的，除非最多只有一个基类有实现，其他基类都是接口或纯虚的。

1. **访问控制**

所有数据成员都是私有的，使用accessor和mutator 。

1. **写短函数**

尽量写短小和功能单一的函数，有利于重构和重用。40行左右是比较理想的，但并不绝对。

1. **引用类型的参数**

所有引用传递的参数都应标记为const，需要被修改的对象请用指针传递。

1. **casting**

使用C++的转型算子而不要使用c的强制转型。

1. **preincrement & predecrement**

优先使用preincrement和predecrement，因为它们不会产生左值。

1. **const的使用**

推荐在任何应该使用const的地方都使用它。包括变量，函数，函数参数等。

1. **0和NULL**

为了避免混乱，对整数使用0，实数用0.0，指针用NULL，字符串用'\0'

1. **sizeof**

尽可能使用sizeof(变量名)，而不是sizeof(类型名)。

## 关于代码中命名部分的说明

1. **一般命名规则**

所有名字都应该具有描述性，避免易混淆的缩写。变量和类型时名词，函数时动词。

一般局部变量使用格式如下：

int numErrors; // 单词连写，首字母小写。

缩写一般只在专有术语或广为人知的情况下使用，比如

int numDNSConnections;

类型名以大写字母开头，单词连写，每个单词首字母大写。避免下划线。

class UrlTable { ....  
typedef hash\_map<UrlTableProperties\*, string> PropertiesMap;

类成员变量名一般以m\_开头，如果时指针类型以m\_p开头，其他部分单词连写，每个单词首字母大写。

string m\_tableName;

UrlTable \*m\_pUrlTable;

全局变量以g\_开头，其他同局部变量。

extern int g\_windowList;

一般函数命名类似局部变量，单词连写，除第一个单词外，首字母大写。

如果是数据成员的accessor和mutator，则应该是setVariableName和variableName格式。

class Foo {

int numEntries() const { return m\_numEntries; }

void setNumEntries(int i) const {　m\_numEntries = i; }

private：

int m\_numEntries;

};

1. **函数注释**

本节重点关注类、函数及重要结构的注释要求，对其他方面不做硬性要求函数声明的注释紧靠声明的开头，注释中至少写明参数意义，参数取特殊值的意义等。并且应该遵照doxygen注释规则，以便生成代码文档。以下为参考格式，具体的细节请参考doxygen的格式说明（www.doxygen.org）。

以下为一个完整的类注释举例：

/\*\*

\* \class ExampleClass

\*

\* \ingroup PackageName

\* (Note, this needs exactly one \defgroup somewhere)

\*

\* \brief Provide an example

\*

\* This class is meant as an example. It is not useful by itself

\* rather its usefulness is only a function of how much it helps

\* the reader. It is in a sense defined by the person who reads it

\* and otherwise does not exist in any real form.

\*

\* \note Attempts at zen rarely work.

\*

\* \author (last to touch it) $Author: bv $

\*

\* \version $Revision: 1.5 $

\*

\* \date $Date: 2005/04/14 14:16:20 $

\*

\* Contact: bv@bnl.gov

\*

\* Created on: Wed Apr 13 18:39:37 2005

\*

\* $Id: doxygen-howto.html,v 1.5 2005/04/14 14:16:20 bv Exp $

\*

\*/

#ifndef EXAMPLECLASS\_H

#define EXAMPLECLASS\_H

class ExampleClass

{

public:

/// Create an ExampleClass

ExampleClass();

/// Create an ExampleClass with lot's of intial values

ExampleClass(int a, float b);

~ExampleClass();

/// This method does something

void DoSomething();

/\*\* This is a method that does so

\* much that I must write an epic

\* novel just to describe how much

\* it truly does. \*/

void DoNothing();

/\*\* \brief A useful method.

\* \param level an integer setting how useful to be

\* \return Output that is extra useful

\*

\* This method does unbelievably useful things.

\* And returns exceptionally useful results.

\* Use it everyday with good health.

\*/

void\* VeryUsefulMethod(bool level);

private:

const char\* fQuestion; ///< the question

int fAnswer; ///< the answer

}; // end of class ExampleClass

#endif // EXAMPLECLASS\_H

1. **代码格式化**

行长一般不超过80字符。

非ascii字符尽量不要使用，如果使用，应该使用UTF-8编码。

缩进使用4个空白字符。

函数声明看起来如下：

ReturnType ClassName::FunctionName(Type par\_name1, Type par\_name2) {

DoSomething();

...

}

或者

ReturnType ClassName::ReallyLongFunctionName( Type par\_name1,

Type par\_name2,

Type par\_name3) {

DoSomething();

...

}

# Python 代码编程规范

## 代码的布局

1. **缩进**

对新编的代码使用四个空格进行缩进（这也是Emacs Python Mode默认的缩进），对于已有的项目如果使用了制表符缩进，为了保持项目的兼容性应该继续使用制表符。请不要使用空格和制表符的混合缩进方式。

1. **行的最大长度**

请将所以行最大长度限制在79个字符以内，对顺序存放的大块文本（文档字符串或注释），推荐将长度限制在72个字符。折叠长行请尽量使用Python支持的圆括号，方括号和花括号内的行的延续，否则使用反斜杠，并确认恰当的缩进了延续的行。

1. **空行**

用两个空行分割顶层函数和类的定义，类内方法的定义用单个空行分割，在函数内使用空行时，空行用于表示一个段落

1. **编码**

尽量使用ASCII或Latin-1编码，如果必须使用其他编码，请一律使用UTF-8编码，在原文注释和文档字符串中一律使用英文，如果必须要求中文文档，请另附文档文件如README.zh\_CN

## 导入（import）

通常应该在单独的行中导入，如：

*import os*

*import sys*

*from types import StringType, ListType*

Imports通常放在文件顶部，仅在模块注释和文档字符串之后，在模块的全局变量和常量之前，并且有顺序的成组放置，在每组导入直接放置一个空格。

1. 标准库的导入
2. 相关主包的导入，如gtk，qt等
3. 特定应用的主包的导入

## 空格

以下空格规范是源自Python创始人Guido，请尽量保持与本规范相同，有利于我们代码风格的一致，增加可读性。

下列情况不应该出现空格：

1. 紧挨着圆括号、方括号和花括号，如“spam(ham[1], {egg:2})”
2. 紧贴在逗号、分号或冒号前的，如“if x == 4: print x, y; x, y = y, x”
3. 紧跟着函数调用的参数列表前开括号的，如“spam(1)”
4. 紧贴着下标的开括号前，如“dict['key'] = list[index]”
5. 不要在用于指定关键字参数和默认参数的‘=’使用空格，如“def complex(real, imag=0.0)”

始终在一下这些运算符两边放置一个空格：赋值（=）比较（==，<，>，!=，<=，>=，in，not in，is，is not），布尔运算（and，or，not），另外请尽量不要将多条语句写在同一行。

## 注释

1. **代码注释**

同代码不一致的注释比没有注释更差，当代码修改时，始终优先更新注释。

注释应该使用完整的句子。如果注释是一个短语或句子，首字母应该大写，除非它是一个以小写字母开头的标识符。

如果注释很短，最好省略末尾的句号注释块通常由一个或多个完整的句子构成的段落组成，每一个句子应该以句号结尾。

应该在每个句子的句号末使用两个空格，以便和大多数编辑器兼容。

请一律是用英文书写注释。

注释块用于注释随后的一些代码并和这些代码拥有相同的缩进，注释块中每行以‘#’和一个空格开始。注释块内的段落以仅含单个‘#’的行分割。注释块的上下方最好各留一个空行。

行内注释和语句在同一行，行内注释要谨慎使用，行内注释应该以一个‘#’和一个空格开始，并至少用两个空格和前面的语句分开。

1. **文件头版权注释**

请至少在公司正规立项的项目代码的源文件中添加版权声明文件头，并注明代码所使用的License，文件头中应至少包含以下字符串(这里请将[YEAR]替换成年分）：

*# Copyright (C) [YEAR] Red Flag Software Co., Ltd.*

如以下为使用GPLv3的示例：

# Copyright (C) 2008, 2009 Red Flag Software Co., Ltd.

# Author: Tom Wang <tomwang@redflag-linux.com>

# This file is part of LIME3.

# LIME3 is free software: you can redistribute it and/or modify

# it under the terms of the GNU General Public License as published by

# the Free Software Foundation, either version 3 of the License, or

# (at your option) any later version.

# LIME3 is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License

# along with LIME3. If not, see <http://www.gnu.org/licenses/>.

## 文档化

必须为公共模块、函数、类和方法编写文档字符串，给非公开的方法编写文档字符串是非强制的，但你应该由描述这个方法是干什么的的注释，这个注释应该在def之后。

请确定为每一个源代码文件编写了文档字符串。

使用PEP 257[[1]](#footnote-0)描述的文档字符串约定。注意，多行文档字符串结尾的”””应该单独成行（对单行来说”””可以位于同一行）。

## 命名约定

1. **通用规则**
   1. 以一个下划线作前导：是若的“内部使用”标志，from module import \*，不会导入它们。
   2. 以一个下划线结尾：用于避免与Python关键字冲突，如：func(m, class\_='classname')
   3. 以双下划线前导：用于类的私有变量名。
   4. 以双下划线开始和以双下划线结尾：通常是特殊的对象或属性，存在于用户控制的名字空间中，例如：\_\_init\_\_，\_\_import\_\_或\_\_file\_\_，应该限制使用这类标示符。
   5. 请永远使用英文命名（虽然Python 3.0支持UTF-8命名）。
   6. 不要使用l（小写字母el），O（大小字母oh)或I（大小字母I）作为但字符的变量名，在某些字体中无法与数字1和0区分。
2. **模块名**

模块名应是不含有下划线的、简短的、小写的的名字，当一个用C或C++写的扩展模块有一个伴随的Python模块，这个Python模块提供了一个更高层的接口时，C/C++模块有一个前导下划线（如：\_socket）。Python包应该是一个不含下划线简短的小写名字。

1. **类名**

类名应该总是使用首字母大写单词串（CapWords）的约定。

1. **异常名**

如果模块对所有情况定义了单个异常，它通常被叫做“error”或“Error”。

1. **函数名**

函数名应该为小写，可能用下划线风格的单词以增加可读性。

1. **方法名和实例变量名**

命名规则同函数，当不作为类的公共接口的内部方法或变量应该使用一个前导下划线，这里并是不强制，同时也可以使用两个前导下划线作为Python强制的私有名字。

1. **继承**

设计时要确定一个类中的方法或实例变量是否被公开，一般永远不要将数据变量公开，除非你实现的本质只是记录，通常应该给变量提供一个接口作为替换。为属性起名字时注意：公共属性没有前导和后置下划线，如果他们的名字与保留字冲突，可以在变量名后面加一个后置下划线，如class\_。

## 其他编码规则

1. 同象None之类的单值进行比较时要使用'is'和'is not'。
2. 使用基于类的异常，模块和包应该定义它们域内的异常基类，这个域内的异常基类应该是内建Exception的子类，而且该类要有内容明确的文档字符串。
3. 在检查字符串的前后缀时请使用‘startwiths’和‘endwiths’函数。
4. 对象类型的比较用isinstance()，如应该用“if isinstance(obj,int)”，而不是“if type(obj) is type(1)。检查是否是字符串时应该用“isinstance(obj,basestring)”（Python 2.3版本之后)。
5. 不要用‘==’来比较布尔型以确定是“True”还是“False”

# RPM打包规范

## 包文件名定义规范:

包文件名的一般形式为：

%{name}-%{version}-%{release}%{?dist}.platform.rpm

* 1. Version为版本号，在做同一版本系统时，通常会确定每个包的版本号，没有特殊需求的情况Version号不应改变。如果更新了版本号，那么release号回到1(有一种特例情况release号不归1将在后面说明)。
  2. Release为发行号，每次重新编译后，无论是否有作修改，提交到仓库中的包必须++发行版本号。
  3. 如果包来源于其他发行版或开源社区，那么第一次打包时，release号应该更新为%{release}.1，例如release号为1.2的,那么应该变成1.2.1；以后每次更新增加release号末尾数值。
  4. Release后应该跟%{?dist}，如果打包环境中存在相应处理脚本，编译后%{?dist}会被替换为AXS3,EL等字符串。也就是1.2 -> 1.2.1%{?dist}；如果原来就有%{?dist}那么把它挪到后面：1.2%{?dist} -> 1.2.1%{?dist}。
  5. 如果源自其他发行版或开源社区的包release号包含有RHEL或类似的字符串（不是变量），那么去掉它，无论字符串在什么位置。比如1.3RHEL5或 1.3.el5或1.el5.3，升级后都改为1.3.1%{?dist}。
  6. 如果是自建的新包，版本号和发行号中一律不允许出现数字和”.”之外的字符
  7. 如果从其他系统中引入的包中含有beta,rc等字符串，一律去掉，并在changelog中写明对应关系。例如3.1beta-1->3.1-1.1%{?dist}; 3.1-1beta->3.1-1.1%{?dist}。
  8. 如果再次拿到上述其他系统更新后的包，版本号或发行号与原来的包在数字上一致而差别只在非数字字符串，例如原包版本为3.1-1.1%{?dist}并对应3.1beta-1，需要更新到3.1-1时，那么丢弃非数字字符串，只增加发行号，变为:3.1-1.2%{?dist}。再如： 3.2-1.4%{?dist}对应某发行版3.2-1beta，并更新了3次release号，当需要更新到3.2-1rc时，变为：3.2-1.5%{?dist}。

## 打包行为规范

1. 编译以及安装动作动作能够使用标准宏(例如%configure、%makeinstall等)的地方必须使用宏。
2. 涉及安装路径可以使用标准宏的地方（例如%{\_bindir}, %{\_libdir}等)必须使用宏，禁止使用绝对路径。
3. 每次重新编译必须添加ChangeLog，ChangeLog中要写明当前版本号(如果是从其他发行版或社区直接获得的包并更新了版本号要写明版本对应关系)、为什么重新编译、涉及哪些修改。
4. 如果是为修复BUG而作的更新，要写明对应的BUG号。
5. ChangeLog和Patch只允许添加，禁止修改已有内容。
6. 一个SRPM包至少要在x86和x86-64平台上测试，保证可以正确编译，安装和运行（源代码本身的问题除外）。
7. 新建的包SPEC文件中禁止出现中文、日文等一切含ASCII值0x80以上字符的文字。
8. 如果已有的包中含有上述字符，ChangeLog可以保留,Summary，Descrīption以及注释等内容视情况改为英文或删除。

# 代码规范通用部分

1. 如果修改其他项目的代码，应该尽量遵循原有代码的风格与规范。
2. 在编码过程中缩写要慎用，不得产生歧义。
3. 在合适的地方加上空行和注释，尤其是新代码。

# 附录：

无

# 相关支持文件

|  |  |  |
| --- | --- | --- |
| 序号 | 名称 | 编号 |
| 1 | 《软件开发控制程序》 | ZKRF-QM-CX08 |
| 2 | 《代码评审规程》 | GC-CX08-01 |

# 质量记录

无

|  |
| --- |
|  |

本程序由产品研发部提出。

1. <http://www.python.org/dev/peps/pep-0257/> [↑](#footnote-ref-0)