1. **M\_变量声明与命名风格**
2. 代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。反例：\_name / \_\_name / $name / name\_ / name$ / name\_\_
3. 代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。 说明：正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式 也要避免采用。

正例：alibaba / taobao / youku / hangzhou 等国际通用的名称，可视同英文。

反例：DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量 = 3

1. 变量名和函数名以小写字母开头，开头之后的部分每个单词以大写字母开头。

正例：short counter; / char itemDelimiter = ' ';

1. 类名以大写字母开头。

正例：XmlService / User

1. 常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。 正例：MAX\_STOCK\_COUNT
2. 抽象类命名使用 Abstract 或 Base 开头；异常类命名使用 Exception 结尾；测试类 命名以它要测试的类的名称开始，以 Test 结尾。
3. 避免短的或无意义的命名。使用尽量完整的单词 组合来表达其意。

正例：：AtomicReferenceFieldUpdater。

反例：变量 int a 的随意命名方式。

1. 枚举成员名称需要全大写，单词间用下划线隔开。
2. 单个字符的变量名只适用于用来计数的临时变量，因为此时该变量的用途十分明显
3. 声明每一个变量都要用独立的一行

正例：int height; / int width;

1. 当一个变量被用到时再声明它。
2. 对于指针或引用，在类型名和\*或&之间用一个空格，但是在\*或&和变量名之间没有空格

正例：char \*x; / const QString &myString; / const char \* const y = "hello";

**二．代码格式**

1. 大括号的使用约定。如果是大括号内为空，则简洁地写成{}即可，不需要换行；如果 是非空代码块统一如下例：

if ( ... )  
{  
...  
}  
else  
{  
...  
}

1. 左小括号和字符之间不出现空格；同样，右小括号和字符之间也不出现空格

反例：if (空格 a == b 空格)

1. if/for/while/switch/do 等保留字与括号之间都必须加空格。

正例：if (foo)

1. 任何二目、三目运算符的左右两边都需要加一个空格。

正例：lineF == lineN

1. 注释的双斜线与注释内容之间有且仅有一个空格。

正例： // 这是示例注释，请注意在双斜线之后有一个空格

1. 方法参数在定义和传入时，多个参数逗号后边必须加空格。即逗号左边没有空格，逗号右边有一个空格。

正例: QT\_REQUIRE\_VERSION(argc, argv, "4.0.2")

1. 左引号的左边和右引号的右边都有一个空格，左引号的右边和右引号的左边都没有空格。如果右引号右边是右括号的话，它们之间没有空格。

正例: qDebug() << Q\_FUNC\_INFO << "was called with value1:" << value1 << "value2:" << value2;

QT\_REQUIRE\_VERSION(argc, argv, "4.0.2")

1. 不同逻辑、不同语义、不同业务的代码之间插入一个空行分隔开来以提升可读性。
2. 当条件语句的执行部分多于一句的时候才使用花括号。然而在if-else语句块中，如果if或else中的一个包含了多行，另一个为了对称性原则，也要用花括号。

正例：if (address.isEmpty())

return false;

1. 每行代码不多于100个字符，超出需要换行。换行时：

第二行相对第一行缩进 4 个空格，从第三行开始，不再继续缩进。

运算符与下文一起换行。

方法调用的点符号与下文一起换行。

方法调用中的多个参数需要换行时，在逗号后进行。

正例：if (longExpression

+ otherLongExpression

+ otherOtherLongExpression) { }

**三．类相关**

（1）避免通过一个类的对象引用访问此类的静态变量或静态方法，无谓增加编译器解析成 本，直接用类名来访问即可。

（2）构造方法里面禁止加入任何业务逻辑，如果有初始化逻辑，请放在 init 方法中。

（3）类内方法定义的顺序依次是：公有方法->公有槽->信号->保护方法->私有方法->私有槽。

（4）当一个类有多个构造方法，或者多个同名方法，这些方法应该按顺序放置在一起， 便于阅读。

（5）成员变量一般为名词。函数成员一般为动词/动词+名词，当返回值为Bool型变量时，函数名一般以前缀’is’开头。

正例：void setDirty(bool b); / bool isDirty() const;

（6）每一个对象，只要它不是基础类型(int, float, bool, enum, or pointers)，都应该以常量引用的形式传递。这条使得代码运行得更快。

正例：QString myMethod(const QString &foo, const QPixmap &bar, int number);

（7）为了使构造函数被错误使用的可能性降到最小，每一个构造函数（除了拷贝构函数）都应该检查自己是否需要加上explicit 符号。

（8）尽量减少在头文件中包含其他头文件的数量，可以用前置声明法。

正例：class KBar;

class KFoo : public KFooBase

{ … };

（9）假如你有一个Foo类，有Foo.h文件和Foo.cpp文件，在你的Foo.cpp文件中，要先包含Foo.h文件再包含其他头文件。

正例：// .h文件

class Foo

{ …};

// .cpp文件

#include "foo.h"

#include "bar.h"

1. **控制相关**
2. case标签和switch在同一列。每一个case语句的末尾都要有一个break语句或return语句。在一个 switch 块内，都必须包含一个 default 语句并且 放在最后，即使空代码。
3. 除常用方法（如 getXxx/isXxx）等外，不要在条件判断中执行其它复杂的语句，将 复杂逻辑判断的结果赋值给一个有意义的布尔变量名，以提高可读性
4. **注释相关**
   1. 方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释 使用/\* \*/注释，注意与代码对齐。
   2. 对于注释的要求：第一、能够准确反应设计思想和代码逻辑；第二、能够描述业务含 义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同 天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给继任者看 的，使其能够快速接替自己的工作。
   3. 好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的 一个极端：过多过滥的注释，代码的逻辑一旦修改，修改注释是相当大的负担。