# 一、QtWidgets C++部分规范

# 1.文件

## 文件头

在.h/.hpp 的开头应有一段格式统一的说明，内容包括：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 关键字 | 关键字含义 | 说明 |
|  | @brief | 文件简要功能描述 |  |
|  | @details | 文件详细功能描述 |  |
|  | @author | 作者名称 |  |
|  | @date | 创建、修改日期 |  |
|  | @version | 文件版本 |  |
|  | @warning | 特殊警告说明 |  |
|  | @copyright | 公司版权说明 |  |
|  | @note | @date @author 修改说明历史n | @date格式为yyyymmdd,@author一般为拼音首字母缩写,@date/@author/修改说明历史之间使用1个tab分割。 |

关键字和关键字含义之间使用2个tab分割，保持书写格式优美。

1个tab采用4个空格。

样例：

/\*\*

\*@brief 文件简要功能描述

\*@author 作者名称

\*@date 创建、修改日期

\*@version 文件版本

\*@warning 特殊警告说明

\*@copyright 公司版权说明

\*@note @date @author 修改说明历史n

\*@note @date @author 修改说明历史n-1

\*......

\*@note @date @author 修改说明历史2

\*@note @date @author 修改说明历史1

\*/

## 头文件组成

头文件一般包括：文件头、编译宏定义开始、包含系统文件、包含工程文件、类/结构前置声明、、extern全局函数/变量宏定义、类定义、编译宏定义结束等。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 组成部分 | 样例 | 说明 |
|  | 文件头 |  |  |
|  | 编译宏定义开始 | #ifndef DLG\_DEV\_H |  |
|  | 包含系统文件 | #include <QtCore/qstring.h> | 将包含文件到最小最直接 |
|  | 包含工程文件 | #include “DevManager.h” |  |
|  | 类/结构前置声明 | class CDevCfg; |  |
|  | 扩展全局变量 | extern int g\_i32DevCount; |  |
|  | 扩展全局函数 | extern CDevCfg\* GetDevCfg(int i32Index); |  |
|  | 常量定义 | const int MAX\_DEV\_NAME\_LEN; |  |
|  | 枚举定义 | enum DEV\_TYPE |  |
|  | 类/结构定义 | class RTX\_MOD CDlgDev: public QDialog, CRTDialog |  |
|  | 编译宏定义结束 | #endif //DLG\_DEV\_H |  |

样例：

/\*\*

\*@brief 文件简要功能描述

\*@author 作者名称

\*@date 创建、修改日期

\*@version 文件版本

\*@warning 特殊警告说明

\*@copyright 公司版权说明

\*@note @date @author 修改说明历史n

\*@note @date @author 修改说明历史n-1

\*......

\*@note @date @author 修改说明历史2

\*@note @date @author 修改说明历史1

\*/

#ifndef DIALOGDEV\_H

#define DIALOGDEV\_H

#include <QtCore/qstring.h>

#include <QtGui/qmessagebox.h>

#include "DevManager.h"

#include "DevThread.h"

//前置声明

class CDevCfg;

class CDevInfo;

extern int g\_i32DevCount;

extern int g\_i32DevState;

extern CDevCfg\* GetDevCfg(int i32Index);

extern CDevInfo\* HandleDevInfo(const QString& strDevCode);

const int MAX\_DEV\_NAME\_LEN = 64;

enum DEV\_TYPE

{

DT\_1500 = 0,

DT\_208 = 1,

DT\_212 = 2

};

/\*\*

\* @class 设备窗口操作类

\*/

class RTX\_MOD CDialogDev : public QDialog, CRTDialog

{

Q\_OBJECT

public:

explicit CDialogDev(QObject \*parent = 0);

signals:

public slots:

};

#endif // DIALOGDEV\_H

## 文件独立

1. 定义和实现分开。内联函数除外。
2. 应该在一个单独的.h /.hpp文件中定义一个类，在一个单独的.cpp 文件中实现这个类。
3. .h 与.cpp 文件的文件名必须与类名相同。除了类名的首个大写字母C。
4. 相关模块、相似功能的文件命名规则一致，统一放到相应的功能代码目录。

## 文件合并

1. 若几个类的规模都不大，关系又很密切，比如业务关系，则可在一个.h 文件中定义这些类，在一个.cpp 文件中实现。
2. 对于附属于较大规模类的一个很小规模的类， 可以写在那个大规模类的.h 和.cpp 里。

## 文件大小

1. 一个头文件中类定义最好不要超过5个。
2. 一个头文件中最好能控制在500行之内（不包括注释）。
3. 一个头文件中全局或类的接口/函数最好控制在100个之内。
4. 一个实现文件最好能控制在5000行之内（不包括注释）。

# 注释

除非极其简单，否则对关键信息应有注释说明。内容包括：功能、入口/出口参数，必要时还可有备注或

补充说明。

1. 注释用途：代码交流、代码阅读、代码review、新人培训、工作移交等。
2. 统一采用业界的文档标准：doxygen。
3. 代码进行了修改，注释需要同步修改。
4. 注释内容意思要与代码保持一致。

注释分为：块注释和行注释。

块注释：

/\*\*

\*@brief 块注释内容说明

\*/

行注释：变量注释一般为行注释。

//注释内容说明。

## 类注释

对每个类进行简单的注释说明。内容包括：类名称含义、类主要用途、作者、创建日期。

样例：

/\*\*

\* @class 设备窗口操作类

\* @brief 操作工通过该窗口进行设备查询、设备状态、设备配置等操作

\* @author 作者名称

\* @date 创建、修改日期

\*/

class RTX\_MOD CDialogDev : public QDialog, CRTDialog

## 函数注释

对每个函数/接口进行相应的注释说明。内容包括：函数含义、传入参数、传出参数、返回值、特殊说明（内存管理）等。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 关键字 | 关键字函数 | 说明 |
|  | @brief | 函数功能简要说明 |  |
|  | @remark | 特殊说明 |  |
|  | @param[in] | 传入参数 |  |
|  | @param[out] | 传出参数 |  |
|  | @param[in|out] | 传入/传出参数 |  |
|  | @return | 返回类型 |  |
|  | @retval | 返回值 |  |

1. 有传入参数、有传出参数、有返回值必须需要函数注释。
2. 代码行数小于10行的函数可以不需要函数注释。（不包括注释和空行）
3. 代码行数大于10行的行数应该需要函数注释。（不包括注释和空行）
4. 代码行数大于50行的函数必须进行函数注释。（不包括注释和空行）

样例：

/\*\*

\*@brief 通过规则id获取条码规则详情列表

\*@remark 条码规则的指针和内存共用

\*@param[in] 规则id

\*@retval 符合规则id的条码规则详情列表

\*/

CEQue<CRTBarcodeRuleDetail\*> GetDetail(int i32RuleID);

## 变量注释

对每个变量进行相应的注释说明。内容包括：变量名称含义、变量类型、变量长度、变量值范围、变量默认值等。

1. 注释内容相对简单的，放在变量名称后面。
2. 注释内容相对复杂的，放在变量名称上一行。

样例：

QString m\_strDevName; //设备名称

//设备类型：1为1500型号，2为208型号，3为212型号，4为410型号，默认值为2

quint32 m\_ui32DevType;

## 关键业务注释

对系统/项目中的关键业务进行相关的注释说明。内容包括：业务规则说明、处理细节、特殊说明、修改人、修改时间等。

样例：

//add by hls begin 20140723

//修改实施部qc311：质检表单内有质检项目存在空值时限制提交

//目的：减少车间管理难度，由人工检查质检数据有效性变为系统防错

//现有流程：1、终端质检时点击增加，没有输入数据直接确定时系统把【】值写到每个质检项中，

//建议改善如下：1、终端质检时点击增加，没有输入数据直接确定时限制操作，同时弹窗提示；

//2、在刷卡提交质检表单时，判断是否有空值的质检项目，有空值时限制提交并弹窗提示；

int irow = 0;

int icol = 0;

string strCellValue = "";

for (irow = 0;irow < pGrid->RTGetRowCount();irow++)

{

//列名称：项目名称、记录、记录.....、记录n、判定、修正值

//需要剔除开始的项目名称和结尾的判定与修正值

//RTGetColCount()会多返回一行，所有默认列字段需要减（除了判定和修正值）

for (icol=1; icol < pGrid->RTGetColCount()-3; icol++)

{

strCellValue = "";

strCellValue = pGrid->RTGetValue(irow,icol);

//有任一一个质检项目的记录值为空值都不能通过，需要弹窗提示

if (strCellValue.empty())

{

string str = "";

util::string\_format(str, "第[%d]行第[%d]列的质检项目的记录值的内容不能为空值，请重新填写", irow+1, icol+1);

PopMsg(str.c\_str());

return;

}

}

}

//add by hls end 20140723

## 关键算法注释

对系统/项目中的关键算法进行相关的注释说明。内容包括：算法功能说明、算法公式说明、特殊说明、修改人、修改时间等。

## 内嵌代码的注释

对于if、while、do 等其大括号内嵌代码块比较长(需拖动滚动条来看)或者多层嵌套时，在结尾的“}”后要加上其所对应的语句的注释说明。

例：

if (bIsOk == TRUE)

{

代码段

}// if ((bIsOk == TRUE)...

## 缺省值的注释

函数入口参数有缺省值时，应注释说明。

例:

BOOL MyFunction(

const char cszParam1 [],

BOOL bParam2 // = TRUE

);

或者：

BOOL MyFunction(

const char cszParam1[],

BOOL bParam2 /\* = TRUE \*/

);

## 代码段的注释

描述某个代码段的注释，可以给注释描述的代码段外围加上{}，帮助阅读。

例：

//代码段实现功能或意图描述

{

代码段

}

# 类

## 类定义

格式为：class [导入导出宏定义] 类名称 [:[public/protected/private] 接口类1名称][,[public/protected/private] 父类1名称]

1. 尽量采用单继承，慎用多继承。
2. 优先采用接口继承和抽象类继承。

样例：

/\*\*

\* @class 设备窗口操作类

\* @brief 操作工通过该窗口进行设备查询、设备状态、设备配置等操作

\*/

class RTX\_MOD CDialogDev : public QDialog, CRTDialog

## 类命名

1. 首字母为英文大写C。表示类的意思。
2. 采用大小写混合，不使用下划线。多个单词或单词缩写之间靠大写字母来区分。如：CDevCfgManager。
3. 一般采用形容词+名称组合。
4. 类名称长度最好能控制在20个字母之内，可以适当考虑英文缩写。
5. 相关模块、相似功能的类命名规则一致。
6. 优先采用行业内、公司内、部门内的业务术语。
7. 尽可能采用英文，避免采用拼音及拼音缩写。
8. 命名中不要使用a/b/c 1/2/3类似的序号来区分不同的类。

## 类组成

包括：默认构造函数、构造函数、析构函数、赋值函数、拷贝赋值函数、初始化函数、清空函数、虚函数、信号、槽函数、静态类变量、对象变量。

构造函数采用数据成员初始化序列，并按照数据成员变量在头文件中声明的先后顺序。

顺序为：从上到下依次为：public、protected、private。

样例：

/\*\*

\* @class 设备窗口操作类

\* @brief 操作工通过该窗口进行设备查询、设备状态、设备配置等操作

\*/

class RTX\_MOD CDialogDev : public QDialog, CRTDialog

{

Q\_OBJECT

public:

explicit CDialogDev(QObject \*parent = 0);

CDialogDev(quint32 ui32DevID, const QString& strDevCode);

~CDialogDev(void);

CDialogDev(const CDialogDev& src);

CDialogDev& operator(const CDialogDev& src);

void Init();

void Clear();

protected:

void InitToolbar();

void InitStatusbar();

private:

void ClearDevInfo();

void ClearDevParam();

signals:

sig\_ChangeDevCode(const QString& strDevCode);

sig\_ClearDevInfo(CDevInfo\* pinfo);

public slots:

slot\_EditDevInfo(CDevInfo\* pinfo);

slot\_SaveDevInfo(CDevInfo\* pinfo);

public:

QString m\_strDevName; //设备名称

//设备类型：1为1500型号，2为208型号，3为212型号，4为410型号，默认值为2(208型号)

quint32 m\_ui32DevType;

protected:

QPushButton\* m\_pbtnF1Query;

QPushButton\* m\_pbtnF2Save;

private:

QByteArray m\_baRecvedData;

quint64 m\_ui64RecvedDataSize;

};

#endif // DIALOGDEV\_H

## 类层次

层次：根类到叶子类的所有层级数量。

1. 类层次一般不超过7层。
2. 通用方法、通用方法过程、通用变量提升到基类。如f1~f10按钮。
3. 耦合度高的建议采用继承，耦合度低的建议采用组合。
4. 子类只处理自己的函数和变量。
5. 父类的虚函数，子类根据需要可以有选择的重载实现。

## 类扇目

扇目：基类所拥有的下级子类数量。

1. 扇目一般不超过10个。

## 类大小

1. 一个类定义中最好能控制在500行之内（不包括注释）。
2. 一个类定义中全局或类的接口/函数最好控制在100个之内。
3. 一个类实现最好能控制在5000行之内（不包括注释）。

# 函数

## 函数定义

1. 格式为：[static/virtual] 函数返回值 函数名称(参数列表);
2. 能区分纯虚函数、虚函数和非虚函数；

样例：

protected:

/\*\*

\* @brief 具体实现函数，实现窗口下方的按钮F1的点击事件，若子类窗口该按钮有具体内容则需要子类重载实现，无具体内容则不需要子类重载实现

\* @author hls

\* @date 2014-06-25

\* @param 无

\* @return 无

\*/

virtual void OnHandleF1Clicked(void);

public slots:

/\*\*

\* @brief 槽函数，处理窗口下方的按钮F1的点击事件

\* @author hls

\* @date 2014-06-25

\* @param 无

\* @return 无

\*/

void HandleF1Clicked(void);

## 函数命名

1. 采用大小写混合，不使用下划线。多个单词或单词缩写之间靠大写字母来区分。如：qint64 HandleRecvedData(char\* pdata)。
2. 函数首字母大写为公司函数，小写为qt函数。
3. 一般采用动词+名称组合。
4. 布尔型函数一般为：Is函数名称。如：IsRT208Dev()。
5. 函数名称长度最好能控制在20个字母之内，可以适当考虑英文缩写。
6. 相关模块、相似功能的函数命名规则一致。
7. 优先采用行业内、公司内、部门内的业务术语。
8. 尽可能采用英文，避免采用拼音及拼音缩写。
9. 命名中不要使用a/b/c 1/2/3类似的序号来区分不同的函数及参数列表中的参数名称。

# 变量

变量包括：全局变量、静态类变量、对象变量、参数变量、临时变量等。

1. 变量尽可能延后定义。
2. 变量定时时赋予默认值或错误值。
3. 变量能“望名知意”。
4. 注意变量的生命周期和作用域。
5. 变量一名一意，不要重名。

## 变量定义

1. 格式为：变量类型 变量名称; //变量注释
2. 5个变量一组；
3. 变量名称的首字母为列25；
4. 变量注释的首字母为列51；

样例：

int m\_i32TaskID; //任务id

string m\_strTaskCode; //任务编号

int m\_i32OrderID; //工单id

string m\_strOrderCode; //工单编号

int m\_i32ProcessID; //工序id

string m\_strProcessCode; //工序编号

int m\_i32MaterialID; //物料id

string m\_strMaterialCode; //物料编号

unsigned int m\_ui32RetryCount; //重试次数

//区分一个任务被数据终端多次申请部分加工数

unsigned int m\_ui32BeginSeq; //该工单的开始序号

unsigned int m\_ui32EndSeq; //该工单的结束序号

unsigned int m\_ui32CurSeq; //该工单的当前序号

int m\_i32LabelID; //条码标签id

int m\_i32RuleID; //条码规则id

## 变量命名

1. 格式为:[作用域类型\_]变量类型前缀变量名称[ = 默认值]。
2. 全局变量为：g\_i32DevCount;静态类变量为：sm\_i32DevCount;对象变量为：m\_i32DevCount;参数变量、临时变量为i32DevCount;
3. 采用大小写混合，不使用下划线。多个单词或单词缩写之间靠大写字母来区分。如：m\_i32DevCount。
4. 一般采用形容词+名称组合。
5. 布尔型变量一般为：bIs变量名称。如：m\_bIsRT208。
6. 变量名称长度最好能控制在20个字母之内，可以适当考虑英文缩写。
7. 相关模块、相似功能的变量命名规则一致。
8. 优先采用行业内、公司内、部门内的业务术语。
9. 尽可能采用英文，避免采用拼音及拼音缩写。
10. 命名中不要使用a/b/c 1/2/3类似的序号来区分不同的变量。

# 每行代码长度

每行代码的长度推荐为100 个ASCII 字符，最长不得超过150；折行以对齐为准。

例：

DWORD MyFunction(const char cszParam1[],

BOOL nParam2);

或者：

BOOL MyFunction(

DWORD dwParam1,

void \*pvParam2,

int nParam3,

int \*pnParam4

);

# 合并行的问题

循环、分支代码，判断条件与执行代码不得在同一行上。

例：

正确：

if (n == -2)

n = 1;

else

n = 2;

不得写做：

if (n == -2) n = 1;

else n = 2;

# 指针中\*号的位置

指针的定义，\* 号既可以在变量名之前，也可以紧接类型。

例：

可写做： int \*pnsize;

**也可写做： int\* pnsize;**

但不得写做： int \* pnsize;

# 全局函数的调用

在类的成员函数内调用全局函数时，在全局函数名前必须加上“::”。

# 关于 if...else if

else if 必须写在一行。

# 与“{”、“}”有关的各项规定

## “{”、“}”的使用

在“{”之前不允许有除空格与Tab 之外的其他任何字符；在它之后可有注释，但不允许有代码。与“{”对应的“}”必须在同一列上。

例：

正确：

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

{ // .....

printf("number %d:", i);

printf("%s\n", pBuffer[i]);

}

不得写做：

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

{ printf("number %d:", i);

printf("%s\n", pBuffer[i]);

}

也不得写做：

for (i = 0; i < count; i++){

printf("number %d:", i);

printf("%s\n", pBuffer[i]);

}

## “{”、“}”的省略

在循环、分支之后若只有一行代码，在无歧义的情况下可省略“{”、“}”,但是一般情况下，不建议省略。

例：

if (n == -2)

n = 1;

else

n = 2;

# 与空格有关的各项规定

## 运算符

在所有两目、三目运算符的两边都必须有空格。在单目运算符两端不必空格。但在“->”、“::”、“.”、“[”、“]”等运算符前后，及“&”（取地址）、“\*”（取值）等运算符之后不得有空格。

例：

正确：

int n = 0, nTemp;

for (int i = nMinLine; i <= nMaxLine; i++)

不得写做：

int n=0, nTemp;

for ( int i=nMinLine; i<=nMaxLine; i++ )

## 关键字

for、while、if 等关键字之后应有1 个空格，再接“(”。

例：

正确：

if (-2 == n)

不得写做：

if(-2 == n)

## 函数

调用函数、宏时，“(”前不得有空格。

例：

正确：

printf("%d\n", nIndex);

不得写做：

printf ("%d\n", nIndex);

## 类型转换

类型强制转换时，“(”“)”前后不得有空格

例：

可写做：

(char \*)pBuffer;

也可写做：

(char\*)pBuffer

不得写做：

( char \* )pBuffer

( char\* ) pBuffer

# 与缩进有关的各项规定

## TAB单位

缩进以Tab 为单位。要求在编辑器中将1 个Tab 设置为4 个空格宽度。

## 缩进

下列情况，代码缩进一个Tab:

### 函数体相对函数名及“{”、“}”。

例：

int MyFunction(int x)

{

return (x \* x);

}

### if、else、for、while、do 等之后的代码。

### 折行

一行之内写不下，折行之后的代码，应在合理的位置进行折行。

若有+ - \* / 等运算符，则运算符应在上一行末尾，而不应在下一行的行首。

## 不缩进

下列情况，不必缩进：switch 之后的case、default。

例：

switch (nID)

{

case BEGIN:

......

break;

case END:

......

break;

default:

......

break;

}

# 关于出错处理

### 断言

模块内部的编程和设计错误应该通过断言标记。

### 容错

变量使用之前，必须初始化。使用中，必须有相应的条件判断其合法性。

允许使用goto语句，但不能过度使用而导致出现资源泄漏的问题。

### 异常

减少不必要的异常，但是使用后，不能忽略抛出的异常，同时避免资源泄露。

* 构造和析构函数不能抛出异常

说明：如果构造和析构函数执行失败则无法安全地撤销和回滚，故这些函数不能向外抛出异常。为了降低复杂性，建议在这类函数中实现最简单的逻辑。

* 通过传值的方式抛出，通过引用的方式捕获

说明：抛出异常时，如果抛出指针，谁释放指针就成为问题。捕捉时如果是传值，会存在拷贝，拷贝可能不成功(比如异常是由于内存耗尽造成的)，而且拷贝得不到派生类对象，因为在拷贝时，派生类对象会被切片成为基类对象。

* 确保抛出的异常一定能被捕捉到

说明：异常未被捕捉到，系统的默认行为是终止程序运行，所以要确保程序产生的异常都能被捕捉。

* 确保异常发生后资源不泄漏

说明：异常发生后，当前代码执行序列被打断，需要查看分配的内存、文件和内核句柄等资源是否正确释放，避免资源泄漏，尤其每个可能的返回点是否正确释放资源。

* 独立编译模块或子系统的外部接口禁止抛异常

说明：异常处理没有普遍通用的二进制标准，所以不允许跨模块抛异常。

# switch

1. 命中率高的case放在switch前部；
2. 补全默认值 default: break;
3. 每个case使用大括号{};

样例：

switch ()

{

case :

break;

case :

break;

case :

break;

default:

break;

}

# 枚举

1. 枚举值：一种采用连续序号(0/1/2/3/4/5/6.....)，一种采用二进制位值(0/1/2/4/8/16/......),还有一种是不规则数字（如串口波特率）。
2. 枚举值格式：枚举类型变量 = 枚举类型值, //枚举类型说明
3. 枚举值最后标明：默认值、最大值、最小值。

样例：

//条码生成结果：

enum BARCODE\_CREATE\_RESULT

{

BCR\_FAILED = 0, //生成失败

BCR\_SUCCEED = 1, //生成成功

BCR\_NO\_DEV = 2, //无相关的设备id

BCR\_NO\_PRINTER = 3, //无相关的打印机id

BCR\_NO\_RULE = 4, //无相关的条码规则id

BCR\_NO\_ORDER = 5, //无相关的工单id

BCR\_NO\_TASK = 6, //无相关的任务id

BCR\_NO\_MATERIAL = 7, //无相关的物料id

BCR\_NO\_PROCESS = 8, //无相关的工序id

BCR\_NO\_LABEL = 9, //无相关的标签id

BCR\_NO\_DETAIL = 10, //无相关的条码规则详情id

BCR\_NO\_RULE\_DETAIL = 11, //无相关的条码详情

BCR\_OVER\_MAX\_SEQ = 12, //已超过最大条码序号

BCR\_FORMAT\_ERROR = 13, //数据终端发来的格式不对

//默认值、最小值和最大值

BCR\_DEFAULT = BCR\_FAILED,

BCR\_MIN = BCR\_FAILED,

BCR\_MAX = BCR\_FORMAT\_ERROR

};

# 二、QtQuick qml部分附加规范

# Import说明

1. 导入模块的顺序为：基础的模块在前面；扩展的模块在后面。

例如

import QtQuick 2.5

import QtQuick.Controls 1.4

import QtQuick.Dialogs 1.2

1. 导入模块别名：如果有两个模块的类名冲突了，那么其中的一个模块就要使用别名进行替换：

import QtQuick 2.5

import QtQuick.Controls 1.4 as BuiltIn

import QtQuick.Controls.Styles 1.4

BuiltIn.Label

{

color: "#FF650602"

}

3)如果想要导入的是一个文件夹，其中包含了qmldir文件，那么这样的文件夹应该放在后面。

import QtQuick 2.5

import QtQuick.Controls 1.4

import "../shared"

import "../style"

4) 如果想要导入javascript文件，那么此类import也要放在最后

import QtQuick 2.5

import QtQuick.Window 2.2

import Qt.labs.settings 1.0

import QtDream.Render 2.3

import "Misc.js" as Misc

# 对象声明

1）按照对象所在的类的继承关系对属性进行赋值

所有对象为了引用，都有ID属性

QtObject的属性有objectName

Item的属性比较多

Rectangle除了Item外，还有color等等

Rectangle

{

id: rect

objectName: "Test rectangle"

width: 100

height: 100

color: "yellow"

}

2）子对象声明

子对象声明要放在父对象属性赋值后，父对象信号处理器以及父对象自定义Javascript函数之前

3）对象信号处理器

和槽概念类似，它定义在子对象之后，父对象自定义Javascript函数之前

# 其他说明

1）一个QML代码应是以QML对象为主

2）如果要执行页面逻辑，那么请使用信号、信号处理器以及Javascript函数

3）如果要执行的是业务逻辑，那么使用C++的函数来操作

4）让复杂的Javascript函数尽量放在一个Javascript文件中，除非牵涉到较多QML的id以及需要QML定义的一个属性

5）少写new操作符，如果您想创建一个字符串，那么请使用var a = "";，如果您想创建一个对象，那么请使用var a = { }，如果您想创建一个数组，那么请使用var a = [ ];

6）使用var代替variant作为不定类型的变量注记。

7）回调函数如果没有闭包的操作，最好将回调函数单独写一个函数。

8）如果是类似ListView，ListModel和Component这样的，最好让它们处于同一个父对象下；

9）使用Javascript文件来描述常量；使用QML单例来替代全局变量