

# CommonEdit DynamicFunction

## CEDF 动态函数系统

### 目录

1. 需求背景.....	2
2. 使用 CEDF 功能：建立可用 CEDF 函数的类： .....	3
3. 使用 CEDF 功能：创建 CEDF 函数.....	4
4. 使用 CEDF 功能：CEDF 函数的返回值和参数.....	5
5. 使用 CEDF 功能：添加一个 CEDF 函数调用计划到 CEDF 函数管理队列.....	8
6. 使用 CEDF 功能：重现 CEDF 分发槽.....	9
7. 使用 CEDF 功能：套壳型返璞归真.....	11
8. 使用 CEDF 功能：CEDF 计划的针对性操作.....	12

## 1. 需求背景

倘若有一个程序，需要在每一个执行周期内执行一些过程，例如：

```
void cycle(void){
    Func1();
    Func2();
    Func3();
    Func4();
    ...
}
```

如果上述过程在 **cycle** 函数的全部执行情况内不变，则这是一种任何程序员都能应对的最基本情况。

但是如果我们考虑，这一周期过程需要在一定条件下多执行一些其他函数，

亦或是不执行某些函数，那么一般的解决方法可能如下：

```
void cycle(bool condition){
    Func1();
    Func2();
    Func3();
    if(condition){
        Func4();
    }
    ...
}
```

但是假如 **cycle** 内计划了数十余函数，亦或是条件表达式非常繁杂，那么 **cycle** 函数的声明将会是一个地狱般的过程。

为此，CommonEdit 提供 **DynamicFunction**，即 CEDF 函数，CEDF 函数可以被加入 CEDF 函数管理队列，称为一个 CEDF 函数调用计划。CEDF 函数调用计划可以被十分便捷的管理。

## 2. 使用 CEDF 功能：建立可用 CEDF 函数的类：

要使用 CEDF 功能，您应当为您的函数准备 CEDF 环境。

在需要使用 CEDF 函数的类的声明中加入 `CE` 宏关键字 `enableCEDynamicFunc`，并且在类的定义的第一行使用 `CE` 宏功能 `CE_DYNAMIC`

例如：

```
class CEQueueFuncTest : enableCEDynamicFunc
{
    CE_DYNAMIC
```

CEDF 的容器以及拓展功能基于 Qt 开发，因此使用 CEDF 必须同时使用 Qt。

宏关键字 `enableCEDynamicFunc` 会以 `public` 方式继承 `CEQueueFuncBase` 和 `QObject`。

宏功能 `CE_DYNAMIC` 会以 `private` 方式创建类 `CEQueueFunc` 的实例 `CEDynamicFuncQueue`（即 CEDF 函数管理队列），以 `public` 方式定义三个可用于调用 CEDF 函数的 Qt 槽函数 `doCEDynamicFunc()`，`doCEDynamicFuncFirst()`，`doCEDynamicFuncAt()`，并且同时会调用 Qt 宏功能 `Q_OBJECT`。

也就是说，当一个类是一个可用 CEDF 函数的类时，它也是一个可用 Qt 元对象编程（信号与槽机制等）的类。

### 3. 使用 CEDF 功能：创建 CEDF 函数

CEDF 函数的声明与普通函数声明并不一致。

普通函数声明为 返回值类型 函数名(参数)

CEDF 函数声明为 CEDF\_def(函数名)

上述声明使用了 CE 宏关键字 CEDF\_def，该宏会扩展到：

**public : CEDF\_ReturnStruct 函数名(CEDF\_ParaLists)**

也就是说，CEDF 函数均声明为 **public**，并且有统一的返回值类型

**CEDF\_ReturnStruct** 和统一的参数类型 **CEDF\_ParaLists**，这两个“类型”

本质上也是 CE 宏关键字，会进一步向下扩展，但无需了解。

CEDF 函数声明例如：

```
CEDF_def(Func2) {  
    qDebug() << "FUNC2";  
    CE_Return CEDF_ReturnVOID;  
};
```

#### 4. 使用 CEDF 功能：CEDF 函数的返回值和参数

在上面的例子中，我们注意到该函数的返回值写法用了两个 **CE** 宏关键字 **CE\_Return** **CE\_ReturnVOID**。从字面上，我们可以将其类比为 **return void**

上文已经提到，CEDF 函数的返回值类型是 **CEDF\_ReturnStruct**，该类型顾名思义是一个结构体。该结构体内有三个变量，第一个变量是一个枚举值，决定了该函数在执行完毕后在 CEDF 函数管理队列中如何处理，第二个变量是一个 **QStringList**，可以用来存放可以被转换成 **QString** 的一般变量，第三个变量是一个 **QList<void\*>**，可用来存放指针变量。

与此同时，传入 CEDF 函数的两个变量依次也为 **QStringList** 和 **QList<void\*>**，用于向 CEDF 函数提供参数。他们的对象名分别为 **CEDF\_NormalList** 和 **CEDF\_PointerList**。

现在您可能会问，什么是“该函数在执行完毕后在 CEDF 函数管理队列中如何处理”？，我们现在来回答这个问题。

在第二节中，我们指出了 **CE\_DYNAMIC** 宏会定义的三个用于调用 CEDF 函数的函数。其中 **doCEDynamicFunc()** 意味着，把 CEDF 函数管理队列中的每一个调用计划都遍历一遍，**doCEDynamicFuncFirst()** 意味着，把 CEDF 函数管理队列中的第一个调用计划执行一次，**doCEDynamicFuncAt(CEDynamicFunctionID)** 意味着，把具有 **CEDynamicFunctionID** 的执行计划执行一次。

在遍历执行中，我们可能会希望某函数搞一点“特例”，比如说在执行完一次该函数后，在满足一定条件的情况下再执行一遍，然后再去执行队列里的其

他函数。这个时候，我们就可以用如下返回值：

**CE\_Return CE\_ReturnREDO;**

此时 CEDF 函数管理队列会将当前 CEDF 函数执行计划重新执行。若

**CE\_ReturnREDO** 不在某判断下，则会造成死循环。

有的时候，我们希望一个执行计划在执行完毕之后就从队列中移除，以免下

次再被执行，那么我们就可以用如下返回值：

**CE\_Return CE\_ReturnREMOVE;**

那么在执行完该执行的计划之后，CEDF 函数管理队列会把提交了上述返回

值的执行计划移出队列。

有的时候我们可能不仅仅需要执行计划重新执行，还希望能够在重新执行之

前修改计划中传入 CEDF 函数的参数，这个时候 CEDF 函数的返回值才会真

正起效，我们称一个 CEDF 函数执行后将自己的返回值作为参数再执行一遍

的操作为“复写重启”，即使用如下返回值：

**CE\_Return CE\_ReturnCYCLE;**

此时函数会将自己的返回值提交给 CEDF 函数管理队列，CEDF 函数管理队

列会将该返回值覆盖原计划的参数，并且重新执行该计划。

由于 **CE\_ReturnCYCLE** 提交的返回值实质上就是调用时的参数

**CEDF\_NormalList** 和 **CEDF\_PointerList**，因此如果要使用该宏，应当

在函数内直接操作 **CEDF\_NormalList** 和 **CEDF\_PointerList**，或者在操

作完成后将结果同步到 **CEDF\_NormalList** 和 **CEDF\_PointerList**，供宏

正常工作。

例如：

```

CEDF_def(Func1) {
    qDebug() << CEDF_NormalList[0];
    CEDF_NormalList[0] = QString::number(CEDF_NormalList[0].toInt() - 1);
    if (CEDF_NormalList[0] != "0") {
        CE_Return CEDF_ReturnCYCLE;
    }
    else {
        CE_Return CEDF_ReturnREMOVE;
    }
};

```

有时候我们希望，CEDF 函数仅做复写，而不做重启，那么使用如下返回值：

CE\_Return CE\_ReturnEDIT;即可。

## 5. 使用 CEDF 功能：添加一个 CEDF 函数调用计划到 CEDF 函数管理队列

使用 CE 宏函数 `addCEDF(FuncName, QStringList, QList<void*>);` 可以提交一个 CEDF 函数调用计划到 CEDF 函数管理队列。

特别的，当传入 CEDF 函数的两个列表为空（即相当于传统函数的 `void`）时，可以使用 CE 宏关键字 `CEDF_BothVOID`，若只有一个列表为空，为空的列表可用 `CEDF_VOID` 代替，例如：

```
qDebug()<<addCEDF(Func1, { "30" }, CEDF_VOID);
qDebug()<<addCEDF(Func2, CEDF_BothVOID);
```

实际上，`addCEDF` 函数还会返回一个 `long long` 值（在实际应用中已经被 `typedef` 为 `CEDynamicFunctionID`），代表该 CEDF 函数执行计划在管理队列中的唯一 ID。

上图中通过 Qt 宏函数 `qDebug()` 输出了该值

这也就是说，

```
qDebug()<<addCEDF(Func1, { "30" }, CEDF_VOID);
qDebug()<<addCEDF(Func1, { "30" }, CEDF_VOID);
qDebug()<<addCEDF(Func2, CEDF_BothVOID);
```

实际上是三个 CEDF 调用计划，虽然第一个和第二个计划内容一样，但是其 `CEDynamicFunctionID` 并不一致：

```
164847292910026410
164847292910024530
164847292910002469
```



## 6. 使用 CEDF 功能：重现 CEDF 分发槽

说到底，总要有一个方法实现所谓的“动态”，即从文字函数名转换到实际函数的过程。由于 CE 并未准备预编译机制，因此这个过程，即 CEDF 分发槽，需要您自己实现。

CEDF 分发槽需要一个 CE 宏关键字定义：

**CEDF\_slot{槽体}**

在槽体内只需要一种函数，即 CE 宏函数 **ifIsCEDF(CEDF\_name)**；用该函数覆盖类内的所有 CEDF 函数的函数名，即可令 CEDF 函数管理队列正常执行 CEDF 函数执行计划。

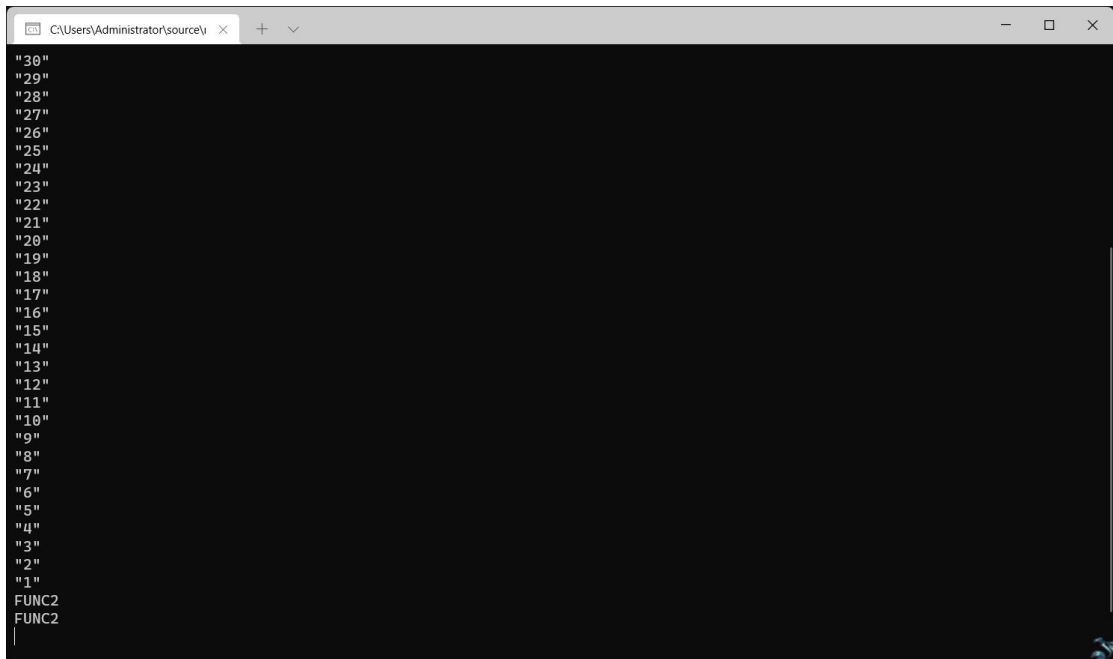
例如：

```
CEDF_slot{
    ifIsCEDF(Func1);
    ifIsCEDF(Func2);
}
```

若如上述例图：

```
qDebug()<<addCEDF(Func1, { "30" }, CEDF_VOID);
qDebug()<<addCEDF(Func2, CEDF_BothVOID);
```

提交了两个 CEDF 调用计划到 CEDF 函数管理队列，并调用两次 **doCEDynamicFunc()**之后，程序输出如下：



```
C:\Users\Administrator\source\ >
"30"
"29"
"28"
"27"
"26"
"25"
"24"
"23"
"22"
"21"
"20"
"19"
"18"
"17"
"16"
"15"
"14"
"13"
"12"
"11"
"10"
"9"
"8"
"7"
"6"
"5"
"4"
"3"
"2"
"1"
FUNC2
FUNC2
|
```

这是符合预期的，在第一次 `doCEDynamicFunc()`后进行的队列遍历时，  
**Func1** 如上文图片定义的那样进行了复写重启，并在满足条件后声明将本计划移出队列。在第二次 `doCEDynamicFunc()`后进行的队列遍历时，就只剩下一个调用了 **Func2** 的 **CEDF** 调用计划。

## 7. 使用 CEDF 功能：套壳型返璞归真

鉴于 CEDF 函数的返回值并不为 `void`，因此在 Qt 开发环境中会造成很大问题，因为 Qt 最常用的信号与槽机制中，要求槽函数返回值为 `void`。一般来说，如果真的需要 CEDF 函数成为 Qt 槽函数，那么我们大可手动调用函数并自行忽略返回值，也就是说给 CEDF 函数套一个 `void` 壳。

为了节省编码时间，CE 提供了给 CEDF 函数套壳为 Qt 槽函数用的 CE 宏函数 `CEDF_toQtSlot(CEDF, QtSlot)`；例如：

```
CE,
CEDF_toQtSlot(Func1,qFunc1);
```

该函数会给 CEDF 函数 `Func1` 加一层 `void` 函数的壳，新函数为 `qFunc1`，并且已经在宏内部声明为 Qt 槽函数。

## 8. 使用 CEDF 功能：CEDF 计划的针对性操作

我们上文已经提到，能够区分 CEDF 执行计划的唯一方式是使用 **CEDynamicFunctionID**，所以说，如果想要从 CEDF 函数管理队列的角度去删除一个调用计划，而不是从一个 CEDF 函数内部去删除调用计划，唯一方式是向队列提供 **CEDynamicFunctionID**。即使用 CE 宏函数

**removeCEDF(CEDFID)**；即可移除具有指定 **CEDFID** 的执行计划。

同理，若想针对性的仅执行某一个函数调用计划，唯一方式也是向队列提供 **CEDynamicFunctionID**，即使用函数 **doCEDynamicFuncAt(CEDFID)**，即可执行具有指定 **CEDFID** 的执行计划。