NULL&nullptr

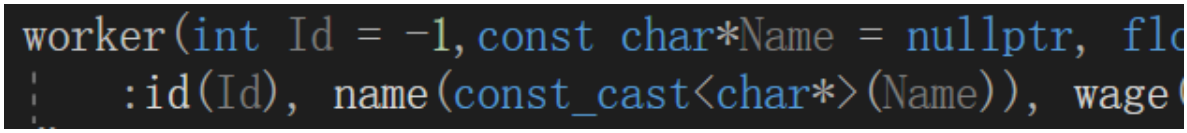
一、综述

在C++的指针初始化之中，我们通常会出现使用以下语法：

```
void* pt = NULL
```

但有时候，NULL在类的初始化中会出现一些奇怪的问题。比如：

即出现未定义标识符的情况。而假设我们改成 `nullptr`，就可以避免类似的报错。



显然的，IDE的高亮标识也出现了差异，紫色的表示宏定义的变量，此处应该是系统内部实现的就是如此，而蓝色就是保留字了。根据查到的一些文章，我们可以发现，一般情况下我们建议使用 `nullptr` 进行指针类型的初始化，本篇文章就来探讨一下其中的原因。

二、关于 NULL

C++中的NULL的宏定义就是0。本质上，`NULL,0` 是一样的。都是一个 int...仅此而已.....

我们在之前已经讨论过，一个指针的本质就是 unsigned int 以16进制的方式访问内存空间。这样，为我们使用 0 来进行定义 unsigned int 提供了可能性。不过虽然大家都是0，但是含义却并不相同。显然地，我们不能够使用 一个unsigned int 的 0 当作指针使用去访问内存空间。考虑如下代码：

代码：

```
unsigned int a = 0;
printf("%p\n",a);
```

报错：

```
C:\Users\dell\Desktop\in.cpp: In function 'int main()':
C:\Users\dell\Desktop\in.cpp:5:17: warning: format '%p' expects argument of type 'void*', but argument 2 has type 'unsigned int' [-Wformat=]
  printf("%p\n",a);
```

所以，关于 `NULL` 的定义。我们不能够简单的进行如下的定义：

```
#define NULL 0
```

而应该如此定义

```
#define NULL (void*)0
//这样在输出、编译时就不会出错
#define a (void*)0
printf("%p\n",a);
system("pause");
return 0;
//因为此时的a就只是指针而已
```

输出：

```
0000000000000000
请按任意键继续. . .
```

据说C++中的对于NULL的定义是这样的：

```
ifdef __cplusplus
define NULL 0
else
define NULL ((void *)0)
endif
```

这样，在NULL中就需要一种推导，关于 NULL应该被解释成为哪种类型。这很容易会出现二义性。因此我们使用nullptr就会好很多。

三、关于nullptr

```
const
class nullptr_t
{
public:
    template<class T>
    inline operator T*() const
        { return 0; }


    template<class C, class T>
    inline operator T C::*() const
        { return 0; }

private:
    void operator&() const;
} nullptr = {}
```

nullptr的实际类型就是上面的`std:nullptr_t`。可以看到，它不是整型\指针，却可以转换成为任意类型。