C++ explicit关键字详解

首先，C++中的explicit关键字只能用于修饰只有一个参数的类构造函数，它的作用是表明该构造函数是显示的，而非隐式的，跟它相对应的另一个关键字是implicit，意思是隐藏的,类构造函数默认情况下即声明为implicit(隐式)。

那么显示声明的构造函数和隐式声明的有什么区别呢？我们来看下面的例子：

```
class CxString  // 没有使用explicit关键字的类声明，即默认为隐式声明
{
public:
    char *_pstr;
    int _size;
    CxString(int size)
    {
        _size = size;                // string的预设大小
        _pstr = malloc(size + 1);    // 分配string的内存
        memset(_pstr, 0, size + 1);
    }
    CxString(const char *p)
    {
        int size = strlen(p);
        _pstr = malloc(size + 1);    // 分配string的内存
        strcpy(_pstr, p);            // 复制字符串
        _size = strlen(_pstr);
    }
    // 析构函数这里不讨论，省略...
};

    // 下面是调用：

    CxString string1(24);     // 这样是OK的，为CxString预分配24字节的大小的内存
    CxString string2 = 10;    // 这样是OK的，为CxString预分配10字节的大小的内存
    CxString string3;         // 这样是不行的，因为没有默认构造函数，错误为:"CxString": 没有合适的默认构造函数可用
    CxString string4("aaaa"); // 这样是OK的
    CxString string5 = "bbb"; // 这样也是OK的，调用的是CxString(const char *p)
    CxString string6 = 'c';   // 这样也是OK的，其实调用的是CxString(int size)，且size等于'c'的ascii码
    string1 = 2;              // 这样也是OK的，为CxString预分配2字节的大小的内存
    string2 = 3;              // 这样也是OK的，为CxString预分配3字节的大小的内存
    string3 = string1;        // 这样也是OK的，至少编译是没问题的，但是如果析构函数里用free释放_pstr内存指针的时候可能会报错，完整的代码必须重载运算符"="，并在其中处理内存释放
```

上面的代码中，"CxString string2 = 10;" 这句为什么是可以的呢？在C++中，如果的构造函数只有一个参数时，那么在编译的时候就会有一个缺省的转换操作:将该构造函数对应数据类型的数据转换为该类对象。也就是说 "CxString string2 = 10;" 这段代码，编译器自动将整型转换为CxString类对象，实际上等同于下面的操作：

```
CxString string2(10);
或
CxString temp(10);
CxString string2 = temp;
```

但是，上面的代码中的_size代表的是字符串内存分配的大小，那么调用的第二句 "CxString string2 = 10;" 和第六句 "CxString string6 = 'c';" 就显得不伦不类，而且容易让人疑惑。有什么办法阻止这种用法呢？答案就是使用explicit关键字。我们把上面的代码修改一下，如下：

```
class CxString  // 使用关键字explicit的类声明，显示转换
{
public:
    char *_pstr;
    int _size;
    explicit CxString(int size)
    {
        _size = size;
        // 代码同上，省略...
    }
    CxString(const char *p)
    {
        // 代码同上，省略...
    }
};

    // 下面是调用：

    CxString string1(24);     // 这样是OK的
    CxString string2 = 10;    // 这样是不行的，因为explicit关键字取消了隐式转换
```


< 2019年7月 >

日	一	二	三	四	五	六
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

导航

博客园
首页
新随笔
联系
订阅XML
管理

统计

随笔 - 115
文章 - 0
评论 - 4
引用 - 0

公告

昵称:矮油~
园龄:4年8个月
粉丝:18
关注:1
+加关注

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔档案

2019年4月 (1)
2018年12月 (1)
2018年11月 (3)
2018年10月 (1)
2018年9月 (1)
2018年7月 (28)
2018年6月 (2)
2018年5月 (1)
2018年1月 (3)
2017年12月 (2)
2017年10月 (2)
2017年9月 (1)
2017年6月 (3)
2017年4月 (2)
2016年10月 (3)
2016年8月 (3)
2016年1月 (2)
2015年11月 (2)
2015年9月 (1)
2015年8月 (1)
2015年6月 (1)
2015年5月 (3)
2015年4月 (2)
2015年3月 (15)
2015年2月 (5)
2015年1月 (7)
2014年11月 (8)
2014年10月 (11)

最新评论

1. Re:c++中vector与list的区别

STL 中的list 就是一 双向链表，可高效地进行插入删除元素。
list不支持随机访问。所以没有 at(pos)和operator[]。

--矮油~

2. Re:全数字锁相环(PLL)的原理简介以及verilog设计代码

上面的代码有一些小问题，用的时候需要注意

```
    CxString string3;          // 这样是不行的，因为没有默认构造函数
    CxString string4("aaaa"); // 这样是OK的
    CxString string5 = "bbb"; // 这样也是OK的，调用的是CxString(const char *p)
    CxString string6 = 'c';   // 这样是不行的，其实调用的是CxString(int size)，且
size等于'c'的ascii码，但explicit关键字取消了隐式转换
    string1 = 2;               // 这样也是不行的，因为取消了隐式转换
    string2 = 3;               // 这样也是不行的，因为取消了隐式转换
    string3 = string1;         // 这样也是不行的，因为取消了隐式转换，除非类实现操作
符"="的重载
```

explicit关键字的作用就是防止类构造函数的隐式自动转换.

上面也已经说过了，explicit关键字只对有一个参数的类构造函数有效，如果类构造函数参数大于或等于两个时，是不会产生隐式转换的，所以explicit关键字也就无效了．例如：

```
class CxString  // explicit关键字在类构造函数参数大于或等于两个时无效
{
public:
    char *_pstr;
    int _age;
    int _size;
    explicit CxString(int age, int size)
    {
        _age = age;
        _size = size;
        // 代码同上，省略...
    }
    CxString(const char *p)
    {
        // 代码同上，省略...
    }
};

    // 这个时候有没有explicit关键字都是一样的
```

但是，也有一个例外，就是当除了第一个参数以外的其他参数都有默认值的时候，explicit关键字依然有效，此时，当调用构造函数时只传入一个参数，等效于只有一个参数的类构造函数，例子如下：

```
class CxString  // 使用关键字explicit声明
{
public:
    int _age;
    int _size;
    explicit CxString(int age, int size = 0)
    {
        _age = age;
        _size = size;
        // 代码同上，省略...
    }
    CxString(const char *p)
    {
        // 代码同上，省略...
    }
};

    // 下面是调用：

    CxString string1(24);      // 这样是OK的
    CxString string2 = 10;     // 这样是不行的，因为explicit关键字取消了隐式转换
    CxString string3;          // 这样是不行的，因为没有默认构造函数
    string1 = 2;               // 这样也是不行的，因为取消了隐式转换
    string2 = 3;               // 这样也是不行的，因为取消了隐式转换
    string3 = string1;         // 这样也是不行的，因为取消了隐式转换，除非类实现操作
符"="的重载
```

以上即为C++ explicit关键字的详细介绍.

总结：

explicit关键字只需用于类内的单参数构造函数前面。由于无参数的构造函数和多参数的构造函数总是显示调用，这种情况在构造函数前加explicit无意义。

google的c++规范中提到explicit的优点是可以避免不合时宜的类型变换，缺点无。所以google约定所有单参数的构造函数都必须是显示的，只有极少数情况下拷贝构造函数可以不声明称explicit。例如作为其他类的透明包装器的类。

effective c++中说：被声明为explicit的构造函数通常比其non-explicit兄弟更受欢迎。因为它们禁止编译器执行非预期（往往也不

修改一下

--矮油~

3. Re:数组指针和指针数组

@garbageMan嗯。a表示首地址，a+1相当于a[1];&a表示指向这个数组的指针;&a+1指向以数组的大小为单位的下一个地址，比如int *a[10] 中a相当于a[0]相当于&a,a+1相当.....

--矮油~

4. Re:数组指针和指针数组

引用指针数组定义 int *p[n];[]优先级高，先与p结合成为一个数组，再由int*说明这是一个整型指针数组，它有n个指针类型的数组元素。这里执行p+1是错误的错...

--garbageMan

被期望）的类型转换。除非我有一个好理由允许构造函数被用于隐式类型转换，否则我会把它声明为explicit，鼓励大家遵循相同的政策。

矮油~
关注 - 1
粉丝 - 18
+加关注

« 上一篇：命名空间 extern的用法 static全局变量
» 下一篇：C++对象的内存分布和虚函数表

posted on 2018-07-12 14:21 矮油~ 阅读(18824) 评论(0) 编辑 收藏

相关博文：

· C++ explicit关键字详解
· C++ explicit关键字详解
· C++ explicit关键字详解
· C++ explicit关键字详解
· C++ explicit关键字详解

最新新闻：

· 银河系最耀眼“烟火表演”：超级耀斑或将威胁地球
· 禁令暂时解除 华为还要研发操作系统吗
· 科学家们发现黑洞并非总是由恒星残骸形成的
· 法国气温达 45.9 摄氏度，破历史记录
· 全球首艘混动游轮起航，从挪威驶往北极再到南极
» 更多新闻...