38 How to CREATE_INSTANTIATE OBJECTS in C++

怎么去实例化我们的类呢？一般我们有两种选择，这两种选择的区别是内存来自哪里，我们的对象实际上会创建在哪里。
当我们在C++中创建对象时，它需要占用一些内存。即使我们写了一个完全为空的类：没有类成员，什么都没有，那它也必须要占用最少一个字节的内存，但这并不常见，因为我们的类中有很多的成员，它们需要被存储在某个地方。应用程序会把内存分为两个主要部分：*heap and stack* **堆和栈**

在C++中，我们可以选择对象在哪里创建，在堆上还是在栈上，它们有不同的功能性差异。

比如说*stack*对象有一个自动的**生存周期**，它们的生存周期是由它声明的地方的*scope*（作用域）决定的，一旦超出了这个作用域，它的内存很快就会被释放掉。因为当这个作用域结束时，栈会弹出，放在这个栈上、这个作用域的所有东西都会被释放。

但*heap*不一样，堆是一个很大很神秘的地方，一旦你在堆上分配一个对象，你实际上已经在堆上创建了一个一直存在那里的对象。直到你决定：不需要它了，我想释放这个对象，你可以用那段内存做任何事。

1. 栈上创建

C++

```cpp
#include <iostream>
#include <string>
#define LOG(x) std::cout << x << std::endl;

using String = std::string;

class Entity
{
private:
    String m_Name;
public:
    Entity() : m_Name("Unknown"){}
    Entity(const String& name) : m_Name(name) {}

    const String& GetName() const { return m_Name; }
};


int main()
{
    Entity entity("Cherno");           // Cherno
    Entity entity1 = Entity("Cherno"); // Cherno
    LOG(entity.GetName());
    LOG(entity1.GetName());
    std::cin.get();
}
```

什么时候像这样在栈上创建对象？

几乎所有时候。如果你可以这样创建对象的话，那就这么来创建，这是基本规则。
因为在C++中这是初始化对象最快的方式和最受管控的方式。

现在让我们来讨论下：

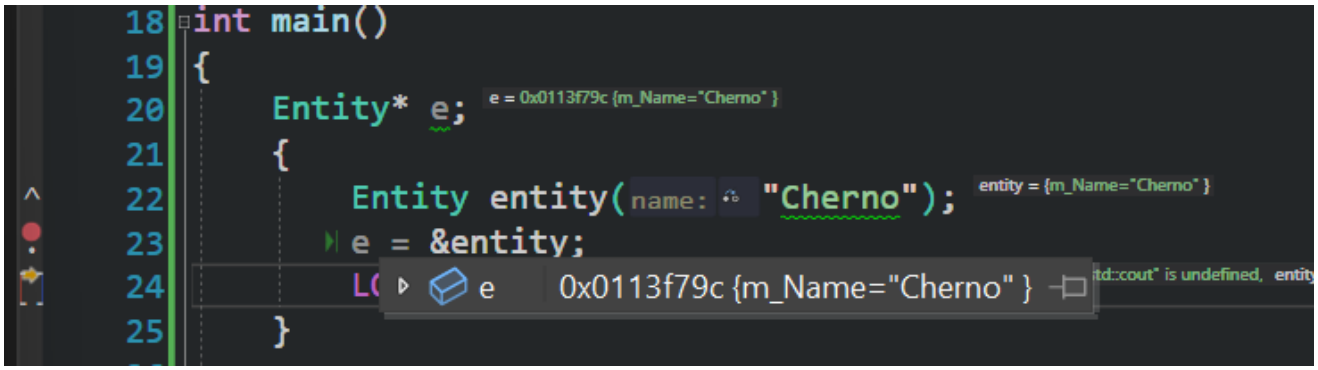
为什么在有些情况下你不能这么做：

如果你想让它在函数生存期之外也能存活。

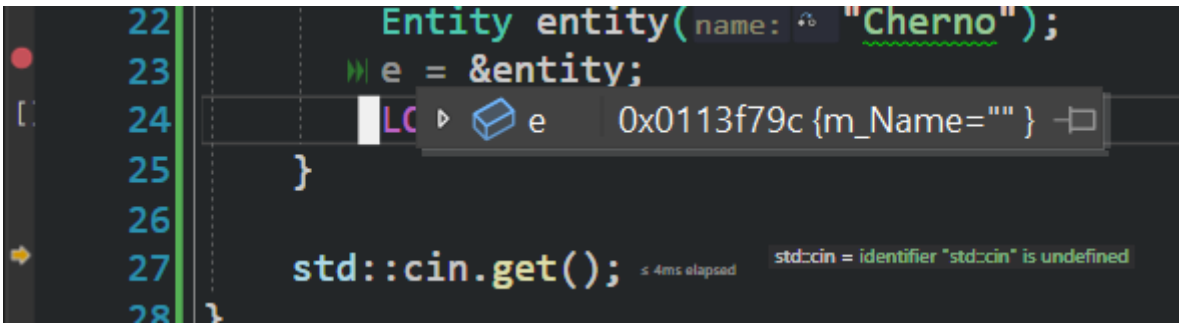
- 函数

```C++
void Function()
{
    Entity entity = Entity("Cherno");
}                       //一旦达到这个大括号，这个entity就会从内存中销毁
```

- 实例化



断点调试到离开大括号范围时，e还是指向相同的地址，但是m_Name已经没有了。



因为那个对象被释放掉了或者销毁了，这个叫Cherno的entity对象已经不存在了，它已经不存在栈结构里了。

另一个我们不想在栈上分配的原因可能是：如果这个entity太大了，同时我们可能有很多的entity，我们就可能没有足够的空间来进行分配，因为栈通常都很小，一般是一两兆，这取决于你的平台和编译器。
因此你可能不得不在*heap*上进行分配。

2. 堆上分配

```C++
Entity* entity = new Entity("Cherno");
```

这里最大的区别不是那个指针，而是这个*new*关键字。
我们调用`new Entity`时，实际发生的就是我们在堆上分配了内存，我们调用了构造函数，然后这个`new Entity`实际上会返回一个`Entity*`(指针)。它返回了这个entity在堆上分配的内存地址，这就是为什么我们要声明成`Entity*`类型。

不过不应该到处使用*new*，简单来说就是因为性能的原因，在堆上分配要比在栈上花费更长的时间，而且如果在堆上分配的话，那你**必须手动释放分配的内存**

```C++
delete entity;
```

用*new*就得*delete*，这是C++的运行方式。

```C++
int main()
{
    Entity* e;
    {
        Entity* entity = new Entity("Cherno");
        e = entity;                                 //entity已经是指针了，删除&
        LOG((*entity).GetName());                   // 解引用
        /* 或者 */
        LOG(entity->GetName());                     // →箭头运算符

    }

    std::cin.get();
    delete e;
}
```

这里要注意的是，`entity` 是在内部作用域中创建的局部变量，其生命周期仅限于该作用域。当程序执行完离开该作用域时，`entity` 指针将超出其作用域，并且无法再访问到该指针。如果尝试在离开作用域后使用 `delete entity`，将导致未定义的行为，因为 `entity` 指针已经无效。

相反，`e` 是在外部作用域中定义的指针，并且将 `entity` 的值赋给了 `e`。因此，在离开内部作用域后，`e` 仍然保留了对动态分配的 `Entity` 对象的指向。因此，使用 `delete e` 可以正确地释放该对象的内存。

所有如果对象非常大，或者你想显示地控制对象的生存周期，那就在堆上创建。
如果不是这两种情况的话，那就在栈上分配，这更简单，而且会自动回收也更快。
堆上分配要手动*delete*，如果忘记调用会导致内存泄漏。

后面会使用*smart pointers*(智能指针)，让我们实际上仍然可以在堆上进行分配，而且仍然可以获得那种大小优势，还有就是当指针超出作用域时，对象会被自动删除。