2.7 拷贝构造函数_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 2.7 拷贝构造函数涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

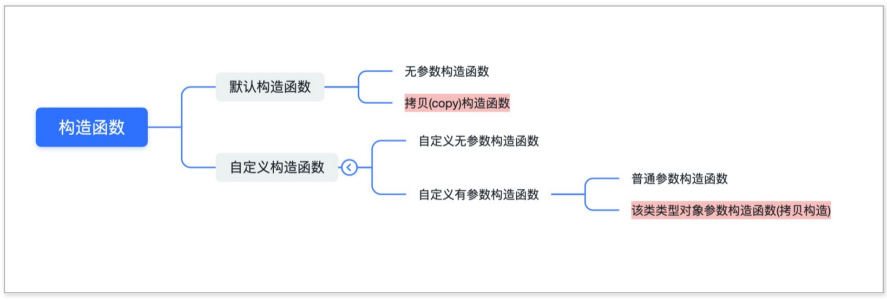
7. 拷贝构造函数

程序运行结果如下:

```
linux@ubuntu:~/CPP_Class/lesson4$ ./a.out
String(const char *str)
h,ascii code:104
e,ascii code:101
l,ascii code:108
l,ascii code:108
o,ascii code:111
--------------------------
h,ascii code:104
e,ascii code:101
l,ascii code:108
l,ascii code:108
o,ascii code:111
~String()
~String()
*** Error in `./a.out': double free or corruption (fasttop): 0x09c3e008 ***
Aborted (core dumped)
```

思考:

- 创建两个对象只调用了一次构造函数?
- 为什么出现了 double free 错误?



默认构造函数编译器自动提供，如果我们自己手动编写了构造函数，编译器就不会在提供了。

拷贝构造函数是一种特殊的构造函数，它在创建对象时，是使用同一类中之前创建的对象来初始化新创建的对象。

类名 (const 类名 & 引用)

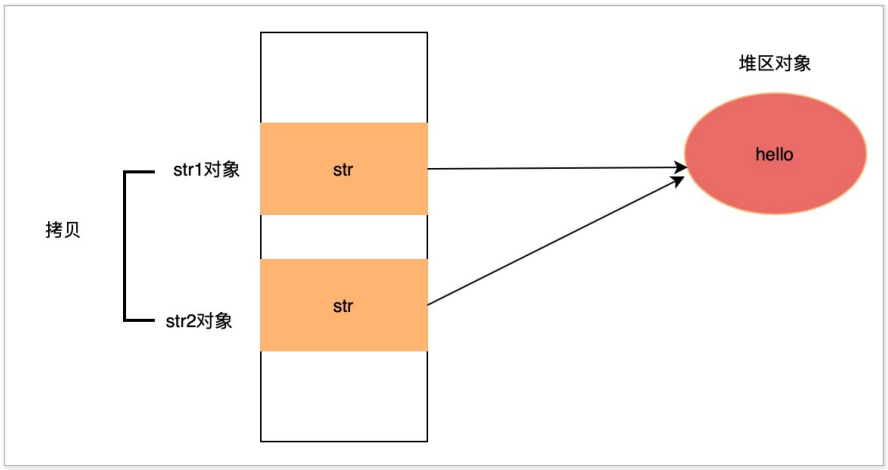
```
class String{


public:
        String(const char *str = NULL);
        String(const String &other);
        ~String(void);
        void show(void);
private:
        char *str;
};
```

拷贝构造函数的参数部分必须是一个引用，不能是一个对象

```
String::String(const String &other)
{
     this->str = other.str;
}
```

- 浅拷贝只是做对象数据成员的直接赋值，编译器默认提供的拷贝构造就是浅拷贝的实现方式
- 浅拷贝容易导致两个对象共享一个同一个资源的情况，例如: 在 String 类中如果使用浅拷贝的形式，就会导致两个对象的 str 指向同一个堆区内存，在释放的时候会出现 double free 的 bug




```
String::String(const String &other)
{
    if(!other.str){
           this->str = other.str;
    }else{
          int len = strlen(other.str) + 1;
          this->str = new char[len];
          strcpy(this->str,other.str);
```

```
    }
}
```

深拷贝的采用了重新分配资源的方式，让对象相互之间独立性更好

一个对象的创建需要通过另一个对象来初始化 **, 体现在以下三个场景:**

```
String str1("Hello");
String str2 = str1;

void function(String object);
void function(String &object);
void function(String *pobject);

String function(void)
{
    String object("hello");
    return object;
}
```

注意: 有些时候 g++ 编译器会优化代码，不创建临时对象，看不到调用拷贝构造函数的现象。可以在编译时候加上 **-fno-elide-constructors 编译参数 **，让编译器不要做此优化。

```
-fno-elide-constructors
The C++ standard allows an implementation to omit creating a temporary that
is only used to initialize another object of the same type.  Specifying this
option disables that optimization,and forces G++ to call the copy constructor
in all cases.
```

找出如下代码的错误，编译代码的时候使用 **-fno-elide-constructors 编译参数, 分析构造函数 (普通构造和拷贝构造) 和析构函数调用的次数 **

```
#include <iostream>

using namespace std;

class Test{
public:
    Test(int size){
        cout << "Test(int size)" << endl;
        data = new int[size];
    }
    Test(const Test obj){
        cout << "Test(const Test obj)" << endl;
        *this = obj;
    }
```

```
    ~Test(void){
        cout << "~Test()" << endl;
        delete data;
    }

private:
    int *data;
};

Test function(Test obj)
{
    Test tmp = obj;
    return tmp;
}

int main(void)
{
    Test obj1(3);
    Test obj2 = function(obj1);

    return 0;
}
```

全文完