

- 1、新增 ubuntu 下的 C++ 获取系统时间和使用 fstream （文件流）将数据输出到文件中。 --
-2022.0325
- 2、更新 深蓝学院 chapter1 《C++初探》 的关键点、代码、ppt及注释。 --- 2022.0502
- 3、更新 深蓝学院 chapter2 《对象与基本类型》 的关键点、代码、ppt及注释， 并提交了作业。 -----
-2022.0509
- 4、更新 深蓝学院 chapter3 《数组 vector 与字符串》 的关键点、代码、ppt及注释， 这一章节没有作
业 -----2022.0513
- 5、更新 语句、循环部分， 其中 **达夫设备** 这个暴力求解的优化方法挺有趣的。 ----- 2022.07.03
- 6、更新函数的参数、函数体、返回值和重载与重载的解析。 ----- 2022.07.05
- 5、更新委托构造函数的用法，减少代码冗余。

1.简介

委托构造函数（Delegating Constructor）由C++11引入，是对C++构造函数的改进，允许构造函数通过初始化列表调用同一个类的其他构造函数，目的是简化构造函数的书写，提高代码的可维护性，避免代码冗余膨胀。

通俗来讲，一个委托构造函数使用它所属的类的其他构造函数执行自己的初始化过程，或者说它把自己的一些（或者全部）职责委托给了其他构造函数。和其他构造函数一样，一个委托构造函数也有一个成员初始化列表和一个函数体，成员初始化列表只能包含一个其它构造函数，不能再包含其它成员变量的初始化，且参数列表必须与构造函数匹配。

首先看一下一个不使用委托构造函数造成代码冗余的例子。

```
class Foo
{
public:
    Foo() :type(4), name('x') {initRest()
    Foo(int i) : type(i), name('x') {ini
    Foo(char c) :type(4), name(c) {initR
```

```
private:  
    void initRest() /* init othre membe  
    int type;  
    char name;  
    //...  
};
```

从上面的代码片段可以看出，类Foo的三个构造函数除了参数不同，初始化列表、函数体基本相同，其代码存在着很多重复。在C++11中，我们可以使用委托构造函数来减少代码重复，精简构造函数。

```
class Foo  
{  
public:  
    Foo() {initRest();}  
    Foo(int i) : Foo() {type = i;}  
    Foo(char e) : Foo() {name = e;}  
private:  
    void initRest() { /* init othre memb  
    int type{1};  
    char name{'a'};  
};
```

一个委托构造函数想要委托另一个构造函数，那么被委托的构造函数应该包含较大量参数，初始化较多的成员变量。而且在委托其他构造函数后，不能再进行成员列表初始化，而

只能在函数体内对其他成员变量进行赋值。

2. 注意事项

(1) 不要形成委托坏。在构造函数较多的时候，我们可能拥有多个委托构造函数，而一些目标构造函数很可能也是委托构造函数，这样依赖，我们就可能在委托构造函数中形成链状的委托构造关系，形成委托坏（Delegation Cycle）。

```
class Foo
{
public:
    Foo(int i) : Foo('c') { type = i; }
    Foo(char c) : Foo(1) { name = c; }
private:
    int type;
    char name;
};
```

其中Foo(int i)与Foo(char c)相互委托就形成了委托坏，这样会导致编译错误。

(2) 如果在委托构造函数中使用try，可以捕获目标构造函数中抛出的异常。

```
#include <iostream>
using namespace std;

class Foo
{
public:
    Foo(int i) try: Foo(i,'c')
    {
        cout<<"start assignment"<<endl;
        type = i;
    }
    catch(...)
    {
        cout<<"caught exception"<<endl;
    }

private:
    Foo(int i,char c)
    {
        cout<<"throw exception"<<endl;
        throw 0;
    }
    int type;
    char name;

};

int main()
{
    Foo f(1);
    return 0;
}
```

程序输出结果：

```
throw exception  
caught exception
```

可见在目标构造函数Foo(int i,char c)中抛出异常，在委托构造函数Foo(int i)中可以进行捕获，并且目标构造函数体内的代码并不会被执行。这样的设计是合理的，因为目标构造函数抛出异常说明对象并没有完成初始化，在委托构造函数中进行赋值操作都是一些无意义的动作。

更多文章 · [C++11的智能指针和自动内存管理](#)