第12章 委托构造函数

《现代C++语言核心特性解析》 谢丙堃

冗余的构造函数

- 重复的成员初始化代码

```
class X
{
public:
    X() : a_(0), b_(0.) { CommonInit(); }
    X(int a) : a_(a), b_(0.) { CommonInit(); }
    X(double b) : a_(0), b_(b) { CommonInit(); }
    X(int a, double b) : a_(a), b_(b) { CommonInit(); }
private:
    void CommonInit() {}
    int a_;
    double b_;
};
```

冗余的构造函数

- “简洁”的构造方法

```
class X2 {
public:
    X2() { CommonInit(0, 0.); }
    X2(int a) { CommonInit(a, 0.); }
    X2(double b) { CommonInit(0, b); }
    X2(int a, double b) { CommonInit(a, b); }
private:
    void CommonInit(int a, double b) {
        a_ = a;
        b_ = b;
        c_ = "hello world";
    }
    int a_;
    double b_;
    std::string c_;
};
```

使用委托构造函数

- 基本语法

```
class X
{
public:
    X() : X(0, 0.) {}
    X(int a) : X(a, 0.) {}
    X(double b) : X(0, b) {}
    X(int a, double b) : a_(a), b_(b) { CommonInit(); }
private:
    void CommonInit() {}
    int a_;
    double b_;
};
```

使用委托构造函数

- 注意事项
  1. 每个构造函数都可以委托另一个构造函数为代理
  2. 不要递归循环委托
  3. 委托构造函数的执行顺序是先执行代理构造函数的初始化列表，然后执行代理构造函数的主体，最后执行委托构造函数的主体
  4. 如果在代理构造函数执行完成后，委托构造函数主体抛出了异常，则自动调用该类型的析构函数
  5. 如果一个构造函数为委托构造函数，那么其初始化列表里就不能对数据成员和基类进行初始化

使用委托构造函数

- 注意事项5举例：

```
class X
{
public:
    X() : a_(0), b_(0) { CommonInit(); }
    X(int a) : X(), a_(a) {}    // 编译错误
    X(double b) : X(), b_(b) {} // 编译错误
private:
    void CommonInit() {}
    int a_;
    double b_;
};
```

委托模板构造函数

```
class X {
    template<class T> X(T first, T last) : l_(first, last) { }
    std::list<int> l_;
public:
    X(std::vector<short>&);
    X(std::deque<int>&);
};
X::X(std::vector<short>& v) : X(v.begin(), v.end()) { }
X::X(std::deque<int>& v) : X(v.begin(), v.end()) { }

int main() {
    std::vector<short> a{ 1,2,3,4,5 };
    std::deque<int> b{ 1,2,3,4,5 };
    X x1(a);
    X x2(b);
}
```

值得注意的用法

- 捕获委托构造函数的异常
  - 使用Function-try-block去捕获委托构造函数异常

- 委托参数较少的构造函数

```
basic_fstream::basic_fstream(const char* s, ios_base::openmode mode)
    : basic_fstream()
{
    if (open(s, mode) == 0)
        setstate(failbit);
}
```

感谢聆听
欢迎关注