

Branch: master ▾

Find file

Copy path

[hello-world](#) / [C++远征之封装篇](#) / [第五章-const再现江湖](#) / [常对象成员和常成员函数](#) / [常对象成员和常成员函数.md](#) YHim 33

9b40e0d 13 days ago

1 contributor

63 lines (37 sloc) 2.21 KB

常对象成员和常成员函数

例子：

```
class Coordinate
{
public:
    Coordinate(int x, int y);
private:
    const int m_iX;
    const int m_iY;
};
```



这是一个坐标类，定义的两个数据成员都用const来修饰。

```
Coordinate::Coordinate(int x, int y)
```

```
{  
    m_iX = x;  
    m_iY = y;  
}
```

✗

```
Coordinate::Coordinate(int x, int y):m_iX(x),m_iY(y)
```

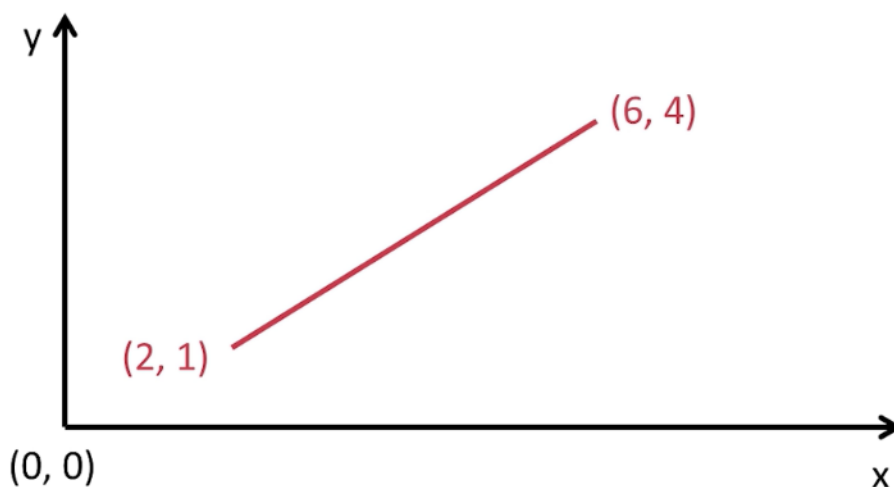
```
{  
}  
}
```

✓



上面的初始化方法是错误的，必须通过下面的这种初始化列表的方式才能成功。如果对象作为数据成员，const能否修饰呢？当然可以。我们把这种数据成员称为**常对象数据成员**。

为方便理解，以线段为例：



有如图的线段，当线段的位置被确定后，就不允许更改了(即一旦起点(2,1)和终点(6,4)被确定后)。要想达到这个目的，就要将代码写成下面的样子：

```
class Line
{
public:
    Line(int x1, int y1, int x2, int y2);
private:
    const Coordinate m_coorA;
    const Coordinate m_coorB;
};
```



这是一个线段的类，它有两个对象成员，一个A点，一个B点。因为要实现一旦初始化A、B两个点就不能再修改，所以就将两个点定义为const，即**常对象**。想要初始化这两个点就要使用初始化列表的方式：

```
Line::Line(int x1, int y1, int x2, int y2):
    m_coorA(x1, y1), m_coorB(x2, y2)
{
    cout << "Line" << endl;
}
```



调用：

```
int main(void)
{
    Line *p = new Line(2, 1, 6, 4);

    delete p;
    p = NULL;
    return 0;
}
```



用const修饰的成员函数就称为**常成员函数**。写法：

```
class Coordinate
{
public:
    Coordinate(int x, int y);
    void changeX() const; ← 常成员函数
    void changeX();
private:
    int m_iX;
    int m_iY;
};
```



定义成员函数：

```
void Coordinate::changeX() const
{
    m_iX = 10; ✗
};
```

```
void Coordinate::changeX()
{
    m_iX = 20; ✓
};
```



为什么在常成员函数中不能修改数据成员的值呢？当定义changeX()这个成员函数的时候，看上去这个成员函数貌似没有任何的参数，而实际上却隐含着一个参数(this指针)。

```
void Coordinate::changeX()
{
    m_iX = 10;
};
```



```
void changeX(Coordinate *this)
{
    this-> m_iX = 10;
};
```



当定义一个常成员函数时，编译器就会理解成这个样子：

```
void Coordinate::changeX() const
{
    m_iX = 10;
};
```

```
void changeX(const Coordinate *this)
{
    this-> m_iX = 10; ✗
};
```



它的参数中隐含有一个this指针，但这个this指针是用const修饰的。显然，这个this指针已经变成了一个常指针，通过常指针去改变该指针指向的数据，是不被允许的。所以，在常成员函数中修改数据成员的值就是错误的。

```
class Coordinate
{
public:
    Coordinate(int x, int y);
    void changeX() const;
    void changeX();
private:
    int m_iX;
    int m_iY;
};
```

} 互为重载



这两个changeX()互为重载。但这种写法不推荐。

```
int main(void)
{
    Coordinate coordinate(3, 5);
    coordinate.changeX();
    return 0;
}
```

调用的是哪个
成员函数呢？

慕课网

这样就可能会疑惑。这里调用的是不带const的成员函数。那么，什么情况下可以调用常成员函数呢？必须要写成这样：

```
int main(void)
{
    const Coordinate coordinate(3, 5);
    coordinate.changeX();
    return 0;
}
```

常对象

调用的是常
成员函数

慕课网

在实例化对象时，要用const来修饰这个对象。也把这样实例化出来的对象称之为常对象。

Branch: master

Find fileCopy path

hello-world / C++远征之封装篇 / 第五章-const再现江湖 / 常对象成员和常成员函数 / 常对象成员和常成员函数-例子.md

YHim 22

e4b0b35 2 days ago

1 contributor

165 lines (140 sloc)4.14 KB

常对象成员和常成员函数例子

要求如下：

```
/*
 * 常对象成员、常函数、常对象
 * 要求：
 *     定义两个类：
 *         坐标类：Coordinate
 *         数据成员：横坐标m_iX，纵坐标m_iY
 *         成员函数：构造函数、析构函数、数据封装函数
 *         线段类：Line
 *         数据成员：点A m_coorA，点B m_coorB
 *         成员函数：构造函数、析构函数、数据封装函数、信息打印函数
 */
/*
 *
 */
/*
 *
 */
*/
```

Coordinate.h

```
class Coordinate
{
public:
    Coordinate(int x, int y);
    ~Coordinate();
    void setX(int x);
    int getX() const;
    void setY(int y);
    int getY() const;

private:
    int m_iX;
    int m_iY;
};
```

Coordinate.cpp

```
#include <iostream>
#include "Coordinate.h"
using namespace std;

Coordinate::Coordinate(int x, int y)
{
    m_iX = x;
    m_iY = y;
    cout << "Coordinate() " << m_iX << "," << m_iY << endl;
}

Coordinate::~~Coordinate()
{
    cout << "~Coordinate() " << m_iX << "," << m_iY << endl;
}

void Coordinate::setX(int x)
{
    m_iX = x;
}
```

https://github.com/YHim/hello-world/blob/master/C%2B%2B%E8%BF%9C%E5%BE%81%E4%B9%8B%E5%B0%81%E8%A3%85%E7%AF%87/...

1/4


```
}

int Coordinate::getX() const
{
    return m_iX;
}

void Coordinate::setY(int y)
{
    m_iY = y;
}

int Coordinate::getY() const
{
    return m_iY;
}
```

Line.h

```
#include "Coordinate.h"

class Line
{
public:
    Line(int x1, int y1, int x2, int y2);
    ~Line();
    void setA(int x, int y);
    void setB(int x, int y);
    void printInfo();
    void printInfo() const;
private:
    const Coordinate m_coorA; //const既可以写在类名前面,
    Coordinate const m_coorB; //也可以写在类名后面。
};
```

Line.cpp

```
#include <iostream>
#include "Line.h"
using namespace std;

Line::Line(int x1, int y1, int x2, int y2):m_coorA(x1,y1),m_coorB(x2,y2)
{
    cout << "Line(int x1, int y1, int x2, int y2)" << endl;
}

Line::~Line()
{
    cout << "~Line()" << endl;
}

void Line::setA(int x, int y)
{
    //m_coorA.setX(x);
    //m_coorA.setY(y);
}

void Line::setB(int x, int y)
{
    //m_coorB.setX(x);
    //m_coorB.setY(y);
}

void Line::printInfo()
{
    cout << "printInfo()" << endl;
    cout << "(" << m_coorA.getX() << "," << m_coorA.getY() << ")" << endl;
    cout << "(" << m_coorB.getX() << "," << m_coorB.getY() << ")" << endl;
}

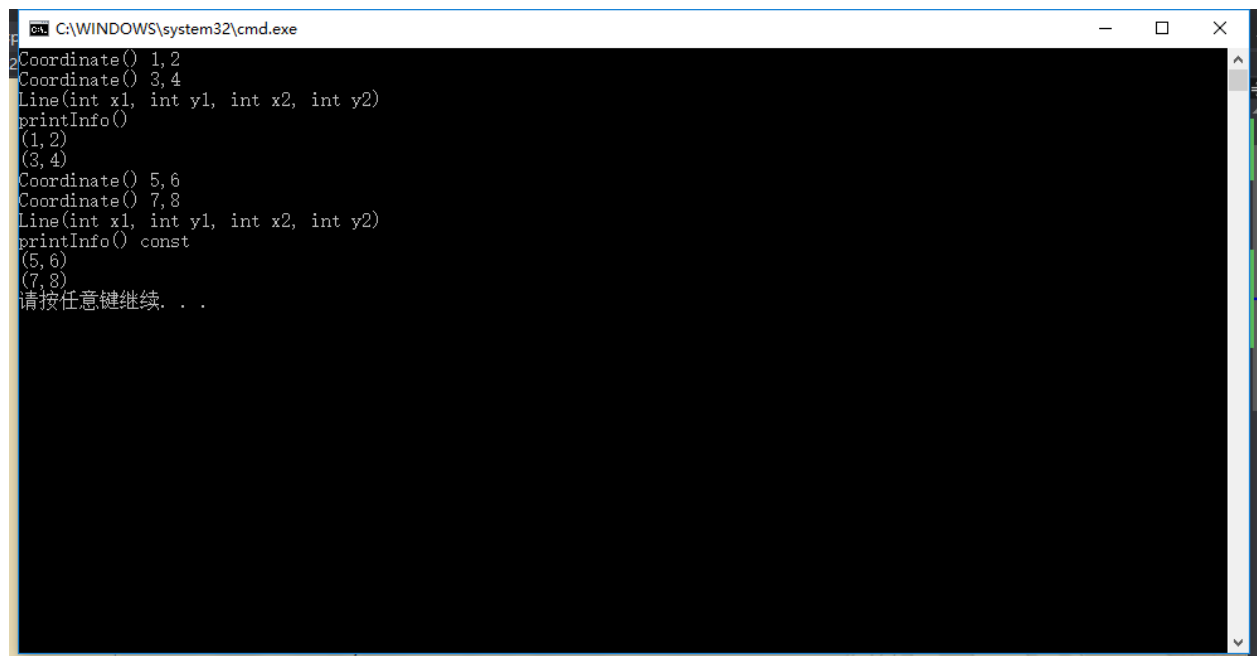
void Line::printInfo() const
{
    cout << "printInfo() const" << endl;
    cout << "(" << m_coorA.getX() << "," << m_coorA.getY() << ")" << endl;
}
```

```
cout << "(" << m_coorB.getX() << "," << m_coorB.getY() << ")" << endl;  
}
```

demo.cpp

```
#include <iostream>  
#include <stdlib.h>  
#include "Line.h"  
using namespace std;  
  
int main()  
{  
    Line line(1, 2, 3, 4);  
    line.printInfo();//调用的是普通成员函数  
  
    const Line line2(5,6,7,8);  
    line2.printInfo();//调用的是常成员函数  
  
    system("pause");  
    return 0;  
}
```

运行结果：



总结：

1. 常对象只能调用常成员函数。
2. 普通对象可以调用全部成员函数。
3. 当对一个对象调用成员函数时，编译程序先将对象的地址赋给this指针，然后调用成员函数，每次成员函数存取数据成员时，由隐含使用this指针。
4. 当一个成员函数被调用时，自动向它传递一个隐含的参数，该参数是一个指向这个成员函数所在的对象的指针。
5. 在C++中，this指针被隐含地声明为：X const this,这意味着不能给this 指针赋值；在X类的const成员函数中，this指针的类型为：const X const,这说明this指针所指向的这种对象是不可修改的（即不能对这种对象的数据成员进行赋值操作）；
6. 由于this并不是一个常规变量，所以，不能取得this的地址。
由于a是const对象，所以a只能调用类A中的常成员函数。那么为什么会提示：“不能将this指针.....”的语句呢 因为对于c++的成员函数（当然不是静态成员函数）都会含有一个隐藏的参数，对于上例A中的int GetValue()函数，在编译后会变成：int GetValue(A * const this); //不能修改this变量，但可以修改this指向的内容，即：this是常量指针。而对于int GetValue()const，编译后是：int GetValue(const A* const this); 所以this指针是const类型，从编译后的结果看就很清楚了，因为a是const，所以其this指针就对应：const A* const this; 而print函数被编译出来后对应的是void print(A* const this); 在进行参数匹配时，所以就会提示“不能将this指针从const A ..” this指针的出现就解释了，用哪一个对象的数据成员。通常情况下，this指针是隐含存在的，也可以将其显示的表示出来（即如上例中的 this->mValue。不过this指针只能在类中使用）还有就是 this指针是一个const指针；

