C++中 this 指针的用法

this 指针只能在一个类的成员函数中调用，它表示当前对象的地址。下面是一个例子：

```
void Date::setMonth( int mn )
{
  month = mn; // 这三句是等价的
  this->month = mn;
  (*this).month = mn;
}
```

1. this 只能在成员函数中使用。

全局函数，静态函数都不能使用 this。

实际上，成员函数默认第一个参数为 T* const register this。

如：

class A{public: int func(int p){}};

其中，func 的原型在编译器看来应该是： int func(A* const register this, int p);

2. 由此可见，this 在成员函数的开始前构造的，在成员的结束后清除。

这个生命周期同任一个函数的参数是一样的，没有任何区别。

当调用一个类的成员函数时，编译器将类的指针作为函数的 this 参数传递进去。如：

A a;

a.func(10);

此处，编译器将会编译成： A::func(&a, 10);

嗯，看起来和静态函数没差别，对吗？不过，区别还是有的。编译器通常会对 this 指针做一些优化的，因此，this 指针的传递效率比较高－－如 vc 通常是通过 ecx 寄存器来传递 this 参数。

3. 回答

#1:this 指针是什么时候创建的?

this 在成员函数的开始执行前构造的，在成员的执行结束后清除。

#2:this 指针存放在何处? 堆,栈,全局变量,还是其他?

this 指针会因编译器不同，而放置的位置不同。可能是栈，也可能是寄存器，甚至全局变量。

#3:this 指针如何传递给类中函数的?绑定?还是在函数参数的首参数就是 this 指针.那么 this 指针又是如何找到类实例后函数的?

this 是通过函数参数的首参数来传递的。this 指针是在调用之前生成的。类实例后的函数，没有这个说法。类在实例化时，只分配类中的变量空间，并没有为函数分配空间。自从类的函数定义完成后，它就在那儿，不会跑的。

#4:this 指针如何访问类中变量的/?

如果不是类，而是结构的话，那么，如何通过结构指针来访问结构中的变量呢？如果你明白这一点的话，那就很好理解这个问题了。

在 C++中，类和结构是只有一个区别的：类的成员默认是 private，而结构是 public。

this 是类的指针，如果换成结构，那 this 就是结构的指针了。

#5:我们只有获得一个对象后,才能通过对象使用 this 指针,如果我们知道一个对象 this 指针的位置可以直接使用吗?

this 指针只有在成员函数中才有定义。因此，你获得一个对象后，也不能通过对象使用 this 指针。所以，我们也无法知道一个对象的 this 指针的位置（只有在成员函数里才有 this 指针的位置）。当然，在成员函数里，你是可以知道 this 指针的位置的（可以&this 获得），也可以直接使用的。

#6:每个类编译后,是否创建一个类中函数表保存函数指针,以便用来调用函数? 普通的类函数（不论是成员函数，还是静态函数），都不会创建一个函数表来保存函数指针的。只有虚函数才会被放到函数表中。但是，既使是虚函数，如果编译器能明确知道调用的是哪个函数，编译器就不会通过函数表中的指针来间接调用，而是会直接调用该函数。 # 7:这些编译器如何做到的?8:能否模拟实现?

知道原理后，这两个问题就很容易理解了。

其实，模拟实现 this 的调用，在很多场合下，很多人都做过。

例如，系统回调函数。系统回调函数有很多，如定时，线程啊什么的。

举一个线程的例子：

```
class A{
int n;
public:
static void run(void* pThis){
A* this_ = (A*)pThis;
this_->process();
}
```

```
void process(){}
};
main(){
A a;
_beginthread( A::run, 0, &a );
}
```

这里就是定义一个静态函数来模拟成员函数。

也有许多 C 语言写的程序，模拟了类的实现。如 freetype 库等等。

其实，有用过 C 语言的人，大多都模拟过。只是当时没有明确的概念罢了。

如：

```
typedef struct student{
int age;
int no;
int scores;
}Student;
void initStudent(Student* pstudent);
void addScore(Student* pstudent, int score);
...
```

如果你把 pstudent 改成 this，那就一样了。

它相当于：

```
class Student{
public:
int age; int no; int scores;
void initStudent();
void addScore(int score);
}
```

const 常量可以有物理存放的空间，因此是可以取地址的

///this 指针是在创建对象前创建.

this 指针放在栈上,在编译时刻已经确定.

并且当一个对象创建后,并且运行整个程序运行期间只有一个 this 指针.

问题的提出：编写程序实现对象资源的拷贝（要求使用 this 指针）。

```
#include <iostream>
#include <string>
```

```
using namespace std;
      class student{
private:
  char *name;
       int id;
public:
       student(char *pName="no name",int ssId=0)
       {
                  id=ssId;
         name=new char[strlen(pName)+1];
         strcpy(name,pName);
         cout<<"construct new student "<<pName<<endl;
       }
       void copy(student &s)
       {
                  if (this==&s)
                  {
       cout<<"Erro:can't copy one to oneself!"<<endl;
       return;
                  }else
      {
       name=new char[strlen(s.name)+1];
       strcpy(name,s.name);
       id=s.id;
       cout<<"the function is deposed!"<<endl;
      }
       }
       void disp()
       {
         cout<<"Name:"<<name<<"    Id:"<<id<<endl;
       }
       ~student()
       {
```

```
        cout<<"Destruct "<<name<<endl;
        delete name;
            }

};
int main()
{
        student a("Kevin",12),b("Tom",23);
   a.disp();
   b.disp();
   a.copy(a);
   b.copy(a);
   a.disp();
   b.disp();
     return 0;
}
     end!
```

在书中对于 this 指针的说明：对于每个成员函数(包括构造函数和析构函数)都有一个 this 指针。this 指针指向的是调用该函数的对象，如果方法需要应用整个调用对象。可以使用表达式*this，在函数的括号后面使用 const 限定符将 this 限定为 const，这样将不能使用 this 指针来修改对象的值。
所以在创建线程的时候，可以传递 this 指针来传递整个调用对象。

```
#include<iostream>
#include<string.h>
using namespace std;
class CTest
{
  public:
       int x;
       int y;
       CTest() {x=1;y=1;}
```

```
    ~CTest() {}
    void show()
    {
        cout<<x<<endl<<y<<endl;
    }
    void set(int x,int y)//void set(int a x,int b)
    {                                //{
        x = x;                       //    x=a;
        y = y;                       // y=b;
    }                                //}

};

int main()
{
    CTest ctest;
    ctest.show();
    ctest.set(5,8);
    ctest.show();
    return 0;
}
```

输出的结果是

1

1

1

1

我开始有点怀疑这是为什么

```
void set(int x,int y)
    {
        x = x;
        y = y;
    }
```

不行，但是

```
void show()
{
    cout<<x<<y<<endl;
}
```
却没有问题，
后来发现这是个变量的有效范围(作用域的问题)
显然，类的成员变量在类的成员函数中是可见的，所有的类方法都将 this 指针设置为调用他的对象的地址
在 void show()
函数中 x 和 y 其实是 this->x 和 this->y 的简写。
但是在 void set(int x,int y)中
x=x;
y=y;都是参数变量在其作用任何 x 或者 y 都不是 this->x 或者 this->y 的简写，因为参数变量和成员变量同名，在 void set(int x,int y)
就隐含(屏蔽)了成员变量
好比是：
main.cpp:
```
int i=5;
void set(int i);
int main()
{
    int n = 3;
     set(n);
    return 0;
}
void   set(int i)
{
     i=i; / / 显然不能对全局变量赋值成功，局部变量"屏蔽"了全局变量//要用::i=i
    cout<<i<<endl;//要数据全局变量的话也要 cout<<::i<<endl;
}
```
同样的如果想赋值成的话必须标记使用的是哪个变量
```
void set(int x,int y)
```

```
{
    this->x = x;
    this->y = y;
}
```

总之就是变量的作用域在作怪