问题

C++11新增了很多新特性，这也成为了面试中非常常见的问题，这里介绍一些常用的新特性。C++11新特性有很多，这里就简单整理几个很常见的，应该足以应对面试中的问题了。

C++11新特性

• 初始化列表

初始化列表，即用花括号来进行初始化。C++11中可以直接在变量名后面跟上初始化列表来进行对象的初始化，使用起来更加方便，例如：

```
vector<int> vec;            //C++98/03给vector对象的初始化方式
vec.push_back(1);
vec.push_back(2);

vector<int> vec{1,2};       //C++11给vector对象的初始化方式
vector<int> vec = {1,2};
```

• auto关键字

C++11之前，**在使用表达式给变量赋值的时候需要知道表达式的类型，如char、int等，然而有的时候要做到这一点并不容易**，因此，为了解决这个问题，C++11引入了`auto`关键字，编译器可以分析表达式的结果来进行类型推导。当然，直接定义变量的时候也可以使用`auto`来推导类型，可以理解为`auto`相当于一个占位符，在编译期间会自动推导出变量的类型。

```
auto a = 2;      //推导出a为int类型
auto b = 2.5;    //推导出b为double类型
auto c = &a;     //推导出c为int*类型

vector<int> vec = {1,2,3,4};
vector<int>::iterator it = vec.begin();      //初始化迭代器
auto it = vec.begin();                       //使用auto后更加方便
```

使用`auto`时**必须对变量进行初始化**；另外，也可以使用`auto`定义多个变量，但**必须注意，多个变量推导的结果必须为相同类型**，如：

```
auto a;      //错误，没有初始化
int a = 2;
auto *p = &a, b = 4;     //正确，&a为int*类型，因此auto推导的结果是int类型，b也是int类型
auto *p = &a, b = 4.5;   //错误，auto推导的结果为int类型，而b推导为double类型，存在二义性
```

`auto`使用的限制：

①`auto`定义变量时必须初始化

②`auto`不能在函数的参数中使用

③`auto`不能定义数组，例如：`auto arr[] = "abc"`，（`auto arr = "abc"`这样是可以的，但arr不是数组，而是指针）

④`auto`不能用于类的非静态成员变量中

● decltype关键字

有时候会遇到这样的情况：希望从表达式的类型中推断出要定义的变量的类型，但是想用该表达式的值来初始化变量。C++11中引入了`decltype`关键字来解决这个问题，编译器通过分析表达式的结果来返回相应的数据类型。

格式：

```
decltype(表达式) 变量名 [=初始值];    //[]表示可选,下面用exp来表示表达式
```

`decltype`的使用遵循以下3条规则：

①若exp是一个不被括号`()`包围的表达式，或者是单独的变量，其推导的类型将和表达式本身的类型一致

②若exp是函数调用，则`decltype(exp)`的类型将和函数返回值类型一致

③若exp是一个左值，或者是一个被括号`()`包围的值，那么`decltype(exp)`的类型将是exp的引用

具体示例：

```
class Base{
public:
    int m;
};
int fun(int a, int b){
    return a+b;
}

int main(){
    int x = 2;
    decltype(x) y = x;           //y的类型为int，上述规则1
    decltype(fun(x,y)) sum;      //sum的类型为函数fun()的返回类型，上述规则2

    Base A;
    decltype(A.m) a = 0;         //a的类型为int
    decltype((A.m)) b = a;       //exp由括号包围，b的类型为int&，符合上述规则3

    decltype(x+y) c = 0;         //c的类型为int
    decltype(x=x+y) d = c;       //exp为左值，则d的类型为int&，符合上述规则3
    return 0;
}
```

`decltype`和`auto`的区别：（两者都可以推导出变量的类型）

- `auto`是根据等号右边的初始值推导出变量的类型，且变量必须初始化，`auto`的使用更加简洁
- `decltype`是根据表达式推导出变量的类型，不要求初始化，`decltype`的使用更加灵活

• 范围for循环

类似于python中的for-in语句，使用格式及例子如下：

```
vector<int> nums = {1,2,3,4};
//使用冒号（:）来表示从属关系，前者是后者中的一个元素，for循环依次遍历每个元素，auto自动推导为int类型
for(auto num : nums){
    cout << num << endl;
}
```

• nullptr关键字

C++11使用`nullptr`代替了`NULL`，原因是`NULL`有时存在二义性，有的编译器可能将`NULL`定义为`((void*)0)`，有的则直接定义为0。

```
void fun(int x) {
    cout << x << endl;
}
void fun(int *p) {
    if (p != NULL) cout << *p << endl;
}

int main() {
    fun(0);      //在C++98中编译失败，存在二义性，在C++11中编译为fun(int)
    return 0;
}
```

`nullptr`是一种特殊类型的字面值，可以被转换成任意其他的指针类型，也可以初始化一个空指针。

```
int *p = nullptr;   //等价于 int *p = 0;
```

• lambda表达式

lambda表达式定义了一个匿名函数，一个lambda具有一个返回类型、一个参数列表和一个函数体。与函数不同的是，lambda表达式可以定义在函数内部，其格式如下：

```
[capture list] (parameter list) -> return type { function body }
//[捕获列表] (参数列表) -> 返回类型 { 函数体 }
```

• capture list（捕获列表）：定义局部变量的列表（通常为空）

• parameter list（参数列表）、return type（返回类型）、function body（函数体）和普通函数一样

• 可以忽略参数列表和返回类型，但**必须包括捕获列表和函数体**

示例：

```cpp
auto sum = [](int a, int b) -> int { return a+b; };      //一个完整的lambda表达式
cout << sum(1, 2) << endl;  //输出3

auto fun = [] { return 4; };     //省略参数列表和返回类型
cout << fun() << endl;       //打印结果为：4
```

lambda表达式可以定义在函数内：

```cpp
//使用lambda表达式和sort排序自定义一个降序排序算法
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

//bool cmp(const int a, const int b) {
//  return a > b;  // 前者大于后者返回true，因此为降序排序
//}

int main() {
    vector<int> nums{ 13, 5, 3, 7, 43 };
    //sort(nums.begin(), nums.end(), cmp);  // 1.使用函数来定义，需要自定义一个cmp函数来调用
    //2.直接使用lambda表达式
    sort(nums.begin(), nums.end(), [](int a, int b)-> int { return a > b; });
    for (auto i : nums) {
        cout << i << " ";
    }
    cout << endl;
    system("pause");
    return 0;
}
```

使用捕获列表：

- [] 不捕获任何变量
- [&] 捕获外部作用域中所有变量，并作为引用在函数体中使用（按引用捕获）。
- [=] 捕获外部作用域中所有变量，并作为副本在函数体中使用（按值捕获）。
- [=，&x] 按值捕获外部作用域中所有变量，并按引用捕获 x 变量。
- [x] 按值捕获 x 变量，同时不捕获其他变量。

```cpp
//下面使用lambda表达式编写一个函数，从数组中找到第一个大于给定长度的字符串
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    vector<string> str = {"abcd", "hello", "hi", "hello world", "hello abcd"};
    int len = 5;
```

```
11
12      //使用lambda表达式，len为按值捕获的变量
13      auto iter = find_if(str.begin(), str.end(), [len](const string &s)
    {return s.size() > len; });
14
15      cout<<"The length of first word longer than "<<len<<" is :  "<<*iter<<
    endl;
16      //system("pause");
17      return 0;
18  }
```

• 智能指针

C++提供了4中智能指针，`auto_ptr`、`unique_ptr`、`share_ptr`、`weak_ptr`，其中第一种为C++98中引入的，在C++11中已经被弃用，后三种是C++中引入的。

使用智能指针主要的目的是**为了更安全且更加容易地管理动态内存**。

关于智能指针的详细介绍，请参考 C++基础中的问题 ***05_请说一下你理解的 C++ 中的四个智能指针*** ，这里就不具体展开啦。

• 右值引用

右值引用的介绍，请参考 C++基础问题 ***31_c++中的左值引用与右值引用*** 。

参考资料

C++11教程：C++11新特性大汇总

《C++ Primer》第五版