```cpp
//一:用法简介
//c++11: 也是一种可调用对象。
//lambda表达式，它定义了一个匿名函数，并且可以捕获一定范围内的变量。
//auto f = [](int a)->int {
//auto f = [](int a){
auto f = [](int a=8) {
    return a + 1;
};
```

```cpp
std::cout << f() << std::endl;
//特点:
//a)是个匿名函数，也可以理解为“可调用的代码单元”，或者理解成 未命名的内联函数。
//b)它也有一个返回类型，一个参数列表，一个函数体
//c)与函数不同的是，lambda表达式可以在函数内部定义,这个是常规函数做不到的;
//格式:
//[捕捉列表](参数列表)->返回类型{函数体};
//a)这是一个返回类型后置这种语法（lambda表达式的返回类型后置是必须的,这个语法就这么规定);
//因为很多时候lambda表达式返回值特别明显，所以允许lambda表达式返回类型 省略，编译器可以自动推导;
//lambda参数可以有默认值
//大家要注意，编译器并不是总能推断出返回值类型，如果编译器推断不出来的时候，它会报错，这个时候你就要显式给出具体的返回值类型。
//b)没有参数的时候，参数列表可以省略，甚至()也能省略，所以如下这些格式都合法:
```

```cpp
auto f1 = [] {return 1; };
auto f2 = [] {return 2; };
cout << f1() << endl;
cout << f2() << endl;
```

```cpp
//c)捕获列表[]和函数体不能省，必须时刻包含。
//d)lambda调用方法和普通函数相同，都是使用()这种函数调用运算符;
//e)lambad表达式可以不返回任何类型，不反悔任何类型就是void;
auto f3 = [] { };
//f)函数体末尾的分号不能省略;
```

Lambda表达式在无法推导出返回值类型时,需指定返回值类型且只能使用后置返回类型

```cpp
//auto f = [](int a)->int {
```

```cpp
//二: 捕获列表    :[捕捉列表];通过捕获列表来捕获一定范围内的变量，范围是啥意思呢?
//a)[] 不捕获任何变量 ，但不包括静态局部变量。lambda可以直接使用局部静态变量(局部静态变量是不需要捕获的);
```

```cpp
int i = 9;
auto f1 = [] {
    return i;//报错（无法捕获外部变量）
}
```

```cpp
//b)[&] 捕获外部作用域中所有变量，并作为引用在函数体内使用
int i = 9;
auto f1 = [&]
{
    i = 5; //因为&的存在，那么就允许给i赋值，从而也就改变了i的值
    return i;
};
```

```
//c)[=] : 捕获外部作用域中所有变量，并作为副本（按值）在函数中使用，也就是可以 用它的值，但不许给它赋值。
//int i = 9;
//auto f1 = [=]
//{
//  //i = 5; //非法，不可以给它赋值，因为是以值的方式捕获。
//  return i;
//};
//cout << f1() << endl;
```

```
//d)[this]：一般用于类中，捕获当前类中this指针，让lambda表达式有和当前类成员函数同样的访问权限。
//如果[]中已经使用了 & 或者 =，那么默认就已经使用了this，说白了，捕获this的目的就是为了在lambda中使用当前类的成员函数和成员变量；
class CT
{
public:
    int m_i = 5;
    void myfuncpt(int x, int y)
    {
        auto mylambda1 = [this] //无论是this，还是& 或者 = 都可以达到访问类成员的目的；
        {
            return m_i; //因为有了this，这个访问合法的，用& =也行；
        };
        cout << mylambda1() << endl;
    }

};
```

```
//e)[变量名] ：如果是多个变量名，则彼此之间用,分隔。[变量名]表示按值捕获变量名代表的变量，同时不捕获其他变量；
//[变量名]：按引用捕获变量名代表的变量，同时不捕获其他变量；
```

```
//f)[=,&变量名]:
//按值捕获所有外部变量，但按引用捕获&中所指的变量，这里这个=必须写在开头位置，开头 这个位置表示 默认捕获方式；
//
```

```
//g)[&, 变量名]：按引用来捕获所有外部变量，但按值来捕获变量名所代表的变量，这里这个&必须写在开头位置，开头这个东西表示 默认捕获方式。
```

```
//总结：lambda表达式对于能访问的外部变量控制的非常细致；
```

```
//四：lambda表达式中的mutable(易变的)
int x = 5;
auto f = [=]() mutable //注意要加mutable，则()参数列表之外的这个圆括号不能省略；
{
    x = 6;
    return x;
};
```

```
//五：lambda表达式的类型及存储
//c++11中 lambda表达式的类型被称呼为 “闭包类型(Closure Type)”；
//闭包：函数内的函数（可调用对象）。 本质上就是lambda表达式创建的运行时期的对象；
//lambda表达式是一种比较特殊的，匿名的，类类型【闭包类】的对象（也就是定义了一个类类型，又生成一个匿名的改类类型的对象(闭包)
//我们可以认为它是一个带有operator()的类类型对象,也就是仿函数（函数对象）；
//所以，我们也可以用std::function和std::bind来保存和调用lambda表达式。每个lambda都会触发编译器给咱们生成 一个独一无二的类类型；
std::function<int(int)> fc1 = [](int tv) {return tv; };
cout << fc1(15) << endl;

std::function<int(int)> fc2 = std::bind( //bind第一个参数是函数指针，第二个参数开始就是真正的函数参数。
    [](int tv) {
        return tv;
    },
    16
);
cout << fc2(15) << endl;
```

```
//不捕获任何变量的lambda表达式，也就是捕获列表为空，可以转换成一个普通的函数指针；
using functype = int(*) (int); //定义一个函数指针类型；
functype fp = [](int tv) {return tv; };
cout << fp(17) << endl;
```

```cpp
//六：lambda表达式再演示和优点总结
//（6.1） for_each简介:是个函数模板；
/*vector<int> myvector = { 10,20,30,40,50 };
for_each(myvector.begin(), myvector.end(), myfunc);*/
vector<int> myvector = { 10,20,30,40,50 };
int isum = 0;
for_each(
    myvector.begin(), myvector.end(), [&isum](int val) {
    isum += val;
    cout << val << endl;
    }
);
```

```cpp
//（6.2） find_if简介 ：函数模板find_if
//用来查找一个东西，查什么东西呢，取决于它的第三个参数，它的第三个参数也是个函数对象（lambda表达式）
vector<int> myvector = { 10,20,30,40,50 };
auto result = find_if(myvector.begin(), myvector.end(), [](int val)
{
    cout << val << endl;
    if (val > 15)
        return true; //返回true表示停止遍历。
    return false; //只要我返回false，那么find_if就不停的遍历myvecotr。一直到返回true或者遍历完为止
});
//如果find_if第三个参数这个可调用对象（lambda)返回true，find_if就停止遍历；
//find_if的调用返回一个迭代器，指向第一个满足条件的元素，如果这样的元素不存在，则这个迭代器会指向myvector.end
```

```cpp
if (result == myvector.end())
{
    cout << "没找到" << endl;
}
else
{
    cout << "找到了，结果为：" << *result << endl;
}
```

```cpp
//总结：善用lambda表达式，让代码更简洁，更灵活，更强大，提高开发效率，提高可维护性等等；
```