函数匹配和const的关系

const对函数匹配的影响

参数传递会忽略顶层const

1. 形参是非引用或非指针类型

```
// 以下两个版本调用时会引起二义性，不知道调用哪一个
void foo(const int a);  // 这里的const是顶层const，传递参数时会忽略
void foo(int a);
```

而且，从另一个角度，由于形参是非引用，因此采用拷贝方式初始化形参。故

- const int a既可以用int类型的对象初始化，也可以用const int类型的初始化，因此会忽略该顶层const
- int a也是，既可以用int。。。

> 拷贝初始化，形参和实参互相独立，互不影响。

2. 形参是指针类型

```
void foo(const int *a); // 这里的const不是顶层const，是底层const，因此不会忽略
void foo(int *a);
```

- 这里的const int *a，很显然可以指向（绑定）到一个const int也可以绑定到int
- 第二个int *a，很显然不可以指向一个const int对象
  因此，这是两个可以在匹配过程可以区分的两个函数。

3. 形参是引用类型

```
void foo(const int &a); // 这里的const是底层const，顶层const隐含在引用中，即引用可以看成一个指针常量
                        // & ==== *const ，两者是等价的。
void foo(int &a);
```

和指针一样，第二个不可以绑定到const int类型对象。第一个两个都可以。因此，也可以区分两个函数。

成员函数可以通过有无const来区分

由前面所述，成员函数中const修饰的是this指针，故可以区分。