|  |  |
| --- | --- |
| 1. 可调用对象与function 2. 不同类型可能具有相同的调用形式 | Int add(int I,intj);  Auto mod[] = [](int I, int j){return I %j;}  Struct di{  Int operator()(int denominator,int divisor){  Return denominator / divisor;  }  }  Int(int,int);// 以上都是int(int,int)形式  函数表  Map<string,int(\*)(int,int)> binops;//不可  Map<string,function<int(int,int)>> binops;  头文件functional中。  // 添加  Binops[“+”] = add;  Binops[“-”] = divide();  // 使用  Binops[“+”](10,5);  Int (\*fp)(int,int) = add;  Binops.insert({“+”,fp});  Binops.insert({“+”,[](int a,int b){return add(a,b)}); |
| 1. 可调用对象与function 2. 与对象一样，可调用的对象也有类型。函数及函数指针的类型由其返回值类型和实参类型决定 3. 问题：两个不同类型的可调用对象可能共享同一种 调用形式。 4. 调用形式：指明返回的类型和实参类型。 5. 一种调用形式对应一个函数类型 6. 不同类型可能具有相同的调用形式 7. 定义一个函数表，来存储指向这些可调用对象的 “指针”，但是虽然调用形式一样，但是类型不一样，不可以以指针形式存储 8. add是函数指针，lambda有自己的类类型 9. 标准库function类型 10. 为解决上述问题，function标准库类型，可以存储同一种调用形式的函数，lambda或者类型 11. 重载的函数与function 12. 不能将重载的函数放入function中，因为二义性，不知道调用哪个。即使形参类型不一样，或个数不一样。 13. int(\*fp1)(int, int) = overadds; 14. binops3.insert({ "+",fp1 });// 可以用函数指针，会指定特定版本 15. 或者使用lambda代替重载的函数 | |

