C++入门：C++中的一些小的知识点

1. 对C语言中的缺陷进行改进
2. C++中引入的新的语法特性

命名空间：新的作用域---用来解决名字冲突

1. 概念
2. 定义---普通　嵌套　相同名称
3. 使用：包含命名空间的名字 using N::name; using namespace N;

缺省参数---备胎：

1. 全缺省参数：所有参数都有缺省值
2. 半缺省参数：只有部分参数具有缺省值--（只能从右往左）

注意：声明&定义--->最好声明

函数重载：

前提：必须在相同的作用域(全局，局部，命名空间)

条件：名字必须要相同，参数列表一定不同(参数个数不同、参数类型不同、类

型的次序不同)

返回值类型是否相同没有关系

```
int TestFunc(int a){}
double TestFunc(int a){}
int main(){
   TestFunc(1);
   TestFunc(2);
   return 0;
}
```

此时编译器不知道应该调用哪一个函数。所以是否构成重载与返回值类

型无关

```
int Add(int left, int right){
    return left + right;
}
double Add(double left, double right){
    return left + right;
}
int main(){
    Add(1, 2);      // 编译器进行推演，结果调用第一个加法函数
    Add(1.0, 3.0);  // 编译器进行推演，结果调用第二个加法函数
    return 0;
}
void TestFunc() {}
void TestFunc(int a = 0) {}
int main(){
    TestFunc();
    return 0;
}
```

上边这两个函数构成重载，但是在编译执行main函数的时候，就会报错

因为这两个函数虽然构成重载，但是第一个函数没有参数，第二个函数

看似有一个参数，而实际上这个参数是缺省参数，所以在调用TestFunc

这个函数时编译器就不能确定该调用哪一个函数。

C语言不支持函数重载：在C语言中，编译器对函数名字的修饰规则：只是在函数名前面加一个下划线。

因为C语言的命名规则非常简单，只是在函数名的前边加一个下划线“_”

验证为什么C++支持函数重载

```
int Add(int left, int right);              // ?Add@@YAHHH@Z
HHH=>返回值
                                                  类型以及参数列表的类型
double Add(double left, double right);  // ?Add@@YAHHH@Z
char Add(char left, char right);           // ?Add@@YADDD@Z
int Add(int left, char right);             // ?Add@@YAHHD@Z
//
int main() {
    Add(1, 2);
    Add(1.0, 3.0);
    Add('1', '4');
    Add(1, '2');
    return 0;
}
```

在函数前边加上一个 extern "C" 编译器就会把C++工程中的某个函数按照C语言的风格进行编译。

extern "C"的功能：在C++的工程中把一个函数按照C语言的风格进行编译

引用：应用层面

1. 概念：引用就是一个别名，与其引用的实体共用一句爱内存空间
2. 定义：

    a)引用必须与其实体的类型一致

    b)必须实例化

```
int a = 10; int& ra = a;
void TestFunc(int& a){}
```

3. 引用特性：

    a)

    b)

    c)

4. const 引用：
5. 引用的使用场景：

    a) 引用做参数：他可以达到和指针类似的效果，而且比指针使用起来更加

    简洁方便，代码的可读性很高。而且效率和指针是类似的，因为在底层，应

    用就是按照指针的方式处理的。

    b) 引用做函数的返回值：如果一个函数以引用的方式作为返回值，那么

    的这个东西的生命周期不能受函数的限制，如果出了函数的作用域，这个变

    量就不存在了，那么就不可以把这个东西按照引用的方式返回。如果要按照

引用类型返回，那么就需要返回的这个实体的生命周期比函数的生命周期长

就可以，比如：全局变量、函数中有一个引用类型的参数，将来在这个函数

体中对这个引用类型的参数进行一系列的操作，再把它返回出去也可以、在

这个函数体中一个静态类型的变量，将来把这个静态的变量返回去也可以，

因为静态的变量不在栈上，只要你返回的东西不在栈上就都可以、在这个函

数体中动态开辟一块空间，然后把这块空间的地址返回去也是可以的，因为

动态开辟的空间是在堆上，函数运行结束了，栈被操作系统回收了，但是堆

得空间仍旧存在)

6. 引用传参 -- 返回值(效率比较)

a) 传值

b) 传地址

c) 传引用

传值的效率最低，指针和引用的效率基本类似

自定义类型：能用引用的尽量用引用 T&(需要修改),const T&(不需要修改)

7. 指针和引用到底有什么区别？

a) 底层实现：引用实际上就是一个指针。因为在底层，编译器只要看到引

用这个类型，他将来要翻译这个代码，就把引用按照指针的方式进行翻译

了。

int a = 10; int& ra = a; ra 在底层的类型：int *const p = &a;

const int& cra = a; cra 在底层的类型：const int *const

引用的概念：编译器不会为引用变量重新开辟内存空间

在底层的实现方式：引用实际上是有空间的

b) 应用

```
int a = 10;
const int&& ra = 10;  // 右值引用 C++11
```

底层引用