指针与引用

指针的基本用法

1. 指针的定义
2. 使用指针改变地址
3. 使用指针更改变量的值

```
int *p = NULL;
int a = 0, b = 5;
p = &a; //指针的赋值：将a的内存地址赋值指针p
*p = b; //指针的赋值：将(int)b=5的值赋给(int)a
```

注：& 表示取址、* 表示取值

&a 的数据类型为int型指针；*p数据类型为整型int

结构体指针

```
#incldue<iostream>
using namespace std;

struct node{
    int value;
    node* next;
    node(int _value){
        value = _value;
    }
};
node* head;

int main(){
    head = new node(1);//用new动态开辟head指向node的空间
    head->next = new node(2);//对head内部的next进行操作（用new动态开辟head的next指向node的空间）
    node* second = head->next;//声明一个node型的指针名为second

    cout << head->value << second->value << head->next->value;
    return 0;
}
```

指针作为参数传递

```cpp
int swap(int *a, int *b){
    int c;
    c = *a, *a = *b, *b = c;
}
int main(){
    int a = 3, b = 4;
    swap(a, b);
    std::cout << a << " " << b;
    return 0;
}
```

c++引用

引用是某一个变量或对象的**别名**，对引用的操作与对其所绑定的变量或对象的操作完全等价。

```cpp
//常量的引用
//语法：类型 &引用名=目标变量名；
int a = 5;
int &b = a;
std::cout << a << " " << b;

//数组的引用
//语法：类型 (&引用名)[数组中元素数量]=数组名；
int a[3] = {1, 2, 3};
int (&b)[3] = a;
std::cout<< &a[0] << " " << &b[0];
```

注意：

1. &不是求地址运算符，而是起标志作用
2. 引用的类型必须和其所绑定的变量的类型相同
3. 声明引用的同时**必须对其初始化**，否则系统会报错
4. 引用相当于变量或对象的别名，因此**不能再将已有的引用名作为其他变量或对象的名字或别名**
5. 引用不是定义一个新的变量或对象，因此**内存不会为引用开辟新的空间存储这个引用**

引用作为参数传递

```cpp
int swap(int &x, int &y){
    int c;
    c = a, a = b, b = c;
}

int main(){
    int a = 3, b = 4;
    swap(a, b);
    std::cout << a << " " << b;
    return 0;
}
```

常引用

来源

常引用声明方式：const 类型标识符 &引用名=目标变量名；

用这种方式声明的引用，不能通过引用对目标变量的值进行修改,从而使引用的目标成为const，达到了引用的安全性。

```
int a ;
const int &ra=a;
ra=1; //错误
a=1; //正确
```

引用作为返回值