总结：引用

本文链接：https://knights.blog.csdn.net/article/details/107027359

1. 普通引用

> - **概念：** **可以理解为是一个已定义变量的别名**
> - **注意：** **普通引用 必须要初始化**（int &b = a），但引用 **做形参时不需要初始化**
> - **优点：** **具有更好的可读性和实用性**

```
int a = 10;
int &b = a; // b就是a的别名
b = 200;    // 修改变量a中的内容

printf("&a = %d \n &b = %d \n", &a, &b);  // a 与 b 的地址是相同的

/* 在C中相当于以下效果 */
int * const b = &a;
*b = 200;
```

1.1 普通引用 举例

```
void swap(int &a, int &b){  // 交换数据
    int temp = a;
    a = b;
    b = temp;
}
```

1.2 引用所占空间大小

```
/* QT64位环境测试 */
struct myType{
    int a;          // 4字节
    int arary[10];  // 40字节
    int &b;         // ?字节
    char &c;        // ?字节
};

/* 打印结果： */
void printSize(){
    printf("sizeof(int) : %d \n", sizeof(int));             // 4字节
    printf("sizeof(arary[10]) : %d \n", sizeof(int) * 10);  // 40字节
    printf("sizeof(int *) : %d \n", sizeof(int *));         // 8字节
    printf("sizeof(char *) : %d \n", sizeof(char *));       // 8字节
    printf("sizeof(myType) : %d \n", sizeof(myType));       // 64字节(自动4字节对齐，实际为60字节)
}
```

> **结论：** **实际引用所占空间大小 与指针类型所占空间相同 。32位系统占4字节， 64位系统占8字节**

1.3 引用的本质

> - **引用在C++中的内部实现是一个 常指针（相当于 type *const var）**
> - **C++编译器在编译过程中使用常指针作为引用的内部实现，因此引用 所占用的空间大小与指针相同**
> - **从使用的角度，引用会让人误会其只是一个别名，没有自己的存储空间。这是C++为了实用性而做出的 细节隐藏**

1.4 函数返回值是引用

函数返回值是局部变量当引用

```
/* QT64位 环境测试 */
#include <iostream>

int GetValueA(){    // 返回局部变量a的值
    int a = 10;
    return a;
}

int& GetValueB(){   // 返回局部变量b的引用
    int b = 20;
    return b;       // 此处有警告
}

int main(int argc, const char *argv[]){
    int a   = GetValueA();  // 可以正常获取返回值
    int b1  = GetValueB();  // Qtx64位环境编译通过(有警告)，但运行出错    vs2013编译通过，也可正常运行
    int &b2 = GetValueB();  // Qtx64位环境编译通过(有警告)，但运行出错    vs2013编译通过，调试方式不一样，一个正常显示，  一个显示地址

    GetValueB() = 10;       // 测试作为左值接收，编译通过，但运行出错

    std::cout << "a  = "  << a  << std::endl;
    std::cout << "b1 = " << b1 << std::endl;
    std::cout << "b2 = " << b2 << std::endl;

    return 0;
}
```

> - **结论：当函数返回值为引用时， 若返回 局部变量（栈变量）时（条件）**
>   1. **不能成为其它引用的初始值**
>   2. **不能作为左值使用**
> - **原因：栈变量使用完后被系统自动释放资源**

函数返回值是static变量当引用

```
/* QT64位环境测试 */
#include <iostream>

int GetValueA(){  // 返回局部变量a的值
    static int a = 10;
    return a;
}

int& GetValueB(){ // 返回局部变量b的引用
    static int b = 20;
    return b;     // 此处有警告
}

int main(int argc, const char *argv[]){
    int a   = GetValueA();  // 可以正常获取返回值
    int b1  = GetValueB();  // 可以正常获取返回值
    int &b2 = GetValueB();  // 可以正常获取返回值

    std::cout << "a = "  << a  << std::endl;    // 正常打印：10
    std::cout << "b1 = " << b1 << std::endl;    // 正常打印：20
    std::cout << "b2 = " << b2 << std::endl;    // 正常打印：20

    GetValueB() = 100;                         // 测试作为左值，测试正常
    std::cout << "GetValueB() =  "  << GetValueB()  << std::endl;   // 正常打印：100

    return 0;
}
```

> - **结论：当函数返回值为引用时， 若返回返回 静态变量 或 全局变量（条件）**
>   1. **可以成为其它引用的初始值**
>   2. **可以作为左值使用，也可以作为右值使用**

1.5 指针引用

```
#include "iostream"

using namespace std;

struct Teacher{
    char name[64];
    int age ;
};

// 1.C语言中的二级指针
int getTeacher1(Teacher **p){
    if (p == NULL){
        return -1;
    }

    Teacher *tmp = (Teacher *)malloc(sizeof(Teacher));
    if (tmp == NULL){
        return -2;
    }

    tmp->age = 33;
    *p = tmp;
}

// 2.C++中的指针引用
int getTeacher2(Teacher* &pt){
   pt = (Teacher *)malloc(sizeof(Teacher));
   if(NULL == pt){
       return -1;
   }

    pt->age = 40;
}

// 释放内存
void FreeTeacher(Teacher *pT1){
    if (pT1 == NULL){
        return ;
    }

    free(pT1);
}

int main(int argc, const char *argv[]){
    Teacher *pT1 = NULL;

    getTeacher1(&pT1);
    cout<<"age:"<<pT1->age<<endl;
    FreeTeacher(pT1);

    getTeacher2(pT1);
    cout<<"age:"<<pT1->age<<endl;
    FreeTeacher(pT1);
}
```

> **结论：指针引用可以理解为C语言中的 二级指针**

---

2. 常引用

> **作用：让引用 只读属性， 不能通过引用修改变量**

2.1 常引用基本形式

```
int a = 10;
const int &b = a;
b = 20;              // 编译器报错... 修改不了内存中的值
```

2.2 常引用初始化

> - **常引用初始化分为两种情况：**
>   1. **用变量初始化常引用（2.1例程）**
>   2. **用字面量初始化常引用 (以下例程)**

```
const int a = 10;    // C++编译器将a放在符号表中
int &b = 10;         // 编译器报错，原因：字面量没有内存地址
const int &c = 10;   // 编译通过：C++编译器会分配内存空间
```

2.3 常引用做函数参数

```
int printInfo(const Teacher &t){}  // 拥有只读属性，函数内不能修改内容
```

---

3. 引用总结

> - **c++中 *const & int name;* 相当于 C中 const int *const name;**
> - **普通引用相当于C中 int *const name;**
> - **当使用常量（字面量）对const引用进行初始化时，C++编译器会为常量值分配空间，并将引用名作为这段空间的别名**
> - **使用字面量对const引用初始化后，将生成一个只读变量**

👈 **写文不易 且行且珍惜** 👉

👈 **MrWang** 👉