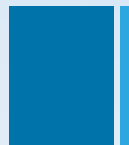




# C++

## 第四讲 C++对C的改进（三）

C++备课组 丁盟



# 自我介绍

丁盟

qq : 2622885094





# 上一讲教学目标

- 理解C++中函数与C语言中函数的不同
- 掌握C++中函数重载的用法
- 理解C++中的内存处理机制
- 了解C++中的异常处理机制



# 本讲教学目标

- 掌握C++中引用的使用
- 掌握C++中引用作为函数参数的使用方法
- 了解C++中的强制类型转换

1

## 引用 ( reference )

普通变量与引用

指针变量的引用

const与引用

函数与引用

2

## 类型强制转换

# 普通变量与引用

**value(变量名)**

**3**

**ref(变量的别名)**

❖ 引用：**通常指变量的别名**

```
#include <iostream>
using namespace std;
int main(void)
{
    int value = 3;
    int &ref = value;
    int *ptr = &ref;
    cout << ref << " "
         << *ptr << endl;

    return 0;
}
```

# 普通变量与引用

```
int main(void)
{
    int ival = 10;
    int &rival = ival;
    ival = ival*ival;

    cout << &rival << setw(10) << &ival << endl;
    cout << ival << setw(10) << rival << endl;
    rival /= 5;
    cout << ival << setw(10) << rival << endl;

    return 0;
}
```

# 普通变量与引用

## ❖ 引用的错误声明

- 没有void引用，引用的引用，指向引用的指针、引用数组、空引用

```
int x = 10, a[10];  
int &rx = x;
```

int &&r;	// 引用的引用
int &*p;	// 引用的指针
int &arr[3];	// 引用数组
void &r;	// void类型的引用
int &r = NULL;	// 空引用



# 普通变量与引用

## ❖ 引用与指针的区别

```
double  dval = 10, dgrade = 90;
double* pn = &dval;
double& rn = dval;
cout << &dval << " " << &rn << " "
      << &pn << endl;
cout << sizeof(pn) << " "
      << sizeof(rn) << endl;
pn = &dgrade;
rn = dgrade;
cout << *pn << " " << rn << endl;
//指针变量定义的时候可以赋空值，引用不行
int *p = NULL;      //int &q = NULL;
```

# 普通变量与引用

## ❖ [注意]

- 声明引用**必须初始化**（除引用做型参和返回值时）

```
int &rival = ival;
```

- 声明引用后，该引用不能作为其他变量的引用

```
int a1, a2;
```

```
int &ra = a1;
```

```
ra = a2; // 赋值
```

- 所有对引用的操作都将转换成对所引用变量的操作

1

## 引用 ( reference )

普通变量与引用

指针变量的引用

const与引用

函数与引用

2

## 类型强制转换

# 指针变量的引用

❖ 指针变量也是变量，因此有指针变量的引用

类型\* &指针引用名 = 指针；

必须初始化

```
int main(void)
{
    int n = 10;
    int *pn = &n;
    //int *&rn = &n 错误,右值只能是指针变量,不能为表达式
    int *&rn = pn;
    (*pn)++;    cout << "n=" << n << endl;
    (*rn)++;    cout << "n=" << n << endl;
    return 0;
}
```

1

## 引用 ( reference )

普通变量与引用

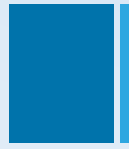
指针变量的引用

const与引用

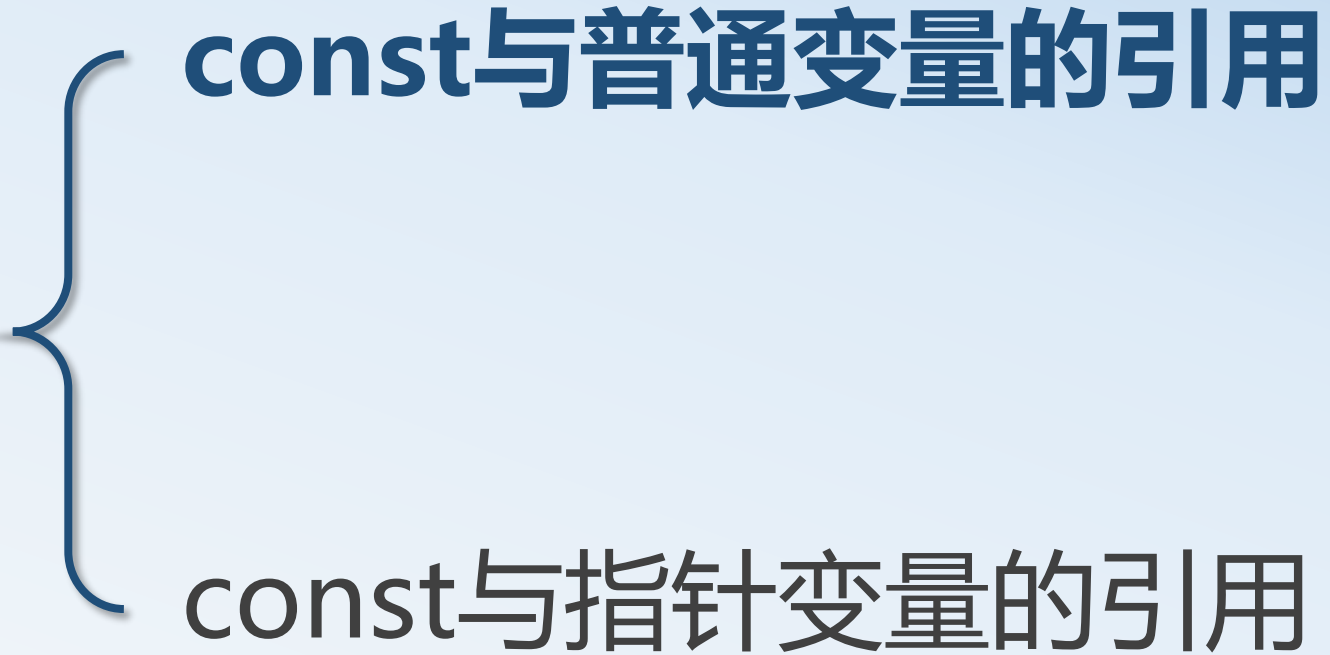
函数与引用

2

## 类型强制转换



# const与引用



# const与普通变量的引用

## ❖ 格式：

```
const 类型 &引用名 = 表达式;  
类型 const &引用名 = 表达式;
```

必须初始化

## ❖ 含义：不能通过引用修改变量的值

```
int x = 2;  
x++;  
const int &n = x;  
cout<< "x=" << x << endl;  
n++; // Error
```

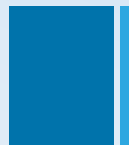
## const与普通变量的引用

❖注意：const int to int & is error

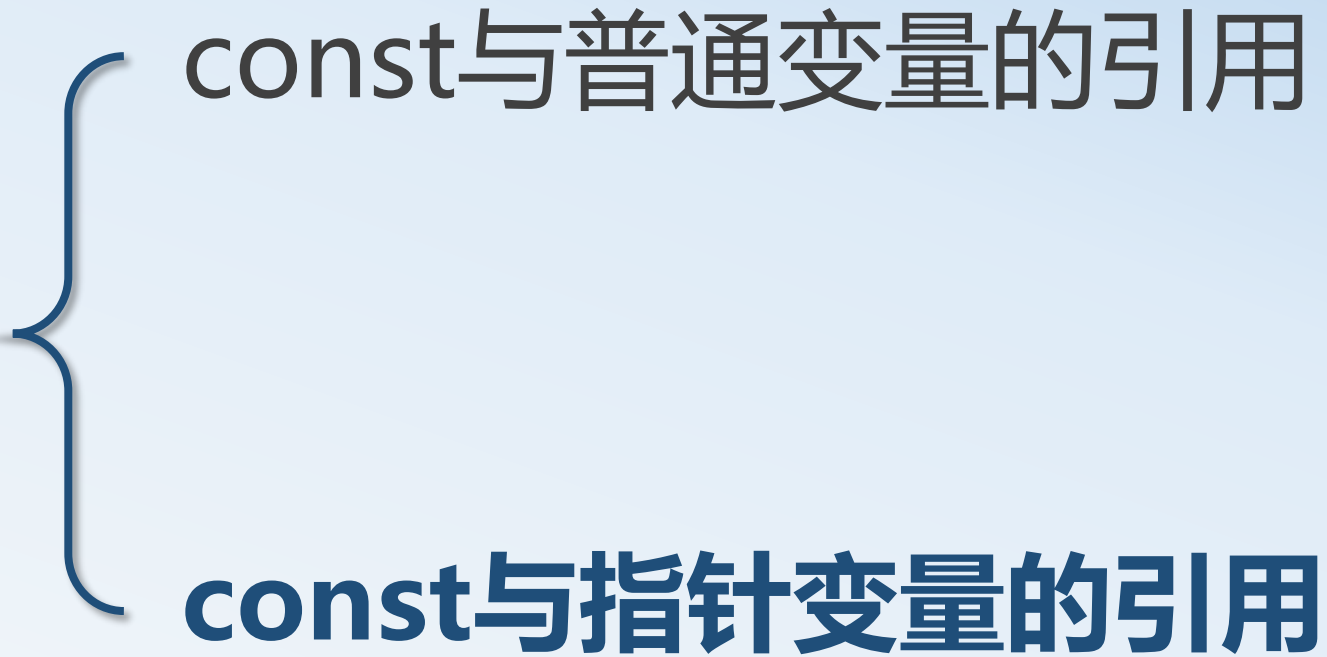
```
int main(void)
{
    //const int to int& is error
    const int ival = 3;
    int &rival = ival;

    return 0;
}
```





# const与引用



# const与指针变量的引用

## ❖ 指向常量的指针变量的引用

**const** 类型 \* &指针引用名 = 常量指针;

```
int a = 10;
```

```
const int * pa = &a;
```

```
const int * &ra = pa;
```

限定\*ra

必须初始化

## ❖ 常指针的引用

类型 \* **const** &指针引用名 = 指针;

```
int * const p = &a;
```

```
int * const &rb = p;
```

限定rb

1

## 引用 ( reference )

普通变量与引用

指针变量的引用

const与引用

函数与引用

2

## 类型强制转换

# 函数与引用

**引用作为函数参数**

引用作为函数返回值

函数引用

# 引用作为函数参数

## 1、普通变量作为函数参数

值  
传  
递

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

void swap(int a,int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

```
int main(void)
{
    int i = 3,j = 5;
    swap(i , j);
    cout<<"i="<<i<<endl
        <<"j="<<j<<endl;

    return 0;
}
```

# 引用作为函数参数

## 2、指针变量作为函数参数

值  
传  
递

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

void swap(int *x,int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

```
int main(void)
{
    int i = 3,j = 5;
    swap( &i,  &j );
    cout << "i= "
          << i << endl;
    cout << "j= "
          << j << endl;

    return 0;
}
```

# 引用作为函数参数

## 3、引用作为函数参数

引用传递

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

void swap(int &x ,int &y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

```
int main(void)
{
    int i = 3,j = 5;
    swap(i,j);
    cout << "i= "
          << i << endl;
    cout << "j= "
          << j << endl;

    return 0;
}
```

# 引用作为函数参数

## 4、常引用作为函数参数

引用传递

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

void swap(const int &x,
          const int &y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

```
int main(void)
{
    int i = 3, j = 5;
    swap( i, j );
    cout << "i= "
          << i << endl;
    cout << "j= "
          << j << endl;

    return 0;
}
```



# 引用作为函数参数

## ❖ [注意]

- 指针变量占内存,内容为地址,是值传递,值是地址;引用传递,效率高
- 引用作为形参时,对应的实参必须是变量,不能是常量或者表达式 ( But `const reference` is OK )

```
void display(int &x)
{
    cout << x;
}
... ..
display(3); //Error
```

```
void display(const int &x)
{
    cout << x;
}
... ..
display(3); //Right
```

# 引用作为函数参数

- 常引用作为函数参数时，可以作为重载的依据

```
void ff(const int&)  
{    cout << "const int&" << endl; }  
void ff(int&)  
{    cout << "int&" << endl; }  
int main(void)  
{    const int KIval = 100;  
      ff(KIval);  
      int ival = 0;  
      ff(ival);  
      return 0;  
}
```

# 函数与引用

引用作为函数参数

**引用作为函数返回值**

函数引用

# 引用作为函数返回值

❖ 概念：函数的返回值是引用类型

❖ 格式：**类型名 &函数名（形参表）**

```
int& fun(int &x, int &y);
```

❖ 特点：函数调用即可为右值，也可作为左值

```
d = fun(a, b);  
fun(a, b) = 7;
```

## 引用作为函数返回值

```
int& foo(int &raIval)
{
    return raIval;
}


int main(void)
{
    int ival = 3;
    foo(ival) = 5;
    cout << ival << endl;
    return 0;
}
```

函数调用作为左值

# 引用作为函数返回值

## ❖ [注意]

- 返回变量引用的函数 return 后面必须为变量，不能为常量或者表达式
- 局部变量不能作为引用返回

```
int &fun( int &x , int y )  
{  
    ... ..  
    return y;   
}
```

结果不可预知！

## 引用作为函数返回值

```
int sum = 0;
int &foo(int arg)
{
    sum += arg;
    return sum;
}
int main(void)
{
    foo(10) += 20;
    cout << "sum = " << sum << endl;

    return 0;
}
```

## 引用作为函数返回值

- ❖ 当返回常量时,则函数返回类型也应该为const

```
const int knum = 3;  
const int &foo()  
{  
    return knum;  
}  
  
int main(void)  
{  
    cout << foo() << endl;  
    return 0;  
}
```



# 引用作为函数返回值

## ❖ 引用返回与传值返回的区别

```
... ..  
int fun(int y)  
{  
    return y;  
}
```

```
... ..  
int* fun(int *y)  
{  
    return &y;  
}
```

```
... ..  
int& fun(int &y)  
{  
    return y;  
}
```

```
int main(void)  
{  
    int ival = 2;  
    cout << fun(ival) << endl;  
    return 0;  
}
```

# 函数与引用

引用作为函数参数

引用作为函数返回值

**函数引用**

## 函数引用

```
int reffunction(int aival)
{
    cout << "Hello world!" << endl;
    return 1;
}
int ( &rfn1 )(int) = reffunction;
int ( &rfn2 )(int) = rfn1;
int main(void)
{
    rfn1(1);
    rfn2(1);
    return 0;
}
```

1

## 引用 ( reference )

普通变量与引用

指针变量的引用

const与引用

函数与引用

2

## 类型强制转换

# 类型强制转换

## ❖ C++对旧式强制类型转换的改进

### 1. C的旧式强制转换：**(类型名)(表达式)**

```
(int) x  
(int) 5  
(int) (x + y)
```

### 2. C++的强制转换：（兼容了C的转换）

**类型名 (表达式)**

```
int (x)  
int (x + y)
```

# 类型强制转换

## 3. 命名的强制类型转换

**const\_cast**

**static\_cast**

**reinterpret\_cast**

**dynamic\_cast** (后续课程中讲解)

# 类型强制转换

➤ **const\_cast**: **const\_cast<type\_id>(expression)**

功能：

- ① 用于去除const
- ② 反过来也可加上const

注意：只针对指针或引用

```
const int ival = 10;  
int ivall = const_cast<int>(ival); //无法转换  
int *p_ival = const_cast<int *>(&ival);  
const int *q_ival = const_cast<const int *>(p_ival);
```

# 类型强制转换

```
void fun(const char* src)
{
    char *p = const_cast<char*>(src);
    while( *p != '\0' )
    {
        *p = 'x';
        p++;
    }
}
int main(void)
{
    char buf[] = "hello world";
    fun( buf );
    cout << buf << endl;
    return 0;
}
```



# 类型强制转换

➤ **static\_cast:** `static_cast<type_id>(expression)`

功能：

- ① 用于内置数据类型之间的转换，如：int转换成char
- ② 空指针转为目标类型指针或目标类型指针转换空指针  
(参与转换的两个指针之中至少一个是void \*)
- ③ 把任何类型的表达式转换成void类型
- ④ 用于类层次结构中基类和子类之间指针或引用的转换
- ⑤ 具有继承关系的类类型对象之间转换  
(基类不可以转换成派生类对象，派生类对象可以转换成基类对象)

# 类型强制转换

## ➤ reinterpret\_cast:

**reinterpret\_cast<type\_id> (exdivssion)**

功能：

- ① 它可将指针转为整数，也能将整数转为指针（先把指针转成整数，再把整数转为原类型的指针，还可得到原先的指针值）
- ② 指针之间的相互转换
- ③ 引用之间的转换

功能：

- ① 它可将指针转为整数，也能将整数转为指针（先把指针转成整数，再把整数转为原类型的指针，还可得到原先的指针值）
- ② 指针之间的相互转换

# 类型强制转换

[建议]

强制转换关闭或挂起了正常类型检查，强烈建议**尽量**  
**少的使用强制类型转换！**



# 本讲教学目标

- 掌握C++中引用的使用
- 掌握C++中引用作为函数参数的使用方法
- 了解C++中的强制类型转换



THANKS

