



北京大学
PEKING UNIVERSITY

信息科学技术学院

程序设计实习

C++面向对象程序设计

郭炜 微博 <http://weibo.com/guoweiofpku>



北京大学
PEKING UNIVERSITY

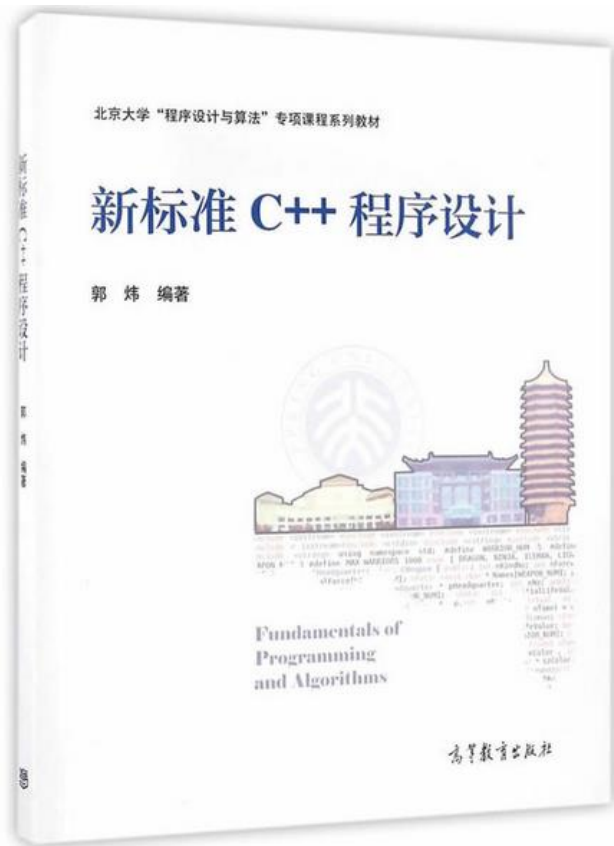
信息科学技术学院

配套教材:

高等教育出版社

《新标准C++程序设计》

郭炜 编著





北京大学
PEKING UNIVERSITY

信息科学技术学院

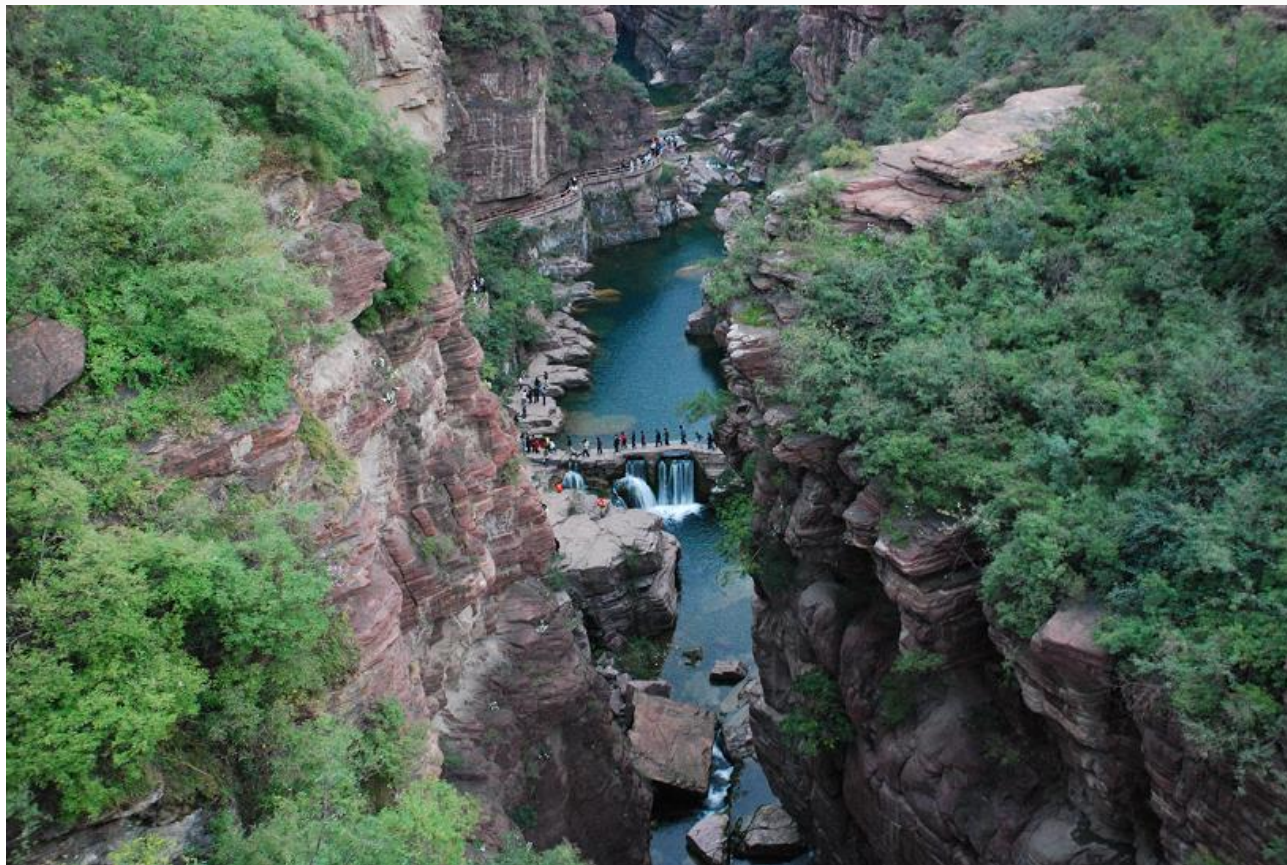
从C到C++



北京大学
PEKING UNIVERSITY

信息科学技术学院 郭炜

“引用”的概念 和应用



河南云台山红石峡

引用的概念

➤ 下面的写法定义了一个引用，并将其初始化为引用某个变量。

类型名 & 引用名 = 某变量名;

```
int n = 4;
```

```
int & r = n;    // r引用了 n, r的类型是
```

引用的概念

➤ 下面的写法定义了一个引用，并将其初始化为引用某个变量。

类型名 & 引用名 = 某变量名;

```
int n = 4;
```

```
int & r = n;    // r引用了 n, r的类型是 int &
```

引用的概念

- 下面的写法定义了一个引用，并将其初始化为引用某个变量。

类型名 & 引用名 = 某变量名;

```
int n = 4;
```

```
int & r = n;    // r引用了 n, r的类型是 int &
```

- 某个变量的引用，等价于这个变量，相当于该变量的一个别名。

引用的概念

```
int n = 4;  
int & r = n;  
r = 4;  
cout << r;    //输出 4  
cout << n;  
n = 5;  
cout << r;
```


引用的概念

```
int n = 4;  
int & r = n;  
r = 4;  
cout << r; //输出 4  
cout << n; //输出 4  
n = 5;  
cout << r;
```

引用的概念

```
int n = 4;  
int & r = n;  
r = 4;  
cout << r;    //输出 4  
cout << n;    //输出 4  
n = 5;  
cout << r;    //输出5
```

引用的概念

- 定义引用时一定要将其初始化成引用某个变量。

引用的概念

- 定义引用时一定要将其初始化成引用某个变量。
- 初始化后，它就一直引用该变量，不会再引用别的变量了。

引用的概念

- 定义引用时一定要将其初始化成引用某个变量。
- 初始化后，它就一直引用该变量，不会再引用别的变量了。
- 引用只能引用变量，不能引用常量和表达式。

引用的概念

```
double a = 4, b = 5;  
double & r1 = a;  
double & r2 = r1;    // r2也引用 a  
r2 = 10;  
cout << a << endl;  
r1 = b;  
cout << a << endl;
```

引用的概念

```
double a = 4, b = 5;  
double & r1 = a;  
double & r2 = r1;      // r2也引用 a  
r2 = 10;  
cout << a << endl;    // 输出 10  
r1 = b;  
cout << a << endl;
```

引用的概念

```
double a = 4, b = 5;  
double & r1 = a;  
double & r2 = r1;      // r2也引用 a  
r2 = 10;  
cout << a << endl;    // 输出 10  
r1 = b;                // r1并没有引用b  
cout << a << endl;
```


引用的概念

```
double a = 4, b = 5;  
double & r1 = a;  
double & r2 = r1;      // r2也引用 a  
r2 = 10;  
cout << a << endl;    // 输出 10  
r1 = b;                // r1并没有引用b  
cout << a << endl;    // 输出 5
```

引用应用的简单示例

C语言中，如何编写交换两个整型变量值的函数？

引用应用的简单示例

C语言中，如何编写交换两个整型变量值的函数？

```
void swap( int * a, int * b)
{
    int tmp;
    tmp = * a; * a = * b; * b = tmp;
}
int n1, n2;
swap(& n1, & n2) ; // n1, n2的值被交换
```

引用应用的简单示例

➤有了C++的引用:

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

int n1, n2;
swap(n1, n2) ; // n1,n2的值被交换
```

引用作为函数的返回值

```
int n = 4;
int & SetValue() { return n; }
int main()
{
    SetValue() = 40;
    cout << n;
    return 0;
}
```

引用作为函数的返回值

```
int n = 4;  
int & SetValue() { return n; }  
int main()  
{  
    SetValue() = 40;  
    cout << n;  
    return 0;  
} //输出: 40
```

常引用

定义引用时，前面加const关键字，即为“常引用”

```
int n;
```

```
const int & r = n;
```

r 的类型是

常引用

定义引用时，前面加const关键字，即为“常引用”

```
int n;
```

```
const int & r = n;
```

r 的类型是 **const int &**

常引用

不能通过常引用去修改其引用的内容:

```
int n = 100;  
const int & r = n;  
r = 200;    // 编译错  
n = 300;    // 没问题
```

常引用和非常引用的转换

const T & 和 T & 是不同的类型!!!

T & 类型的引用或T类型的变量可以用来初始化
const T & 类型的引用。

const T 类型的常变量和const T & 类型的引用则
不能用来初始化T &类型的引用，除非进行**强制类型转换**。

下面程序片段哪个没错?

- ☐ A `int n = 4; int & r = n * 5;`
- ☐ B `int n = 6; const int & r = n; r = 7;`
- ☐ C `int n = 8; const int & r1 = n; int & r2 = r1;`
- ☒ D `int n = 8; int & r1 = n; const int r2 = r1;`

提交

下面程序片段输出结果是什么？

```
int a = 1,b = 2;  
int & r = a;  
r = b;  
r = 7;  
cout << a << endl;
```

- ☐ A 1 ☐ B 2
- ☒ C 7 ☐ D 都不是

提交



北京大学
PEKING UNIVERSITY

信息科学技术学院 郭炜

“const” 关键字 的用法



河南郭亮村

1) 定义常量

```
const int MAX_VAL = 23;
```

```
const string SCHOOL_NAME = "Peking University";
```

2) 定义常量指针

□ 不可通过常量指针修改其指向的内容

```
int n,m;  
const int * p = &n;  
* p = 5;  
n = 4;  
p = &m;
```

2) 定义常量指针

□ 不可通过常量指针修改其指向的内容

```
int n,m;  
const int * p = &n;  
* p = 5;    //编译出错  
n = 4;  
p = &m;
```


2) 定义常量指针

□ 不可通过常量指针修改其指向的内容

```
int n,m;  
const int * p = &n;  
* p = 5;    //编译出错  
n = 4;      //ok  
p = &m;
```

2) 定义常量指针

□ 不可通过常量指针修改其指向的内容

```
int n,m;  
const int * p = &n;  
* p = 5;    //编译出错  
n = 4;      //ok  
p = &m;     //ok, 常量指针的指向可以变化
```

2) 定义常量指针

□ 不能把常量指针赋值给非常量指针，反过来可以

```
const int * p1; int * p2;  
p1 = p2;      //ok  
p2 = p1;      //error  
p2 = (int * ) p1; //ok, 强制类型转换
```

2) 定义常量指针

- 函数参数为常量指针时，可避免函数内部不小心改变参数指针所指地方的内容

```
void MyPrintf( const char * p )  
{  
    strcpy( p, "this" ); //编译出错  
    printf( "%s", p );    //ok  
}
```

3) 定义常引用

□ 不能通过常引用修改其引用的变量

```
int n;  
const int & r = n;  
r = 5; //error  
n = 4; //ok
```

下面说法哪种是对的？

- ☐ A 常引用所引用的变量，其值不能被修改
- ☒ B 不能通过常量指针，去修改其指向的变量
- ☐ C 常量指针一旦指向某个变量，就不能再指向其他变量
- ☐ D 以上都不对

提交



北京大学
PEKING UNIVERSITY

信息科学技术学院 郭炜

动态内存分配



庐山如琴湖

用new 运算符实现动态内存分配

□ 第一种用法，分配一个变量：

P = new T;

T是任意类型名，P是类型为T * 的指针。

动态分配出一片大小为 sizeof(T)字节的内存空间，并且将该内存空间的起始地址赋值给P。比如：

```
int * pn;  
pn = new int;  
* pn = 5;
```


用new 运算符实现动态内存分配

□ 第二种用法,分配一个数组:

P = new T[N];

T :任意类型名

P :类型为T * 的指针

N :要分配的数组元素的个数, 可以是整型表达式

动态分配出一片大小为 `sizeof(T)*N`字节的内存空间, 并且将该内存空间的起始地址赋值给P。

用new 运算符实现动态内存分配

□ 动态分配数组示例：

```
int * pn;  
int i = 5;  
pn = new int[i * 20];  
pn[0] = 20;  
pn[100] = 30; //编译没问题。运行时导致数组越界
```

用delete运算符释放动态分配的内存

- 用 “new” 动态分配的内存空间，一定要用 “delete” 运算符进行释放

delete 指针; //该指针必须指向new出来的空间

```
int * p = new int;
```

```
* p = 5;
```

```
delete p;
```

```
delete p; //导致异常，一片空间不能被delete多次
```

用delete运算符释放动态分配的数组

□ 用 “delete” 释放动态分配的数组，要加 “[]”

delete [] 指针; //该指针必须指向new出来的数组

```
int * p = new int[20];  
p[0] = 1;  
delete [] p;
```

表达式 “new int” 的返回值类型是：

- ☐ A int
- ☒ B int *
- ☐ C int &
- ☐ D void

提交

下面小段程序，哪个是正确的：

- ☐ A

```
char * p = new int;  
p = 'a';  
delete p;
```
- ☐ B

```
int *p = new int[25];  
p[10] = 100;  
delete p;
```
- ☒ C

```
char * p = new char[10];  
p[0] = 'K';  
delete [] p;
```
- ☐ D 都不对

提交



北京大学
PEKING UNIVERSITY

信息科学技术学院 郭炜

内联函数
函数重载
函数缺省参数



新加坡金沙酒店无边泳池

内联函数

- 函数调用是有时间开销的。如果函数本身只有几条语句，执行非常快，而且函数被反复执行很多次，相比之下调用函数所产生的这个开销就会显得比较大。
- 为了减少函数调用的开销，引入了内联函数机制。编译器处理对内联函数的调用语句时，是将整个函数的代码插入到调用语句处，而不会产生调用函数的语句。

内联函数

```
inline int Max(int a,int b)
{
    if( a > b) return a;
    return b;
}
```

函数重载

- 一个或多个函数，名字相同，然而参数个数或参数类型不相同，这叫做函数的重载。

- 以下三个函数是重载关系：

```
int Max(double f1, double f2) { }
```

```
int Max(int n1, int n2) { }
```

```
int Max(int n1, int n2, int n3) { }
```

- 函数重载使得函数命名变得简单。
- 编译器根据调用语句中的实参的个数和类型判断应该调用哪个函数。

函数重载

(1) `int Max(double f1, double f2) { }`

(2) `int Max(int n1, int n2) { }`

(3) `int Max(int n1, int n2, int n3) { }`

`Max(3.4, 2.5);` //调用 (1)

`Max(2, 4);` //调用 (2)

`Max(1, 2, 3);` //调用 (3)

`Max(3, 2.4);` //error, 二义性

函数的缺省参数

- C++中，定义函数的时候可以让最右边的连续若干个参数有缺省值，那么调用函数的时候，若相应位置不写参数，参数就是缺省值。

```
void func( int x1, int x2 = 2, int x3 = 3)
{ }
```

`func(10) ; //等效于 func(10,2,3)`

`func(10,8) ; //等效于 func(10,8,3)`

`func(10, , 8) ; //不行,只能最右边的连续若干个参数缺省`

函数的缺省参数

- 函数参数可缺省的目的在于提高程序的可扩充性。
- 即如果某个写好的函数要添加新的参数，而原先那些调用该函数的语句，未必需要使用新增的参数，那么为了避免对原先那些函数调用语句的修改，就可以使用缺省参数。

下面说法正确的是：

- ☐ A 多个重载函数的参数个数必须不同。
- ☐ B 两个函数，参数表相同，返回值类型不同，它们是重载关系。
- ☐ C 调用一个第二个和第三个参数都有有缺省值的函数时，可以不写第二个实参而写第三个实参。
- ☒ D 使用内联函数的目的是提高程序的运行速度。