本文作者：黄邦勇帅

本文是学习 C++的附加内容，主要介绍了 C++中的 string 类的各种成员函数，及成员函数的功能与作用，是作为学习 C++的参考使用的。
本文内容完全属于个人见解与参考文现的作者无关，其中难免有误解之处，望指出更正。

主要参考文献：

1、C++.Primer.Plus.第五版.中文版 [美]Stephen Prata 著 孙建春 韦强译 人民邮电出版社 2005 年 5 月
2、C++.Primer.Plus.第四版.中文版 Stanley B.Lippman、Barbara E.Moo 著 李师贤等译 人民邮电出版社 2006 年 3 月
3、C++.Primer.Plus.第三版.中文版 Stanley B.Lippman 等著 潘爱民 张丽译 中国电力出版社 2002 年 5 月
4、C++入门经典 第三版 [美]Ivor Horton 著 李予敏译 清华大学出版社 2006 年 1 月
5、C++参考大全 第四版 [美]Herbert Schidt 著 周志荣 朱德芳 于秀山等译 电子工业出版社 2003 年 9 月
6、21 天学通 第四版 C++ [美]Jesse Liberty 著 康博创作室 译 人民邮电出版社 2002 年 3 月

第 20 章 string 类

1、 string 类用于处理字符串，要使用 string 类需要包含 string 头文件。
2、 注意 string 是一个类，它具有类的特性，也就是说 string 类有构造函数，有重载的操作符，有成员函数。string 对象可以自动调整大小，但有一些限制，string 对象有个最大允许的长度，该长度由 string 类的静态常量 string::nops 设定，通常是最大的 unsigned int 值，在 vc++中被设为-1。
3、 string 类是模板类 basic_string 类的 char 具体化版本，basic_string 类的原型为：template<class charT, class traits=char_traits<charT>, class Allocator=allocator<charT> > class basic_string{….}; 对于 string 类具有预定义的具体化版本 typedef basic_string<char> string;也就是说 string 是 basic_string 模板类的 char 具体化版本的别名。
4、 string 类的 size_type 类型，size_type 是 string 中的配套类型，一般被定义为 unsigned 类型。可以使用限定名的方法来使用 size_type 类型，比如 string::size_type a;
5、 string 类的构造函数：string 类有 6 种形式的构造函数，string 类是 basic_string 类的特化版本，因此他的构造函数就是 basic_string 模板类的 char 特化版本的构造函数，在这里我们省略掉其他复杂的形式，得到以下的 string 构造函数版本，具体的 basic_string 模板类的构造函数这里不讨论。
   a、 **string(const char *s);**将 string 对象初始化为 s 指向的传统 C 字符串(即以空字符结束的字符串)。比如 string one(“hyong”);将 string 对象 one 用字符串 hyong 来初始化。这就意味着可以把 char 类型的数组转换为 string 对象，比如 string a; char b[]=”ldki”; 则 a=b; string c=b;都是正确的。但不能将 string 对象的字符串转换为 char 类型的数组。
   b、string(size_type n,char c);创建一个包含 n 个元素的对象，其中每个元素都被初始化为字符 c。比如 string two(10, ‘c’);将 string 对象 two 初始化包含为十个字符 c。这意味着不可以用单个字符来初始化 string 类型的对象，比如 string a=’s’;将发生错误。
   c、string();创建一个默认的 string 对象，长度为 0。比如 string three;即表示创建一个长度为 0 的字符串对象 three。
   d、**string(const string &str, size_type pos=0, size_type n=npos);**将 string 对象初始化为 string 对象的字符串从 pos 开始到结尾的字符，或从 pos 开始的 n 个字符。
   e、string(const char *s, size_type n);将 string 对象初始化为 s 指向的传统 C 字符串中的前 n 个字符，即使超出了字符串的范围，操作仍会进行。比如 char all[]=”hyong”; string five(all, 5);表示用字符数组 all 的前 5 个字符来初始化 string 对象 five。注意即使复制的长度超出了数组的长度，操作仍将进行，也就是说如果把 5 改为 10 的话，将导致 5 个无用的字符被复制到对象中。
   f、tempalte<class Iter>string(Iter begin, Iter end);将 string 对象初始化为[begin, end]间的字符，其中 begin 和 end 就像指针，用于指定位置，范围包括 begin 在内，但不包括 end，注意不包括 end。还要注意 begin 和 end 被看着指针，也就是说 char all[ ]=”hyongilfmm”; string seven(all+2, all+4)将使用从第 all+2 的字符 o 开始到第 all+3 的字符 n 初始化对象 seven，最后 seven 为”on”。因为字符串从 0 开始计数，所以这里从第 3 个字符开始。而 all+3 指的是第 4 个字符，注意 all+4 指的是第 5 个字符，这里不以第 5 个字符 g 结束，因为该构造函数不包括 end 在内。还要注意，对 begin 和 end 被看着指针指向某一位置，也就是说 string a=”hyong”;string b(a+3, a+5);是错误的，因为 a 是 string 类对象，不是指针，所以 a+3 没有意义，所以这里应这样 string b(&a[3], &a[5])这里 a[3]是一个 char 值，&a[3]是一个地址。
6、 注意，不能用单个字符来初始化 string 对象，比如 string a=’d’;将是错误的，但是 string a(1,’d’)是正确的，这里将把’d’初始化为”d”。

7、 string 对象只能接收以空字符结尾的字符串，比如语句 char b[ ]={‘a’, ‘b’, ‘c’}; string a=b;将是错误的。

8、 string 对象的输入：可以使用 cin 和 getline 函数对 string 对象进行输入，注意不能使用 get 函数，getline()函数读取字符到 string 对象中直到达到以下条件时停止：1、到达文件尾，这种情况下输入流的 eofbit 将被设置。2、遇到分界符。3、读取的字符达到 string 对象的最大允许值，这时将设置 failbit。

9、 string 类的输入函数 getline()：该函数的原型为：template<class charT, class traits, class Allocator> basic_istream<charT, traits>& getline(basic_istream<charT, traits>& is, basic_string<charT, traits, Allocator>& str, charT delim);该函数表示，对于 string 类，简省的 getline 函数为 string & getline(istream& is, string& str, char delim);表示把输入流 is 中的字符读入到字符串 str 中，直到遇到分界符 delim，到达文件尾或到达字符串的最大长度，分界符 delim 将被读取并删除。第二个版本的 getline()函数没有最后的分界符参数。注意这个版本的 getline 函数与前面介绍的不一样。调用这个版本的 getline 函数是直接调用，比如 string a; getline(cin, a, ‘z’);把输入流 cin 中的字符读入到 string 对象 a 中，直到遇到分界符 z 为止。而这些调用都是错误的，a.getline(cin,a,’z’)；cin.getline(cin, a, ‘z’)；另一点就是第一个参数必须要是一个输入流的对象，在这里输入流是标准输入流 cin。还要注意的是这个版本的 getline 函数只能适用于 string 对象，而 string 对象也只能使用这个版本的 getline 函数，比如这些使用方法都是错误的，char a[3]; getline(cin, a);错误字符数组 a 无法转换为 string 对象，string a; cin.getline(a, 4);试图从输入流中读取 4 个字符到 string 对象 a 中，该函数的原型接受的是 char 类型的数组，所以也存在无法将 string 对象 a 转换为 char 类型的错误，string 对象只能使用 现在介绍的这个 getline()版本。

10、string 类重载的操作符如下：string 重载了关系运算符，赋值运算符，[]运算符，+加运算符。对于关系运算符，数字小于大写字符，大写字符小于小写字符。重载的[]运算符使得 string 对象可以 像使用数组一样。重载的+运算符可以让两个 string 对象相加，**注意对于+运算符操作数的两边必须有一个是 string 类型的对象，不能将两个字符串直接相加，**比如 string b=”iwf”; string a=”hyon”+”onfg”+b;就是错误的，但 string a=”hyon”+(”onfg”+b);就是正确的。

11、string 类的成员函数：

11.1、find()方法：find 方法有 几个重载版本，分别如下，在这里介绍的 find 函数原型是 basic_string 模板类的 char 特化版本 string 的原型，也就是 说对basic_string 模板类中的 find()函数作了简化。

a、size_type find(const string &str, size_type pos=0) const 从位置 pos 开始查找子字符串 str，如果找到，则反回该子字符串 首次出现时其首字符的索引，否则反回 string::npos，由前所述 npos 是类 string 中的静态常量，表示 string 对象能 存储的最大长度。比如 string a=”hyongilsdfmgkd”; string b=”ong”; int c; c=a.find(b,1)表示从 string 对象 a 的第 1 个位置开始即字符 y 开始，注意这符串以 0 开始，查找 string 对象 b 中的字符串”ong”如果找到则反回起始位置，如果没找到则反回 string::pos。

b、size_type find(const char *s, size_type pos=0) const 从位置 pos 开始查找子字符串 s，如果找到，则反回该子字符串首次出现时其首字符的索引，否则反回 string::npos，这个版本的 find 函数和第一个版本一样，只不过第一个参数是 char 类型的字符数组，而不是 string 对象。

c、size_type find(const char *s, size_type pos, size_type n) 从 string 类型的字符串的 pos 位置开始，查找 char 类型的数组 s 的前 n 个字符组 成的子字符串，如果找到，则反回该子字符串首次出现时其首字符的索引，否则反回 string::npos。注意这个版本的 find 函数只适用于 char 类型的数组，而不适用于 string 对象，比如 string a=”hyongsldjfoidfj”; string b=”ong”; char c[]=”ong”;int d;则 d=a.find(c,1,3)则表示从 string 对象的第 1 个位置开始查找由字符数组的前 3 个字符组 成的字符串，如果找到则反回起始位置，如果没找到则反回 string::pos。但是语句 d=a.find(b,1,3)将发生错误，因为 string 对象 b 不是字符数组，所以发生错误。还要注意的是函数的第二个参数 pos 指的是调用该函数的 string 对象的位置，比如 string a=”edklkkk”; char b[]=”jidkik”; int c=a.fing(b,2,2);这里并不是从 string 类型 a 的字符串的 开始位置查找由char 类型的数组 b 从位置 2 开始的两个字符”dk”，而是指从 string 类型的对象 a 的第 2 个位置 k 开始查找由 char 类型的数组 b 的前两个字符”ji”，所以最后未找到这两个字符而反回 -1。

d、size_type find(char ch, size_type pos=0) const 从位置 pos 开始查找字符 ch，如果找到，则反回该字符首次出现时的位置，否则反回 string::npos

11.2、at()成员函数：at()函数的 功能与[ ]操作符相似，都是用于 访问string 对象中的 单一的字符，区别在于 at()函数会执行边界检查，而[ ]操作符不会。其原型为 reference at(size_type n);const_reference at(size_type n) const;第一个函数用于 检索或更改字符串的值，第二个函数用于 const 对象，只能用于 检索其值。调用方法为:string a=”hyong”; cout<<a.at(1);输出下标为 1 的单个字符 y。

11.3、substr()成员函数：该函数用于 反回原始字符串的子字符串。其原型为 basic_string substr(size_type pos=0, size_type n=npos) const;表示 反回由 pos 开始的 n 个字符，其默认参数为反回整个字符串。调用方法为：string a=”hyongkdlsi”; cout<<a.substr(2, 5);表示 反回从下标 2 开始的 5 个字符 ongkd，如果指定的字符数超过了 string 对象的结尾，则 substr()函数只 反回从指定位置到string 对象尾的字符。如果指定的起始位置超出了 string 对象的结尾，则抛出一个异常。

对于以下的成员函数要注意的是 string 是 basci_string 模板类的 char 特化版本，也就是说如果是 string 对象则类型 charT 应被替换为 char，basic_string 应被替换为 string。

11.4、compare()成员函数：该函数用于比 较两个字符串的大小，其功能与使用关系运算符进行比较相似。其比较规则为如果第一个字符串大于第二个字符串则反回一个大于 0 的值，如果第一个字符串小于第二个字符串的值则反回一个小于 0 的值，如果两个字符串相等则反回 0，如果第一个字符串比第二个字符串短则与第二个字符串的前部分相同，则反回一个小于 0 的值。注意 compare()函数是 string 类的成员函数，所以使用 compare 函数时 应使用 string 类的对象 来调用。compare()函数的 五个原型为:

a、int compare(const basic_string &str) const;比如 string a="hyongldfki"; string b="hyongilfdke"; int c=a.compare(b);表示把 string 对象 a 与对象 b 进行比较，比较的结果是 a 小于 b 函数反回一个小于 0 的值。

b、int compare(size_type pos1, size_type n1, const basic_string &str) const;将第一个字符串 从pos1 位置开始的 n1 个字符与第二个字符串进行比 较比如 string a="ahyongldlkfi"; b="hyongldiekf"; int c=a.compare(1,5,b);将 string 对象 a 从第标号为 1 开始的 5 个字符与第二个字符串 相比，结果对象 a 小于对象 b，反回一个小于 0 的值。

c、int compare(size_type pos1,size_type n1, const basic_string &str, size_type pos2, size_type n2) const;将第一个字符串从 pos1 开始的 n1 的字符，与第二个字符串 从pos2 开始的 n2 个字符进行比 较。比如 string a="ahyongidlfjm"; b="adhyongldi"; int c=compare(1,5,b,2,5);将对象 a 从位置 1 开始的 5 个字符与对象 b 从位置 2 开始的 5 个字符进行比较，结果两个对象相等，反回值 0。

d、int compare(const charT *s) const;这个版本与第 1 个版本 相同，只不过该版本用于 char 类型的数组。

e、int compare(size_type pos1, size_type n1, const charT* s, size_type n2=npos) const;这个版本与第 3 个版本 相同，只不过这个版本用于 char 类型的数组。注意，这里的函数原型有 4 个参数，只适 合于 char 类型的数组对象，对于 string 的对象就是错误的用法，比如 string a="hyong"; string b="kdl"; int c=a.compare(1,2,b,2);就是错误的，这里试图将 string 对象 a 从位置 1 开始的 2 个字符与 string 对象 b 的前两个字符进行比较，这里就错在 string 对象 b 与 compare 的函数原型中的 charT*类型不 相符合。

f、对于 compare() 函数可以直接用字符串进行比较，比如 a.compare("dfoei")； a.compare(1,2,"dkife")；a.compare(1,2,"dosdife",3); a.compare(1,2,"dfsl",1,2);都是正确的。

11.5、append()成员函数：该函数用于将一个字符串 追加到另一个字符串的后面 其使用功效与使用+=相似。append()函数有以下 几个原型:

a、basic_string &append(const basic_string& str);将 str 的内容追加到原字符串的末尾。

b、basic_string& append(const basic_string& str, size_type pos, size_type n);

c、template<class InputIterator> basic_string& append(InputIterator first, IputIterator last);

d、basic_string& append(const charT* s);

e、basic_string& append(const charT* s, size_type n);

f、basic_string& append(size_type n, charT c);

11.6、assign()成员函数：该函数用于将一个字符串 赋给另一个字符串 功能与使用=赋值运算符相似。assign()函数有以下几个原型:

a、basic_string& assign(const basic_string& str); 用字符串 str 替换原有字符串的内容，比如 string a="hyong"; string b="hemi"; a.assign(b);表示用 string 类型 b 替换掉 a 的内容，最后 a 为"hemi"。

b、basic_string& assign(const basic_string& str, size_type pos, size_type n);使用 str 中位置从 pos 开始的 n 个字符 替换原有 内容，比如 string a="hkel"; string b="hyong"; a.assign(b, 2,3);使用从 b 的位置 2 开始的 3 个字符"ong"替换原有的 内容，最后 a 的内容为"ong"。

c、basic_string& assign(const charT* s, size_type n);使用 char 类型的数组的前 n 个字符 替换原有的内容，比如 string a="kdi"; char b[ ]="hyongk"; a.assign(b, 3);表示用数组 b 的前 3 个字符"hyo"替换掉 a 的内容，最后 a 为"hyo"，这里要注意，该函数的第一个参数 charT*只适用于 char 类型的数组，如果传递给第一个的参数是 string 类型的对象则会出错。

d、basic_string& assign(const charT* s);使用 char 类型的以空字符结尾的字符串 替换原有的内容

e、basic_string& assign(size_type n, charT c);使用 n 个字符 c 替换原有的内容。

f、template<class InputIterator> basic_string& assign(InputIterator first, InputIterator last);

11.7、insert()成员函数：该函数可以将一个字符串 插入到一个string 对象中，该函数与 append()类似，不过 insert()接受另一个 指定插入位置的参数，该参数可以是 位置也可以是迭代器。数据被插入到插入点的前面。insert()函数有以下 几个原型:

a、basic_string& insert(size_type pos1, const basic_string& str);将 string 类型的字符串 插入到pos1 的位置的前面，比如 string a="hyongdk"; string b="u"; a.insert(2,b);将字符串 b 中的字符 插入到a 中的第 2 个位置 o 的前面，最后 a 为"hyaongdk"。

b、basic_string& insert(size_type pos1,const basic_string& str, size_type pos2, size_type n);将 string 类型的字符串 str 从位置 pos2 开始的 n 个字符 插入到string 类型的对象 位置在pos1 前面的位置中。比如 string a="hyong"; string b="iklfmgie"; a.insert(2,b,3,2);表示将 string 类型的字符串 b 从位置 3 开始的 2 个字符"fm"插入到 string 类型的字符串 a 在位置 2 的前面最后 a 成为"hyfmong"。注意：该语句中 basic_string& str 也可以用于 char 类型的数组，因为 char 类型的数组可以转换为 sring 对象。

c、basic_string& insert(size_type pos, const charT* s, size_type n);表示将 由char 类型的字符数组的前 n 个字符 插入到 string 类型的对象中在 位置pos 的前面。比如 string a="hyong"; char b[ ]="ikldke"; a.insert(2, b, 3);表示将字符数组的前 3 个字符"ikl"插入到 string 对象的 位置2 的前面，最后 a 成为"hyiklong"。

d、basic_string& insert(size_type pos, const charT* s);表示将 整个char 类型的数组 插入到string 对象中的 位置在pos 的前面。

e、basic_string& insert(size_type pos, size_type n, charT c);插入 n 个字符 c 到 string 类型对象 位置在pos 的前面。比如 string a="hyong"; a.insert(2,4,'c');插入 4 个字符 c 到 a 的位置 2 的前面，最后 a 成为"hyaaaaong"。

f、iterator insert(iterator p, charT c=charT());

g、void insert(iterator p, size_type n, charT c);

h、template<class InputIterator>void insert(iterator p, InputIterator first, InputIterator last);

11.8、erase()成员函数：该函数从字符串中删除字符。erase()函数有以下几个原型。

a、basic_string& erase(size_type pos=0, size_type n=npos);从位置 pos 开始删除 n 个字符或删除到字符串尾。

b、iterator erase(iterator position);删除迭代器位置引用的字符，并反回指向下一个元素的迭代器，如果后面没有其他元素则反回 end();

c、iterator earse(iterator first, iterator last);删除区间[first, last)中的字符，即删除从包括 first 开始的位置到不包括 last 的位置之间的字符。它反回一个迭代器，该迭代器指向最后一个被删除的元素后面的一个元素。

11.9、replace()成员函数：该函数用于替换字符串的内容。有如下几个原型:

a、basic_string& replace(size_type pos1, size_type n1, const basic_string& str);将 string 类型的对象从位置pos1 开始的 n1 个字符替换为string 类型的对象 str，比如 string a=”hyongkdik”; string b=”ffff”; a.replace(5,4,b);把 a 从位置 5 开始的 4 个字符”kdik”替换为 ffff，最后 a 成为”hyongffff”。

b、basic_string& replace(size_type pos1, size_type n1, const basic_string& str, size_type pos2, size_type n2);表示把 string 对象从pos1 开始的 n1 个字符替换为字符串str 从 pos2 开始的 n2 个字符。比如 string a=”hyongdkk”; string b=”hemi”; a.replace(5,3,b,2,2);将 a 从位置 5 开始的 3 个字符”dkk”替换为 b 从位置 2 开始的 2 个字符”mi”，最后 a 为”hyongmi”。

c、basic_string& replace(size_type pos,size_type n1,const charT* s, size_type n2);表示把 string 对象从位置pos 开始的 n1 个字符替换为char 数组类型的前 n2 个字符。

d、basic_string& replace(size_type pos, size_type n1, const charT* s);表示把 string 对象从位置pos 开始的 n1 个字符替换为 char 数组类型的字符。

e、basic_string& replace(size_type pos, size_type n1, size_type n2, charT c);表示把 string 对象从位置pos 开始的 n1 个字符替换为n2 个 char 类型的字符 c。比如 string a=”hyongldkfi”; a.replace(5,5,3,’s’);表示把 a 从位置 5 开始的 5 个字符”ldkfi”替换为 3 个字符 s，最后 a 为”hyongsss”。

f、basic_string& replace(iterator i1, iterator i2, const basic_string& str);

g、basic_string& replace(iterator i1, iterator i2, const charT* s, size_type n);

h、basic_string& replace(iterator i1, iterator i2, const charT* s);

i、basic_string& replace(iterator i1, iterator i2, size_type n, charT c);

j、template<class InputIterator> basic_string& replace(iterator i1, iterator i2, InputIterator j1, InputIterator j2);

11.10、copy()成员函数：该函数用于将 string 对象或其中一部分复制到指定的字符串数组中，其原型为：size_type copy(charT* s, size_type n, size_type pos=0) const;将 string 对象中从位置pos 开始的地方复制 n 个字符到指定的数组 s 中。函数反回复制的字符数，该函数不追加空值字符，且不进行数组的长度的检查。

11.11、swap()函数：该函数交换两个string 对象的内容，其原型为：void swap(basic_string<charT, traits, Allocator>&);

11.12、length(), size()成员函数：这两个成员函数功能相同，使用不带参数的length,size 函数能反回string 对象中的实际元素数。

11.13、capacity()成员函数：该成员函数反回 string 对象的容量，也就是说一个string 对象最多能存储多少个字符，这个数字可能大于 size()，length()的数目，capacity()-size()表示 string 对象还能存储多少个字符。

12、C 风格字符串：c 风格字符串是以空字符 null 结束的字符数组。比如 char a[ ]={‘a’,’b’}就不是一个 C 风格字符串，因为该数组不是以空字符结束的字符数组而 char b[ ]={‘a’,’b’,’\0’}则是 C 风格的字符串。再如 char a[3]=”abc”;将发生数组溢出的错误，因为字符串”abc”有 4 个字符，最后还有一个空字符’\0’，反以数组 a 应该声明为有 4 个成员的数组才对，即 char a[4]=”abc”。

13、C 风格字符串的标准库函数，在使用 C 风格字符串的库函数时须要包含头文件#include <cstring>，传递给 C 风格字符串的库函数的参数都必须是指针或者地址，调用这些函数时可以直接调用，具体内容如下：

a、strlen(s);反回 s 的长度，不包括字符结束符 null。注意该函数处理的对象一定是要以空字符null 结束的字符数组，比如 char a[ ]={‘a’,’b’}; strlen(a);这时就会出错，因为当调用 strlen 函数时，系统将从实参 a 指向的地址开始搜索，直到遇到 null 结束符为止，strlen()反回这一段空间的长度，在该例中由于数组a 没有 null 结束符，所以最后的结果是无法预料的。

使用以下的库函数时一定要确定数组 s1 的大小有足够的空间，还要注意最后的 null 结束符要占一个位置。

b、strcmp(s1,s2);比较字符串 s1 和 s2，如果 s1 大于 s2 反回正数，如果 s1 小于 s2 反回负数，如果 s1 等于 s2 反回 0。

c、strcat(s1,s2);将字符串 s2 连接到 s1 后，并反回 s1。

d、strncat(s1,s2,n);将 s2 的前 n 个字符连接到s1 后面，并反回 s1。

e、strcpy(s1,s2);将 s2 复制给 s1，并反回 s1。

f、strncpy(s1,s2,n);将 s2 的前 n 个字符复制给s1，并反回 s1。

14、智能指针 auto_ptr 类：

智能指针的作用：智能指针能够防止使用动态内存 new 分配时忘记使用 delete 释放指针所指向的对象的情况，还可以防止两个指针指向同一个对象时一个指针释放了该对象而另一个指针还认为该对象存在的情况。比如 void f(){string *p=new string(“dk”); ….return;}在函数 f 结束时就忘记使用了 delete 语句来释放指针 p 所指向的对象。再比如 void f(){string *p=new string(“dk”); string *p1; p1=p; delete p;cout<<*p1;}在函数 f 中，把动态指针 p 赋给指针 p1，这时指针 p 和 p1 指向同一个对象，但语句 delete p 删除了他所指向的内容，而指针 p1 这时并不知道指针p 指向的内容已经不存在了，这时指针p1 指向了一个不存在的对象，所以语句 cout<<*p1;将是错误的。智能指针就能防止以上两种错误的情况出现。

15、auto_ptr 类是一个 模板类，其原型为：template<class X> class auto_ptr{public: explicit auto_ptr(X* p=0)throw();……}; 其使用方法为：auto_ptr< double> p1(new double);这里 new double 分配的内存空间反回的指针被传递给 auto_ptr 类构造函数的形参 p，在这里就为 p1 动态分配了一个智能指针的存储空间，使用 auto_ptr 类声明的指针有和常规指针相似的特征，但可以不用使用 delete 语句来释放内存空间，auto_ptr 类会执行这个操作。例如 void f(){auto_ptr<string> p1(new string("dk"));….return;}在函数结 束时不再需要delete 语句，auto_ptr 类会自动执行这一操作。

16、使用 auto_ptr 类的注意事项：

a、使用 auto_ptr 类需要包含头文件 memory，且不需再使用 delete 语句。

b、auto_ptr 类的构造函数是显示的，即 explicit 类型，也就是 说不存在从指针到auto_ptr 对象的 隐式转换，以下的使用都将是错误的： double *p=new double(2); auto_ptr<double> pd=p; 错误，不存在隐式转换，不可以这样 初始化 auto_ptr 对象；

17、使用计数器自已实现智能指针类的方法：该方法可以 解决类中含有指针对象的情况。

a、首先应将该智能指针类中的计数器与类的对象相关联，计数器跟踪该类有多少个对象共享同一个指针，当计数器为 0 时删除对象。比如将类 hyong 声明为智能指针类 u_ptr 的友元。

b、每次创建类的新对象时，初始化指针，并将 计数器设为 1。

c、当对象作为另一对象的副本而创建时，复制构造函数复制指针并增加与之相应的计数器的值。

d、对一个对象进行 赋值时，赋值操作符减少左操作数所指对象的计数器的值(如果计数器为 0，则删除对象)，并增加右操作数所指对象的计数器的值。

e、最后，调用析构函数时，析构函数减少计数器的值，如果计数器减至 0，则删除基础对象。

具体例子如下

例：定义智能指针类：

```
class u_ptr{friend class hyong; //把要使用 智能指针的类hyong 声明为智能指针类的友元。
int *ip; size_t use;   //声明一个计数器。
u_ptr(int *p):ip(p),use(1){} //定义带一个参数的构造函数，并将 计数器初始化为 1。
~u_ptr(){delete ip;}};//析构函数。
//带指针的类 hyong 的定义
class hyong{u_ptr *ptr;       int val;
public:
hyong(int *p,int i):ptr(new u_ptr(p)),val(i){} //定义构造函数
hyong(const hyong& orig):ptr(orig.ptr),val(orig.val){++ptr->use} //复制构造函数，当复制一个对象时增加计数器的值。
hyong& operator=(const hyong&   rhs )   //重载赋值操作符
{++rhs.ptr->use; //将右操作数的计数器加 1，该操作可以防止自身赋值。
 if (--ptr->use= =0) delete ptr; //将左操作数的计数器减 1，如果为 0 则释放该指针指向的对象。
ptr = rhs.ptr ; val=rhs.val; return *this;} //复制对象，并反回这个对象。
~hyong(){if (--ptr->use = =0)   delete ptr;}};//调用 析构函数，调用一 次就将计数器减1，如果计数器为 0 就释放指针指向的对象。
```

作者：黄邦勇帅