

# string类

郭 炜 刘家瑛

北京大学 程序设计实习



# string类

- **string 类** 是一个模板类, 它的定义如下:  
`typedef basic_string<char> string;`
- 使用string类要包含头文件 `<string>`
- string对象的初始化:
  - `string s1("Hello");`    // 一个参数的构造函数
  - `string s2(8, 'x');`    // 两个参数的构造函数
  - `string month = "March";`

# string类

- 不提供以**字符和整数**为参数的构造函数

- 错误的**初始化方法：

- `string error1 = 'c';` // 错
- `string error2('u');` // 错
- `string error3 = 22;` // 错
- `string error4(8);` // 错

- 可以将字符赋值给string对象

- `string s;`  
`s = 'n';`



# string类 程序样例

```
#include <iostream>
#include <string>
using namespace std;
int main(int argc, char* argv[]){
    string s1("Hello");
    cout << s1 << endl;
    string s2(8, 'x');
    cout << s2 << endl;
    string month = "March";
    cout << month << endl;
    string s;
    s='n';
    cout << s << endl;
    return 0;
}
```

程序输出:

Hello

xxxxxxx

March

n

# string类

- 构造的string太长而无法表达 → 会抛出length\_error异常
- string 对象的长度用成员函数 length()读取;
  - string s("hello");  
cout << s.length() << endl;
- string 支持流读取运算符
  - string s; cin >> s;
- string 支持getline函数
  - string s;  
getline(cin, s);

# string的赋值和连接

## 用 '=' 赋值

- string s1("cat"), s2;

s2 = s1;

## 用 assign成员函数复制

- string s1("cat"), s3;

s3.assign(s1);

## 用 assign成员函数部分复制

- string s1("catpig"), s3;

s3.assign(s1, 1, 3);

//从s1 中下标为1的字符开始复制3个字符给s3

# string的赋值和连接

- 单个字符复制

```
s2[5] = s1[3] = 'a';
```

- 逐个访问string对象中的字符

```
string s1("Hello");
```

```
for(int i=0; i<s1.length(); i++)
```

```
    cout << s1.at(i) << endl;
```

- 成员函数at会做范围检查, 如果超出范围, 会抛出out\_of\_range异常, 而下标运算符不做范围检查

# string的赋值和连接

## 用 + 运算符连接字符串

```
string s1("good "), s2("morning!");
```

```
s1 += s2;
```

```
cout << s1;
```

## 用成员函数 append 连接字符串

```
string s1("good "), s2("morning!");
```

```
s1.append(s2);
```

```
cout << s1;
```

```
s2.append(s1, 3, s1.size()); //s1.size(), s1字符数
```

```
cout << s2;
```

```
//下标为3开始, s1.size()个字符
```

```
//如果字符串内没有足够字符, 则复制到字符串最后一个字符
```



# 比较string

## 用关系运算符比较string的大小

- ==, >, >=, <, <=, !=
- 返回值都是bool类型, 成立返回true, 否则返回false
- 例如:

```
string s1("hello"), s2("hello"), s3("hell");  
bool b = (s1 == s2);  
cout << b << endl;  
b = (s1 == s3);  
cout << b << endl;  
b = (s1 > s3);  
cout << b << endl;
```

输出:

1  
0  
1

# 比较string

- 用成员函数compare比较string的大小

```
string s1("hello"), s2("hello"), s3("hell");  
int f1 = s1.compare(s2);  
int f2 = s1.compare(s3);  
int f3 = s3.compare(s1);  
int f4 = s1.compare(1, 2, s3, 0, 3);    //s1 1-2; s3 0-3  
int f5 = s1.compare(0, s1.size(), s3);    //s1 0-end  
cout << f1 << endl << f2 << endl << f3 << endl;  
cout << f4 << endl << f5 << endl;
```

# 比较string

## 输出

```
0 // hello == hello
```

```
1 // hello > hell
```

```
-1 // hell < hello
```

```
-1 // el < hell
```

```
1 // hello > hell
```

# 子串

- 成员函数 `substr()`

```
string s1("hello world"), s2;
```

```
s2 = s1.substr(4,5); //下标4开始5个字符
```

```
cout << s2 << endl;
```

输出:

o wor

# 交换string

- 成员函数 `swap()`

```
string s1("hello world"), s2("really");  
s1.swap(s2);  
cout << s1 << endl;  
cout << s2 << endl;
```

输出:  
really  
hello world



# string的特性

- ▲ 成员函数 `capacity()`  
返回无需增加内存即可存放的字符数
- ▲ 成员函数 `maximum_size()`  
返回string对象可存放的最大字符数
- ▲ 成员函数 `length()`和`size()`相同  
返回字符串的大小/长度
- ▲ 成员函数 `empty()`  
返回string对象是否为空
- ▲ 成员函数 `resize()`  
改变string对象的长度



# string的特性

```
string s1("hello world");
cout << s1.capacity() << endl;
cout << s1.max_size() << endl;
cout << s1.size() << endl;
cout << s1.length() << endl;
cout << s1.empty() << endl;
cout << s1 << "aaa" << endl;
s1.resize(s1.length()+10);
cout << s1.capacity() << endl;
cout << s1.max_size() << endl;
cout << s1.size() << endl;
cout << s1.length() << endl;
cout << s1 << "aaa" << endl;
s1.resize(0);
cout << s1.empty() << endl;
```

# string的特性

```
31 // capacity
4294967293 // maximum_size
11 // length
11 // size
0 // empty
hello worldaaa // string itself and "aaa"
31
4294967293
21
21
hello worldaaa
1
```

不同编译器上可能会不一样



# 寻找string中的字符

## 成员函数 `find()`

- `string s1("hello world");`

`s1.find("lo");`

//在s1中从前向后查找“lo”第一次出现的地方

//如果找到, 返回“lo”开始的位置, 即 l 所在的位置下标

//如果找不到, 返回 `string::npos` (`string`中定义的静态常量)

## 成员函数 `rfind()`

- `string s1("hello world");`

`s1.rfind("lo");`

//在s1中从后向前查找“lo”第一次出现的地方

//如果找到, 返回“lo”开始的位置, 即 l 所在的位置下标

//如果找不到, 返回 `string::npos`

# 寻找string中的字符

## 成员函数 `find_first_of()`

- `string s1("hello world");`

`s1.find_first_of("abcd");`

//在s1中从前向后查找“abcd”中任何一个字符第一次出现的地方  
//如果找到, 返回找到字母的位置; 如果找不到, 返回 `string::npos`

## 成员函数 `find_last_of()`

- `string s1("hello world");`

`s1.find_last_of("abcd");`

//在s1中查找“abcd”中任何一个字符最后一次出现的地方  
//如果找到, 返回找到字母的位置; 如果找不到, 返回 `string::npos`



# 寻找string中的字符

## 成员函数 `find_first_not_of()`

- `string s1("hello world");`

```
s1.find_first_not_of("abcd");
```

//在s1中从前向后查找不在“abcd”中的字母第一次出现的地方  
//如果找到, 返回找到字母的位置; 如果找不到, 返回 `string::npos`

## 成员函数 `find_last_not_of()`

- `string s1("hello world");`

```
s1.find_last_not_of("abcd");
```

//在s1中从后向前查找不在“abcd”中的字母第一次出现的地方  
//如果找到, 返回找到字母的位置; 如果找不到, 返回 `string::npos`

# 寻找string中的字符

```
string s1("hello world");  
cout << s1.find("ll") << endl;  
cout << s1.find("abc") << endl;  
cout << s1.rfind("ll") << endl;  
cout << s1.rfind("abc") << endl;  
cout << s1.find_first_of("abcde") << endl;  
cout << s1.find_first_of("abc") << endl;  
cout << s1.find_last_of("abcde") << endl;  
cout << s1.find_last_of("abc") << endl;  
cout << s1.find_first_not_of("abcde") << endl;  
cout << s1.find_first_not_of("hello world") << endl;  
cout << s1.find_last_not_of("abcde") << endl;  
cout << s1.find_last_not_of("hello world") << endl;
```

输出:

2	4294967295
9	4294967295
1	4294967295
11	4294967295
0	4294967295
10	4294967295

# 替换string中的字符

## 成员函数erase()

```
string s1("hello world");  
s1.erase(5);  
cout << s1;  
cout << s1.length();  
cout << s1.size();
```

// 去掉下标 5 及之后的字符

输出:  
hello55

# 替换string中的字符

## 成员函数 `find()`

```
string s1("hello world");  
cout << s1.find("l", 1) << endl;  
cout << s1.find("ll", 2) << endl;  
cout << s1.find("ll", 3) << endl;
```

//分别从下标1, 2, 3开始查找“l”

输出:

2 2 9

# 替换string中的字符

## 成员函数 `replace()`

```
string s1("hello world");  
s1.replace(2,3, "haha");  
cout << s1;
```

//将s1中下标2 开始的3个字符换成“haha”

输出:  
hehaha world

# 替换string中的字符

## 成员函数 `replace()`

```
string s1("hello world");  
s1.replace(2,3, "haha", 1,2);  
cout << s1;
```

//将s1中下标2 开始的3个字符  
//换成“haha” 中下标1开始的2个字符

输出:  
heah world



# 在string中插入字符

## 成员函数 insert()

```
string s1("hello world");  
string s2("show insert");
```

```
s1.insert(5, s2); // 将s2插入s1下标5的位置
```

```
cout << s1 << endl;
```

```
s1.insert(2, s2, 5, 3);
```

```
//将s2中下标5开始的3个字符插入s1下标2的位置
```

```
cout << s1 << endl;
```

输出:

```
helloshow insert world  
heinslloshow insert world
```

# 转换成C语言式char \*字符串

- 成员函数 `c_str()`

```
string s1("hello world");  
printf("%s\n", s1.c_str());
```

```
// s1.c_str() 返回传统的const char * 类型字符串  
// 且该字符串以 '\0' 结尾
```

输出:  
hello world

# 转换成C语言式char \*字符串

## 成员函数data()

```
string s1("hello world");  
const char * p1=s1.data();  
for(int i=0; i<s1.length(); i++)  
    printf("%c",*(p1+i));
```

输出:  
hello world

//s1.data() 返回一个char \* 类型的字符串  
//对s1 的修改可能会使p1出错。

# 转换成C语言式char \*字符串

## 成员函数copy()

```
string s1("hello world");  
int len = s1.length();  
char * p2 = new char[len+1];  
s1.copy(p2, 5, 0);  
p2[5]=0;  
cout << p2 << endl;  
// s1.copy(p2, 5, 0) 从s1的下标0的字符开始,  
//制作一个最长5个字符长度的字符串副本并将其赋值给p2  
//返回值表明实际复制字符串的长度
```

输出:  
hello