

### 1.string 类的构造函数：

string(const char \*s); //用 c 字符串 s 初始化

string(int n,char c); //用 n 个字符 c 初始化

此外，string 类还支持默认构造函数和复制构造函数，如 string s1；string s2="hello"；都是正确的写法。当构造的 string 太长而无法表达时会抛出 length\_error 异常

### 2.string 类的字符操作：

const char &operator[](int n)const;

const char &at(int n)const;

char &operator[](int n);

char &at(int n);

operator[]和 at()均返回当前字符串中第 n 个字符的位置，但 at 函数提供范围检查，当越界时会抛出 out\_of\_range 异常，下标运算符[]不提供检查访问。

const char \*data()const; //返回一个非 null 终止的 c 字符串数组

const char \*c\_str()const; //返回一个以 null 终止的 c 字符串

int copy(char \*s, int n, int pos = 0) const; //把当前串中以 pos 开始的 n 个字符拷贝到以 s 为起始位置的字符数组中，返回实际拷贝的数目

### 3.string 的特性描述：

int capacity()const; //返回当前容量（即 string 中不必增加内存即可存放的元素个数）

int max\_size()const; //返回 string 对象中可存放的最大字符串的长度

int size()const; //返回当前字符串的大小

int length()const; //返回当前字符串的长度

bool empty()const; //当前字符串是否为空

void resize(int len,char c); //把字符串当前大小置为 len，并用字符 c 填充不足的部分

string 类的输入输出操作：

string 类重载运算符 operator>>用于输入，同样重载运算符 operator<<用于输出操作。

函数 getline(istream &in,string &s);用于从输入流 in 中读取字符串到 s 中，以换行符'\n'分开。

### 4.string 的赋值：

string &operator=(const string &s); //把字符串 s 赋给当前字符串

string &assign(const char \*s); //用 c 类型字符串 s 赋值

string &assign(const char \*s,int n); //用 c 字符串 s 开始的 n 个字符赋值

string &assign(const string &s); //把字符串 s 赋给当前字符串

string &assign(int n,char c); //用 n 个字符 c 赋值给当前字符串

string &assign(const string &s,int start,int n); //把字符串 s 中从 start 开始的 n 个字符赋给当前字符串

string &assign(const\_iterator first,const\_iterator last); //把 first 和 last 迭代器之间的部分赋给字符串

### 5.string 的连接：

string &operator+=(const string &s); //把字符串 s 连接到当前字符串的结尾

string &append(const char \*s); //把 c 类型字符串 s 连接到当前字符串结尾

string &append(const char \*s,int n); //把 c 类型字符串 s 的前 n 个字符连接到当前字符串结尾

string &append(const string &s); //同 operator+=()

string &append(const string &s,int pos,int n); //把字符串 s 中从 pos 开始的 n 个字符连接到当前字符串的结尾

string &append(int n,char c); //在当前字符串结尾添加 n 个字符 c

string &append(const\_iterator first,const\_iterator last); //把迭代器 first 和 last 之间的部分连接到当前字符串的结尾

### 6.string 的比较：

bool operator==(const string &s1,const string &s2)const; //比较两个字符串是否相等

运算符">","<",">=","<=","!="均被重载用于字符串的比较；

```

int compare(const string &s) const;//比较当前字符串和 s 的大小
int compare(int pos, int n,const string &s)const;//比较当前字符串从 pos 开始的 n 个字符组成的字符串
与 s 的大小
int compare(int pos, int n,const string &s,int pos2,int n2)const;//比较当前字符串从 pos 开始的 n 个字
符组成的字符串与 s 中 pos2 开始的 n2 个字符组成的字符串的大小
int compare(const char *s) const;
int compare(int pos, int n,const char *s) const;
int compare(int pos, int n,const char *s, int pos2) const;
compare 函数在>时返回 1 , <时返回-1 , ==时返回 0
string 的子串 :
string substr(int pos = 0,int n = npos) const;//返回 pos 开始的 n 个字符组成的字符串
string 的交换 :
void swap(string &s2); //交换当前字符串与 s2 的值

```

#### 7.string 类的查找函数 :

```

int find(char c, int pos = 0) const;//从 pos 开始查找字符 c 在当前字符串的位置
int find(const char *s, int pos = 0) const;//从 pos 开始查找字符串 s 在当前串中的位置
int find(const char *s, int pos, int n) const;//从 pos 开始查找字符串 s 中前 n 个字符在当前串中的位置
int find(const string &s, int pos = 0) const;//从 pos 开始查找字符串 s 在当前串中的位置
//查找成功时返回所在位置 , 失败返回 string::npos 的值
int rfind(char c, int pos = npos) const;//从 pos 开始从后向前查找字符 c 在当前串中的位置
int rfind(const char *s, int pos = npos) const;
int rfind(const char *s, int pos, int n = npos) const;
int rfind(const string &s,int pos = npos) const;
//从 pos 开始从后向前查找字符串 s 中前 n 个字符组成的字符串在当前串中的位置 , 成功返回所在位置 ,
失败时返回 string::npos 的值
int find_first_of(char c, int pos = 0) const;//从 pos 开始查找字符 c 第一次出现的位置
int find_first_of(const char *s, int pos = 0) const;
int find_first_of(const char *s, int pos, int n) const;
int find_first_of(const string &s,int pos = 0) const;
//从 pos 开始查找当前串中第一个在 s 的前 n 个字符组成的数组里的字符的位置。查找失败返回
string::npos
int find_first_not_of(char c, int pos = 0) const;
int find_first_not_of(const char *s, int pos = 0) const;
int find_first_not_of(const char *s, int pos,int n) const;
int find_first_not_of(const string &s,int pos = 0) const;
//从当前串中查找第一个不在串 s 中的字符出现的位置 , 失败返回 string::npos
int find_last_of(char c, int pos = npos) const;
int find_last_of(const char *s, int pos = npos) const;
int find_last_of(const char *s, int pos, int n = npos) const;
int find_last_of(const string &s,int pos = npos) const;
int find_last_not_of(char c, int pos = npos) const;
int find_last_not_of(const char *s, int pos = npos) const;
int find_last_not_of(const char *s, int pos, int n) const;
int find_last_not_of(const string &s,int pos = npos) const;
//find_last_of 和 find_last_not_of 与 find_first_of 和 find_first_not_of 相似 , 只不过是 从后向前查找

```

#### 8.string 类的替换函数 :

```

string &replace(int p0, int n0,const char *s);//删除从 p0 开始的 n0 个字符 , 然后在 p0 处插入串 s
string &replace(int p0, int n0,const char *s, int n);//删除 p0 开始的 n0 个字符 , 然后在 p0 处插入字
符串 s 的前 n 个字符
string &replace(int p0, int n0,const string &s);//删除从 p0 开始的 n0 个字符 , 然后在 p0 处插入串 s

```

```

string &replace(int p0, int n0,const string &s, int pos, int n);//删除 p0 开始的 n0 个字符，然后在 p0 处插入串 s 中从 pos 开始的 n 个字符
string &replace(int p0, int n0,int n, char c);//删除 p0 开始的 n0 个字符，然后在 p0 处插入 n 个字符 c
string &replace(iterator first0, iterator last0,const char *s);//把[first0 , last0 ) 之间的部分替换为字符串 s
string &replace(iterator first0, iterator last0,const char *s, int n);//把[first0 , last0 ) 之间的部分替换为 s 的前 n 个字符
string &replace(iterator first0, iterator last0,const string &s);//把[first0 , last0 ) 之间的部分替换为串 s
string &replace(iterator first0, iterator last0,int n, char c);//把[first0 , last0 ) 之间的部分替换为 n 个字符 c
string &replace(iterator first0, iterator last0,const_iterator first, const_iterator last);//把[first0 , last0 ) 之间的部分替换成[first , last ) 之间的字符串

```

#### 9.string 类的插入函数：

```

string &insert(int p0, const char *s);
string &insert(int p0, const char *s, int n);
string &insert(int p0,const string &s);
string &insert(int p0,const string &s, int pos, int n);
//前 4 个函数在 p0 位置插入字符串 s 中 pos 开始的前 n 个字符
string &insert(int p0, int n, char c);//此函数在 p0 处插入 n 个字符 c
iterator insert(iterator it, char c);//在 it 处插入字符 c，返回插入后迭代器的位置
void insert(iterator it, const_iterator first, const_iterator last);//在 it 处插入[first , last ) 之间的字符串
void insert(iterator it, int n, char c);//在 it 处插入 n 个字符 c

```

#### 10.string 类的删除函数

```

iterator erase(iterator first, iterator last);//删除[first , last ) 之间的所有字符，返回删除后迭代器的位置
iterator erase(iterator it);//删除 it 指向的字符，返回删除后迭代器的位置
string &erase(int pos = 0, int n = npos);//删除 pos 开始的 n 个字符，返回修改后的字符串

```

#### 11.string 类的迭代器处理：

string 类提供了向前和向后遍历的迭代器 iterator，迭代器提供了访问各个字符的语法，类似于指针操作，迭代器不检查范围。

用 string::iterator 或 string::const\_iterator 声明迭代器变量，const\_iterator 不允许改变迭代的内容。常用迭代器函数有：

```

const_iterator begin()const;
iterator begin(); //返回 string 的起始位置
const_iterator end()const;
iterator end(); //返回 string 的最后一个字符后面的位置
const_iterator rbegin()const;
iterator rbegin(); //返回 string 的最后一个字符的位置
const_iterator rend()const;
iterator rend(); //返回 string 第一个字符位置的前面
rbegin 和 rend 用于从后向前的迭代访问，通过设置迭代器
string::reverse_iterator,string::const_reverse_iterator 实现

```

#### 12.字符串流处理：

通过定义 ostream 和 istream 变量实现，<sstream>头文件中

例如：

```

string input("hello,this is a test");
istream is(input);
string s1,s2,s3,s4;

```

```
is>>s1>>s2>>s3>>s4;//s1="hello,this",s2="is",s3="a",s4="test"  
ostringstream os;  
os<<s1<<s2<<s3<<s4;  
cout<<os.str();
```