构造函数（ (constructor)）

接口说明

```
explicit basic_string(const A& al = A());
    basic_string(const basic_string& rhs);
    basic_string(const basic_string& rhs, size_type pos, size_type n,
        const A& al = A());
    basic_string(const E *s, size_type n, const A& al = A());
    basic_string(const E *s, const A& al = A());
    basic_string(size_type n, E c, const A& al = A());
    basic_string(const_iterator first, const_iterator last,
        const A& al = A());
```

① basic_string() （重点）

构造空的string类对象，即空字符串

②basic_string(const _Elem *_Ptr)

用C-string来构造string类对象

③basic_string(size_type _Count, _Elem _Ch)

string类对象中包含n个字符c

④basic_string(const _Myt& _Right)

拷贝构造函数

⑤clear （重点）

清空有效字符

```
string s("Hello");
    s.clear();
    cout<<s.size()<<endl;  //0
    cout<<s.capacity()<<endl;  //15
```

①

```
string s;
```

②

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

③

```
string s2(10, 'a');
```

④

```
string s2(10, 'a');
string s3(s2);
```

```
basic_string& operator=(const basic_string& rhs);
    basic_string& operator=(const E *s);
    basic_string& operator=(E c);
    iterator begin();
    const_iterator begin() const;
    iterator end();
    const_iterator end() const;
    reverse_iterator rbegin();
    const_reverse_iterator rbegin() const;
    reverse_iterator rend();
    const_reverse_iterator rend() const;
    const_reference at(size_type pos) const;
```

```
reference at(size_type pos);
```

```
string s("Hello");
for(int i=0; i<s.size(); ++i)
        cout<<s.at(i);
```

```
const_reference operator[](size_type pos) const;
reference operator[](size_type pos);
```

```
string s("Hello");
    cout<<s<<endl;   //1
    for(int i=0; i<s.size(); ++i)
        cout<<s[i];
```

```
const E *c_str() const;
```

```
string s("Hello");  //15
    cout << s.c_str() << endl;  //Hello
    cout<<strlen(s.c_str())<<endl;  //0
```

c_str会将string对象转换为C语言中的字符串变量

```
const E *data() const;
```

```
size_type length() const;
size_type size() const;
size_type capacity() const;
```

```
string s("Hello");
    cout<<s.size()<<endl;  //5
    cout<<s.length()<<endl;  //5
    cout<<s.capacity()<<endl;  //15
```

size和length都不算'\0'

capacity是由系统决定的

```
size_type max_size() const;
```

```
string s;
    cout <<"s.max_size = "<< s.max_size() << endl;  //4294967294
```

max_size的值是4294967294

```
    void resize(size_type n, E c = E());
string s;
    cout<<s.size()<<endl;  //0
    cout<<s.capacity()<<endl;  //15

    s.resize(100);
    cout<<s.size()<<endl;  //100
    cout<<s.capacity()<<endl;  //111

    s.resize(10);
    cout<<s.size()<<endl;  //10
    cout<<s.capacity()<<endl;  //111
```

resize直接改变的是size的大小，capacity大小会自动适应

capacity大才大，小不小

```
string s("Hello");
s.resize(10); //Hello00000
```

只给第一个参数时默认用0补

```
string s("Hello");
s.resize(10, 'x'); //Helloxxxxx
```

给定第二个参数时就用第二个参数补要增容却空缺的位

```
    void reserve(size_type n = 0);
string s;
    s.reserve(100); ////////
    cout << s.size() << endl;  //0
    cout << s.capacity() << endl;  //111
```

```
    bool empty() const;
    basic_string& operator+=(const basic_string& rhs);
string s("Hello");  //15
    string s1("BitWorld.");
    char *str = "Bit.";
```

```
    s += s1;
    cout<<s<<endl;  //HelloBitWorld.
```

```
    basic_string& operator+=(const E *s);
```

```
  string s("Hello");  //15
    char arr[] = " good!";
    s += arr;
    cout << s << endl;  //Hello good!
```

```
    basic_string& operator+=(E c);
```

```
  string s("Hello");  //15
    s += '!';
    cout << s << endl;
```

```
    basic_string& append(const basic_string& str);
    basic_string& append(const basic_string& str,
        size_type pos, size_type n);
    basic_string& append(const E *s, size_type n);
    basic_string& append(const E *s);
    basic_string& append(size_type n, E c);
    basic_string& append(const_iterator first, const_iterator
last);
    basic_string& assign(const basic_string& str);
    basic_string& assign(const basic_string& str,
        size_type pos, size_type n);
    basic_string& assign(const E *s, size_type n);
    basic_string& assign(const E *s);
    basic_string& assign(size_type n, E c);
    basic_string& assign(const_iterator first, const_iterator
last);
    basic_string& insert(size_type p0,
        const basic_string& str);
    basic_string& insert(size_type p0,
```

```
    const basic_string& str, size_type pos, size_type n);
basic_string& insert(size_type p0,
    const E *s, size_type n);
basic_string& insert(size_type p0, const E *s);
basic_string& insert(size_type p0, size_type n, E c);
iterator insert(iterator it, E c);
void insert(iterator it, size_type n, E c);
void insert(iterator it,
    const_iterator first, const_iterator last);
basic_string& erase(size_type p0 = 0, size_type n = npos);
iterator erase(iterator it);
iterator erase(iterator first, iterator last);
basic_string& replace(size_type p0, size_type n0,
    const basic_string& str);
basic_string& replace(size_type p0, size_type n0,
    const basic_string& str, size_type pos, size_type n);
basic_string& replace(size_type p0, size_type n0,
    const E *s, size_type n);
basic_string& replace(size_type p0, size_type n0,
    const E *s);
basic_string& replace(size_type p0, size_type n0,
    size_type n, E c);
basic_string& replace(iterator first0, iterator last0,
    const basic_string& str);
basic_string& replace(iterator first0, iterator last0,
    const E *s, size_type n);
basic_string& replace(iterator first0, iterator last0,
    const E *s);
basic_string& replace(iterator first0, iterator last0,
    size_type n, E c);
```

```
basic_string& replace(iterator first0, iterator last0,
    const_iterator first, const_iterator last);
size_type copy(E *s, size_type n, size_type pos = 0) const;
void swap(basic_string& str);
```

```
size_type find(const basic_string& str, size_type pos = 0)
const;
```

```
string s("HelxoBit.");

    size_t index = s.find(sub);  //5
    index = s.find(sub, 5);  //5
index = s.find(sub, 6);  //4294967295
    if(index == string::npos) //npos
        cout<<"Error."<<endl;
```

第一个参数是string类的对象，第二个参数是默认值为0的参数，表示调用它的string类对象的下标

```
size_type find(const E *s, size_type pos, size_type n) const;
```

```
string s("HelxoBit.");  //15
    string sub("Bi");
    size_t index = s.find("xo", 3, 1);
```

第一个参数是要查找的字符串常量，第二个参数是从string对象中第几个位置开始查找，第三个参数是要查询常字符串前几个字符

```
size_type find(const E *s, size_type pos = 0) const;
```

```
string s("HelxoBit.");
    index = s.find("xo");  //3
```

第一个参数是要查找的字符串常量，第二个参数是默认值为0从string对象中第几个位置开始查找

```
size_type find(E c, size_type pos = 0) const;
```

```
string s("HelxoBit."); //15
index = s.find('H', 0);
```

第一个参数是要查找的字符常量，第二个参数是从string对象中第几个位置开始查找

```
size_type rfind(const basic_string& str,
    size_type pos = npos) const;

size_type rfind(const E *s, size_type pos,
    size_type n = npos) const;

size_type rfind(const E *s, size_type pos = npos) const;

size_type rfind(E c, size_type pos = npos) const;


size_type find_first_of(const basic_string& str,
    size_type pos = 0) const;
size_type find_first_of(const E *s, size_type pos,
    size_type n) const;
size_type find_first_of(const E *s, size_type pos = 0) const;
size_type find_first_of(E c, size_type pos = 0) const;
size_type find_last_of(const basic_string& str,
    size_type pos = npos) const;
size_type find_last_of(const E *s, size_type pos,
    size_type n = npos) con/t;
size_type find_last_of(const E *s, size_type pos = npos) const;
size_type find_last_of(E c, size_type pos = npos) const;
size_type find_first_not_of(const basic_string& str,
```

```
        size_type pos = 0) const;
    size_type find_first_not_of(const E *s, size_type pos,
        size_type n) const;
    size_type find_first_not_of(const E *s, size_type pos = 0)
const;
    size_type find_first_not_of(E c, size_type pos = 0) const;
    size_type find_last_not_of(const basic_string& str,
        size_type pos = npos) const;
    size_type find_last_not_of(const E *s, size_type pos,
         size_type n) const;
    size_type find_last_not_of(const E *s,
        size_type pos = npos) const;
    size_type find_last_not_of(E c, size_type pos = npos) const;
    basic_string substr(size_type pos = 0, size_type n = npos)
const;
    int compare(const basic_string& str) const;
    int compare(size_type p0, size_type n0,
        const basic_string& str);
    int compare(size_type p0, size_type n0,
        const basic_string& str, size_type pos, size_type n);
    int compare(const E *s) const;
    int compare(size_type p0, size_type n0,
        const E *s) const;
    int compare(size_type p0, size_type n0,
        const E *s, size_type pos) const;
    A get_allocator() const;
```

```
int main()
{
   string s;
```

```
    //cin>>s;  //' '  \n
    getline(cin, s);  //  \n  jfla falfjla
    cout<<s<<endl;
    return 0;
}
```

cin遇到空格输入就中断了，因此string类的对象

在C++中本质上有两种getline函数，（称为第一种）一种在头文件<istream>中，是istream类的成员函数。

（称为第二种）一种在头文件<string>中，是普通函数。