介绍一下 自己学习时期 用vs写代码的一般套路

项目的一般文件分类

写项目一般 在vs中创建三个文件分别为：这里拿string项目做例

$$存储函数声明：string.h函数的实现部分：string.cpp项目测试文件：test.cpp$$

一般分为这三个文件、

我们分开讲解各个文件的作用

string.h 这个文件是用来存储函数声明，类的声明，结构体的声明。一般头文件的调用也写在这个文件中

string.cpp 这个文件是用来存储函数的实现，比如string.h中声明了 push_back函数，那么就需要在string.cpp中将他的功能实现出来。一般头文件只需要调用string.h即可

test.cpp 这个文件用来测试函数，主函数也要写在这里一般头文件只需要调用string.h即可 因为头文件是展开的，而在string..h中已经展开，如果再调用。不加防止重复引用头文件的指令就会报错了。

看了这三个文件的作用，那我们就先看一下实际操作把。比如我们要写一个通讯录（这里不对他进行详细写，只借用他来讲解一下步骤）：

```
// Phone.h中的代码：
#include<iostream>
using namespace std;
struct PhoneTxt {
    char* name;
    int id;
};
// 初始化
void Init(struct PhoneTxt& P);
```

```
//Phone.cpp中的代码
#include "Phone.h"
void Init(struct PhoneTxt& P)
{
    P.name = (char* )malloc(sizeof(char));
    P.id = 0;
}
```

```
//test.cpp中的代码
#include "Phone.h"
int main()
{
    struct PhoneTxt P;
    Init(P);
    return 0;
}
```

虽然上面的代码写的很挫但是很好理解啊

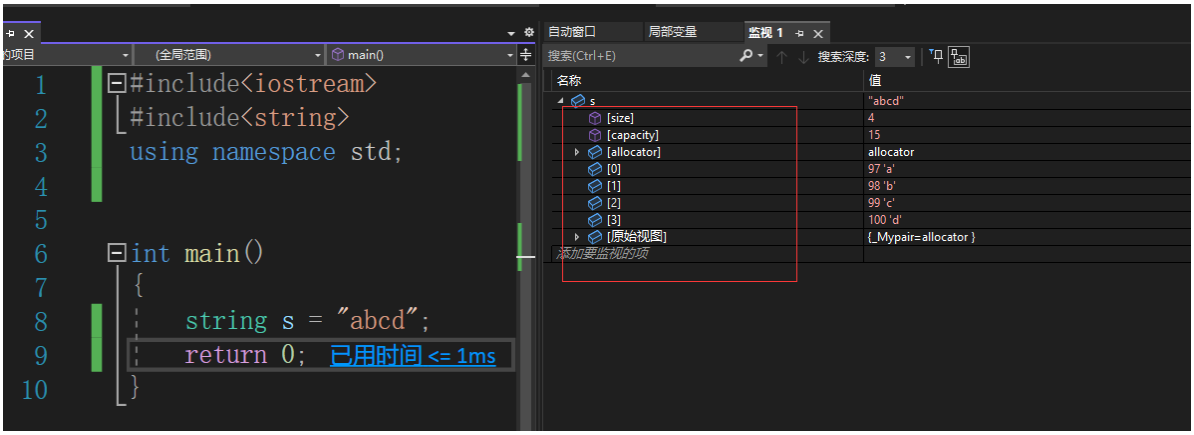
下面我们就开始正文把，开始讲解string的模拟实现把。下期我再写一个通讯录的模拟实现。

这里我说明一个东西熬，虽然我们一般分为三个，但是我们的函数代码功能部分可以写到string.h中，我们下面介绍的就是直接在,h中实现的部分

string的实现需要注意那些地方

一、string的成员变量究竟有哪些呢?

其实这里很容易就能知道。，我们直接用vs创建一个string对象，我们就可以直接监视对象，就可以了呀。看下图



其实我们这里很容易看出来，其实主要就是三个，其一：size 记录 数据的多少个，其二：capacity记录数据的容量 其三：就是记录数据

二、我们可以很轻松的直接写出来string的成员变量

下面看我代码

```
#pragma once
namespace wr {
    class string
    {
    private:
        char* _str;
        size_t _size;
        size_t _capacity;
    };
}
```

首先 因为库中有string ，所以我们如果再直接写一个string类的话便会报错。那么下一步，我们开始查文档。，看看string的各个类的接口，然后再来实现

三、我们看一下string有那些接口

1、构造函数

```
string(); // 这里是默认构造函数
string(const char* str); // 这里是一个字符串拷贝构造函数
string(const char* str,size_type length); // 以str为初值 (长度任意),
string(string& s, size_type index, size_type length ); // 这里是以index为索引开始的子串，长度为length
```

例如

```
    string str1( 5, 'c' );
    string str2( "Now is the time..." );
    string str3( str2, 11, 4 );
    cout << str1 << endl;
    cout << str2 << endl;
    cout << str3 << endl;
```

运行结果

```
ccccc
Now is the time...
time
```

经过这个用例应该很容易就能理解用法了。

二、运算符（operntor）

运算符的重载

```
== // 返回为bool
> // 返回为bool
< // 返回为bool
>= // 返回为bool
<=// 返回为bool
!= // 返回为bool
+ // 返回为string + string = string
+= // 同上
[] // 返回为char
```

介绍：你可以用 ==, >, <, >=, <=, 和 !=比较字符串. 可以用 + 或者 += 操作符连接两个字符串，并且可以用[]获取特定的字符.

三、尾插一个字符（push_back）

尾插

```
    string str1 = "War and Peace";
    str1.push_back('a');
    cout << str1 << endl;
```

打印结果为

```
War and Peacea
```

四、添加文本(append)

语法:

```
1.basic_string &append( const basic_string &str );
2.basic_string &append( const char *str );
3.basic_string &append( const basic_string &str, size_type index, size_type len
);
4.basic_string &append( const char *str, size_type num );
5.basic_string &append( size_type num, char ch );
6.basic_string &append( input_iterator start, input_iterator end );
```

功能:

1. 在字符串结尾添加字符串str
2. 在字符串结尾添加str
3. 在字符串结尾添加字符串str的字串,以index开始长度为len
4. 在字符串结尾添加str中的num个字符
5. 在字符串结尾添加num个ch
6. 在字符串的末尾添加以迭代器start开始以end结束的字符串序列

实操案例

```
    string str = "Hello World";
    str.append( 10, '!' );
    cout << str << endl;
```

运行结果

```
Hello World!!!!!!!!!
```

其实这里一般直接用+操作符来代替了就

五、begin () 和 end （)

语法

```
 iterator begin();
 iterator end();
```

介绍

```
begin()函数返回一个迭代器,指向字符串的第一个元素.
end()函数返回一个迭代器,指向字符串的最后一个元素的下一个.
```

```
### 六、容量(capacity)  和 大小  size
```

语法

```
size_t capacity(); size_t size();
```

介绍

```
capacity()函数返回在重新申请更多的空间前字符串可以容纳的字符数. 这个数字至少与 size()一样大.
size()表示当前存储了多少个数字
```

七、比较(compare)

语法

```
1.  int compare( const basic_string &str );
2.  int compare( const char *str );
3.  int compare( size_type index, size_type length, const basic_string &str );
4.  int compare( size_type index, size_type length, const basic_string &str,
size_type index2, size_type length2 );
5.  int compare( size_type index, size_type length, const char *str, size_type
length2 );
```

介绍

1. 比较自己和str,
2. 比较自己和str,
3. 比较自己的子串和str,子串以index索引开始，长度为length
4. 比较自己的子串和str的子串，其中index2和length2引用str，index和length引用自己
5. 比较自己的子串和str的子串，其中str的子串以索引0开始，长度为length2，自己的子串以index开始，长度为length

八、拷贝（copy）

语法

```
 size_type copy( char *str, size_type num, size_type index );
```

介绍

```
拷贝自己到str中以index开始的num个字符
```

九、data

语法

```
const char* data();
```

介绍

```
返回指向自己的第一个字符的指针
```

十、判空（empty）

语法

```
bool empty();
```

介绍

```
如果字符串为空则empty()返回真(true)，否则返回假(false).
```

十一、删除（erase）

语法

```
iterator erase( iterator start, iterator end );
basic_string &erase( size_type index = 0, size_type num = npos );
```

介绍

```
删除从start到end的所有字符，返回一个[迭代器](iterators.html),指向被删除的最后一个字符的下一个位置
删除从index索引开始的num个字符，返回*this.
```

十二。查找

语法

```
size_type find( const basic_string &str, size_type index );
size_type find( const char *str, size_type index );
size_type find( const char *str, size_type index, size_type length );
size_type find( char ch, size_type index );
```

介绍

1. 返回str在字符串中第一次出现的位置（从index开始查找）。如果没找到则返回**string::npos**
2. 返回str在字符串中第一次出现的位置（从index开始查找）。如果没找到则返回**string::npos**,
3. 返回str在字符串中第一次出现的位置（从index开始查找 长度为length）。如果没找到则返回**string::npos**,
4. 返回字符ch在字符串中第一次出现的位置（从index开始查找）。如果没找到就返回**string::npos**

   注：npos 是一个常量为 - 1

十三、插入（insert）

语法

```
  iterator insert( iterator i, const char &ch );
  basic_string &insert( size_type index, const basic_string &str );
  basic_string &insert( size_type index, const char *str );
  basic_string &insert( size_type index1, const basic_string &str, size_type
index2, size_type num );
  basic_string &insert( size_type index, const char *str, size_type num );
  basic_string &insert( size_type index, size_type num,在 char ch );
  void insert( iterator i, size_type num, const char &ch );
  void insert( iterator i, iterator start, iterator end );
```

用法

1. 在迭代器i表示的位置前面插入一个字符ch,
2. 在Index位置插入一个字符串str
3. 在Index位置插入一个字符串str
4. 在index1位置插入sr的字串从index2开始长度为num
5. 在字符串的位置index插入字符串str的num个字符,
6. 在Index位置插入num个字符ch
7. 在迭代器i表示的位置前面插入num个字符ch的拷贝,
8. 在迭代器i表示的位置前面插入一段字符，从start开始，以end结束.

十四、逆向迭代器 （rbegin ） （ rend）

语法

```
const reverse_iterator rbegin();
const reverse_iterator rend();
```

用法

返回一个逆向迭代器，指向字符串最后一个字符

rend()函数返回一个逆向迭代器，指向字符串的开头（第一个字符的前一个位置）

十五、替换(replace)

```
  basic_string &replace( size_type index, size_type num, const basic_string &str
);
  basic_string &replace( size_type index1, size_type num1, const basic_string
&str, size_type index2,size_type num2 );
  basic_string &replace( size_type index, size_type num, const char *str );
  basic_string &replace( size_type index, size_type num1, const char *str,
size_type num2 );
  basic_string &replace( size_type index, size_type num1, size_type num2, char
ch );
  basic_string &replace( iterator start, iterator end, const basic_string &str
);
  basic_string &replace( iterator start, iterator end, const char *str );
  basic_string &replace( iterator start, iterator end, const char *str,
size_type num );
  basic_string &replace( iterator start, iterator end, size_type num, char ch );
```

1. 用str中的num个字符替换本字符串中的字符,从index开始
2. 用str中的num2个字符（从index2开始）替换本字符串中的字符，从index1开始，最多num1个字符

3. 用str中的num个字符（从index开始）替换本字符串中的字符
4. 用str中的num2个字符（从index2开始）替换本字符串中的字符，从index1开始，num1个字符
5. 用num2个ch字符替换本字符串中的字符，从index开始
6. 用str中的字符替换本字符串中的字符,迭代器start和end指示范围
7. 用str中的num个字符替换本字符串中的内容,迭代器start和end指示范围，
8. 用num个ch字符替换本字符串中的内容，迭代器start和end指示范围.

其实这里会前面的 都很好理解的 毕竟十一个人设计的 肯定有他们自己喜欢的方法嘛

十六、保留空间(reserve)

语法

```
  void resize( size_type num );
  void resize( size_type num, char ch );
```

介绍

```
resize()函数改变本字符串的大小到num，新空间的内容不确定。也可以指定用ch填充。
```

十七、逆向查找rfind

和find的区别就是他是逆向的，而find是正向的，他返回的是从后往前第一个，也就是最后面的

十八、字符串截取substr

语法

```
basic_string substr( size_type index, size_type num = npos );
```

介绍

```
substr()返回本字符串的一个子串，从index开始，长num个字符。如果没有指定，将是默认值
string::npos。这样，substr()函数将简单的返回从index开始的剩余的字符串。
```

十九、交换(swap)

语法

```
  void swap( basic_string &str );
```

介绍

```
swap()函数把str和本字符串交换
```