C++17新特性std::string_view

背景：字符串的处理

四种不同字符串 C++

```cpp
// 指针指向静态字符串
const char *str_ptr = "this is a static string.";

// 字符串数组
char str_array[] = "this is a static string.";

// std::string
std::string str = "this is a static string.";

// std::string_view
std::string_view sv = "this is a static string.";
```

1. 静态字符串：会把指针指向静态存储区，字符串只读。若尝试修改，则会报segment fault.
2. 字符串数组：在栈上分配一块空间，将字符串拷入
3. `std::string`：在寄存器设置了字符串的起始指针，调用了`basic_string`的构造函数，中间会有各种检测和字符串拷贝工作，至少会有如下操作：
   a. 确定字符串长度（如`strlen`，遍历一遍字符串）
   b. 按字符串长度（或者预留更多的长度）新建一块内存空间
   c. 拷贝字符串到新建的内存空间（第二次遍历字符串）
4. `std::string_view`：单纯在栈上保存静态字符串的起始指针和长度，没有其它调用

`std::string_view`的实现

`std::string_view`类的成员变量只包含两个：字符串指针和字符串长度。

字符串指针可能是某个连续字符串的其中一段指针

字符串长度也不一定是整个字符串长度，也有可能是某个字符串的一部分长度

`std::string_view`所实现的接口中，完全包含了`std::string`的所有**只读接口**，所以在很多场景下可以轻易使用`std::string_view`来代替`std::string`

一个通常的用法是，生成一个std::string后，如果后续的操作不再对其进行修改，那么可以考虑把std::string转换为std::string_view，后续操作全部使用std::string_view来进行，这样字符串的传递变得更加轻量级

在很多实现上，`std::string`都是用引用计数进行COW方式管理，但是引用计数也会设计锁和原子计数器，而`std::string_view`的拷贝只是单纯拷贝两个数值类型变量（字符串指针以及其长度），效率远远高于前者

std::string_view高效的地方在于，它不管理内存，只保存指针和长度，所以对于只读字符串而言，查找和拷贝是相当简单的

`std::string_view`

构造函数

```
constexpr basic_string_view() noexcept;                                   (1)  (C++17 起)
constexpr basic_string_view( const basic_string_view& other ) noexcept
    = default;                                                            (2)  (C++17 起)
constexpr basic_string_view( const CharT* s, size_type count );          (3)  (C++17 起)
constexpr basic_string_view( const CharT* s );                            (4)  (C++17 起)
template< class It, class End >
constexpr basic_string_view( It first, End last );                        (5)  (C++20 起)
template< class R >
explicit constexpr basic_string_view( R&& r );                            (6)  (C++23 起)
constexpr basic_string_view( std::nullptr_t ) = delete;                   (7)  (C++23 起)
```

其中也可以直接传入`std::string`重载(2)来实现构建

赋值函数

直接使用`=`赋值（等号后面是`std::string_view`类型的变量/

迭代器

`begin` / `cbegin`： 返回指向起始位置的迭代器

`end` / `cend`： 返回指向结尾的迭代器

`rbegin` / `crbegin`： 返回指向起始的反向迭代器

`rend` / `crend`： 返回指向结尾的反向迭代器

元素访问

`operator[]`：访问指定自负

`at`：访问指定自负，带有边界检查

`front`：访问首个字符

`back`：访问最末字符

`data`：返回指向视图首字符的指针

容量

`size` / `length`：返回字符数

`max_size`：返回最大字符数

`empty`：检查视图是否为空

修改器

`remove_prefix`：以后移起点收缩时图

`remove_suffix`：以前移终点收缩时图

`swap`：交换内容

操作

`copy`：复制字符

`substr`：返回子串

`compare`：比较两个视图

`starts_with`：检查 `string_view` 是否始于给定前缀

`ends_with`：检查 `string_view` 是否终于给定后缀

`contains`：检查字符串视图是否含有给定的子串或字符

还有其它的一些查找函数

字面量

`operator""sv`：创建一个字符数组字面量的字符视图