Зебелян_Артем_БПМ_22_3_v231225_06

Зебелян Артём, гр. БПМ-22-3

29.12.2023

Тема: “Стандарстная библиотека - std::string_view”

Введение.

Конструкция std::string_view не самостоятельная строка, а отсылка к подстроке какой-то другой строки. Объекты такого класса не выделяют память для размещения символов строки и не копируют подстроки. Упрощённо их можно представлять себе просто как такие структуры:

```
struct
string_view
        {
        const char* start; // указатель на начало подстроки в чужой памяти
        size_t len; // длина этой подстроки

        };
```

String_view не копирует символы строкового объекта вне зависимости, что он представляет - тип std::string, строковый литерал или символьный массив. Поэтому в качестве типа параметра оптимальнее использовать **std::string_view**, а не константную ссылку const std::string&. При этом не важно, что параметр может передаваться по значению, а не по ссылке - копирование символов все равно не будет происходить. В своей внутренней реализации **std::string_view** лишь копирует длину строки и указатель на последовательность символов.

Рассмотрим следующий код:

```
#include <iostream>
#include <string>

int main() {
std::string s = "Hello, world! How do you do?";
auto sub = s.substr(7, 5); // тип sub — std::string
sub[0] = 'W';
std::cout << sub << "\n"; // World
// В исходной строке всё осталось по-прежнему:
std::cout << s << "\n";    // Hello, world! How do you do?
}
```

Здесь имеется исходная строка s, у которой мы выделяем подстрочку sub. Эта подстрока является самостоятельной копией фрагмента исходной строки. С одной стороны, это позволяет её независимо изменять. С другой стороны, если такие изменения не требуются, на такое копирование тратится время и память.

Перепишем теперь это через **string_view**:

```
#include <iostream>
#include <string>
#include <string_view>

int main() {
std::string s = "Hello, world! How do you do?";
std::string_view sv = s; // смотрим через string_view на строку s
auto sub = sv.substr(7, 5); // теперь тип sub — std::string_view
// sub[0] = 'W'; // изменять через string_view символы строки нельзя
std::cout << sub << "\n"; // world
std::cout << s << "\n"; // Hello, world! How do you do?
}
```

Обёртка string_view рассматривает исходную строчку как константную: она не позволяет её изменять. В то же время через string_view можно читать символы фрагмента исходной строки, а также можно брать новые подстроки и сужать длину подстроки:

```
#include <iostream>
#include <string>
#include <string_view>


int main() {
std::string s = "Hello, world! How do you do?";
std::string_view sv = s;
sv.remove_prefix(7); // убрали из sv первые 7 символов
sv.remove_suffix(16); // убрали последние 16 символов
std::cout << sv << "\n"; // World
std::cout << s << "\n"; // строка s при этом никак не изменилась
}
```

Гибкая инициализация.

Объект std::string_view может быть инициализирован множеством различных типов строк, что показано на следующем примере:

```
#include <iostream>
#include <string>
#include <string_view>

int main()
{
    std::string_view s1 { "Hello, world!" }; // initialize with C-style string literal
    std::cout << s1 << '\n';

    std::string s{ "Hello, world!" };
    std::string_view s2 { s };  // initialize with std::string
    std::cout << s2 << '\n';

    std::string_view s3 { s2 }; // initialize with std::string_view
    std::cout << s3 << '\n';

    return 0;
}
```

Применение.

Обёртка string_view позволяет избежать ненужных копирований и ускорить программу там, где ведётся работа с подстроками одной большой строки. Например, это могут быть фрагменты JSON- или XML-файлов. Однако важно помнить, что string_view валиден только при условии, что объемлющая строка жива. Например, следующая программа некорректна и приведёт к неопределённому поведению:

```
#include <iostream>
#include <string>
#include <string_view>
#include <vector>


int main() {
std::vector<std::string_view> lines;
for (int i = 0; i < 5; ++i) {
std::string line;
std::getline(std::cin, line);
lines.push_back(line); // string_view в векторе ссылается на память строки line
}
for (auto item : lines) {
std::cout << item << "\n"; // Ошибка! Все эти строки уже невалидны!
 }
}
```

Программа компилируется, но при запуске попадает в неопределённое поведение, так как исходные строки line выходят из своей области видимости после каждой итерации цикла, и обращаться к их памяти больше нельзя.

Источники.

[1] Тип std:string_view - https://metanit.com/cpp/tutorial/12.10.php

[2] Яндекс Хэндбук, основы C++ - https://education.yandex.ru/handbook/cpp/article/adapters-and-views#predstavlenie-std-string-view

[3] Introduction to std::string_view - https://www.learncpp.com/cpp-tutorial/introduction-to-stdstring_view/