std::string_viewを使用した文字列解析

高橋 晶 (Akira Takahashi)
faithandbrave@gmail.com
Preferred Networks, Inc.
2019/11/20 (水) C++ MIX #6

std::string_view

- C++17で導入された文字列を参照するクラス
- 文字列リテラルや、char配列の文字列に対して、
  std::stringが持っているような便利なメンバ関数を適用できる

```
// 先頭5文字を取り出す
cout << string_view{"Hello World"}.substr(0, 5) << endl;
// 「Hello」が出力される
```

- メモリ確保や文字列の（ディープ）コピーは起こらない
- ポインタのコピーと、文字列長の保持だけ

string_viewの特徴

- 内部ではconst char*で文字列を参照し、参照の開始位置と、文字列長をもっている
- 受け取った文字列に対して破壊的な変更はせず、できることは参照範囲を参照・変更するだけ
- 注：文字列リテラルの寿命はstaticなので、コピーしなくても寿命が尽きたりはしない

string_viewの基本的な使い道1
string, const char*を受け取るインタフェースの統一

```
// std::string, const char*, std::string_viewのどれでも受け取れる
void f(string_view sv) {
  cout << sv.substr(0, 3) << endl;
}

f("Hello");                 // 文字列リテラル

const char* chars = "Hello";
f(chars);                   // char配列

string s = "Hello";
f(s);                       // std::stringの左辺値

f(std::string("Hello")); // std::stringの一時オブジェクト
```

string_viewの基本的な使い道2
const string&, const char*を返すインタフェースの統一

```
class X {
  string s_;
  const char* chars_;
public:
  // 文字列をコピーでなく参照で返すなら、
  // string_viewによって統一的な返し方ができる
  string_view get_s() const { return s_; }
  string_view get_chars() const { return chars_; }
};
```

string_viewは文字列解析に便利

- string_viewは文字列の参照範囲をずらす、という単純な機能だが、コストが低いので気楽に操作できる
- C++20から追加されたstarts_with()メンバ関数
- C++17からあるremove_prefix()メンバ関数などを使うと、
- かんたんな文字列解析に便利
  - 検索インタフェースも、もちろんある

string_viewは文字列解析に便利

```
// [command][argument] という構文を解析
void parse(string_view sv) {
  const char* text_cmd = "[text]";
  if (sv.starts_with(text_cmd)) { // 先頭にあるコマンドを判定
    // 処理済みのコマンド部分を削除（参照範囲をうしろにずらす）
    sv.remove_prefix(string_view{text_cmd}.length());

    // カッコ [ ] を外す
    sv.remove_prefix(1); sv.remove_suffix(1);
    string_view arg = sv;
  }
  …
}
parse("[text][Hello!]");
parse("[change_scene][title]");
```

まとめ

- string_viewは文字列解析に便利！
- こういう解析は、stringだと部分文字列のためのメモリ確保コストがかかったり、イテレータだと操作がめんどうだったりで、たいへんだった
- 複雑な構文を解析すると単純にいかないのは変わらない
- 自分たちで設計する単純なDSL (ドメイン特化言語) を解析するなら、string_viewを使うことでお手軽に解析できる！
  - splitで済むなら、そっちの方がさらに簡単