stl(vector)の３種類のループの仕方

std::vectorの要素を繰り返し処理（ループ）する代表的な3つの方法（範囲ベースfor文、イテレータ、インデックス）

1. 範囲ベースfor文 (Range-based for loop) - C++11以降

最も現代的で、簡潔かつ安全な方法です。要素を直接参照したい場合に最適です。

```
std::vector<int> vec = {10, 20, 30};
// 参照(&)を使って値を変更可能にする、またはコピーを避ける
for (const auto& value : vec) {
    std::cout << value << std::endl;
}
```

- **メリット:** コードが非常に読みやすい。イテレータの begin/end を意識しなくて良い。
- **用途:** 全要素に対する単純な処理。

2. イテレータ (Iterator)

伝統的な手法で、コンテナの特定の範囲や、要素を削除・挿入しながらのループに有用です。

```
std::vector<int> vec = {10, 20, 30};
for (auto it = vec.begin(); it != vec.end(); ++it) {
    std::cout << *it << std::endl; // ポインタのようにデリファレンスしてアクセス
}
```

- **メリット:** 高機能。ループ途中で要素の削除 (`erase`) が可能。
- **用途:** 要素を削除しながらの走査、STLアルゴリズムとの併用。

3. インデックスによるループ (Index-based)

配列の添え字（0, 1, 2...）を使ってアクセスする方法です。

```
std::vector<int> vec = {10, 20, 30};
for (std::size_t i = 0; i < vec.size(); ++i) {
    std::cout << vec[i] << std::endl; // vec.at(i) も使用可能
}
```

- **メリット:** インデックス（位置情報）自体が必要な場合に便利。
- **用途:** インデックス番号を利用した計算、特定の範囲や飛び飛びの要素のアクセス。

まとめと選び方

方法	特徴・メリット	おすすめの用途
範囲ベースfor	最も簡単、安全、読みやすい	基本的な全要素ループ
イテレータ	柔軟性（要素の削除が可能）	削除・挿入が発生するループ
インデックス	位置（i）を直接扱える	i番目を計算に使う場合

通常は **1. 範囲ベースfor文** を使い、要素削除が必要なら **2. イテレータ** を選ぶのが一般的です。

Revision #2
★Created 25 February 2026 06:25:07 by youe2
Updated 2 June 2026 19:27:52 by youe2