C++20 射影変換

高橋 晶 (Akira Takahashi)
faithandbrave@gmail.com
Preferred Networks, Inc.
2025/06/20 (金) C++ breaktime 2025 / Summer

自己紹介

- Preferred Networks社で、スーパーコンピュータMN-Coreのソフトウェアを作っています
  - エミュレータとかアセンブラとかの低レイヤーなものを作ってます
- 通信系のゲームエンジンを作ってます
- C++の日本語リファレンスサイトcpprefjpを作っています
- 著書
  - 『C++テンプレートテクニック』
  - 『C++ポケットリファレンス』
  - 『プログラミングの魔導書』
- 東京で2〜3ヶ月ごとにC++ MIXという勉強会を開催しています

C++20ではアルゴリズムに射影変換という機能が追加されています

- 最近まで知りませんでした
- Range関係でいっしょに入った機能なので埋もれていました

アルゴリズムおさらい

```
// コンテナはもちろん、
vector<T> v;
auto it = find(
  v.begin(),
  v.end(),
  x);

// 組み込み配列にも使える
T ar[N];
auto p = find(
  ar,
  ar + N,
  x
);
```

- C++標準のアルゴリズムは、コンテナと分離され、イテレータという中間インタフェースを介して各要素にアクセスする

アルゴリズムおさらい

```
// コンテナはもちろん、
vector<T> v;
auto it = ranges::find(
  v,
  x);

// 組み込み配列にも使える
T ar[N];
auto p = ranges::find(
  ar,
  x
);
```

- C++20ではRangeに対応した

findとかcountってじつは不便

```
vector<T> v;
auto it = ranges::find(
  v,
  x);

int n = ranges::count(
  ar,
  x
);
```

- 特定の値を検索するfind
- 特定の値の個数を取得するcount
- これらはそんなに活躍しない
- コンテナの要素型や検索条件は
  そんなに単純ではないから

findとかcountってじつは不便

```
vector<T> v;
auto it = ranges::find_if(
  v,
  [](T x) { return x > 0; });

int n = ranges::count_if(
  ar,
  [](T x) {
    return x.kind == Weapon;
  }
);
```

- より細かい条件を設定できるfind_if / count_ifが使われがち

C++20 射影変換で値を変換できるようになった

```
vector<T> v;
auto it = ranges::find(
  v,
  x,        // xはnameの型
  &T::name); // nameの値を検索

int n = ranges::count(
  ar,
  Weapon,
  [](T x) {
    return x.kind;
  }
);
```

- より細かい条件を設定できるfind_if / count_ifが使われがち
- C++20では**射影変換（projection）**という機能が入り、値を変換した結果で検索ができるようになった
- メンバ変数ポインタ、メンバ関数ポインタ、関数オブジェクトなどを指定できる

変換時の値コピーに注意

```
vector<T> v;
auto it = ranges::find(
  v,
  x,
  // コピーされないよう型を指定
  [](T x) -> const string& {
    return x.name;
  }
);
```

- メンバポインタならパフォーマンス上の問題はないが、
- ラムダ式を指定する場合、パフォーマンス劣化に注意
- 戻り値の型を指定して、参照を返すようにしよう

変換時の値コピーに注意

```
vector<T> v;
auto it = ranges::find(
  v,
  x,
  // コピーされないよう型を指定
  [](T x) -> decltype(auto) {
    return (x.name);
  }
);
```

- decltype(auto)でもOK
- return文にカッコをつけないとコピーになるので注意

射影変換がサポートされているアルゴリズムか確認

```
template <input_range R,
          class Proj = identity,
          class T = projected_value_t<iterator_t<R>, Proj>>
constexpr borrowed_iterator_t<R>
  find(R&& r,
       const T& value,
       Proj proj = {});
```

- アルゴリズムが射影変換に対応しているかは、
  パラメータに「**Proj proj**」があるかで確認できる
- パラメータのどの値が変換されるかは、**projected_value_t**が
  使われているかで確認できる

今回は以上です！

- 射影変換は、いくつかの状況でコードをより単純化できます
- 複雑な条件が必要ない状況で便利に使えます