C++14 LWG.2148

# 列挙型のハッシュサポート

高橋 晶(Akira Takahashi)
faithandbrave@longgate.co.jp

2014/06/27(金) WG21 C++14 DISレビュー会議

# まえがき

- この資料は、C++14に取り入れられる予定の変更、LWG (Library Working Group)のIssue 2148のレビューです。
- 2148. Hashing enums should be supported directly by std::hash
  - http://www.open-std.org/jtc1/sc22/wg21/docs/lwg-defects.html#2148

# 概要

- std::unordered_mapのキーとして列挙型を指定可能にする。
- そのために、enumおよびenum classで定義されたあらゆる列挙型(列挙型コンセプトを満たすあらゆる型)を、std::hashクラスがサポートする。

# 規格の変更内容 1/2

- 20.9 [function.objects], <functional>のヘッダ概要

```
namespace std {
  […]
  // 20.8.12, hash function baseprimary template:
  template <class T> struct hash;
  […]
}
```

# 規格の変更内容 2/2

- 20.8.12 [unord.hash] のパラグラフ1

-1- The unordered associative containers defined in 23.5 [unord] use specializations of the class template hash as the default hash function. For all object types Key for which there exists a specialization hash<Key>, and for all enumeration types (7.2 [dcl.enum]) Key, the instantiation hash<Key> shall: […]

非順序連想コンテナは、デフォルトのハッシュ関数としてクラステンプレートhashの特殊化を使用する。hash<Key>の特殊化が存在するKey型の全てのオブジェクト、および全ての列挙型Keyについて、hash<Key>のインスタンス化は以下を満たさなければならない：

# どのように実装するか 1/2

- libc++は、以下のように実装している：

```
namespace std {
  template <class T> // プライマリテンプレート
  struct hash {
    static_assert(is_enum<T>::value,  “…” );

    size_t operator()(T x) const noexcept
    {
      using type = typename underlying_type<T>::type;
      return hash<type>{}(static_cast<type>(x));
    }
  };
}
```

# どのように実装するか 2/2

- std::hashのプライマリテンプレートを、enum用に使う。
- std::underlying_typeメタ関数を使用して、列挙型のベースとなる整数型を取得し、
- 整数型のstd::hash特殊化に転送する。

# 所感

- 今後、このようなコンセプトによる特殊化が標準ライブラリ内でさらに必要になった場合、std::hashのプライマリテンプレートが複雑になっていく。
  - 具体的には、コンセプトごとにハッシュ関数を切り替えるために、実装用の関数テンプレートを用意し、SFINAEもしくはConceptでオーバーロードしていくことになる。
- そのような提案が上がってきた場合には、std::hashクラステンプレートの仕様を整理する必要がある。
- Boostのように、オーバーロード可能な関数を実装として許可することが考えられる。

# 考えられる仕様

```
namespace std {
  template <Enumerable T>
  size_t hash_value(T x) noexcept;

  template <class T> // プライマリテンプレート
  struct hash {
   size_t operator()(T x) const noexcept
   {
     return hash_value(x); // 関数テンプレートに転送
   }
  };
}
```

- ADLについても考える必要がある。