**concurrent\_unordered\_map**

**concurrency::concurrent\_unordered\_map 클래스는 std::unordered\_map 클래스에 대응하는 병렬 컨테이너입니다.** key/value pair 원소를 컨테이너에 추가하여 key를 가지고 value 값을 검색하는 전형적인 dictionary 개념의 컨테이너죠. concurrent\_unordered\_map 클래스는 여러 스레드 또는 task 에서 컨테이너를 공유하여 동시에 원소를 추가하고 엑세스 하려는 경우에 유용하게 사용될 수 있습니다. 먼저 std::unordered\_map 과의 차이점부터 살펴보도록 하죠.

* erase, bucket, bucket\_count, bucket\_size 메서드의 이름이 unsafe\_erase, unsafe\_bucket, unsafe\_ bucket\_count, unsafe\_bucket\_size 로 각각 변경되었고, 이름에서 예상할 수 있다시피 이 메서드들은 thread-safe 하지 않다.
* 원소의 추가, 엑세스, iterator 엑세스, traversal 작업은 thread-safe 하다. 원소 추가 작업은 iterator를 무효화 시키지 않고 기존에 추가된 원소들의 순서를 변경시키지도 않는다.
* 원소 추가 작업은 equal\_range 메서드가 반환한 iterator를 무효화 시키거나 변경하지 않는다.

원소의 삭제와 관련된 메서드를 제외한 주요 메서드는 대부분 thread-safe 하기 때문에 여러 스레드에서 동시에 호출이 가능합니다. 이번에는 concurrent\_unordered\_map 중 thread-safe 한 메서드와 그렇지 않은 메서드를 구분하여 정리해보겠습니다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **메서드** | **thread\_safe** | **메서드** | **thread\_safe** | **메서드** | **thread\_safe** |
| at | O | count | O | find | O |
| key\_eq | O | begin | O | empty | O |
| get\_allocator | O | max\_size | O | cbegin | O |
| end | O | hash\_function | O | operator[] | O |
| cend | O | equal\_range | O | insert | O |
| size | O | clear | X | max\_load\_factor | X |
| rehash | X | load\_factor | X | operator= | X |
| swap | X | unsafe\_xxx | X |  |  |

unsafe\_xxx 로 적어놓은 것은 unsafe\_ 로 시작하는 모든 메서드를 의미합니다. 그리고 count 메서드는 thread-safe 하게 호출할 수 있지만 count 메서드가 호출됨과 동시에 다른 스레드에서 추가 작업이 발생할 경우 반환된 원소 개수가 정확하지 않을 수 있다는 것을 유의하시기 바랍니다. 그럼 이제 예제로 넘어가 볼까요? 아래에서는 concurrent\_unordered\_map 클래스를 이용하여 'a' ~ 'i'  범위의 값을 key로 하여 임의의 정수값을 추가하는 예제를 보여줍니다.

|  |
| --- |
| #include <ppl.h> #include <concurrent\_unordered\_map.h> #include <iostream>  using namespace concurrency; using namespace std;  int wmain() {     // concurrent\_unordered\_map 객체를 생성하고 원소를 병렬로 추가합니다.     **concurrent\_unordered\_map<char, int> map;**      parallel\_for(0, 10, [&map](int i) {         char key = 'a' + (i%9);   // 'a' ~ 'i' 값으로 key 생성         int value = i;                  // 해당 key에 i 값 할당         **map.insert(make\_pair(key, value));**     });      // concurrent\_unordered\_map 의 원소 출력     for\_each(begin(map), end(map), [](const pair<char, int>& pr) {         wcout << L"[" << pr.first << L", " << pr.second << L"] ";     }); }  // 실행결과  [e, 4] [i, 8] [a, 9] [c, 2] [g, 6] [f, 5] [b, 1] [d, 3] [h, 7] |

몇몇 메서드의 이름 및 동작 방식, thread-safe 여부를 제외하고는 std::unordered\_map 과 사용법이 매우 유사하기 때문에 간단하게 대체하여 사용할 수 있습니다.

**concurrent\_unordered\_multimap**

**concurrency::concurrent\_unordered\_multimap 클래스는 같은 key가 여러개의 value를 가질 수 있다는 점을 제외하고는 앞서 설명드린 concurrency::concurrent\_unordered\_map 클래스와 상당히 유사합니다.** 그리고 두 클래스의 추가적인 차이점을 나열하면 다음과 같습니다.

* insert 메서드가 std::pair<iterator, bool> 타입을 반환하는 대신 iterator 타입을 반환한다.
* operator[] 메서드를 제공하지 않는다.

위에서 설명드렸던 concurrent\_unordered\_map 클래스 예제를 그대로 concurrent\_unordered\_multimap 클래스를 이용하여 작성해 보도록 합시다.

|  |
| --- |
| #include <ppl.h> #include <concurrent\_unordered\_map.h> #include <iostream>  using namespace concurrency; using namespace std;  int wmain() {     // concurrent\_unordered\_multimap 객체를 생성하고 원소를 병렬로 추가합니다.     **concurrent\_unordered\_multimap<char, int> map;**      parallel\_for(0, 10, [&map](int i) {         char key = 'a' + (i%9);   // 'a' ~ 'i' 값으로 key 생성         int value = i;                  // 해당 key에 i 값 할당         **map.insert(make\_pair(key, value));**     });      // concurrent\_unordered\_multimap 의 원소 출력     for\_each(begin(map), end(map), [](const pair<char, int>& pr) {         wcout << L"[" << pr.first << L", " << pr.second << L"] ";     }); }    // 실행결과  [e, 4] [i, 8] [a, 9] [a, 0] [c, 2] [g, 6] [f, 5] [b, 1] [d, 3] [h, 7] |

앞선 예제와는 달리 concurrent\_unordered\_multimap 클래스의 insert 메서드는 기존에 같은 key 값이 존재하더라도 새로운 value를 추가하기 때문에 출력 결과로 'a' key 에 대한 두 개의 값 [a, 9] [a, 0] 이 출력된 것을 볼 수 있습니다.

**concurrent\_unordered\_set**

**concurrency::concurrent\_unordered\_set 클래스는 concurrency::concurrent\_unordered\_map 클래스와 매우 유사하며 key/value pair 원소 대신 value 원소를 가집니다.**또한 concurrent\_unordered\_set 클래스는 operator[] 메서드와 at 메서드를 제공하지 않습니다. 이번엔 concurrent\_unordered\_set 클래스를 이용하여 'a' ~ 'i' 범위의 값을 추가하는 예제를 작성해 보도록 하죠.

|  |
| --- |
| #include <ppl.h> #include <concurrent\_unordered\_set.h> #include <iostream>  using namespace concurrency; using namespace std;  int wmain() {     // concurrent\_unordered\_set 객체를 생성하고 원소를 병렬로 추가합니다.     **concurrent\_unordered\_set<char> set;**      parallel\_for(0, 10000, [&set](int i) {         **set.insert('a' + (i%9));**// 'a' ~ 'i' 값 추가     });      // concurrent\_unordered\_set 의 원소 출력     for\_each(begin(set), end(set), [](char c) {         wcout << L"[" << c << L"] ";     }); }  // 실행결과  [e] [i] [a] [c] [g] [f] [b] [d] [h] |

이 예제는 앞서 설명한 concurrent\_unordered\_map 클래스 예제와 거의 유사하기 때문에 자세한 설명은 생략하고 넘어가도록 하겠습니다.

**concurrent\_unordered\_multiset**

**concurrency::concurrent\_unordered\_multiset 클래스는 여러개의 중복된 value를 가질 수있다는 점을 제외하고는 concurrency::concurrent\_unordered\_set 클래스와 상당히 유사합니다.**그리고  insert 메서드가 std::pair<iterator, bool> 타입을 반환하는 대신 iterator 타입을 반환하죠. 이제 마지막 예제를 작성해보도록 하겠습니다.

|  |
| --- |
| #include <ppl.h> #include <concurrent\_unordered\_set.h> #include <iostream>  using namespace concurrency; using namespace std;  int wmain() {     // concurrent\_unordered\_multiset 객체를 생성하고 원소를 병렬로 추가합니다.     **concurrent\_unordered\_multiset<char> set;**      parallel\_for(0, 40, [&set](int i) {         **set.insert('a' + (i%9));**   // 'a' ~ 'i' 값 추가     });      // concurrent\_unordered\_multiset 의 원소 출력     for\_each(begin(set), end(set), [](char c) {         wcout << L"[" << c << L"] ";     }); }  // 실행결과  [e] [e] [e] [e] [i] [i] [i] [i] [a] [a] [a] [a] [a] [c] [c] [c] [c] [c] [g] [g] [g] [g] [f] [f] [f] [f] [b] [b] [b] [b] [b] [d] [d] [d] [d] [d] [h] [h] [h] [h] |

concurrent\_unordered\_multiset 클래스는 중복된 value의 추가를 허용하기 때문에 총 40개의 value가 출력된 것을 볼 수 있습니다.