Boost.Ref

Jaakko Järvi

Peter Dimov

Douglas Gregor

Dave Abrahams

Table of Contents

Introduction 1
Reference 2
Header <boost/ref.hpp> 2
Acknowledgements 6

Introduction

The Ref library is a small library that is useful for passing references to function templates (algorithms) that would usually take copies of their arguments. It defines the class template `boost::reference_wrapper<T>`, the two functions `boost::ref` and `boost::cref` that return instances of `boost::reference_wrapper<T>`, and the two traits classes `boost::is_reference_wrapper<T>` and `boost::unwrap_reference<T>`.

The purpose of `boost::reference_wrapper<T>` is to contain a reference to an object of type T. It is primarily used to "feed" references to function templates (algorithms) that take their parameter by value.

To support this usage, `boost::reference_wrapper<T>` provides an implicit conversion to `T&`. This usually allows the function templates to work on references unmodified.

`boost::reference_wrapper<T>` is both CopyConstructible and Assignable (ordinary references are not Assignable).

The expression `boost::ref(x)` returns a `boost::reference_wrapper<X>(x)` where X is the type of x. Similarly, `boost::cref(x)` returns a `boost::reference_wrapper<X const>(x)`.

The expression `boost::is_reference_wrapper<T>::value` is true if T is a `reference_wrapper`, and false otherwise.

The type-expression `boost::unwrap_reference<T>::type` is T::type if T is a `reference_wrapper`, T otherwise.

Reference

Header <boost/ref.hpp>

```
namespace boost {
  template<typename T> class reference_wrapper;
  reference_wrapper<T> ref(T&);
  reference_wrapper<T const> cref(T const&);
  template<typename T> class is_reference_wrapper;
  template<typename T> class unwrap_reference;
}
```

Class template reference_wrapper

Class template reference_wrapper -- Contains a reference to an object of type `T`.

```
template<typename T>
class reference_wrapper {
public:
  // types
  typedef T type;

  // construct/copy/destruct
  explicit reference_wrapper(T&);

  // access
  operator T&() const;
  T& get() const;
  T* get_pointer() const;
};

// constructors
reference_wrapper<T> ref(T&);
reference_wrapper<T const> cref(T const&);
```

Description

`reference_wrapper` is primarily used to "feed" references to function templates (algorithms) that take their parameter by value. It provides an implicit conversion to `T&`, which usually allows the function templates to work on references unmodified.

`reference_wrapper` construct/copy/destruct

1.
   ```
   explicit reference_wrapper(T& t);
   ```

   Effects: Constructs a `reference_wrapper` object that stores a reference to `t`.

   Throws: Does not throw.

`reference_wrapper` access

1.
   ```
   operator T&() const;
   ```

   Returns: The stored reference.

   Throws: Does not throw.

2.
   ```
   T& get() const;
   ```

   Returns: The stored reference.

   Throws: Does not throw.

3. `T* get_pointer() const;`

   Returns A pointer to the object referenced by the stored reference.

   Throws Does not throw.

`reference_wrapper` constructors

1. `reference_wrapper<T> ref(T& t);`

   Returns `reference_wrapper<T>(t)`

   Throws Does not throw.

2. `reference_wrapper<T const> cref(T const& t);`

   Returns `reference_wrapper<T const>(t)`

   Throws Does not throw.

Class template is_reference_wrapper

Class template is_reference_wrapper -- Determine if a type `T` is an instantiation of `reference_wrapper`.

```
template<typename T>
class is_reference_wrapper {
public:
  // static constants
  static const bool value = unspecified;
};
```

Description

The `value` static constant will be `true` iff the type `T` is a specialization of `reference_wrapper`.

Class template unwrap_reference

Class template unwrap_reference -- Find the type in a `reference_wrapper`.

```
template<typename T>
class unwrap_reference {
public:
  // types
  typedef unspecified type;
};
```

Description

The typedef `type` is `T::type` if `T` is a `reference_wrapper`, `T` otherwise.

Acknowledgements

`ref` and `cref` were originally part of the Tuple library by Jaakko Järvi. They were "promoted to boost:: status" by Peter Dimov because they are generally useful. Douglas Gregor and Dave Abrahams contributed `is_reference_wrapper` and `unwrap_reference`.