future<RetType>

get the result - blocks until result is ready; return

future_status wait_for(const duration&) wait for the result for a specified period; unblock when result is available or after duration elapsed

wait_for_until(const time_point& wait for the result until the specified point in time; unblock when result is available or after duration elapsed

shared_future<RetType> share() convert future to shared future

void wait() block until result is available

future_status wait_for(const duration&) wait for the result for a specified period; unblock when result is available or after duration elapsed

wait_for_until(const time_point& wait for the result until the specified point in time; unblock when result is available or after duration elapsed

future<RetType> wait() block until result is available

promise<RetType> promise(allocator_arg_t, const Allocator&) construct from Alloc for shared state

future<RetType> get_future() return a future for this promise

void set_value(RetType&) set_value_at_thread_exit() execute the task and signal the future

set_value(RetType, Allocator*) set_value_at_thread_exit() execute the task and signal the future

promise<RetType> promise(allocator_arg_t, const Allocator&) construct from Alloc for shared state

future<RetType> get_future() return a future for this promise

void set_value(RetType&) set_value_at_thread_exit() execute the task and signal the future

promise<RetType> promise(allocator_arg_t, const Allocator&) construct from Alloc for shared state

future<RetType> get_future() return a future for this promise

void set_value(RetType&) set_value_at_thread_exit() execute the task and signal the future

condition_variable

unblock one of the waiting threads

void notify_all() unblock all of the waiting threads

void wait(unique_lock<mutex>&, [Predicate]) unlock the mutex and block the thread until the condition variable is signalled; use Predicate to check for spurious wakeups

cv_status | bool wait_until (unique_lock<mutex>&, const time_point&, [Predicate]) like wait, but only wait until specified time point; return cv_status or, if Predicate is supplied, the value of Predicate

cv_status | bool wait_for (unique_lock<mutex>&, const duration&, [Predicate]) like wait, but only wait for the specified duration; return cv_status or, if Predicate is supplied, the value of Predicate

native_handle_type native_handle() get platform specific handle

condition_variable_any

Same interface as condition_variable, but wait* methods allow a custom lock class in place of unique_lock, and native_handle method isn't available

mutex/recursive_mutex

void lock() recursive_mutex allows multiple calls to lock with increasing levels of ownership

void try_lock() immediately return false if unable to lock

void unlock()

native_handle_type native_handle() get platform specific handle

timed_mutex/

recursive_timed_mutex

Same as mutex/recursive_mutex, with two extra methods:

void try_lock_for(const duration&) try to lock for the specified duration

void try_lock_until(const time_point&) try to lock until the specified time point

Legend

default constructor

C copy constructor

M move constructor

C= copy assignment operator

M= move assignment operator

↔ swap method

cpprocks.com

@cpp_rocks