std::async 与 std::future

std::async

我们可以使用 std::async 来创建一个异步任务，该函数模板返回一个 std::future 对象，通过该对象可获得异步任务执行的结果

基本使用

```
#include <future>
#include <iostream>

using namespace std;

int func(int value) {
    std::this_thread::sleep_for(2s);                                    // 假装运行了很久
    cout<<"func thread id: "<<std::this_thread::get_id()<<endl;
    return value + 1024;
}


int main() {
    std::future<int> result = std::async(func, 255);            // 创建一个异步任务，任务入口为 func

    cout<<"main thread id: "<<std::this_thread::get_id()<<endl;

    cout<<"async task's result is: "<< result.get()<<endl;    // 阻塞在此处，直到异步任务执行完毕

    return 0;
}
```

std::future 是一个类模板，其中模板参数中即为线程入口函数所返回的类型，该例为 int

我们可以通过 std::future::get() 方法来获取异步任务的返回值，如果此时任务仍未执行完毕，那么调用线程将阻塞在此处，等待该异步任务执行结束。并且，我们只能调用一次 get() 方法获取返回值，因为内部由 unique_ptr 实现

std::launch 枚举

```
//enum class launch
_LIBCPP_DECLARE_STRONG_ENUM(launch)
{
    async = 1,
    deferred = 2,
    any = async | deferred
};
```

这些枚举值对于理解 std::async 非常重要

launch::async

该异步任务必然创建新的线程来执行用户传递的任务，若资源紧张无法创建新线程时，结束任务，抛出异常

std::future<int> result = std::async(std::launch::async, func, 255);

Output

```
main thread: 0x117144dc0
func thread: 0x70000958b000
```

launch::deferred

该异步任务必然不会创建新的线程来执行用户传递的任务，并且将发生延迟调用，即用户调用 result.get() 时才会执行该任务。并且，若用户未调用 result.get() 方法，该异步任务便不会被执行

std::future<int> result = std::async(std::launch::deferred, func, 255);

Output

```
main thread: 0x10dbafdc0
func thread: 0x10dbafdc0
```

launch::any

由系统决定到底是使用 launch::async 还是 launch::deferred

该枚举值为 std::async 的默认值

std::future

future，可以把它看成是一个未来量，或者说，期货？反正 future 就是一个对未来的许诺：结果一定在我这儿，至于啥时候能有，我不保证，但是肯定会有的

异常

如果我们的异步任务在执行过程中抛出了异常，我们如何得知？

异步任务若有异常抛出，那么该异常会在调用 get() 时抛出，这个异常我们是可以捕获的

```
int func(int value) {
    throw runtime_error("something went wrong");
    return value + 1024;
}

int main() {
    std::future<int> result = std::async(func, 255);
    try {
        cout << result.get() << endl;
    }
    catch(const std::exception& e) {
        std::cerr << e.what() << endl;
    }
    return 0;
}
```

获取任务执行状态

有的时候我们并不需要无期限的等待，只希望等待一定的时间，此时即可使用 wait_for 方法

```
int main() {

    std::future<int> result = std::async(func, 255);

    std::future_status status = result.wait_for(1s);    // 这里其实也可以 wait_for(0s) 来专门获取状态

    if (status == std::future_status::timeout)
        cout << "超时" << endl;

    else if (status == std::future_status::ready)
        cout << "任务已执行完毕" << endl;

    else if (status == std::future_status::deferred)
        cout << "同步任务" <<endl;
}
```