零声教育出品 Mark 老师 QQ: 2548898954

VSCode C++ 调试环境说明

VSCode 需要的插件

> clangd + CMake + CMake Tools + C/C++
>
> 注意：安装上述插件前，先确保 linux 环境下安装 gcc/g++, cmake, clangd；也可以使用 Mark 老师给 clangd-linux-16.0.2.zip；
>
> 把 bin 目录下的 clangd 拷贝到 `/usr/local/bin` 目录中；
>
> 把 lib 目录下的内容拷贝到 `/usr/local/lib` 目录中；

确保生成 `compile_commands.json`

> cmake 通过编译时指定设置 `CMAKE_EXPORT_COMPILE_COMMANDS=ON` 来生成。
>
> 如果是 Makefile 组织项目代码，需要安装 `bear`；make 时，需要使用 bear make，从而生成 compile_commands.json。

确保 *clangd* 插件能读到 `compile_comands.json`

> 生成 compile_commands.json 的作用是 clangd 需要使用帮助解析项目的代码从而实现精准跳转。
>
> 需要设置 `--compile-commands-dir = ${workspaceFolder}/build`

调试需要使用 C/C++ 插件

> lldb 也可以用来调试，暂时没有 C/C++ 插件好用，lldb 会经常出错。

计数器

> 计数器是框架中一种非常重要的基础任务，计数器本质上是一个不占线程的递减的**信号量**。
> 计数器主要用于工作流的控制，包括匿名计数器和命名计数器两种，可以实现非常复杂的业务逻辑。

并行抓取案例

> 创建一个 ParallelWork 来实现多个 series 并行。

```
void http_callback(WFHttpTask *task)
{
    /* Save http page. */
    ...

    WFCounterTask *counter = (WFCounterTask *)task->user_data;
    counter->count();
}

std::mutex mutex;
std::condition_variable cond;
bool finished = false;

void counter_callback(WFCounterTask *counter)
{
    mutex.lock();
    finished = true;
    cond.notify_one();
    mutex.unlock();
```

```
}

int main(int argc, char *argv[])
{
    WFCounterTask *counter = create_counter_task(url_count, counter_callback);
    WFHttpTask *task;
    std::string url[url_count];

    /* init urls */
    ...

    for (int i = 0; i < url_count; i++)
    {
        task = create_http_task(url[i], http_callback);
        task->user_data = counter;
        task->start();
    }

    counter->start();
    std::unique_lock<std:mutex> lock(mutex);
    while (!finished)
        cond.wait(lock);
    lock.unlock();
    return 0;
}
```

> 以上创建一个目标值为 url_count 的计数器，每个 http 任务完成之后，调用一次 count。
> 注意，匿名计数器的 count 次数不可以超过目标值，否则 counter 可能已经 callback 销毁了，程序行为无定义。

资源池

> 资源池可以解决下列问题：

> - 任务运行时需要先从某个池子里获得一个资源。任务运行结束，则会把资源放回池子，让下一个需要资源的任务运行。
> - 网络通信时需要对某一个或一些通信目标做总的并发度限制，但又不希望占用线程等待（**不占用线程等待**）。
> - 我们有许多随机到达的任务，处在不同的series里。但这些任务必须**串行**的运行。

控制并发度的抓取任务

```
int fetch_with_max(std::vector<std::string>& url_list, size_t max_p)
{
    WFResourcePool pool(max_p);

    for (std::string& url : url_list)
    {
        WFHttpTask *task = WFTaskFactory::create_http_task(url, [&pool](WFHttpTask *task) {
            pool.post(nullptr);
        });
        WFConditional *cond = pool.get(task);  // 无需保存res，可以不传resbuf参数。
        cond->start();
    }

    // wait_here...
}
```

条件任务

> 条件任务是一种任务包装器，通过对条件任务发送信号来触发被包装任务的执行。

延迟 1 秒执行的计算任务

```
int main()
{
    WFGoTask *task = WFTaskFactory::create_go_task("test", [](){ printf("Done\n"); });
    WFConditional *cond = WFTaskFactory::create_conditional(task);
    WFTimerTask *timer = WFTaskFactory::create_timer_task(1, 0, [cond](void *){
        cond->signal(NULL);
    });
    timer->start();
    cond->start();
    getchar();
}
```