推荐学习这个C++开源项目

跟田老师学C++ 今天

以下文章来源于程序喵大人 ， 作者程序喵大人

程序喵大人

要做最牛逼的C++公众号，Java、Android、iOS、音视频、深度学习不定期更新。

大家好，我是程序喵。

最近发现了适合C++开发者进阶的开源项目，这个项目的名字叫workflow，项目地址如下：https://github.com/sogou/workflow

workflow是搜狗公司的服务器引擎，几乎搜狗所有的后端C++服务和其他几十家公司都在使用这个引擎，每日处理超百亿请求。

其实去年在purecpp大会就听过一个美女架构师大佬介绍过这个项目，当时就感慨这个项目怎么这么牛逼🥵。最近听到好多次别人提起这个项目，于是我就仔细看了看项目的介绍，好家伙，一入源码深似海，研究了半个多月源码，由衷佩服作者超强的架构能力，明白了自己和架构师之间巨大的差距，还是得多学习啊，肝。

目录

框架有什么特点？

用户体验相当好：接口简洁，支持常用协议，使用简单，具体怎么简单我下面会介绍；

性能好：不单网络、磁盘IO、CPU计算等，workflow着眼于所有异步资源都尽可能全部调起，有相当充足的测试数据证明该框架的性能较目前主流的服务端框架更好；

稳定性高：搜狗和其他好多公司都在使用这个引擎，稳定性肯定高啊，大家也可以自己去查数据，我就不贴了；

支持多种平台：项目支持Linux、macOS、Windows、Android等操作系统。

解放用户生产力：用户接触到的只有任务(Task)和任务流(series)两种概念，框架将资源高度封装，用户无需接触到线程池、连接池、文件IO与各种异步通知机制等。用户无需关心内部细节，可以将更多精力用在实现复杂的业务逻辑上。

设计理念新颖：源码值得学习，我也是看了这个项目的源码后才推荐给大家的，看完才知道，原来代码可以这么写，继承可以这么玩。

框架能做什么？

框架能做的事情很多，我这里只介绍一些个人认为比较重要的功能，更多功能还需要大家自行解锁。

轻松的搭建server：不用多说，服务端框架如果不能搭建server那还玩啥了，但使用这个框架非常方便，以http server为例，只需要简单几行代码即可：

```
#include <stdio.h>
#include "workflow/WFHttpServer.h"

int main() {
    WFHttpServer server([](WFHttpTask *task) {
        task->get_resp()->append_output_body("Hello World!");
    });
```

```
    if (server.start(8888) == 0) { // start server on port 8888
        getchar(); // press "Enter" to end.
        server.stop();
    }
    return 0;
}
```

轻松高效的发起客户端请求：项目号称可作为万能异步客户端，目前支持http，redis，mysql、websocket和kafka协议，下面是官方给出的一个mysql的客户端示例：

```
int main(int argc, char *argv[]) {
    ...
    WFMySQLTask *task = WFTaskFactory::create_mysql_task(url, RETRY_MAX, mysql_callback);
    task->get_req()->set_query("SHOW TABLES;");
    ...
    task->start();
    ...
}
```

以往的C++ server需要访问mysql时，可能使用的是传统的客户端。在一个线程下以同步阻塞的方式等待数据到来。如果有多个网络请求希望并发，那么用户需要管理好多个mysql cli对象。

workflow完美的解决了这一系列问题，把所有这种用户请求交给内部的poller线程统一管理，实现了高效的非阻塞IO行为，提升了server作为客户端请求数据时的性能表现。再也不用担心这种客户端行为影响server整体的性能。

再举个例子，如果想要完成个wget的功能，只需要在WFHttpTask的回调函数中解析http返回的消息体即可：

```
int main(int argc, char *argv[]) {
    WFHttpTask *task;
    std::string url = argv[1];
    url = "http://" + url;
    task = WFTaskFactory::create_http_task(url, REDIRECT_MAX, RETRY_MAX,
                                           wget_callback);
    protocol::HttpRequest *req = task->get_req();
    req->add_header_pair("Accept", "*/*");
    req->add_header_pair("User-Agent", "Wget/1.14 (linux-gnu)");
    req->add_header_pair("Connection", "close");
    task->start();
    wait_group.wait();
    return 0;
}
```

wget首先发起http请求，在wget_callback中处理http的响应消息体：

```
void wget_callback(WFHttpTask *task) {
    protocol::HttpRequest *req = task->get_req();
    protocol::HttpResponse *resp = task->get_resp();
    int state = task->get_state();
    int error = task->get_error();

    std::string name;
```

```
    std::string value;
    protocol::HttpHeaderCursor req_cursor(req);
    while (req_cursor.next(name, value))
        fprintf(stderr, "%s: %s\r\n", name.c_str(), value.c_str());
    fprintf(stderr, "\r\n");

    /* Print response header. */
    fprintf(stderr, "%s %s %s\r\n", resp->get_http_version(),
                                    resp->get_status_code(),
                                    resp->get_reason_phrase());
    protocol::HttpHeaderCursor resp_cursor(resp);
    while (resp_cursor.next(name, value))
        fprintf(stderr, "%s: %s\r\n", name.c_str(), value.c_str());
    fprintf(stderr, "\r\n");
    /* Print response body. */
    const void *body;
    size_t body_len;
    resp->get_parsed_body(&body, &body_len);
    fwrite(body, 1, body_len, stdout);
    fflush(stdout);
    fprintf(stderr, "\nSuccess. Press Ctrl-C to exit.\n");
}
```

通过workflow轻松的就完成了wget的功能。

支持自定义协议client/server：用户可构建自己的RPC系统，搜狗有个开源项目srpc就是以这个框架为基础实现的。

可构建异步任务流：支持串联，支持并联，支持串并联的组合体，也支持复杂的DAG结构。

异步IO：在Linux系统下可作为文件异步IO工具使用，性能超过任何标准调用。

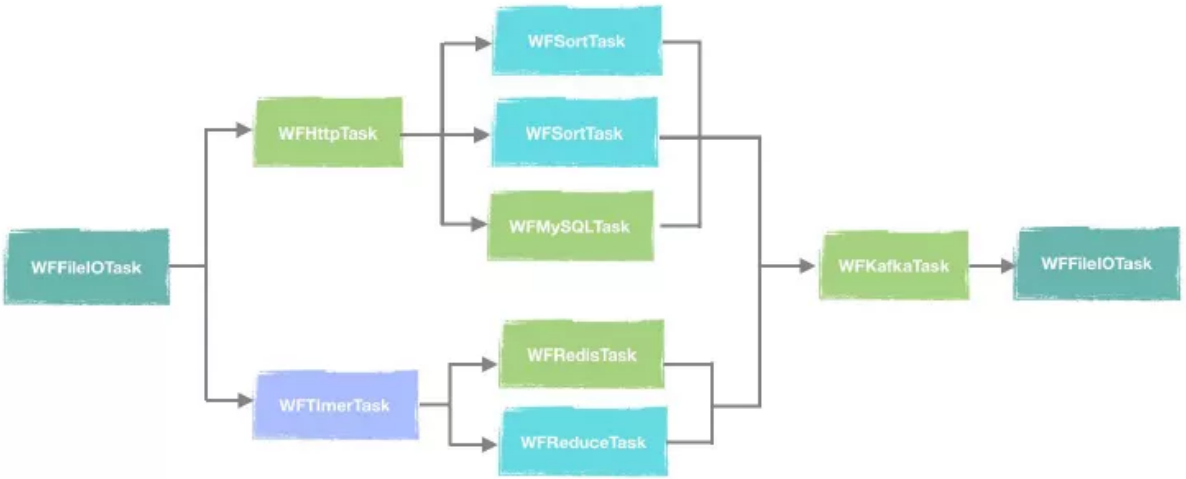
通信与计算一体化：多数框架都着重于网络IO的效率问题，而计算与任务调度等需要用户自己实现，workflow会自动对任务进行调度，打通网络和磁盘等资源，特别适合需要网络通信的重计算模块。

我为什么要推荐这个项目？

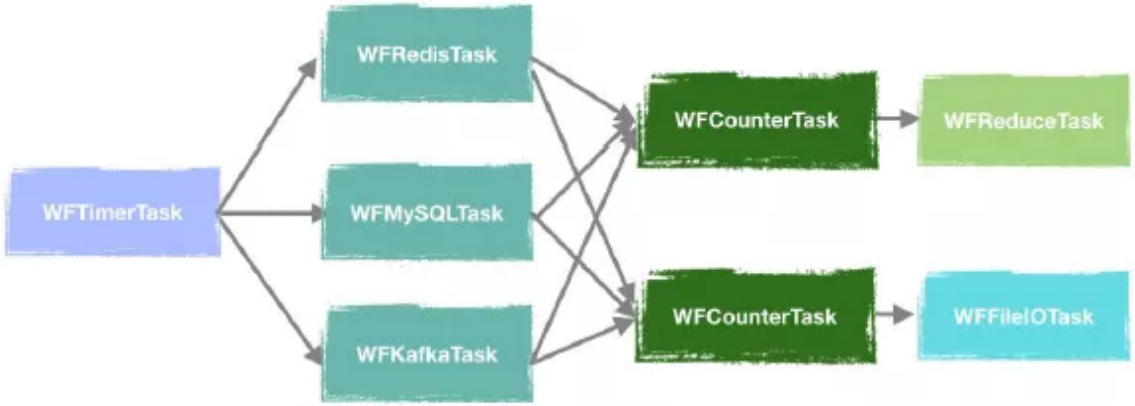
主要就一点：值得学习，适合C++开发者进阶，那具体学习什么？

学习系统的设计，所谓初级重实现，中级重炫技，高级重设计。

在作者的设计理念中，一切业务逻辑皆是任务，多个任务会组成任务流，任务流可组成图，这个图可能是串联图，可能是并联图，也有可能是串并联图，类似于这种：



也有可能是这种复杂的DAG图：



当然图的层次结构可由用户自定义，个人认为框架最牛逼的一点就是支持动态创建任务流。

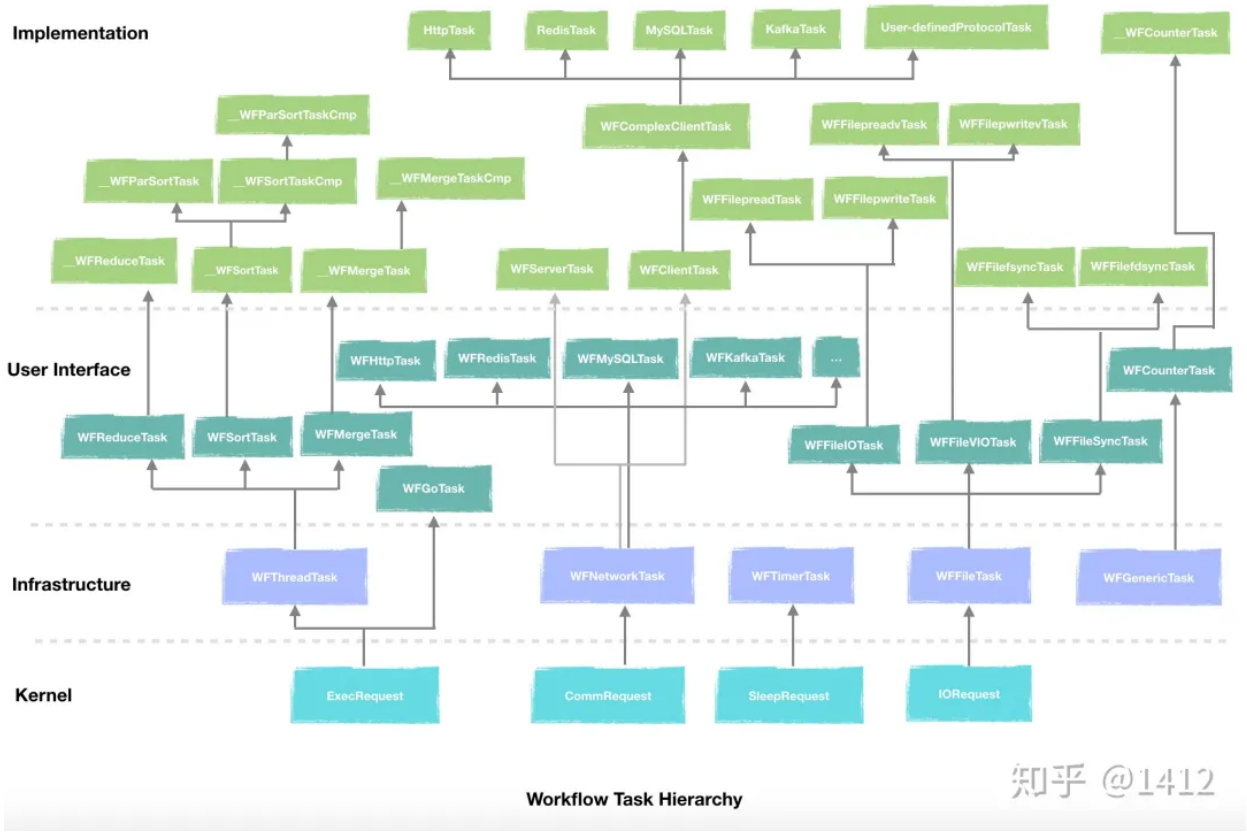
使用下面这段代码可以很直观友好的构造出图的结构，你有没有很好奇这段代码是怎么实现的？

```
WFGraphNode a, b, c, d;
a-->b;
a-->c;
b-->d;
c-->d;
```

这里先卖个关子，感兴趣的自己去看吧，总贴人家代码也不好。

我认为项目最值得大家学习的就是架构的设计，特别任务与任务流的设计，我现在还没看完代码，画不出架构的设计图（也怕画错了），只能笼统的说一句牛逼，因为确实惊艳到我了。

贴一个workflow的关于Task的架构图：



再简单的贴一个定时器Task的实现代码给大家看看：

项目里所有的Task都通过工厂创建：

```
static WFTimerTask *create_timer_task(const std::string& timer_name,
                                      unsigned int microseconds,
                                      timer_callback_t callback);
```

看下WFTimerTask的设计：

```
class WFTimerTask : public SleepRequest {
public:
    void start() {
        assert(!series_of(this));
        Workflow::start_series_work(this, nullptr);
    }
    void dismiss() {
        assert(!series_of(this));
        delete this;
    }
protected:
    virtual SubTask *done() {
        if (this->callback)
            this->callback(this);
    }
};
```

再看下Workflow::start_series_work()的方法：

```
inline void Workflow::start_series_work(SubTask *first, series_callback_t callback) {
    new SeriesWork(first, std::move(callback));
    first->dispatch();
}
```

然后是SleepRequest:

```
class SleepRequest : public SubTask, public SleepSession {
public:
    virtual void dispatch() {
        if (this->scheduler->sleep(this) < 0) {
            this->state = SS_STATE_ERROR;
            this->error = errno;
            this->subtask_done();
        }
    }
protected:
    CommScheduler *scheduler;
    virtual void handle(int state, int error) {
        this->state = state;
        this->error = error;
        this->subtask_done();
    }
};
```

再看下scheduler中的这个sleep()方法:

```
int Communicator::sleep(SleepSession *session) {
    struct timespec value;
    if (session->duration(&value) >= 0) {
```

```
        if (mpoller_add_timer(&value, session, this->mpoller) >= 0)
            return 0;
    }
    return -1;
}
```

然后是SubTask：

```
class SubTask {
public:
    virtual void dispatch() = 0;
private:
    virtual SubTask *done() = 0;
protected:
    void subtask_done();
};
```

然后是subtask_done()方法的实现：

```
void SubTask::subtask_done() {
    SubTask *cur = this;
    ParallelTask *parent;
    SubTask **entry;
    while (1) {
        parent = cur->parent;
```

```
        entry = cur->entry;
        cur = cur->done();
        xxx
        break;
    }
}
```

然后是SleepSession：

```
class SleepSession {
private:
    virtual int duration(struct timespec *value) = 0;
    virtual void handle(int state, int error) = 0;
public:
    virtual ~SleepSession() { }
    friend class Communicator;
};
```

看了这么多源码，那WFTimerTask是如何实现的定时功能呢？

我总结了下面几步：

- **步骤一**

用户调用WFTimerTask的start()；

- **步骤二**

start()中调用Workflow::start_series_work()方法；

- **步骤三**

start_series_work()中调用SubTask的dispatch()方法，这个dispatch()方法由SubTask的子类SleepRequest（WFTimerTask的父类）实现；

- **步骤四**

SleepRequest类的dispath()方法会调用scheduler->sleep()方法；

- **步骤五**

sleep()方法会调用SleepSession的duration()方法获取具体sleep的时间，框架内部用了timerfd把超时时间交给操作系统，时间到了会通知框架层，进而触发SleepSession中的handle()调用；

- **步骤六**

handle()的实现中会调用subtask_done()方法；

- **步骤七**

subtask_done()中会调用SubTask中的done()方法；

- **步骤八**

这个done()方法具体由WFTimerTask覆盖，实现中会调用到具体时间后触发的回调函数。

乍一看可能感觉非常麻烦，为什么实现一个普通的定时功能会搞这么多继承关系，但你真正看了源码后就会发现，项目抽象出的所有Task，比如计数器Task、文件IOTask、网络Task、MySQLTask等，都是通过这种SubTask、XXXRequest、XXXSession的形式来实现，后期再来个XXXTask可以很方便的扩展，这才是优秀项目该有的架构，真的佩服。

读者们，你们可以设计出这么高端的架构吗？反正我要肝完这个项目，也推荐给大家一起学习。

最后总结了该项目中个人认为值得我们学习的地方：

小总结

接口的设计：项目的接入极其简单，几行代码就可搭建个client或者server，几行代码也可构建出简单的任务流图，可用于处理复杂的业务逻辑；

架构的设计：项目中的各种类是如何派生的，作者的设计思路是怎么样的；

网络通信：项目没有使用任何网络框架，而是使用网络裸接口进行网络通信，我们都知道在大型项目中使用网络裸接口进行网络通信需要处理很多异常条件，这里值得学习一波；

任务流的封装：为什么可以动态的构建任务流的串并联图，并在项目内部灵活的调度呢？

文件I/O：项目号称内部文件I/O操作比标准调用性能还好，它是怎么做到的？

内存的管理：项目没有使用任何智能指针，却能管理好内存问题，这是个技术活，当然，也得益于这优秀的架构设计。

我发现workflow团队对这个项目相当重视，还特意建了个QQ交流群（群号码是618773193），对此项目有任何问题都可以在这个群里探讨，也方便了我们学习，真的不错。

参考资料

https://zhuanlan.zhihu.com/p/358869362
https://zhuanlan.zhihu.com/p/165638263

项目地址如下：https://github.com/sogou/workflow，也可以点击阅读原文直达。

在访问GitHub遇到困难时，可使用他们的Gitee官方仓库：https://gitee.com/sogou/workflow

感觉这个项目值得学习的话就给人家个star，不要白嫖哈，对项目团队来说也是一种认可和鼓励。

点击阅读原文直达workflow仓库。

↓ ↓ ↓

END

欢迎一起交流C++开发

请扫描下方二维码加田老师为微信好友

跟田老师学C++

欢迎关注“跟田老师学C++”微信公众号

请手指长按下方二维码图片识别，即可关注

欢迎加入“C++/Qt”知识星球

请手指长按下方二维码图片识别，即可加入

记得帮我点赞/在看哦

阅读原文

喜欢此内容的人还喜欢

从开源项目ADSEC学习域渗透攻防基础

红蓝安全

这是目前见过最好的 OA 开源项目 基于 SpringBoot 开发

IT码徒

如何在Github上面精准搜索开源项目?

Java面试