ZLToolKit笔记12：管道

2025年1月5日　　16:01

在ZLToolKit项目中，给每个Poller线程维护一个内核事件管道_pipe，初始化Poller的时候就已经将监听该管道的事件注册到了内核事件表中：

```
EventPoller::EventPoller(std::string name) {
#if defined(HAS_EPOLL) || defined(HAS_KQUEUE)
    _event_fd = create_event();
    if (_event_fd == -1) {
        throw runtime_error(StrPrinter << "Create event fd failed: "
<< get_uv_errmsg());
    }
    SockUtil::setCloExec(_event_fd);
#endif //HAS_EPOLL
    _name = std::move(name);
    _logger = Logger::Instance().shared_from_this();
    addEventPipe();
}

void EventPoller::addEventPipe() {
    SockUtil::setNoBlocked(_pipe.readFD());
    SockUtil::setNoBlocked(_pipe.writeFD());
    // 添加内部管道事件 [AUTO-TRANSLATED:6a72e39a]
    //Add internal pipe event
    if (addEvent(_pipe.readFD(), EventPoller::Event_Read, [this](int event)
{ onPipeEvent(); }) == -1) {
        throw std::runtime_error("Add pipe fd to poller failed");
    }
}
```

该管道如果发生可读事件，则触发onPipeEvent函数的调用：

```
inline void EventPoller::onPipeEvent(bool flush) {
    char buf[1024];
    int err = 0;
    if (!flush) {
       for (;;) {
         if ((err = _pipe.read(buf, sizeof(buf))) > 0) {
            // 读到管道数据,继续读,直到读空为止 [AUTO-TRANSLATED:47bd325c]
            //Read data from the pipe, continue reading until it's empty
            continue;
         }
         if (err == 0 || get_uv_error(true) != UV_EAGAIN) {
            // 收到eof或非EAGAIN(无更多数据)错误,说明管道无效了,重新打开管道 [AUTO-
TRANSLATED:5f7a013d]
            //Received eof or non-EAGAIN (no more data) error, indicating that the
pipe is invalid, reopen the pipe
            ErrorL << "Invalid pipe fd of event poller, reopen it";
            delEvent(_pipe.readFD());
            _pipe.reOpen();
            addEventPipe();
```

```
            }
            break;
        }
    }
    decltype(_list_task) _list_swap;
    {
        lock_guard<mutex> lck(_mtx_task);
        _list_swap.swap(_list_task);
    }
    _list_swap.for_each([&](const Task::Ptr &task) {
        try {
            (*task)();
        } catch (ExitException &) {
            _exit_flag = true;
        } catch (std::exception &ex) {
            ErrorL << "Exception occurred when do async task: " << ex.what();
        }
    });
}
```

设计技巧

1. **非阻塞 I/O**
   - 使用 `_pipe.read` 和 `UV_EAGAIN` 的判断，确保管道读取是非阻塞的。即使没有数据可读，线程也不会阻塞在读取操作上。
2. **锁时间最小化**
   - 在处理任务队列时，只在关键区域加锁（即 `_list_task` 和 `_list_swap` 的交换），然后释放锁。这种做法显著降低了锁的持有时间，避免了任务执行过程中其他线程无法添加新任务的问题。
3. **异步任务解耦**
   - 异步任务使用 `Task::Ptr` 的形式存储，并在独立的循环中执行，使得任务与事件循环框架松散耦合。这种设计让任务的管理和执行更加灵活，同时便于扩展新任务类型。
4. **异常安全**
   - 在任务执行过程中捕获异常，避免异常扩散导致整个事件循环中断。这是一种稳健的编程策略，特别是在任务可能来源广泛的异步环境中。
5. **双缓冲队列**
   - 使用 `_list_task` 和 `_list_swap` 实现双缓冲设计：
     - `_list_task` 用于接收来自外部线程的任务请求。
     - `_list_swap` 用于当前线程的任务处理。
     - 通过 `swap` 实现高效交换，这种设计既保证了线程安全性，又提高了任务处理的效率。
6. **可扩展性**
   - 管道事件与任务队列分开处理，清晰地分离了 I/O 事件和逻辑任务。通过这种设计，`onPipeEvent` 函数可以轻松扩展为处理其他类型的事件或任务。

管道:

```
class PipeWrap {
public:
    PipeWrap();
    ~PipeWrap();
    int write(const void *buf, int n);
    int read(void *buf, int n);
    int readFD() const { return _pipe_fd[0]; }
    int writeFD() const { return _pipe_fd[1]; }
    void reOpen();
private:
    void clearFD();
private:
    int _pipe_fd[2] = { -1, -1 };
};
```

```
#define checkFD(fd) \
    if (fd == -1) { \
        clearFD(); \
        throw runtime_error(StrPrinter << "Create windows pipe failed: "
<< get_uv_errmsg());\
    }
#define closeFD(fd) \
    if (fd != -1) { \
        close(fd);\
        fd = -1;\
    }
namespace toolkit {
PipeWrap::PipeWrap() {
    reOpen();
}
void PipeWrap::reOpen() {
    clearFD();
#if defined(_WIN32)
    const char *localip = SockUtil::support_ipv6() ? "::1" : "127.0.0.1";
    auto listener_fd = SockUtil::listen(0, localip);
    checkFD(listener_fd)
    SockUtil::setNoBlocked(listener_fd,false);
    auto localPort = SockUtil::get_local_port(listener_fd);
    _pipe_fd[1] = SockUtil::connect(localip, localPort,false);
    checkFD(_pipe_fd[1])
    _pipe_fd[0] = (int)accept(listener_fd, nullptr, nullptr);
    checkFD(_pipe_fd[0])
    SockUtil::setNoDelay(_pipe_fd[0]);
    SockUtil::setNoDelay(_pipe_fd[1]);
    close(listener_fd);
#else
    if (pipe(_pipe_fd) == -1) {
        throw runtime_error(StrPrinter << "Create posix pipe failed: "
```

```
<< get_uv_errmsg());
    }
#endif // defined(_WIN32)
    SockUtil::setNoBlocked(_pipe_fd[0], true);
    SockUtil::setNoBlocked(_pipe_fd[1], false);
    SockUtil::setCloExec(_pipe_fd[0]);
    SockUtil::setCloExec(_pipe_fd[1]);
}
void PipeWrap::clearFD() {
    closeFD(_pipe_fd[0]);
    closeFD(_pipe_fd[1]);
}
PipeWrap::~PipeWrap() {
    clearFD();
}
int PipeWrap::write(const void *buf, int n) {
    int ret;
    do {
#if defined(_WIN32)
        ret = send(_pipe_fd[1], (char *)buf, n, 0);
#else
        ret = ::write(_pipe_fd[1], buf, n);
#endif // defined(_WIN32)
    } while (-1 == ret && UV_EINTR == get_uv_error(true));
    return ret;
}
int PipeWrap::read(void *buf, int n) {
    int ret;
    do {
#if defined(_WIN32)
        ret = recv(_pipe_fd[0], (char *)buf, n, 0);
#else
        ret = ::read(_pipe_fd[0], buf, n);
#endif // defined(_WIN32)
    } while (-1 == ret && UV_EINTR == get_uv_error(true));
    return ret;
}
```