10-Redis事件驱动框架（中）：Redis实现了Reactor模型吗？

你好，我是蒋德钧。今天，我们来聊聊Redis是如何实现Reactor模型的。

你在做Redis面试题的时候，或许经常会遇到这样一道经典的问题：Redis的网络框架是实现了Reactor模型吗？这看起来像是一道简单的“是/否”问答题，但是，如果你想给出一个让面试官满意的答案，这就非常考验你的**高性能网络编程基础和对Redis代码的掌握程度**了。

如果让我来作答这道题，我会把它分成两部分来回答：一是介绍Reactor模型是什么，二是说明Redis代码实现是如何与Reactor模型相对应的。这样一来，就既体现了我对网络编程的理解，还能体现对Redis源码的深入探究，进而面试官也就会对我刮目相看了。

实际上，Reactor模型是高性能网络系统实现高并发请求处理的一个重要技术方案。掌握Reactor模型的设计思想与实现方法，除了可以应对面试题，还可以指导你设计和实现自己的高并发系统。当你要处理成千上万的网络连接时，就不会一筹莫展了。

所以今天这节课，我会先带你了解下Reactor模型，然后一起来学习下如何实现Reactor模型。因为Redis的代码实现提供了很好的参考示例，所以我会通过Redis代码中的关键函数和流程，来给你展开介绍Reactor模型的实现。不过在学习Reactor模型前，你可以先回顾上节课我给你介绍的IO多路复用机制epoll，因为这也是学习今天这节课的基础。

Reactor模型的工作机制

好，首先，我们来看看什么是Reactor模型。

实际上，**Reactor模型就是网络服务器端用来处理高并发网络IO请求的一种编程模型**。我把这个模型的特征用两个“三”来总结，也就是：

- 三类处理事件，即连接事件、写事件、读事件；
- 三个关键角色，即reactor、acceptor、handler。

那么，Reactor模型是如何基于这三类事件和三个角色来处理高并发请求的呢？下面我们就来具体了解下。

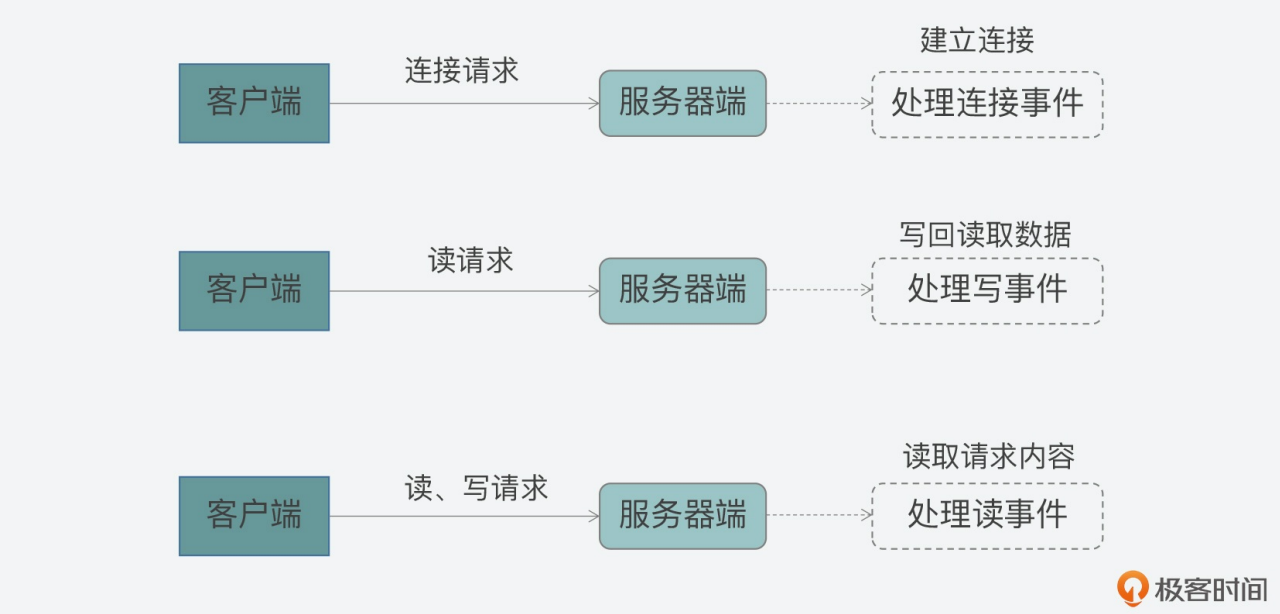
事件类型与关键角色

我们先来看看这三类事件和Reactor模型的关系。

其实，Reactor模型处理的是客户端和服务器端的交互过程，而这三类事件正好对应了客户端和服务器端交互过程中，不同类请求在服务器端引发的待处理事件：

- 当一个客户端要和服务器端进行交互时，客户端会向服务器端发送连接请求，以建立连接，这就对应了服务器端的一个**连接事件**。
- 一旦连接建立后，客户端会给服务器端发送读请求，以便读取数据。服务器端在处理读请求时，需要向客户端写回数据，这对应了服务器端的**写事件**。
- 无论客户端给服务器端发送读或写请求，服务器端都需要从客户端读取请求内容，所以在这里，读或写请求的读取就对应了服务器端的**读事件**。

如下所示的图例中，就展示了客户端和服务器端在交互过程中，不同类请求和Reactor模型事件的对应关系，你可以看下。

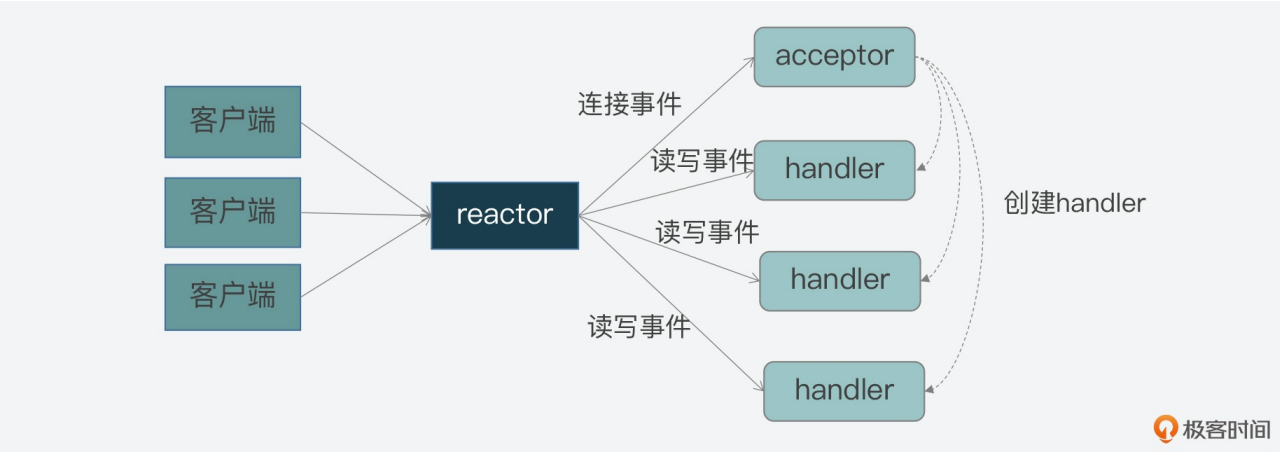


好，在了解了Reactor模型的三类事件后，你现在可能还有一个疑问：这三类事件是由谁来处理的呢？

这其实就是模型中**三个关键角色**的作用了：

- 首先，连接事件由acceptor来处理，负责接收连接；acceptor在接收连接后，会创建handler，用于网络连接上对后续读写事件的处理；
- 其次，读写事件由handler处理；
- 最后，在高并发场景中，连接事件、读写事件会同时发生，所以，我们需要有一个角色专门监听和分配事件，这就是reactor角色。当有连接请求时，reactor将产生的连接事件交由acceptor处理；当有读写请求时，reactor将读写事件交由handler处理。

下图就展示了这三个角色之间的关系，以及它们和事件的关系，你可以看下。



事实上，这三个角色都是Reactor模型中要实现的功能的抽象。当我们遵循Reactor模型开发服务器端的网络框架时，就需要在编程的时候，在代码功能模块中实现reactor、acceptor和handler的逻辑。

那么，现在我们已经知道，这三个角色是围绕事件的监听、转发和处理来进行交互的，那么在编程时，我们又该如何实现这三者的交互呢？这就离不开**事件驱动框架**了。

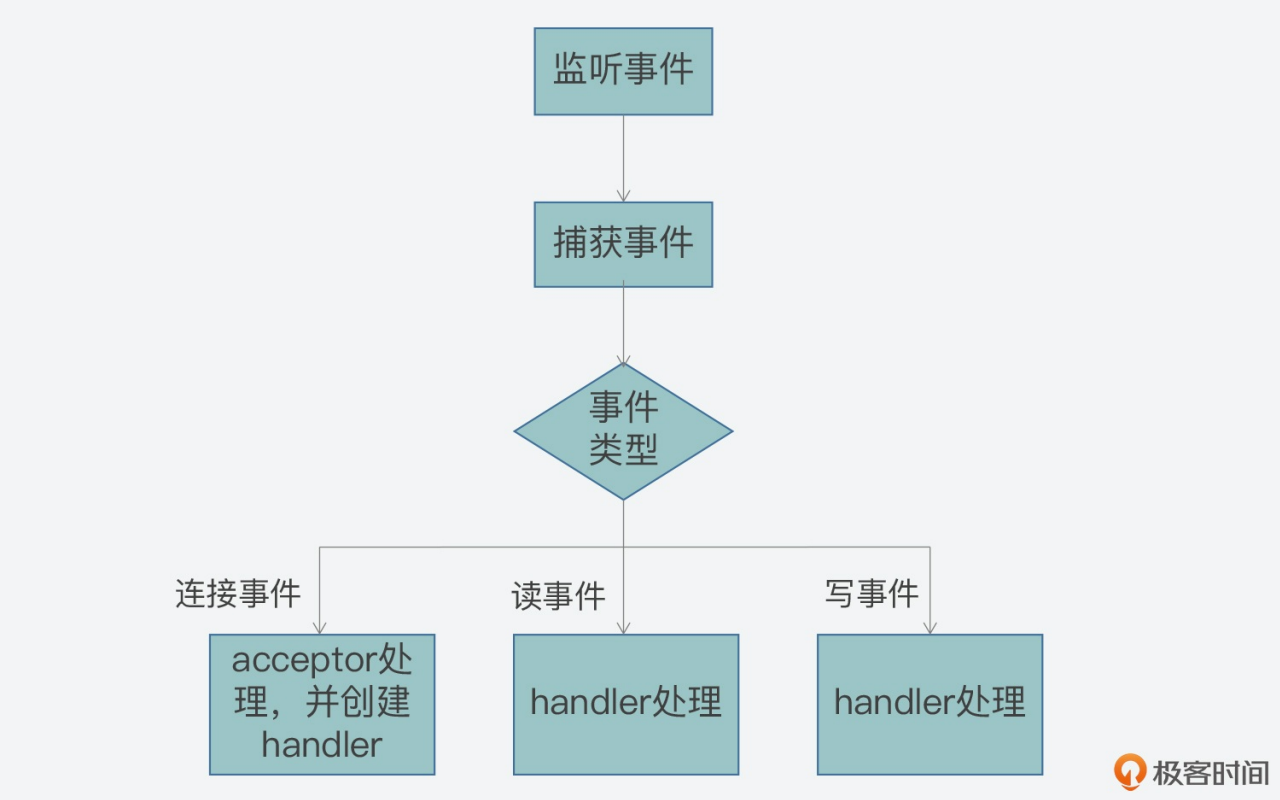
事件驱动框架

所谓的事件驱动框架，就是在实现Reactor模型时，需要实现的代码整体控制逻辑。简单来说，事件驱动框架包括了两部分：一是**事件初始化**；二是**事件捕获、分发和处理主循环**。

事件初始化是在服务器程序启动时就执行的，它的作用主要是创建需要监听的事件类型，以及该类事件对应的handler。而一旦服务器完成初始化后，事件初始化也就相应完成了，服务器程序就需要进入到事件捕获、分发和处理的主循环中。

在开发代码时，我们通常会用一个**while循环**来作为这个主循环。然后在这个主循环中，我们需要捕获发生的事件、判断事件类型，并根据事件类型，调用在初始化时创建好的事件handler来实际处理事件。

比如说，当有连接事件发生时，服务器程序需要调用acceptor处理函数，创建和客户端的连接。而当有读事件发生时，就表明有读或写请求发送到了服务器端，服务器程序就要调用具体的请求处理函数，从客户端连接中读取请求内容，进而就完成了读事件的处理。这里你可以参考下面给出的图例，其中显示了事件驱动框架的基本执行过程：



那么到这里，你应该就已经了解了**Reactor模型的基本工作机制**：客户端的不同类请求会在服务器端触发连接、读、写三类事件，这三类事件的监听、分发和处理又是由reactor、acceptor、handler三类角色来完成的，然后这三类角色会通过事件驱动框架来实现交互和事件处理。

所以可见，实现一个Reactor模型的**关键**，就是要实现事件驱动框架。那么，如何开发实现一个事件驱动框架呢？

Redis提供了一个简洁但有效的参考实现，非常值得我们学习，而且也可以用于自己的网络系统开发。下面，我们就一起来学习下Redis中对Reactor模型的实现。

Redis对Reactor模型的实现

首先我们要知道的是，Redis的网络框架实现了Reactor模型，并且自行开发实现了一个事件驱动框架。这个框架对应的Redis代码实现文件是ae.c，对应的头文件是ae.h。

前面我们已经知道，事件驱动框架的实现离不开事件的定义，以及事件注册、捕获、分发和处理等一系列操作。当然，对于整个框架来说，还需要能一直运行，持续地响应发生的事件。

那么由此，我们从ae.h头文件中就可以看到，Redis为了实现事件驱动框架，相应地定义了**事件的数据结构、框架主循环函数、事件捕获分发函数、事件和handler注册函数**。所以接下来，我们就依次来了解学习下。

事件的数据结构定义：以aeFileEvent为例

首先，我们要明确一点，就是在Redis事件驱动框架的实现当中，事件的数据结构是关联事件类型和事件处理函数的关键要素。而Redis的事件驱动框架定义了两类事件：**IO事件和时间事件**，分别对应了客户端发送的网络请求和Redis自身的周期性操作。

这也就是说，**不同类型事件的数据结构定义是不一样的**。不过，由于这节课我们主要关注的是事件框架的整体设计与实现，所以对于不同类型事件的差异和具体处理，我会在下节课给你详细介绍。那么在今天的课程中，为了让你能够理解事件数据结构对框架的作用，我就以IO事件aeFileEvent为例，给你介绍下它的数据结构定义。

aeFileEvent是一个结构体，它定义了4个成员变量mask、rfileProce、wfileProce和clientData，如下所示：

```
typedef struct aeFileEvent {
    int mask; /* one of AE_(READABLE|WRITABLE|BARRIER) */
    aeFileProc *rfileProc;
    aeFileProc *wfileProc;
    void *clientData;
} aeFileEvent;
```

- **mask**是用来表示事件类型的掩码。对于网络通信的事件来说，主要有AE_READABLE、AE_WRITABLE和AE_BARRIER三种类型事件。框架在分发事件时，依赖的就是结构体中的事件类型；
- **rfileProc和wfileProce**分别是指向AE_READABLE和AE_WRITABLE这两类事件的处理函数，也就是Reactor模型中的handler。框架在分发事件后，就需要调用结构体中定义的函数进行事件处理；
- 最后一个成员变量**clientData**是用来指向客户端私有数据的指针。

除了事件的数据结构以外，前面我还提到Redis在ae.h文件中，定义了支撑框架运行的主要函数，包括框架主循环的aeMain函数、负责事件捕获与分发的aeProcessEvents函数，以及负责事件和handler注册的aeCreateFileEvent函数，它们的原型定义如下：

```
void aeMain(aeEventLoop *eventLoop);
int aeCreateFileEvent(aeEventLoop *eventLoop, int fd, int mask, aeFileProc *proc, void *clientData);
int aeProcessEvents(aeEventLoop *eventLoop, int flags);
```

而这三个函数的实现，都是在对应的ae.c文件中，那么接下来，我就给你具体介绍下这三个函数的主体逻辑和关键流程。

主循环：aeMain函数

我们先来看下aeMain函数。

aeMain函数的逻辑很简单，就是用一个循环不停地判断事件循环的停止标记。如果事件循环的停止标记被设置为true，那么针对事件捕获、分发和处理的整个主循环就停止了；否则，主循环会一直执行。aeMain函数的主体代码如下所示：

```
void aeMain(aeEventLoop *eventLoop) {
    eventLoop->stop = 0;
    while (!eventLoop->stop) {
        …
        aeProcessEvents(eventLoop, AE_ALL_EVENTS|AE_CALL_AFTER_SLEEP);
    }
}
```

那么这里你可能要问了，**aeMain函数是在哪里被调用的呢？**

按照事件驱动框架的编程规范来说，框架主循环是在服务器程序初始化完成后，就会开始执行。因此，如果我们把目光转向Redis服务器初始化的函数，就会发现服务器程序的main函数在完成Redis server的初始化后，会调用aeMain函数开始执行事件驱动框架。如果你想具体查看main函数，main函数在server.c文件中，我们在第8讲中介绍过该文件，server.c主要用于初始化服务器和执行服务器整体控制流程，你可以回顾下。

不过，既然aeMain函数包含了事件框架的主循环，**那么在主循环中，事件又是如何被捕获、分发和处理呢？**这就是由aeProcessEvents函数来完成的了。

事件捕获与分发：aeProcessEvents函数

aeProcessEvents函数实现的主要功能，包括捕获事件、判断事件类型和调用具体的事件处理函数，从而实现事件的处理。

从aeProcessEvents函数的主体结构中，我们可以看到主要有三个if条件分支，如下所示：

```
int aeProcessEvents(aeEventLoop *eventLoop, int flags)
{
    int processed = 0, numevents;

    /* 若没有事件处理，则立刻返回*/
    if (!(flags & AE_TIME_EVENTS) && !(flags & AE_FILE_EVENTS)) return 0;
    /*如果有IO事件发生，或者紧急的时间事件发生，则开始处理*/
    if (eventLoop->maxfd != -1 || ((flags & AE_TIME_EVENTS) && !(flags & AE_DONT_WAIT))) {
       …
    }
```

```
    /* 检查是否有时间事件，若有，则调用processTimeEvents函数处理 */
    if (flags & AE_TIME_EVENTS)
        processed += processTimeEvents(eventLoop);
    /* 返回已经处理的文件或时间*/
    return processed;
}
```

这三个分支分别对应了以下三种情况：

- 情况一：既没有时间事件，也没有网络事件；
- 情况二：有IO事件或者有需要紧急处理的时间事件；
- 情况三：只有普通的时间事件。

那么对于第一种情况来说，因为没有任何事件需要处理，aeProcessEvents函数就会直接返回到aeMain的主循环，开始下一轮的循环；而对于第三种情况来说，该情况发生时只有普通时间事件发生，所以aeMain函数会调用专门处理时间事件的函数processTimeEvents，对时间事件进行处理。

现在，我们再来看看第二种情况。

首先，当该情况发生时，Redis需要捕获发生的网络事件，并进行相应的处理。那么从Redis源码中我们可以分析得到，在这种情况下，**aeApiPoll函数会被调用，用来捕获事件**，如下所示：

```
int aeProcessEvents(aeEventLoop *eventLoop, int flags){
   ...
   if (eventLoop->maxfd != -1 || ((flags & AE_TIME_EVENTS) && !(flags & AE_DONT_WAIT))) {
       ...
       //调用aeApiPoll函数捕获事件
       numevents = aeApiPoll(eventLoop, tvp);
       ...
    }
    ...
」
```

那么，aeApiPoll是如何捕获事件呢？

实际上，Redis是依赖于操作系统底层提供的 **IO多路复用机制**，来实现事件捕获，检查是否有新的连接、读写事件发生。为了适配不同的操作系统，Redis对不同操作系统实现的网络IO多路复用函数，都进行了统一的封装，封装后的代码分别通过以下四个文件中实现：

- ae_epoll.c，对应Linux上的IO复用函数epoll；
- ae_evport.c，对应Solaris上的IO复用函数evport；
- ae_kqueue.c，对应macOS或FreeBSD上的IO复用函数kqueue；
- ae_select.c，对应Linux（或Windows）的IO复用函数select。

这样，在有了这些封装代码后，Redis在不同的操作系统上调用IO多路复用API时，就可以通过统一的接口来

进行调用了。

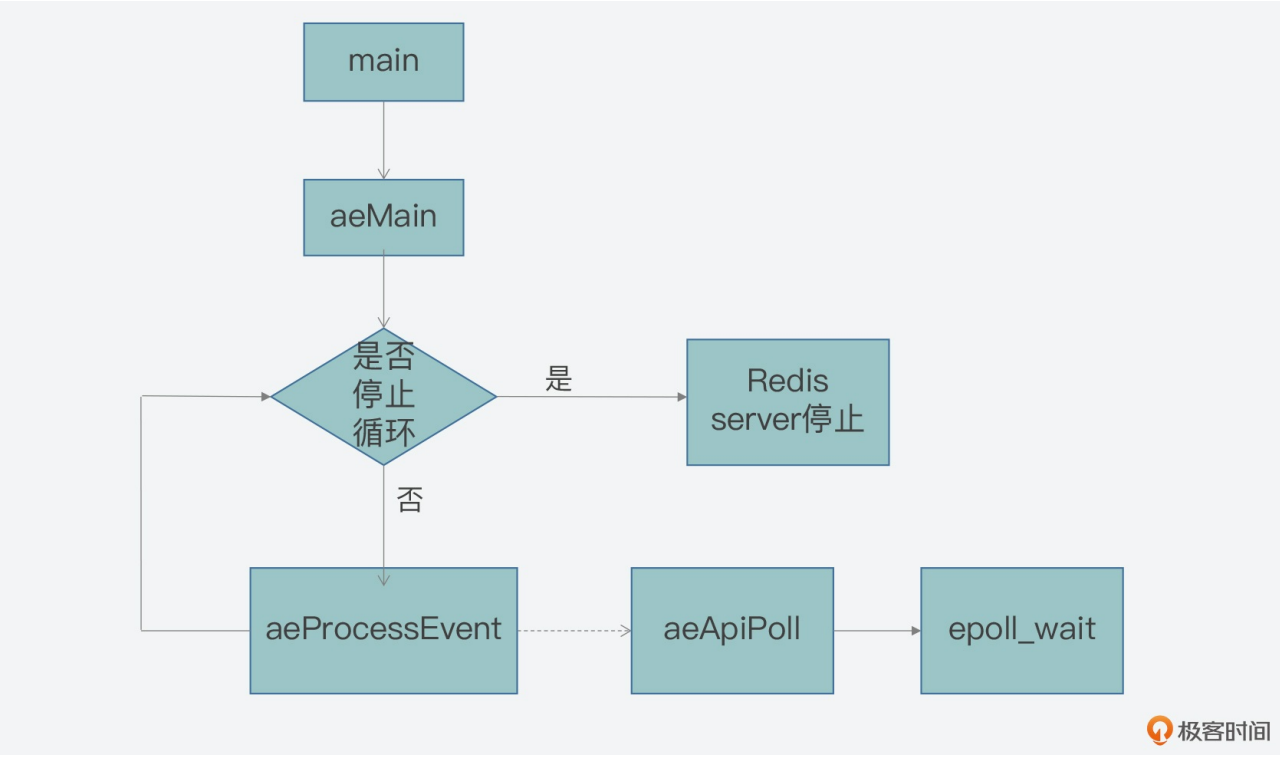
不过看到这里，你可能还是不太明白Redis封装的具体操作，所以这里，我就以在服务器端最常用的Linux操作系统为例，给你介绍下Redis是如何封装Linux上提供的IO复用API的。

首先，Linux上提供了**epoll_wait API**，用于检测内核中发生的网络IO事件。在ae_epoll.c文件中，**aeApiPoll函数**就是封装了对epoll_wait的调用。

这个封装程序如下所示，其中你可以看到，在aeApiPoll函数中直接调用了epoll_wait函数，并将epoll返回的事件信息保存起来的逻辑：

```
static int aeApiPoll(aeEventLoop *eventLoop, struct timeval *tvp) {
    …
    //调用epoll_wait获取监听到的事件
    retval = epoll_wait(state->epfd,state->events,eventLoop->setsize,
            tvp ? (tvp->tv_sec*1000 + tvp->tv_usec/1000) : -1);
    if (retval > 0) {
        int j;
        //获得监听到的事件数量
        numevents = retval;
        //针对每一个事件，进行处理
        for (j = 0; j < numevents; j++) {
             #保存事件信息
        }
    }
    return numevents;
}
```

为了让你更加清晰地理解，事件驱动框架是如何实现最终对epoll_wait的调用，这里我也放了一张示意图，你可以看看整个调用链是如何工作和实现的。



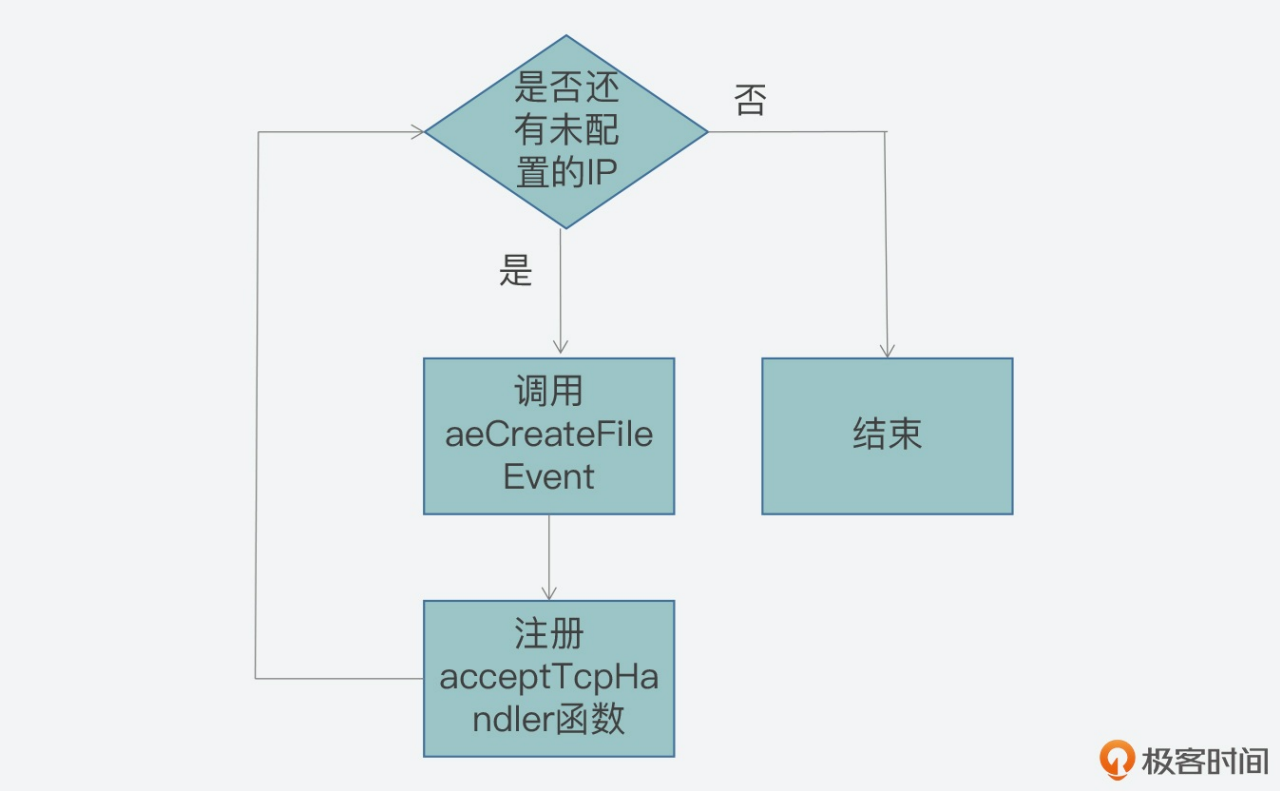
OK，现在我们就已经在aeMain函数中，看到了aeProcessEvents函数被调用，并用于捕获和分发事件的基本处理逻辑。

那么，事件具体是由哪个函数来处理的呢？这就和框架中的aeCreateFileEvents函数有关了。

事件注册：aeCreateFileEvent函数

我们知道，当Redis启动后，服务器程序的main函数会调用initSever函数来进行初始化，而在初始化的过程中，aeCreateFileEvent就会被initServer函数调用，用于注册要监听的事件，以及相应的事件处理函数。

具体来说，在initServer函数的执行过程中，initServer函数会根据启用的IP端口个数，为每个IP端口上的网络事件，调用aeCreateFileEvent，创建对AE_READABLE事件的监听，并且注册AE_READABLE事件的处理handler，也就是acceptTcpHandler函数。这一过程如下图所示：



所以这里我们可以看到，**AE_READABLE事件就是客户端的网络连接事件，而对应的处理函数就是接收TCP连接请求**。下面的示例代码中，显示了initServer中调用aeCreateFileEvent的部分片段，你可以看下：

```
void initServer(void) {
    …
    for (j = 0; j < server.ipfd_count; j++) {
        if (aeCreateFileEvent(server.el, server.ipfd[j], AE_READABLE,
            acceptTcpHandler,NULL) == AE_ERR)
            {
                serverPanic("Unrecoverable error creating server.ipfd file event.");
            }
 }
 …
}
```

那么，aeCreateFileEvent如何实现事件和处理函数的注册呢？这就和刚才我介绍的Redis对底层IO多路复用函数封装有关了，下面我仍然以Linux系统为例，来给你说明一下。

首先，Linux提供了**epoll_ctl API**，用于增加新的观察事件。而Redis在此基础上，封装了aeApiAddEvent函数，对epoll_ctl进行调用。

所以这样一来，aeCreateFileEvent就会调用aeApiAddEvent，然后aeApiAddEvent再通过调用epoll_ctl，来注册希望监听的事件和相应的处理函数。等到aeProceeEvents函数捕获到实际事件时，它就会调用注册的函数对事件进行处理了。

好了，到这里，我们就已经全部了解了Redis中实现事件驱动框架的三个关键函数：aeMain、aeProcessEvents，以及aeCreateFileEvent。当你要去实现一个事件驱动框架时，Redis的设计思想就具有很好的参考意义。

最后我再带你来简单地回顾下，在实现事件驱动框架的时候，你需要先实现一个主循环函数（对应aeMain），负责一直运行框架。其次，你需要编写事件注册函数（对应aeCreateFileEvent），用来注册监听的事件和事件对应的处理函数。**只有对事件和处理函数进行了注册，才能在事件发生时调用相应的函数进行处理。**

最后，你需要编写事件监听、分发函数（对应aeProcessEvents），负责调用操作系统底层函数来捕获网络连接、读、写事件，并分发给不同处理函数进一步处理。

小结

Redis一直被称为单线程架构，按照我们通常的理解，单个线程只能处理单个客户端的请求，但是在实际使用时，我们会看到Redis能同时和成百上千个客户端进行交互，这就是因为Redis基于Reactor模型，实现了高性能的网络框架，**通过事件驱动框架，Redis可以使用一个循环来不断捕获、分发和处理客户端产生的网络连接、数据读写事件。**

为了方便你从代码层面掌握Redis事件驱动框架的实现，我总结了一个表格，其中列出了Redis事件驱动框架的主要函数和功能、它们所属的C文件，以及这些函数本身是在Redis代码结构中的哪里被调用。你可以使用这张表格，来巩固今天这节课学习的事件驱动框架。

关键函数	所属文件	何处调用	主要功能
aeMain	ae.c	server.c中的main函数	执行事件捕获、分发和处理循环
aeProcessEvents	ae.c	ae.c中的aeMain函数	根据事件类型进行相应的处理
aeApiPoll	ae.c	ae.c中的aeProcessEvents函数	调用底层操作系统实现的IO复用API接口
epoll_wait	Linux内核文件	aeApiPoll	检测并返回内核中发生的网络IO事件

最后，我也再强调下，这节课我们主要关注的是，事件驱动框架的基本运行流程，并以客户端连接事件为

例，将框架主循环、事件捕获分发和事件注册的关键步骤串起来，给你做了介绍。Redis事件驱动框架监听处理的事件，还包括客户端请求、服务器端写数据以及周期性操作等，这也是我下一节课要和你一起学习的主要内容。

每课一问

这节课我们学习了Reactor模型，除了Redis，你还了解什么软件系统使用了Reactor模型吗？

精选留言：

- Darren 2021-08-17 10:45:04
  Reactor模型可以分为3种：
  单线程Reactor模式
  一个线程：
  单线程：建立连接（Acceptor）、监听accept、read、write事件（Reactor）、处理事件（Handler）都只用一个单线程。

  多线程Reactor模式
  一个线程 + 一个线程池：
  单线程：建立连接（Acceptor）和 监听accept、read、write事件（Reactor），复用一个线程。
  工作线程池：处理事件（Handler），由一个工作线程池来执行业务逻辑，包括数据就绪后，用户态的数据读写。

  主从Reactor模式
  三个线程池：
  主线程池：建立连接（Acceptor），并且将accept事件注册到从线程池。
  从线程池：监听accept、read、write事件（Reactor），包括等待数据就绪时，内核态的数据I读写。
  工作线程池：处理事件（Handler），由一个工作线程池来执行业务逻辑，包括数据就绪后，用户态的数据读写

  具体的可以参考并发大神 doug lea 关于Reactor的文章。 http://gee.cs.oswego.edu/dl/cpjslides/nio.pdf

  再提一点，使用了多路复用，不一定是使用了Reacto模型，Mysql使用了select（为什么不使用epoll，因为Mysql的瓶颈不是网络，是磁盘IO），但是并不是Reactor模型

  回到问题，那些也是reactor

  nginx：nginx是多进程模型，master进程不处理网络IO，每个Wroker进程是一个独立的单Reacotr单线程模型。

  netty：通信绝对的王者，默认是多Reactor，主Reacotr只负责建立连接，然后把建立好的连接给到从Reactor，从Reactor负责IO读写。当然可以专门调整为单Reactor。

  kafka：kafka也是多Reactor，但是因为Kafka主要与磁盘IO交互，因此真正的读写数据不是从Reactor处理的，而是有一个worker线程池，专门处理磁盘IO，从Reactor负责网络IO，然后把任务交给worker线程池处理。 [7赞]

- Kaito 2021-08-17 02:03:56
  1、为了高效处理网络 IO 的「连接事件」、「读事件」、「写事件」，演化出了 Reactor 模型

2、Reactor 模型主要有 reactor、acceptor、handler 三类角色：

- reactor：分配事件
- acceptor：接收连接请求
- handler：处理业务逻辑

3、Reactor 模型又分为 3 类：

- 单 Reactor 单线程：accept -> read -> 处理业务逻辑 -> write 都在一个线程
- 单 Reactor 多线程：accept/read/write 在一个线程，处理业务逻辑在另一个线程
- 多 Reactor 多线程 / 进程：accept 在一个线程/进程，read/处理业务逻辑/write 在另一个线程/进程

4、Redis 6.0 以下版本，属于单 Reactor 单线程模型，监听请求、读取数据、处理请求、写回数据都在一个线程中执行，这样会有 3 个问题：

- 单线程无法利用多核
- 处理请求发生耗时，会阻塞整个线程，影响整体性能
- 并发请求过高，读取/写回数据存在瓶颈

5、针对问题 3，Redis 6.0 进行了优化，引入了 IO 多线程，把读写请求数据的逻辑，用多线程处理，提升并发性能，但处理请求的逻辑依旧是单线程处理

课后题：除了 Redis，你还了解什么软件系统使用了 Reactor 模型吗？

Netty、Memcached 采用多 Reactor 多线程模型。

Nginx 采用多 Reactor 多进程模型，不过与标准的多 Reactor 多进程模型有些许差异。Nginx 的主进程只用来初始化 socket，不会 accept 连接，而是由子进程 accept 连接，之后这个连接的所有处理都在子进程中完成。 [5赞]

- 曾轼麟 2021-08-19 23:21:11
  上篇文章回答的时候自己提到的-Redis基于多种IO复用实现了同一方法但是多套代码文件的思路，没想到这期老师就提到了。回答老师的问题：还有什么软件系统使用了Reactor模型？
  答：
  最典型的就是netty，java靠netty得以实现了高性能的服务

  总结：
  本篇文章老师主要介绍了Redis是如何实现Reactor模型，其本质上就是围绕三个事件的实现【连接请求，读事件，写事件】，而Redis为了方便实现，封装了事件驱动框架aeEventLoop，其本质上是一个不断处理事件的循环。能同时分发处理来自成百上千个客户端的读，写，连接等请求。

- 屈仁能 2021-08-18 10:20:44
  只知道netty用过Reactor模型，看了评论学到了

- 结冰的水滴 2021-08-17 21:22:57
  kafka使用了Reactor编程模型，它使用一个Acceptor,多个Processor处理网络连接，读写请求，以及一个线程池处理消息读写

- 末日，成欢 2021-08-17 09:34:14

我一直不明白的一点， while循环里使用aeApiPoll得到一些事件后，要对这些事件进行处理， 每个处理函数不耗时吗， 假设每个函数处理耗时1ms， 有1000个事件， 那么下一次循环不得1s后了。