08-Redisserver启动后会做哪些操作?

你好，我是蒋德钧。从这节课开始，我们就来到了课程的第二个模块，在这个模块里，我会带你了解和学习与Redis实例运行相关方面的知识，包括Redis server的启动过程、基于事件驱动框架的网络通信机制以及Redis线程执行模型。今天，我们先来学习下Redis server的启动过程。

我们知道，main函数是Redis整个运行程序的入口，并且Redis实例在运行时，也会从这个main函数开始执行。同时，由于Redis是典型的Client-Server架构，一旦Redis实例开始运行，Redis server也就会启动，而main函数其实也会负责Redis server的启动运行。

> 我在第1讲给你介绍过Redis源码的整体架构。其中，Redis运行的基本控制逻辑是在server.c文件中完成的，而main函数就是在server.c中。

你平常在设计或实现一个网络服务器程序时，可能会遇到一个问题，那就是服务器启动时，应该做哪些操作、有没有一个典型的参考实现。所以今天这节课，我就从main函数开始，给你介绍下Redis server是如何在main函数中启动并完成初始化的。通过这节课内容的学习，你可以掌握Redis针对以下三个问题的实现思路：

1. **Redis server启动后具体会做哪些初始化操作?**
2. **Redis server初始化时有哪些关键配置项?**
3. **Redis server如何开始处理客户端请求?**

并且，Redis server设计和实现的启动过程也具有一定的代表性，你在学习后，就可以把其中的关键操作推而广之，用在自己的网络服务器实现中。

好了，接下来，我们先从main函数开始，来了解下它在Redis server中的设计实现思路。

main函数：Redis server的入口

一般来说，一个使用C开发的系统软件启动运行的代码逻辑，都是实现在了main函数当中，所以在正式了解Redis中main函数的实现之前，我想先给你分享一个小Tips，就是你在阅读学习一个系统的代码时，可以先找下main函数，看看它的执行过程。

那么，对于Redis的main函数来说，我把它执行的工作分成了五个阶段。

阶段一：基本初始化

在这个阶段，main函数主要是完成一些基本的初始化工作，包括设置server运行的时区、设置哈希函数的随机种子等。这部分工作的主要调用函数如下所示：

```
//设置时区
setlocale(LC_COLLATE,"");
tzset();
...
//设置随机种子
char hashseed[16];
getRandomHexChars(hashseed,sizeof(hashseed));
dictSetHashFunctionSeed((uint8_t*)hashseed);
```

这里，你需要注意的是，在main函数的开始部分，有一段宏定义覆盖的代码。这部分代码的作用是，如果定义了REDIS_TEST宏定义，并且Redis server启动时的参数符合测试参数，那么main函数就会执行相应的测试程序。

这段宏定义的代码如以下所示，其中的示例代码就是调用ziplist的测试函数ziplistTest：

```
#ifdef REDIS_TEST
//如果启动参数有test和ziplist，那么就调用ziplistTest函数进行ziplist的测试
if (argc == 3 && !strcasecmp(argv[1], "test")) {
  if (!strcasecmp(argv[2], "ziplist")) {
     return ziplistTest(argc, argv);
  }
  ...
}
#endif
```

阶段二：检查哨兵模式，并检查是否要执行RDB检测或AOF检测

Redis server启动后，可能是以哨兵模式运行的，而哨兵模式运行的server在参数初始化、参数设置，以及server启动过程中要执行的操作等方面，与普通模式server有所差别。所以，main函数在执行过程中需要根据Redis配置的参数，检查是否设置了哨兵模式。

如果有设置哨兵模式的话，main函数会调用initSentinelConfig函数，对哨兵模式的参数进行初始化设置，以及调用initSentinel函数，初始化设置哨兵模式运行的server。有关哨兵模式运行的Redis server相关机制，我会在第21讲中给你详细介绍。

下面的代码展示了main函数中对哨兵模式的检查，以及对哨兵模式的初始化，你可以看下：

```
...
//判断server是否设置为哨兵模式
if (server.sentinel_mode) {
        initSentinelConfig();  //初始化哨兵的配置
        initSentinel();   //初始化哨兵模式
}
...
```

除了检查哨兵模式以外，main函数还会检查是否要执行RDB检测或AOF检查，这对应了实际运行的程序是redis-check-rdb或redis-check-aof。在这种情况下，main函数会调用redis_check_rdb_main函数或redis_check_aof_main函数，检测RDB文件或AOF文件。你可以看看下面的代码，其中就展示了main函数对这部分内容的检查和调用：

```
...
//如果运行的是redis-check-rdb程序，调用redis_check_rdb_main函数检测RDB文件
```

```
if (strstr(argv[0],"redis-check-rdb") != NULL)
   redis_check_rdb_main(argc,argv,NULL);
//如果运行的是redis-check-aof程序，调用redis_check_aof_main函数检测AOF文件
else if (strstr(argv[0],"redis-check-aof") != NULL)
   redis_check_aof_main(argc,argv);
...
```

阶段三：运行参数解析

在这一阶段，main函数会对命令行传入的参数进行解析，并且调用loadServerConfig函数，对命令行参数和配置文件中的参数进行合并处理，然后为Redis各功能模块的关键参数设置合适的取值，以便server能高效地运行。

阶段四：初始化server

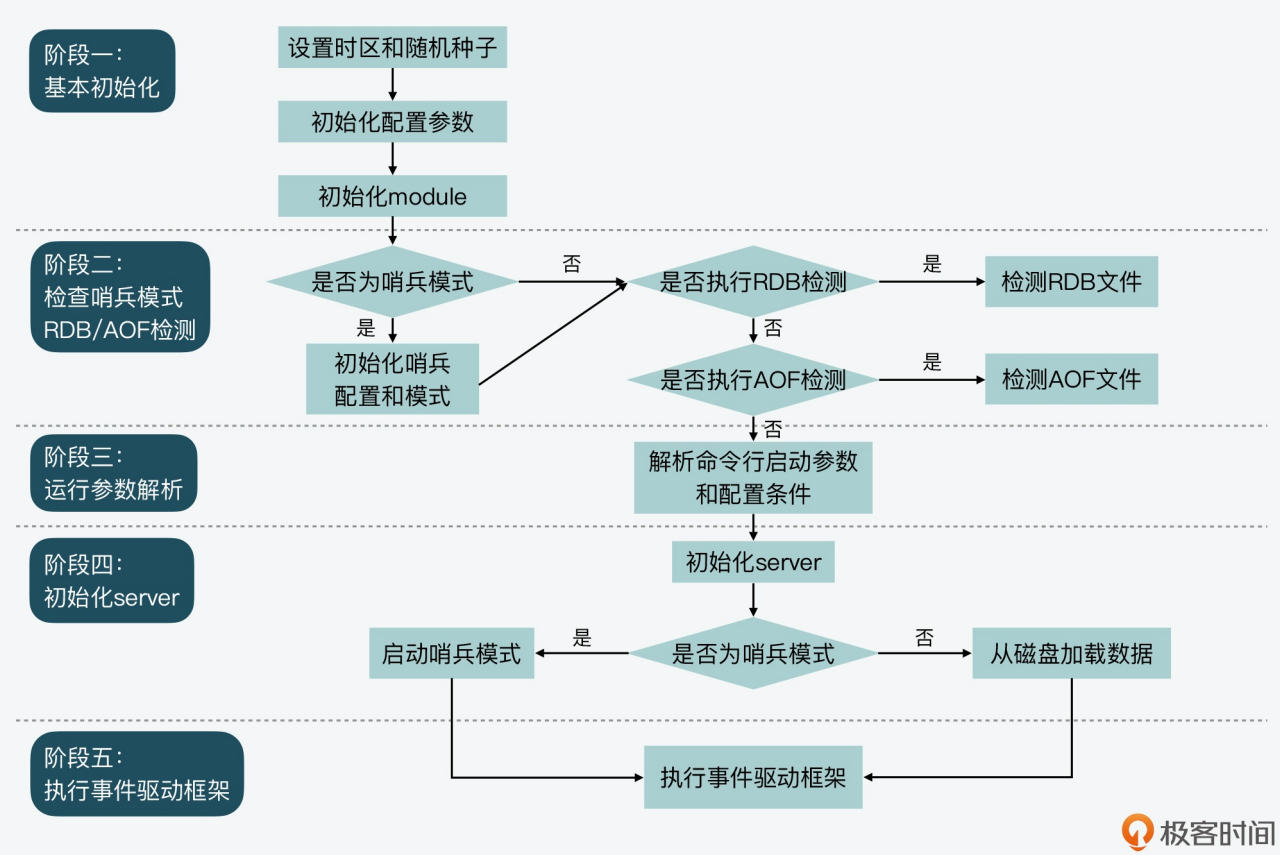
在完成对运行参数的解析和设置后，main函数会调用initServer函数，对server运行时的各种资源进行初始化工作。这主要包括了server资源管理所需的数据结构初始化、键值对数据库初始化、server网络框架初始化等。

而在调用完initServer后，main函数还会再次判断当前server是否为哨兵模式。如果是哨兵模式，main函数会调用sentinelIsRunning函数，设置启动哨兵模式。否则的话，main函数会调用loadDataFromDisk函数，从磁盘上加载AOF或者是RDB文件，以便恢复之前的数据。

阶段五：执行事件驱动框架

为了能高效处理高并发的客户端连接请求，Redis采用了事件驱动框架，来并发处理不同客户端的连接和读写请求。所以，main函数执行到最后时，会调用aeMain函数进入事件驱动框架，开始循环处理各种触发的事件。

我把刚才介绍的五个阶段涉及到的关键操作，画在了下面的图中，你可以再回顾下。



那么，在这五个阶段当中，阶段三、四和五其实就包括了Redis server启动过程中的关键操作。所以接下来，我们就来依次学习下这三个阶段中的主要工作。

Redis运行参数解析与设置

我们知道，Redis提供了丰富的功能，既支持多种键值对数据类型的读写访问，还支持数据持久化保存、主从复制、切片集群等。而这些功能的高效运行，其实都离不开相关功能模块的关键参数配置。

举例来说，Redis为了节省内存，设计了内存紧凑型的数据结构来保存Hash、Sorted Set等键值对类型。但是在使用了内存紧凑型的数据结构之后，如果往数据结构存入的元素个数过多或元素过大的话，键值对的访问性能反而会受到影响。因此，为了平衡内存使用量和系统访问性能，我们就可以通过参数，来设置和调节内存紧凑型数据结构的使用条件。

也就是说，**掌握这些关键参数的设置，可以帮助我们提升Redis实例的运行效率。**

不过，Redis的参数有很多，我们无法在一节课中掌握所有的参数设置。所以下面，我们可以先来学习下Redis的主要参数类型，这样就能对各种参数形成一个全面的了解。同时，我也会给你介绍一些和server运行关系密切的参数及其设置方法，以便你可以配置好这些参数，让server高效运行起来。

Redis的主要参数类型

首先，Redis运行所需的各种参数，都统一定义在了server.h文件的redisServer结构体中。根据参数作用的范围，我把各种参数划分为了七大类型，包括通用参数、数据结构参数、网络参数、持久化参数、主从复制参数、切片集群参数、性能优化参数。具体你可以参考下面表格中的内容。

通用参数	定义了Redis server运行的基本配置，比如server包含的数据库数量、server后台任务运行频率、server运行日志文件和级别。此外，我把Redis运行时资源限制参数也归类为通用参数，比如server运行连接的最大客户端数量、server运行时运行的最大内存容量、客户端缓冲区最大大小等。
数据结构参数	定义了内存紧凑型数据结构的使用条件，比如，基于压缩列表实现Hash的最大哈希项个数，或最大哈希表大小。
网络参数	定义了server的监听地址、端口、TCP缓冲区大小等参数。
持久化参数	定义了AOF和RDB执行时所需的各种配置，比如AOF日志文件名、AOF日志文件落盘方式、AOF文件重写时是否打开日志同步写、RDB镜像文件压缩功能等。
主从复制参数	定义了主从库复制时的关键机制，包括主库信息、主从复制方式、主从复制积压缓冲区大小、从库是否只读等。
切片集群参数	定义了切片集群运行时的关键机制，包括是否启动切片集群、切片集群是否支持故障切换、集群节点心跳超时时间等。
性能优化参数	包括两方面内容，一方面是server运行时的监控机制，比如慢查询的触发条件、延迟监控的触发阈值等；另一方面主要是和Redis提供的优化机制相关，比如惰性删除机制、主动内存碎片整理机制等。

这样，如果你能按照上面的划分方法给Redis参数进行归类，那么你就可以发现，这些参数实际和Redis的主要功能机制是相对应的。所以，如果你要深入掌握这些参数的典型配置值，你就需要对相应功能机制的工作原理有所了解。我在接下来的课程中，也会在介绍Redis功能模块设计的同时，带你了解下其相应的典型参数配置。

好，现在我们就了解了Redis的七大参数类型，以及它们基本的作用范围，那么下面我们就接着来学习下，Redis是如何针对这些参数进行设置的。

Redis参数的设置方法

Redis对运行参数的设置实际上会经过三轮赋值，分别是默认配置值、命令行启动参数，以及配置文件配置值。

首先，Redis在main函数中会**先调用initServerConfig函数，为各种参数设置默认值**。参数的默认值统一定义在server.h文件中，都是以CONFIG_DEFAULT开头的宏定义变量。下面的代码显示的是部分参数的默认值，你可以看下。

```
#define CONFIG_DEFAULT_HZ        10   //server后台任务的默认运行频率
#define CONFIG_MIN_HZ            1    // server后台任务的最小运行频率
#define CONFIG_MAX_HZ            500 // server后台任务的最大运行频率
#define CONFIG_DEFAULT_SERVER_PORT  6379  //server监听的默认TCP端口
#define CONFIG_DEFAULT_CLIENT_TIMEOUT  0  //客户端超时时间，默认为0，表示没有超时限制
```

在server.h中提供的默认参数值，一般都是典型的配置值。因此，如果你在部署使用Redis实例的过程中，对Redis的工作原理不是很了解，就可以使用代码中提供的默认配置。

当然，如果你对Redis各功能模块的工作机制比较熟悉的话，也可以自行设置运行参数。你可以在启动Redis程序时，在命令行上设置运行参数的值。比如，如果你想将Redis server监听端口从默认的6379修改为7379，就可以在命令行上设置port参数为7379，如下所示：

```
./redis-server --port 7379
```

这里，你需要注意的是，Redis的命令行参数设置需要使用**两个减号“–”**来表示相应的参数名，否则的话，Redis就无法识别所设置的运行参数。

Redis在使用initServerConfig函数对参数设置默认配置值后，接下来，main函数就会**对Redis程序启动时的命令行参数进行逐一解析**。

main函数会把解析后的参数及参数值保存成字符串，接着，main函数会**调用loadServerConfig函数进行第二和第三轮的赋值**。以下代码显示了main函数对命令行参数的解析，以及调用loadServerConfig函数的过程，你可以看下。

```
int main(int argc, char **argv) {
…
//保存命令行参数
for (j = 0; j < argc; j++) server.exec_argv[j] = zstrdup(argv[j]);
…
if (argc >= 2) {
   …
   //对每个运行时参数进行解析
   while(j != argc) {
      …
   }
   …
   //
   loadServerConfig(configfile,options);
}
```

这里你要知道的是，loadServerConfig函数是在config.c文件中实现的，该函数是以Redis配置文件和命令行参数的解析字符串为参数，将配置文件中的所有配置项读取出来，形成字符串。紧接着，loadServerConfig函数会把解析后的命令行参数，追加到配置文件形成的配置项字符串。

这样一来，配置项字符串就同时包含了配置文件中设置的参数，以及命令行设置的参数。

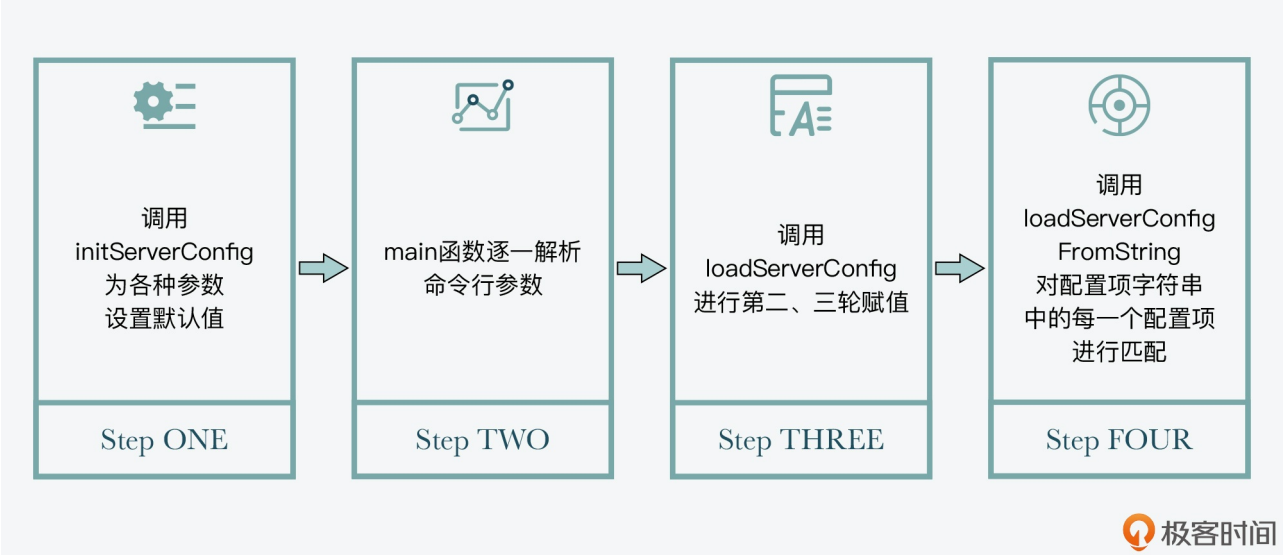
最后，loadServerConfig函数会进一步**调用loadServerConfigFromString函数，对配置项字符串中的每一个配置项进行匹配**。一旦匹配成功，loadServerConfigFromString函数就会按照配置项的值设置server的参数。

以下代码显示了loadServerConfigFromString函数的部分内容。这部分代码是使用了条件分支，来依次比较配置项是否是“timeout”和“tcp-keepalive”，如果匹配上了，就将server参数设置为配置项的值。

同时，代码还会检查配置项的值是否合理，比如是否小于0。如果参数值不合理，程序在运行时就会报错。另外对于其他的配置项，loadServerConfigFromString函数还会继续使用elseif分支进行判断。

```
loadServerConfigFromString(char *config) {
   …
   //参数名匹配，检查参数是否为“timeout“
   if (!strcasecmp(argv[0],"timeout") && argc == 2) {
            //设置server的maxidletime参数
 server.maxidletime = atoi(argv[1]);
 //检查参数值是否小于0，小于0则报错
            if (server.maxidletime < 0) {
                err = "Invalid timeout value"; goto loaderr;
            }
   }
  //参数名匹配，检查参数是否为“tcp-keepalive“
  else if (!strcasecmp(argv[0],"tcp-keepalive") && argc == 2) {
            //设置server的tcpkeepalive参数
 server.tcpkeepalive = atoi(argv[1]);
 //检查参数值是否小于0，小于0则报错
            if (server.tcpkeepalive < 0) {
                err = "Invalid tcp-keepalive value"; goto loaderr;
            }
   }
   …
}
```

好了，到这里，你应该就了解了Redis server运行参数配置的步骤，我也画了一张图，以便你更直观地理解这个过程。



在完成参数配置后，main函数会开始调用initServer函数，对server进行初始化。所以接下来，我们继续来了解Redis server初始化时的关键操作。

initServer：初始化Redis server

Redis server的初始化操作，主要可以分成三个步骤。

- 第一步，Redis server运行时需要对多种资源进行管理。

比如说，和server连接的客户端、从库等，Redis用作缓存时的替换候选集，以及server运行时的状态信息，这些资源的管理信息都会在**initServer函数**中进行初始化。

我给你举个例子，initServer函数会创建链表来分别维护客户端和从库，并调用evictionPoolAlloc函数（在evict.c中）采样生成用于淘汰的候选key集合。同时，initServer函数还会调用resetServerStats函数（在server.c中）重置server运行状态信息。

- 第二步，在完成资源管理信息的初始化后，initServer函数会对Redis数据库进行初始化。

因为一个Redis实例可以同时运行多个数据库，所以initServer函数会使用一个循环，依次为每个数据库创建相应的数据结构。

这个代码逻辑是实现在initServer函数中，**它会为每个数据库执行初始化操作**，包括创建全局哈希表，为过期key、被BLPOP阻塞的key、将被PUSH的key和被监听的key创建相应的信息表。

```
for (j = 0; j < server.dbnum; j++) {
        //创建全局哈希表
        server.db[j].dict = dictCreate(&dbDictType,NULL);
        //创建过期key的信息表
        server.db[j].expires = dictCreate(&keyptrDictType,NULL);
        //为被BLPOP阻塞的key创建信息表
        server.db[j].blocking_keys = dictCreate(&keylistDictType,NULL);
        //为将执行PUSH的阻塞key创建信息表
        server.db[j].ready_keys = dictCreate(&objectKeyPointerValueDictType,NULL);
        //为被MULTI/WATCH操作监听的key创建信息表
        server.db[j].watched_keys = dictCreate(&keylistDictType,NULL);
        …
    }
```

- 第三步，initServer函数会为运行的Redis server创建事件驱动框架，并开始启动端口监听，用于接收外部请求。

注意，为了高效处理高并发的外部请求，initServer在创建的事件框架中，针对每个监听IP上可能发生的客户端连接，都创建了监听事件，用来监听客户端连接请求。同时，initServer为监听事件设置了相应的**处理函数acceptTcpHandler**。

这样一来，只要有客户端连接到server监听的IP和端口，事件驱动框架就会检测到有连接事件发生，然后调用acceptTcpHandler函数来处理具体的连接。你可以参考以下代码中展示的处理逻辑：

```
//创建事件循环框架
server.el = aeCreateEventLoop(server.maxclients+CONFIG_FDSET_INCR);
…
//开始监听设置的网络端口
if (server.port != 0 &&
        listenToPort(server.port,server.ipfd,&server.ipfd_count) == C_ERR)
        exit(1);
…
//为server后台任务创建定时事件
if (aeCreateTimeEvent(server.el, 1, serverCron, NULL, NULL) == AE_ERR) {
```

```
        serverPanic("Can't create event loop timers.");
        exit(1);
}
…
//为每一个监听的IP设置连接事件的处理函数acceptTcpHandler
for (j = 0; j < server.ipfd_count; j++) {
        if (aeCreateFileEvent(server.el, server.ipfd[j], AE_READABLE,
            acceptTcpHandler,NULL) == AE_ERR)
       { … }
}
```

那么到这里，Redis server在完成运行参数设置和初始化后，就可以开始处理客户端请求了。为了能持续地处理并发的客户端请求，**server在main函数的最后，会进入事件驱动循环机制**。而这就是接下来，我们要了解的事件驱动框架的执行过程。

执行事件驱动框架

事件驱动框架是Redis server运行的核心。该框架一旦启动后，就会一直循环执行，每次循环会处理一批触发的网络读写事件。关于事件驱动框架本身的设计思想与实现方法，我会在第9至11讲给你具体介绍。这节课，我们主要是学习Redis入口的main函数中，是如何转换到事件驱动框架进行执行的。

其实，进入事件驱动框架开始执行并不复杂，main函数直接调用事件框架的**主体函数aeMain**（在ae.c文件中）后，就进入事件处理循环了。

当然，在进入事件驱动循环前，main函数会分别调用aeSetBeforeSleepProc和aeSetAfterSleepProc两个函数，来设置每次进入事件循环前server需要执行的操作，以及每次事件循环结束后server需要执行的操作。下面代码显示了这部分的执行逻辑，你可以看下。

```
int main(int argc, char **argv) {
    …
    aeSetBeforeSleepProc(server.el,beforeSleep);
    aeSetAfterSleepProc(server.el,afterSleep);
    aeMain(server.el);
 aeDeleteEventLoop(server.el);
 ...
}
```

小结

今天这节课，我们通过server.c文件中main函数的设计和实现思路，了解了Redis server启动后的五个主要阶段。在这五个阶段中，运行参数解析、server初始化和执行事件驱动框架则是Redis sever启动过程中的三个关键阶段。所以相应的，我们需要重点关注以下三个要点。

第一，main函数是使用initServerConfig给server运行参数设置默认值，然后会解析命令行参数，并通过loadServerConfig读取配置文件参数值，将命令行参数追加至配置项字符串。最后，Redis会调用loadServerConfigFromString函数，来完成配置文件参数和命令行参数的设置。

第二，在Redis server完成参数设置后，initServer函数会被调用，用来初始化server资源管理的主要结构，

同时会初始化数据库启动状态，以及完成server监听IP和端口的设置。

第三，一旦server可以接收外部客户端的请求后，main函数会把程序的主体控制权，交给事件驱动框架的入口函数，也就aeMain函数。aeMain函数会一直循环执行，处理收到的客户端请求。到此为止，server.c中的main函数功能就已经全部完成了，程序控制权也交给了事件驱动循环框架，Redis也就可以正常处理客户端请求了。

实际上，Redis server的启动过程从基本的初始化操作，到命令行和配置文件的参数解析设置，再到初始化server各种数据结构，以及最后的执行事件驱动框架，这是一个典型的网络服务器执行过程，你在开发网络服务器时，就可以作为参考。

而且，掌握了启动过程中的初始化操作，还可以帮你解答一些使用中的疑惑。比如，Redis启动时是先读取RDB文件，还是先读取AOF文件。如果你了解了Redis server的启动过程，就可以从loadDataFromDisk函数中看到，Redis server会先读取AOF；而如果没有AOF，则再读取RDB。

所以，掌握Redis server启动过程，有助于你更好地了解Redis运行细节，这样当你遇到问题时，就知道还可以从启动过程中去溯源server的各种初始状态，从而助力你更好地解决问题。

每课一问

Redis源码的main函数在调用initServer函数之前，会执行如下的代码片段，你知道这个代码片段的作用是什么吗？

```
int main(int argc, char **argv) {
...
server.supervised = redisIsSupervised(server.supervised_mode);
int background = server.daemonize && !server.supervised;
if (background) daemonize();
...
}
```

欢迎在留言区分享你的答案和见解，我们一起交流讨论。如果觉得有收获，也欢迎你把今天的内容分享给更多的朋友。

精选留言：

- Kaito 2021-08-12 00:12:03
  Redis 启动流程，主要的工作有：

  1、初始化前置操作（设置时区、随机种子）

  2、初始化 Server 的各种默认配置（server.c 的 initServerConfig 函数），默认配置见 server.h 中的 CONFIG_DEFAULT_XXX，比较典型的配置有：

  - 默认端口
  - 定时任务频率
  - 数据库数量
  - AOF 刷盘策略

- 淘汰策略
- 数据结构转换阈值
- 主从复制参数

3、加载配置启动参数，覆盖默认配置（config.c 的 loadServerConfig 函数）：

- 解析命令行参数
- 解析配置文件

3、初始化 Server（server.c 的 initServer 函数），例如会初始化：

- 共享对象池
- 客户端链表
- 从库链表
- 监听端口
- 全局哈希表
- LRU 池
- 注册定时任务函数
- 注册监听请求函数

4、启动事件循环（ae.c 的 aeMain 函数）

- 处理请求
- 处理定时任务

这里补充一下，初始化 Server 完成后，Redis 还会启动 3 类后台线程（server.c 的 InitServerLast 函数），协助主线程工作（异步释放 fd、AOF 每秒刷盘、lazyfree）。

课后题：Redis 源码的 main 函数在调用 initServer 函数之前，会执行如下的代码片段，你知道这个代码片段的作用是什么吗？

```
int main(int argc, char **argv) {
...
server.supervised = redisIsSupervised(server.supervised_mode);
int background = server.daemonize && !server.supervised;
if (background) daemonize();
...
}
```

Redis 可以配置以守护进程的方式启动（配置文件 daemonize = yes），也可以把 Redis 托管给 upstart 或 systemd 来启动 / 停止（supervised = upstart|systemd|auto）。 [6赞]

- 曾轼麟 2021-08-13 11:51:05

感谢老师的文章，先回答老师提出的问题：文章中代码片段的作用是？根据关键词我找到代码位于，main函数中loadServerConfig之后执行的，那么这块代码主要任务和顺序如下所示：
1、此时redis的config是已经初始化完成的
2、执行redisIsSupervised其目底主要是判断当redis进程是否运行中
3、判断daemonize是否开启（如果启动设置了daemonize参数那么这里参数已经被填充）
4、之前并没有存活的redis实例，并且开启了daemonize配置，那么执行daemonize函数挂起后台运行

这里需要解释一下，执行daemonize()函数的时候本质是fork()进程，并不是大多文章说的那样是守护线程，父亲fork成功后是直接退出（在前端的效果就是，启动命令任务完成但是ps是有一个redis的进程），剩余的任务是交给fork出的子进程完成的（可以参考《深入理解操作系统》 第8章-异常控制流-进程中的内容）

读完这篇文章后，我个人尝试画出整个redis启动的时序图，发现整个思路就很清晰了，建议同样阅读文章的同学可以尝试画一下，我总结一下我读完文章后的理解：
1、redis的整体启动流程是按照 【初始化默认配置】->【解析启动命令】->【初始化server】->【初始化并启动事件驱动框架】 进行
2、整个运行中的redis，其实就是一个永不停歇的while循环，位于aeMain中（运行中的事件驱动框架）
3、在事件驱动框架中有两个钩子函数 beforeSleep 和 aftersleep，在每次while循环中都会触发这两个函数，后面用来实现事件触发的效果

此外我发现了一个细节点：我在近期版本的redis6的分支上，发现在启动事件驱动框架之前（执行aeMain之前）会执行一个redisSetCpuAffinity函数，其效果有点类似于绑核的效果，那么是否可以认为从redis6开始其实不需要运维帮忙绑核redis自身就能做到绑核的效果呢？

- 末日，成欢 2021-08-12 12:19:43
  起始处设置随机种子是为了做什么？

- 那时刻 2021-08-12 10:17:28
  请问老师，Redis server 会先读取 AOF；而如果没有 AOF，则再读取 RDB。为什么不先读rdb，再读aof呢？

- 可怜大灰狼 2021-08-12 10:04:03
  先判断是否upstart或者systemd托管。再判断是否需要守护进程，内部还是fork+setsid