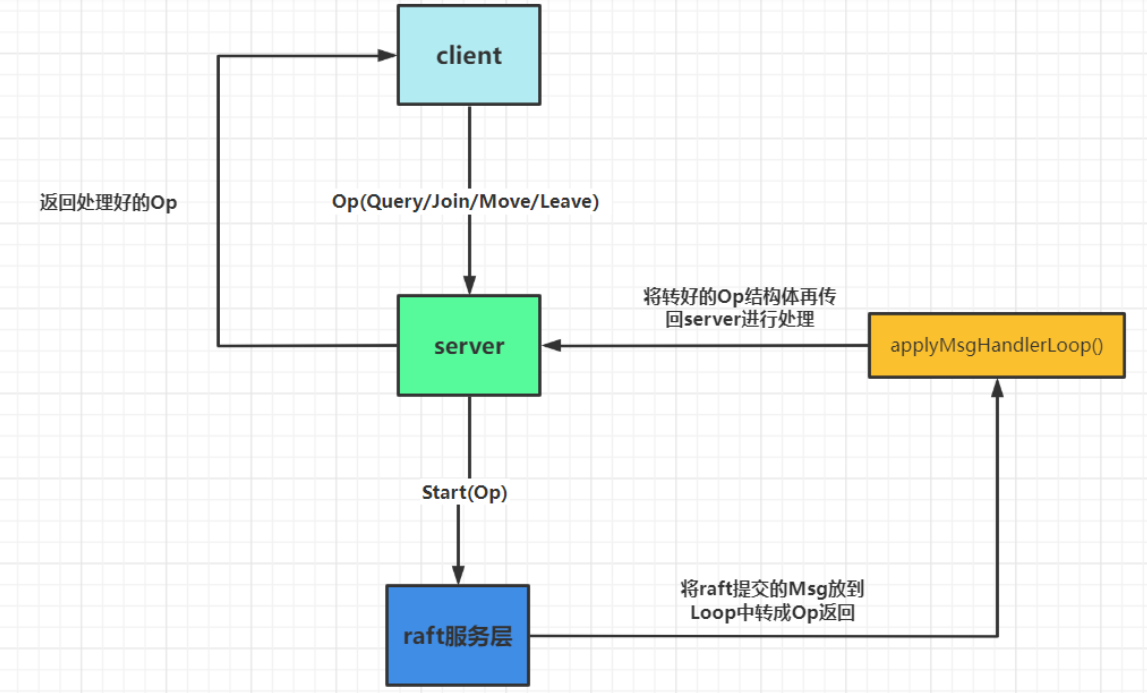
4A的流程如下图所示（写报告时记得重新画图，不要用这个图），具体来说就是搭建一个像lab3那样的包含客户端与服务器的数据库系统，但是这个只是运行规律像是一个服务器，它的本质是一个分片控制器。客户端发一个内容为Op结构体的RPC包，这个Op包含这些操作：Query/Join/Move/Leave，服务器（分片控制器）收到请求之后借助Raft层来执行这些操作，然后raft将执行完毕的反馈信息传给服务器，服务器再给客户端发送反馈。



在这个实验中，分片和组是分布式系统中的两个关键概念，用于构建分片（Sharded）KV 存储系统。以下是对这两个概念的详细解释：

分片（Shard）：

定义： 分片是将数据集划分成较小的部分的过程。在这个实验中，数据集是键值对（Key-Value pairs）的集合，每个键值对由一个唯一的键和相应的值组成。

实现： 分片的目标是将整个键值对集合分散存储在不同的服务器上，以便实现分布式存储和处理。每个分片由一个组（Group）负责管理，组内的服务器存储该分片的数据。每个分片都有一个唯一的编号，用于标识不同的分片。

组（Group）：

定义： 组是一组服务器的集合，负责管理一个或多个分片。组内的服务器协作以提供对其管理的分片的读取和写入操作。

实现： 在这个实验中，组的概念是分片控制器（ShardCtrler）维护的关键组件。每个组有一个唯一的组编号（GID），用于区分不同的组。组内的服务器共同管理一个或多个分片，负责处理针对这些分片的操作。组的成员可能随时间变化，可以通过“加入”（Join）和“离开”（Leave）操作来调整。

操作类型：

Join： 将一组新的服务器加入到系统中，以便它们可以管理分片。

Leave： 从系统中移除一组服务器，不再管理分片。

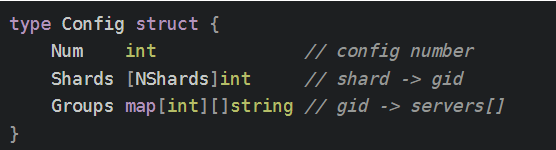
Move： 将一个分片从一个组移动到另一个组，以实现负载均衡或容错性。

Query： 查询系统的当前配置，即哪些组管理哪些分片。

通过这种方式，系统可以水平扩展，处理更多的键值对，同时具备负载均衡和容错性。每个组可以在其管理的分片上并行地处理读取和写入操作，从而提高系统的性能。

那么这个实验其实就是要利用raft进行“配置”的统一，什么叫“配置”？其实就是每个服务器里面有一个config，里面存放得就是一系列版本的配置信息，最新的下标对应的最新的配置。而通过config结构体的定义，可以看出“配置”有三个成员：

一个对应的版本的配置编号，分片所对应的组信息（实验中规定的分片为10个），每个组对应的服务器映射名称列表（也就是组信息）



本文件夹代码修改的是client.go、common.go、server.go，细节如下：

client.go

Clerk结构体：这个就是客户端的结构体，主要在其中添加了seqId字段用于标识每次请求的序列号、leaderId字段用来记录当前leader服务器的编号、clientId字段记录本服务器的编号

MakeClerk函数：用于初始化创建一个Clerk客户端实例

Query函数：用于向服务器发送查询请求

Join函数：用于请求加入新服务器

Leave函数：用于发送离开GID的请求

Move函数：用于发送移动分片的请求

其他函数无可注意的修改

common.go

给增删查改的RPC信息结构体中都增加了seqId字段用于标识每次请求的序列号、clientId字段记录本服务器的编号

server.go

*结构体定义部分：*

常量设置部分设置了一些常量，值得注意的是把加入、离开、移动、查询的超时时间设置为100毫秒。

ShardCtrler结构体：定义了分片控制器，写代码重点在里面添加了以下字段：

·configs数组用来储存配置；

·seqMap用来进行重复性检测，防止Op被重复执行，

具体来说，seqMap 用于存储每个客户端 (ClientId) 最后一次成功执行的操作的序列号 (SeqId)。这样，分片控制器就可以检测到相同客户端发来的相同序列号的重复请求。

在实现中，当分片控制器接收到来自客户端的一个新请求时，首先会检查该请求的客户端 ID 和序列号是否在 seqMap 中存在。如果存在，说明这是一个重复的请求，可以直接忽略。如果不存在，就处理这个请求，并更新 seqMap 中该客户端的最新序列号。

·waitChMap是一个等待通道，具体来说，waitChMap 的用途是：

等待通道的创建： 当分片控制器收到客户端请求，需要向 Raft 模块提交一个操作（Op）时，会为该操作创建一个等待通道 (chan Op)。

操作完成时的通知： 等待通道用于在操作完成后通知等待的线程。在执行操作的过程中，Raft 模块会将完成的操作发送到对应的等待通道，通知等待的线程操作已完成。

操作的幂等性保证： 通过 waitChMap，分片控制器可以确保每个操作只被执行一次。在操作完成后，对应的等待通道会被删除，以防止重复处理相同的操作。

在具体的实现中，当一个操作被提交到 Raft 模块时，分片控制器会创建一个等待通道并将其与操作的索引关联起来。随后，当 Raft 模块完成了对应的操作时，将结果发送到对应的等待通道，通知分片控制器操作已完成。这种机制有助于实现操作的一致性和幂等性。

Op结构体：就是客户端发送的操作RPC包，里面定义了本次操作的类型以及一些操作序列号等相关信息。

*操作定义部分：*

Join函数：这个函数处理客户端加入新服务器的请求。首先，它检查当前节点是否是领导者。然后，封装加入新服务器的操作并将其传递给底层Raft模块进行复制。接着，创建或获取等待通道，并在函数退出时删除通道。设置超时计时器，等待通道的回复或超时。最后，根据接收到的回复Op验证其与请求Op的一致性，并回复相应的操作结果或错误信息。

Leave函数：这个函数处理客户端离开GID的请求。首先，它检查当前节点是否是领导者。然后，封装离开GID的操作并将其传递给底层Raft模块进行复制。接着，创建或获取等待通道，并在函数退出时删除通道。设置超时计时器，等待通道的回复或超时。最后，根据接收到的回复Op验证其与请求Op的一致性，并回复相应的操作结果或错误信息。

Move函数：这个函数处理客户端移动Shard的请求。首先，它检查当前节点是否是领导者。然后，封装移动Shard的操作并将其传递给底层Raft模块进行复制。接着，创建或获取等待通道，并在函数退出时删除通道。设置超时计时器，等待通道的回复或超时。最后，根据接收到的回复Op验证其与请求Op的一致性，并回复相应的操作结果或错误信息。

Query函数：这个函数处理客户端的查询请求。首先，它检查当前节点是否是领导者。然后，封装查询操作并将其传递给底层Raft模块进行复制。接着，创建或获取等待通道，并在函数退出时删除通道。设置超时计时器，等待通道的回复或超时。最后，根据接收到的回复Op验证其与请求Op的一致性，并回复相应的查询结果或错误信息。

*Loop部分：*

applyMsgHandlerLoop函数：applyMsgHandlerLoop 函数是分片控制器 (ShardCtrler) 中的一个循环，负责处理来自 Raft 模块的 applyCh 通道的消息。具体来说，它处理 raft.ApplyMsg 结构体中的命令，执行相应的操作，并将结果发送到等待通道中，以便通知发起该操作的客户端。

以下是该函数的主要作用和流程：

循环监听 applyCh 通道：该函数一直在循环中，监听 applyCh 通道上的消息。applyCh 是 Raft 模块用于将已提交的日志条目传递给状态机的通道。

处理已提交的命令：当从 applyCh 收到消息时，首先判断消息的类型。如果是已提交的命令（msg.CommandValid 为真），则说明有新的操作需要执行。

检查命令的重复性：在执行操作之前，会检查该操作是否是重复的。这是通过 ifDuplicate 函数来实现的，该函数检查操作的客户端 ID 和序列号是否已经在 seqMap 中存在。如果是重复的，就忽略该操作，避免重复执行。

执行相应的操作：根据命令的类型，分片控制器执行相应的操作。当前支持的操作包括 Join（加入新的组）、Leave（离开组）、Move（移动分片）以及Query（查询状态）。在执行操作的过程中，会更新状态，并将结果存储到 seqMap 中，表示该客户端的最新操作序列号。

通知等待通道：在处理完操作后，通过 getWaitCh 函数获取对应的等待通道，并将操作的结果发送到该通道。这样，发起操作的客户端就能够得知操作的执行结果。

总体来说，applyMsgHandlerLoop 负责将已提交的 Raft 日志转化为实际的操作，并确保每个操作只执行一次，同时通知发起操作的客户端操作的执行结果。这是一个典型的状态机应用场景，确保系统在分布式环境下的一致性和正确性。

*handler部分：*

JoinHandler函数：该函数是处理加入新组的操作的函数。具体来说，对于Join进来的新组（servers），函数会创建一个新的配置，将这些新组添加到当前配置的分组中，并进行负载均衡，使得各个组的分片数尽可能均匀。在处理过程中，函数会记录每个分组的分片数，以便进行负载均衡。最终，返回一个更新后的配置。

在配置更新时，该函数允许GID的重用，即一个GID可以在当前配置中Join，然后Leave，然后再次Join。

总的来说，JoinHandler的主要功能是根据新加入的组，更新当前的配置，使得分片分布尽可能均匀。

LeaveHandler函数：该函数是处理离开（Leave）操作的函数。具体来说，对于传入的离开的GIDs（gids），函数会创建一个新的配置，将这些GID对应的分组从当前配置的分组中删除，并进行负载均衡，使得各个组的分片数尽可能均匀。在处理过程中，函数会记录每个分组的分片数，以便进行负载均衡。最终，返回一个更新后的配置。

在配置更新时，LeaveHandler会删除对应GIDs的分组，同时进行负载均衡。离开的GID对应的分片数会置为0，然后重新计算各个分组的分片数，以保持均衡。最终，返回一个更新后的配置。

总的来说，LeaveHandler的主要功能是根据离开的GIDs，更新当前的配置，使得分片分布尽可能均匀。

MoveHandler函数：是用来处理 Move 操作的，其主要功能是将指定的分片（shard）移动到目标分组（gid）。在分布式系统中，这个操作通常涉及到重新配置系统的分片分布，以达到负载均衡或者其他需求。

具体来说，函数的操作步骤如下：

获取当前最新的配置信息 lastConfig，即最后一个配置。

创建一个新的配置 newConfig，复制了当前配置的所有信息。

在新的配置中，将指定的分片 shard 移动到目标分组 gid 中，修改了 newConfig.Shards 的相应项。

返回包含了移动操作的新配置 newConfig。

这个函数实际上并没有涉及到 Raft 协议的操作，而是在应用层面上对分片控制器的配置进行修改。在分布式系统中，Move 操作可能会触发 Raft 协议的日志复制，从而在所有节点上进行相同的配置修改，以保持一致性。

*工具函数部分：*

ifDuplicate函数：检查给定客户端ID和序列号是否代表一个重复的请求

getWaitCh函数：获取或创建给定索引的等待通道

*负载均衡部分：*

loadBalance 函数：

功能：该函数用于在配置改变时进行负载均衡操作，平衡分片的分配，确保各个分组的负载相对均匀。

1. ****计算均分的分片数量：**** 计算均分的分片数量，即 **NShards** 除以当前存在的分组数量。
2. ****计算余数：**** 计算均分后可能剩余的分片数量，以及剩余的分组数量。
3. ****对分组按负载进行排序：**** 调用 **sortGroupShard** 函数，对分组按照分片数量进行排序，负载高的分组排在前面。
4. ****释放多余的分片：**** 遍历排序后的分组，对负载过多的分组进行处理，释放多余的分片。
5. ****为负载较少的分组分配分片：**** 对负载较少的分组进行处理，为其分配额外的分片。
6. ****返回负载均衡后的分片分配情况：**** 返回经过处理后的分片分配情况 **lastShards**。

参数：

GroupMap：表示各分组（GID）的当前分片数量。

lastShards：表示上一个配置中各分片对应的分组（GID）。

返回值：更新后的分片分配情况。

sortGroupShard 函数：

功能：该函数用于根据各分组的分片数量对分组进行排序，以便在负载均衡时优先处理负载高的分组。

参数：GroupMap 表示各分组的当前分片数量。

返回值：返回按照分片数量排序的分组列表。

moreAllocations 函数：

功能：判断当前分组是否需要额外分配分片。在进行负载均衡时，前面的分组会额外分配一个分片，用于保持相对均匀的负载。

参数：

length：表示分组的数量。

remainder：表示剩余的分片数量。

i：表示当前分组的索引。

返回值：若当前分组需要额外分片，则返回 true；否则返回 false。

这些函数共同协作，通过在配置变更时调用 loadBalance 函数，根据当前分组负载情况，对分片进行重新分配，以达到负载均衡的效果。

*初始化部分：*

StartServer函数：创建并初始化服务器（分片控制器）实体