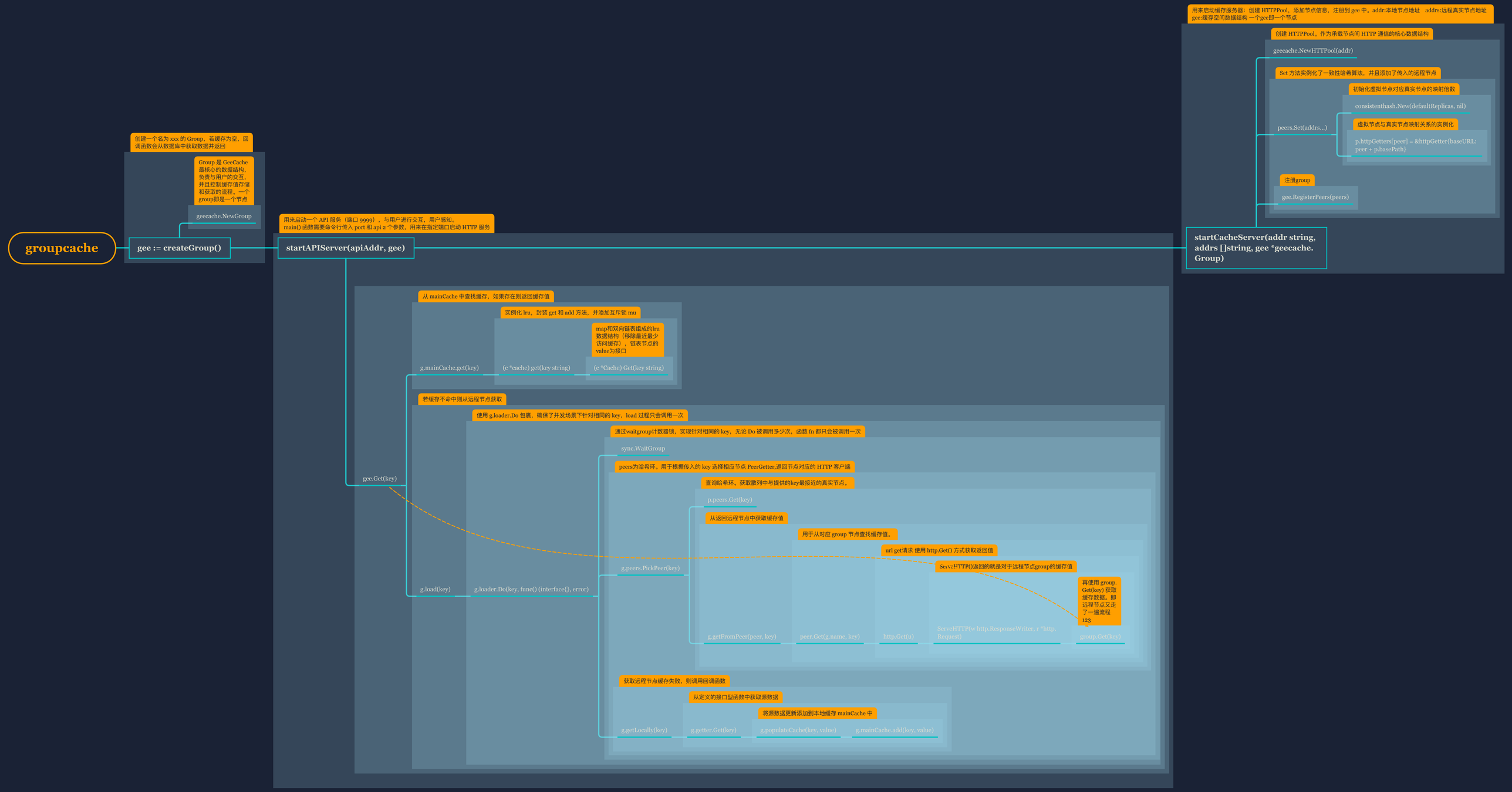
groupcache
gee := createGroup()
创建一个名为 xxx 的 Group，若缓存为空，回调函数会从数据库中获取数据并返回
Group 是 GeeCache 最核心的数据结构，负责与用户的交互，并且控制缓存值存储和获取的流程。一个 group即是一个节点
geecache.NewGroup
startAPIServer(apiAddr, gee)
用来启动一个 API 服务（端口 9999），与用户进行交互，用户感知。
main() 函数需要命令行传入 port 和 api 2 个参数，用来在指定端口启动 HTTP 服务
startCacheServer(addr string, addrs []string, gee *geecache.Group)
用来启动缓存服务器：创建 HTTPPool，添加节点信息，注册到 gee 中。addr:本地节点地址 addrs:远程真实节点地址
gee:缓存空间数据结构 一个gee即一个节点
创建 HTTPPool。作为承载节点间 HTTP 通信的核心数据结构
geecache.NewHTTPPool(addr)
Set 方法实例化了一致性哈希算法，并且添加了传入的远程节点
初始化虚拟节点对应真实节点的映射倍数
consistenthash.New(defaultReplicas, nil)
peers.Set(addrs...)
虚拟节点与真实节点映射关系的实例化
p.httpGetters[peer] = &httpGetter{baseURL: peer + p.basePath}
注册group
gee.RegisterPeers(peers)
从 mainCache 中查找缓存，如果存在则返回缓存值
实例化 lru，封装 get 和 add 方法，并添加互斥锁 mu
map和双向链表组成的lru数据结构（移除最近最少访问缓存），链表节点的value为接口
g.mainCache.get(key)
(c *cache) get(key string)
(c *Cache) Get(key string)
若缓存不命中则从远程节点获取
使用 g.loader.Do 包裹，确保了并发场景下针对相同的 key，load 过程只会调用一次
通过waitgroup计数器锁，实现针对相同的 key，无论 Do 被调用多少次，函数 fn 都只会被调用一次
sync.WaitGroup
gee.Get(key)
peers为哈希环。用于根据传入的 key 选择相应节点 PeerGetter,返回节点对应的 HTTP 客户端
查询哈希环。获取散列中与提供的key最接近的真实节点。
p.peers.Get(key)
从返回远程节点中获取缓存值
用于从对应 group 节点查找缓存值。
url get请求 使用 http.Get() 方式获取返回值
ServeHTTP()返回的就是对于远程节点group的缓存值
g.peers.PickPeer(key)
g.load(key)
g.loader.Do(key, func() (interface{}, error)
再使用 group.Get(key) 获取缓存数据，即远程节点又走了一遍流程
123
g.getFromPeer(peer, key)
peer.Get(g.name, key)
http.Get(u)
ServeHTTP(w http.ResponseWriter, r *http.Request)
group.Get(key)
获取远程节点缓存失败，则调用回调函数
从定义的接口型函数中获取源数据
将源数据更新添加到本地缓存 mainCache 中
g.getLocally(key)
g.getter.Get(key)
g.populateCache(key, value)
g.mainCache.add(key, value)