1. 首先，为什么要用框架呢，因为我们发送http请求的时候其实很多地方是重复的，比方说鉴权、分组等等，需要对每个请求单独考虑，使用框架后，将重复的内容包装起来，开发只要专注于自己的业务逻辑就可以了。
2. 对于一个框架，首先重载ServeHTTP这个函数，使用一个engine将内容包装起来，主要内容包括了路由方式、路由分组、template处理等。
3. 这之后要处理上下文Context，其本质就是把本来的http的Writer和request包装起来。然后实现一些常用的函数，比如说HTML、JSON、String等等。如果不使用这种方式，每次都请求都需要复杂的重复，比如设置请求头啊、请求状态等等。使用了上下文后，开发者可以专注于提交的内容就可以了。
4. 接着将路由本身抽离出来，路由本来是由一个哈希表维护的，就是将method和path拼接起来作为key，处理函数作为value。但是这样处理的话，只能够去应对静态路由。为了应对动态路由，这里使用了前缀树的方式，在trie.go中主要处理前缀树的插入和查找，然后router中放两个哈希映射，一个映射方法，另一个映射指针。添加一个查找路由逻辑，（这个时候需要给context中添加一个哈希映射，映射参数），返回树中节点和参数列表。
5. 然后设计分组，分组的目的就是为了统一的管理，也是为了后面中间件鉴权做准备。新建一个routerGroup结构体，他里面包括了前缀、中间件、父节点和指向engine的指针，主要增加分组函数，依靠prefix和parent来判断是否属于一个族。
6. 然后设计中间件，需要在context中添加中间件数组和index，新建一个next函数，next函数主要目的是保证中间件的运行顺序。这样设计的好处，就是可以在请求前加入中间件，也可以在请求后使用中间件。接着需要修改ServeHTTP函数，当我们接收到一个具体请求时，要判断该请求适用于哪些中间件，在这里我们简单通过 URL 的前缀来判断。
7. 最后是错误处理，（可能是框架本身没有完备的测试，导致在某些情况下出现空指针异常等情况。也有可能用户不正确的参数，触发了某些异常，例如数组越界，空指针等。如果因为这些原因导致系统宕机，必然是不可接受的。）错误处理机制其实就使用之前设计好的中间件，写一个错误处理的中间件，主要是将panic给recovery掉，并在log中打印错误路径。