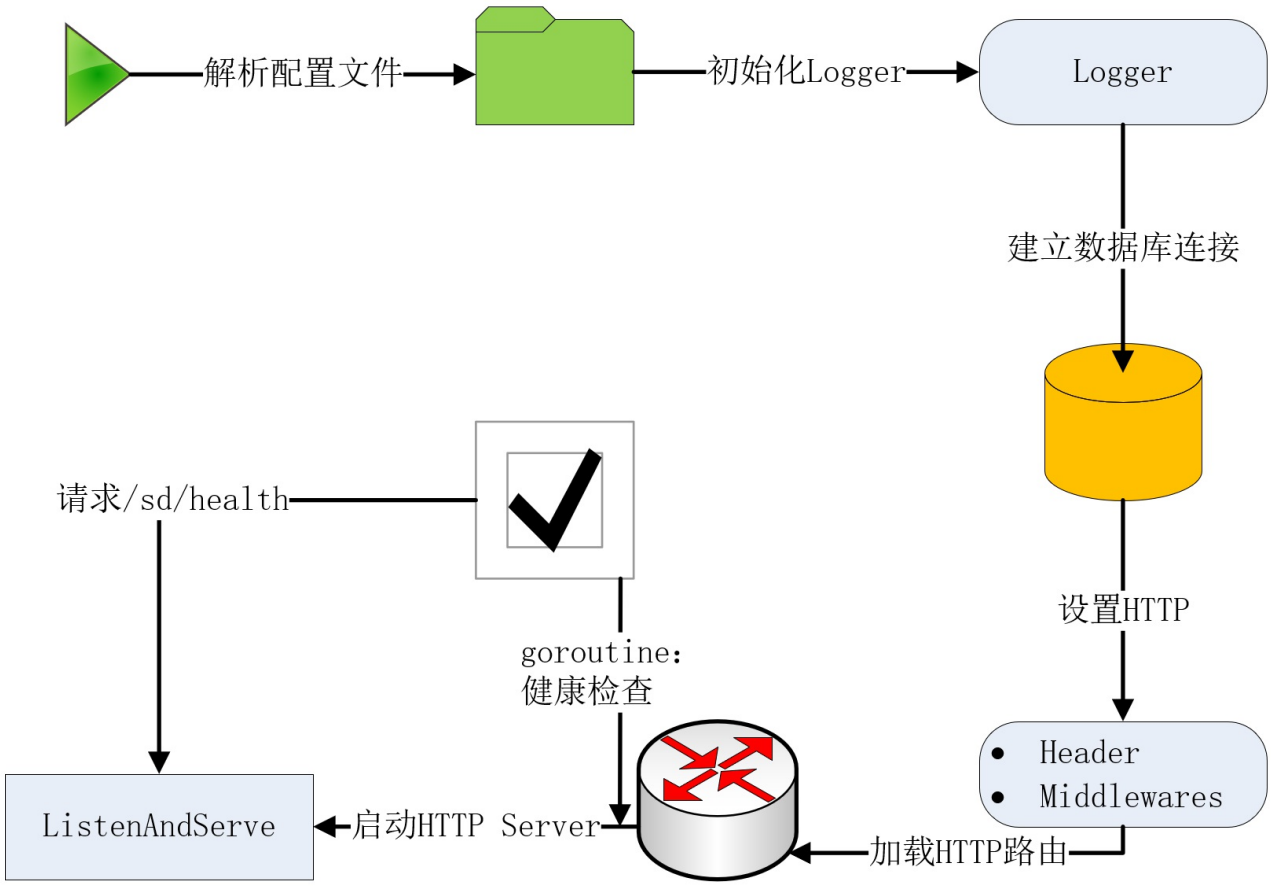
API 流程和代码结构

为了使读者在开始实战之前对 API 开发有个整体的了解，这里选择了两个流程来介绍：

- HTTP API 服务器启动流程
- HTTP 请求处理流程

本小节也提前给出了程序代码结构图，让读者从宏观上了解将要构建的 API 服务器的功能。

HTTP API 服务器启动流程



如上图，在启动一个 API 命令后，API 命令会首先加载配置文件，根据配置做后面的处理工作。通常会将日志相关的配置记录在配置文件中，在解析完配置文件后，就可以加载日志包初始化函数，来初始化日志实例，供后面的程序调用。接下来会初始化数据库实例，建立数据库连接，供后面对数据库的 CRUD 操作使用。在建立完数据库连接后，需要设置 HTTP，通常包括 3 方面的设置：

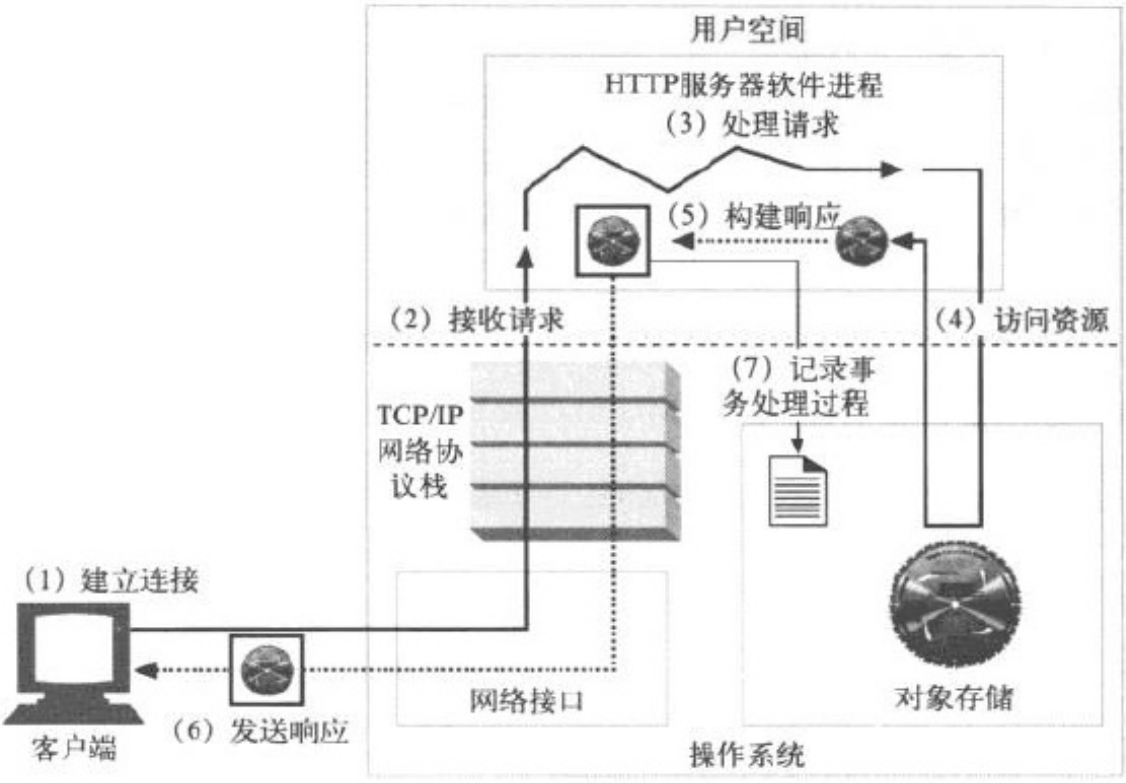
1. 设置 Header
2. 注册路由
3. 注册中间件

之后会调用 net/http 包的 ListenAndServe() 方法启动 HTTP 服务器。

在启动 HTTP 端口之前，程序会 go 一个协程，来ping HTTP 服务器的 /sd/health 接口，如果程序成功启动，ping 协程在 timeout 之前会成功返回，如果程序启动失败，则 ping 协程最终会 timeout，并终止整个程序。

> 解析配置文件、初始化 Log 、初始化数据库的顺序根据自己的喜好和需求来排即可。

HTTP 请求处理流程



一次完整的 HTTP 请求处理流程如上图所示。（图片出自[《HTTP 权威指南》(https://book.douban.com/subject/10746113/)](https://book.douban.com/subject/10746113/)，推荐想全面理解 HTTP 的读者阅读此书。）

1. 建立连接

客户端发送 HTTP 请求后，服务器会根据域名进行域名解析，就是将网站名称转变成 IP 地址：localhost -> 127.0.0.1，Linux hosts文件、DNS 域名解析等可以实现这种功能。之后通过发起 TCP 的三次握手建立连接。TCP 三次连接请参考 [TCP 三次握手详解及释放连接过程(https://blog.csdn.net/oney139/article/details/8103223)](https://blog.csdn.net/oney139/article/details/8103223)，建立连接之后就可以发送 HTTP 请求了。

2. 接收请求

HTTP 服务器软件进程，这里指的是 API 服务器，在接收到请求之后，首先根据 HTTP 请求行的信息来解析到 HTTP 方法和路径，在上图所示的报文中，方法是 GET，路径是 `/index.html`，之后根据

API 服务器注册的路由信息（大概可以理解为：HTTP 方法 + 路径和具体处理函数的映射）找到具体的处理函数。

3. 处理请求

在接收到请求之后，API 通常会解析 HTTP 请求报文获取请求头和消息体，然后根据这些信息进行相应的业务处理，HTTP 框架一般都有自带的解析函数，只需要输入 HTTP 请求报文，就可以解析到需要的请求头和消息体。通常情况下，业务逻辑处理可以分为两种：包含对数据库的操作和不包含对数据的操作。大型系统中通常两种都会有：

1. 包含对数据库的操作：需要访问数据库（增删改查），然后获取指定的数据，对数据处理后构建指定的响应结构体，返回响应包。数据库通常用的是 MySQL，因为免费，功能和性能也都能满足企业级应用的要求。
2. 不包含对数据库的操作：进行业务逻辑处理后，构建指定的响应结构体，返回响应包。

4. 记录事务处理过程

在业务逻辑处理过程中，需要记录一些关键信息，方便后期 Debug 用。在 Go 中有各种各样的日志包可以用来记录这些信息。

HTTP 请求和响应格式介绍

一个 HTTP 请求报文由请求行（request line）、请求头部（header）、空行和请求数据四部分组成，下图是请求报文的一般格式。

- 第一行必须是一个请求行（request line），用来说明请求类型、要访问的资源以及所使用的 HTTP 版本
- 紧接着是一个头部（header）小节，用来说明服务器要使用的附加信息
- 之后是一个空行
- 再后面可以添加任意的其他数据（称之为主体：body）

> HTTP 响应格式跟请求格式类似，也是由 4 个部分组成：状态行、消息报头、空行和响应数据。

目录结构

```
├── admin.sh                                # 进程的start|stop|status|restart控制文件
├── conf                                    # 配置文件统一存放目录
|   ├── config.yaml                         # 配置文件
|   ├── server.crt                          # TLS配置文件
```

```
|   └── server.key
├── config                          # 专门用来处理配置和配置文件的Go package
|   └── config.go
├── db.sql                          # 在部署新环境时，可以登录MySQL客户端，执行source db.sql创建数据库和表
├── docs                            # swagger文档，执行 swag init 生成的
|   ├── docs.go
|   └── swagger
|       ├── swagger.json
|       └── swagger.yaml
├── handler                         # 类似MVC架构中的C，用来读取输入，并将处理流程转发给实际的处理函数，最后返回结果
|   ├── handler.go
|   ├── sd                        # 健康检查handler
|   |   └── check.go
|   └── user                      # 核心：用户业务逻辑handler
|       ├── create.go             # 新增用户
|       ├── delete.go             # 删除用户
|       ├── get.go                # 获取指定的用户信息
|       ├── list.go               # 查询用户列表
|       ├── login.go              # 用户登录
|       ├── update.go             # 更新用户
|       └── user.go               # 存放用户handler公用的函数、结构体等
├── main.go                         # Go程序唯一入口
├── Makefile                        # Makefile文件，一般大型软件系统都是采用make来作为编译工具
├── model                           # 数据库相关的操作
```

```
统一放在这里，包括数据库初始化和对表的增删改查
|   ├── init.go                         # 初始化和连接数据
库
|   ├── model.go                        # 存放一些公用的go
struct
|   └── user.go                         # 用户相关的数据库
CURD操作
├── pkg                                 # 引用的包
|   ├── auth                            # 认证包
|   |   └── auth.go
|   ├── constvar                        # 常量统一存放位置
|   |   └── constvar.go
|   ├── errno                           # 错误码存放位置
|   |   ├── code.go
|   |   └── errno.go
|   ├── token
|   |   └── token.go
|   └── version                         # 版本包
|       ├── base.go
|       ├── doc.go
|       └── version.go
├── README.md                           # API目录README
├── router                              # 路由相关处理
|   ├── middleware                      # API服务器用的是
Gin Web框架，Gin中间件存放位置
|   |   ├── auth.go
|   |   ├── header.go
|   |   ├── logging.go
|   |   └── requestid.go
|   └── router.go
├── service                             # 实际业务处理函数
存放位置
|   └── service.go
```

```
├── util                                    # 工具类函数存放目
录
|   ├── util.go
|   └── util_test.go
└── vendor                                  # vendor目录用来
管理依赖包
    ├── github.com
    ├── golang.org
    ├── gopkg.in
    └── vendor.json
```

Go API 项目中，一般都会包括这些功能项：Makefile 文件、配置文件目录、RESTful API 服务器的 handler 目录、model 目录、工具类目录、vendor 目录，以及实际处理业务逻辑函数所存放的 service 目录。这些都在上述的代码结构中有列出，新加功能时将代码放入对应功能的目录/文件中，可以使整个项目代码结构更加清晰，非常有利于后期的查找和维护。

小结

本小节通过介绍 API 服务器启动流程和 HTTP 请求处理流程，来让读者对 API 服务器中的关键流程有个宏观的了解，更好地理解 API 服务器是如何工作的。API 服务器源码结构也非常重要，一个好的源码结构通常能让逻辑更加清晰，编写更加顺畅，后期维护更加容易，本小册介绍了笔者倾向的源码组织结构，供读者参考。