



下载APP



## 15 | 配置和环境：配置服务中的设计思路(上)

2021-10-20 叶剑峰

《手把手带你写一个Web框架》

课程介绍 &gt;



讲述：叶剑峰

时长 14:03 大小 12.87M



你好，我是轩脉刃。

经过前面几节课对框架的改造，我们能很方便在 hade 框架中创建一个定时任务，或者一个 Web 服务了。但是随着开始编写业务，你会发现，业务中有大量的配置项，比如数据库的用户名密码、缓存 Redis 的 IP 和端口、第三方调用的地址等。如何通过统一的方法快速获取到这些配置项，就是今天要讨论的内容。

当你看到获取配置项这个需求，第一个反应是不是要创建一个读取配置文件的服务。但是，一个程序获取配置项只有读取配置文件这个方法么？其实不是的，获取配置项的方法有很多，**读取本地配置文件、读取远端配置服务、获取环境变量**，都是获取配置项的方法。



获取远端配置服务是通过一个远程调用来获取配置信息的方法，它依赖于远端的服务提供。而读取本地配置文件和获取环境变量，是我们通过进程本身获取配置项最常用的方法，这节课就为框架增加这两种获取配置项的能力。

## 环境变量获取配置思路分析

在现在服务越来越容器化的时代，环境变量越来越重要。因为一个服务一旦被封装为 Docker 镜像，镜像就会被部署在不同的环境中。如何区分不同的环境呢？在容器内部已经把程序、配置文件都进行了打包，唯一能在不同环境变化的就是环境变量了。

所以顾名思义，环境变量也就是为不同环境准备的，不同环境有不同的设置。

为一个程序设置环境变量的方式是多种多样的。如果是容器化的进程，可以在创建镜像的时候设置，也可以在容器启动的时候设置，在 Linux 系统中，我们也可以通过在启动进程的时候，通过前面加上“KEY=VALUE”的方式，为单个进程设置环境变量，比如：

```
1 FOO_ENV=bar ./hade foo
```

[复制代码](#)

**就为当前这个进程设置了一个 key 为 FOO\_ENV，值为 bar 的环境变量。**要记住这种方式，我们后续测试的时候，会用这种方式测试环境变量是否设置成功。

但是不管是哪种设置环境变量的方式，在 Golang 中，都能通过 os 标准库的 `os.Environ()` 获取。Environ 方法，会将进程中设置的所有环境变量，都以字符串数组形式返回，每个字符串为“KEY=VALUE”的环境变量设置。

读取环境变量获取配置的大思路有了，但是考虑到配置的设置还有一个问题。

环境变量可能会有很多。但是我们每次部署一个环境的时候，设置的环境变量可能就只有一两个，那其他的环境变量就需要有一个“默认值”。这个默认值我们一般使用一个以 dot 点号开头的文件 `.env` 来进行设置。

**其实使用 .env 文件来设置默认环境变量，在运行的时候再使用真实的环境变量替换部分默认值**，这种做法，在业界已经是一种非常普遍的加载环境变量的方式了。比如，在 Docker

中有个 env-file 配置项，会在启动的时候读取默认环境变量文件；又比如 Vue 中，Webpack 打包各种不同环境的时候，会根据根目录下.env 文件读取环境配置。

而我们的 hade 框架，也可以借鉴这个思路设计成相同的方式：在 baseFolder 目录下存放一个.env 文件保存默认值，这个.env 文件中的每一行，为一个环境变量 Key=Value 的形式，等运行的时候，进程运行的环境变量设置会覆盖.env 文件的设置。

有了上面对环境变量的设置、读取环境变量获取配置的分析之后，我们就可以开始着手设计了。分成两步来写：

环境变量服务的接口及具体实现

读取配置文件服务的接口及具体实现

## 环境变量服务的接口和实现

首先按照一切皆服务的思想，通过环境变量获取配置项，这本身可以设计成一个服务，存放在服务容器中。所以我们先设计配置项服务 Env 的接口。

环境变量服务设计了四个方法：


AppEnv，获取 APP\_ENV 这个环境变量，这个环境变量代表当前应用所在的环境；

IsExist，判断某个环境变量是否存在；

Get，获取某个环境变量，如果没有设置，则返回空字符串；

All，获取所有环境变量。

在 framework/contract/env.go 文件中：

 复制代码

```
1 package contract
2
3 const (
4     // EnvProduction 代表生产环境
5     EnvProduction = "production"
6     // EnvTesting 代表测试环境
7     EnvTesting = "testing"
8     // EnvDevelopment 代表开发环境
```

```
9     EnvDevelopment = "development"
10    // EnvKey 是环境变量服务字符串凭证
11    EnvKey = "hade:env"
12 )
13 // Env 定义环境变量服务
14 type Env interface {
15     // AppEnv 获取当前的环境，建议分为 development/testing/production
16     AppEnv() string
17     // IsExist 判断一个环境变量是否有被设置
18     IsExist(string) bool
19     // Get 获取某个环境变量，如果没有设置，返回""
20     Get(string) string
21     // All 获取所有环境变量，.env 和运行环境变量融合后结果
22     All() map[string]string
23 }
```

这里有两点你可能觉得有点奇怪。

为什么有 Get 方法为什么没有 Set 方法？

这点我是这么考虑的，环境变量本质代表的是当前程序运行的环境，是一个程序运行时就固定的，它不应该允许程序运行中进行设置，这是一个不安全的行为。所以这里故意没有设计 Set 方法，让环境变量在运行时被修改。

AppEnv 是什么，为什么要单独设置一个 APP\_ENV 这个环境变量？


这是参考 Laravel 和 Vue 这两个项目，它们都会为当前的应用，设置一个固定的环境变量 APP\_ENV，这样在代码中，可以根据这个环境变量加载不同的配置文件、运行不同的业务逻辑。所以我单独为 APP\_ENV 设置了环境变量，同时为这个 APP\_ENV 预设了三个模式：开发模式 development、测试模式 testing、生产模式 production，默认为开发模式。

下面就来说接口的具体实现，在框架文件 framework/env/service.go 中，大致就是按照先读取本地默认.env 文件，再读取运行环境变量。

我们使用一个 map[string]string 来保存最终的环境变量值，设置这个 map 初始有一个 APP\_ENV 的 key，并将它设置为开发环境。

```
2  hadeEnv := &HadeEnv{
3      folder: folder,
4      // 实例化环境变量, APP_ENV 默认设置为开发环境
5      maps: map[string]string{"APP_ENV": contract.EnvDevelopment},
6  }
```

接下来先读取.env 文件中保存的默认环境变量，这里读取就使用 Golang 里面的文件读取方式：先用 os.Open 打开文件，使用 bufio.NewReader 创建一个读取器，然后使用 ReadLine 逐行读取，读取之后将结果保存到 map 中。

 复制代码

```
1  // 打开文件.env
2  fi, err := os.Open(file)
3  if err == nil {
4      defer fi.Close()
5      // 读取文件
6      br := bufio.NewReader(fi)
7      for {
8          // 按照行进行读取
9          line, _, c := br.ReadLine()
10         if c == io.EOF {
11             break
12         }
13         // 按照等号解析
14         s := bytes.SplitN(line, []byte{'='}, 2)
15         // 如果不符合规范, 则过滤
16         if len(s) < 2 {
17             continue
18         }
19         // 保存 map
20         key := string(s[0])
21         val := string(s[1])
22         hadeEnv.maps[key] = val
23     }
24 }
```

最后再使用开头说的 os.Environ 来读取程序的所有环境变量，并且直接覆盖 map 变量。

 复制代码

```
1  // 获取当前程序的环境变量, 并且覆盖.env 文件下的变量
2  for _, e := range os.Environ() {
3      pair := strings.SplitN(e, "=", 2)
4      if len(pair) < 2 {
5          continue
6      }
```

```
7     hadeEnv.maps[pair[0]] = pair[1]
8 }
```

而环境变量服务最终提供的 Get 接口，也就非常简单了，直接从 map 中获取某个 key 的环境变量即可。并且由于在运行过程中没有 Set 等修改环境变量的方法，所以也不需要加锁。

[复制代码](#)

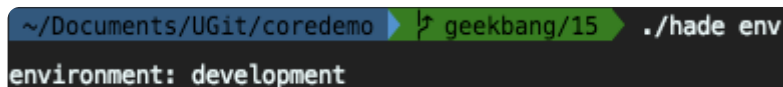
```
1 // Get 获取某个环境变量，如果没有设置，返回""
2 func (en *HadeEnv) Get(key string) string {
3     if val, ok := en.maps[key]; ok {
4         return val
5     }
6     return ""
7 }
```

到这里接口和实现就基本完成了，简单写段代码测试一下。

我们可以创建一个一级命令 `./hade env` 来获取当前 app 的环境，并且验证刚才写的环境变量服务，在这个命令中，我们直接将 AppEnv 打印到控制台。在文件 `framework/command/contract.go` 中：

[复制代码](#)

```
1 // envCommand 获取当前的 App 环境
2 var envCommand = &cobra.Command{
3     Use:     "env",
4     Short:   "获取当前的 App 环境",
5     Run: func(c *cobra.Command, args []string) {
6         // 获取 env 环境
7         container := c.GetContainer()
8         envService := container.MustMake(contract.EnvKey).(contract.Env)
9         // 打印环境
10        fmt.Println("environment:", envService.AppEnv())
11    },
12 }
```



```
~/Documents/UGit/coredemo ➤ geekbang/15 ➤ ./hade env
environment: development
```



再尝试在进程启动的时候注入 APP\_ENV 环境变量为 testing。控制台会将 APP\_ENV 替换为 testing，而不是默认的 development 了。

```
~/Documents/UGit/coredemo ➤ geekbang/15 ● APP_ENV=testing ./hade env  
environment: testing
```

## 读取配置服务的接口

下面我们就来思考配置文件的设计。

基于一切皆服务的思想，我们也可以将配置文件的读取作为服务容器中的一个服务，照旧先定义它的接口。配置文件服务的接口是我们经常使用的，需要充分考虑其易用性。

在项目中会使用一个配置文件还是多个配置文件？这个是需要先思考清楚的。

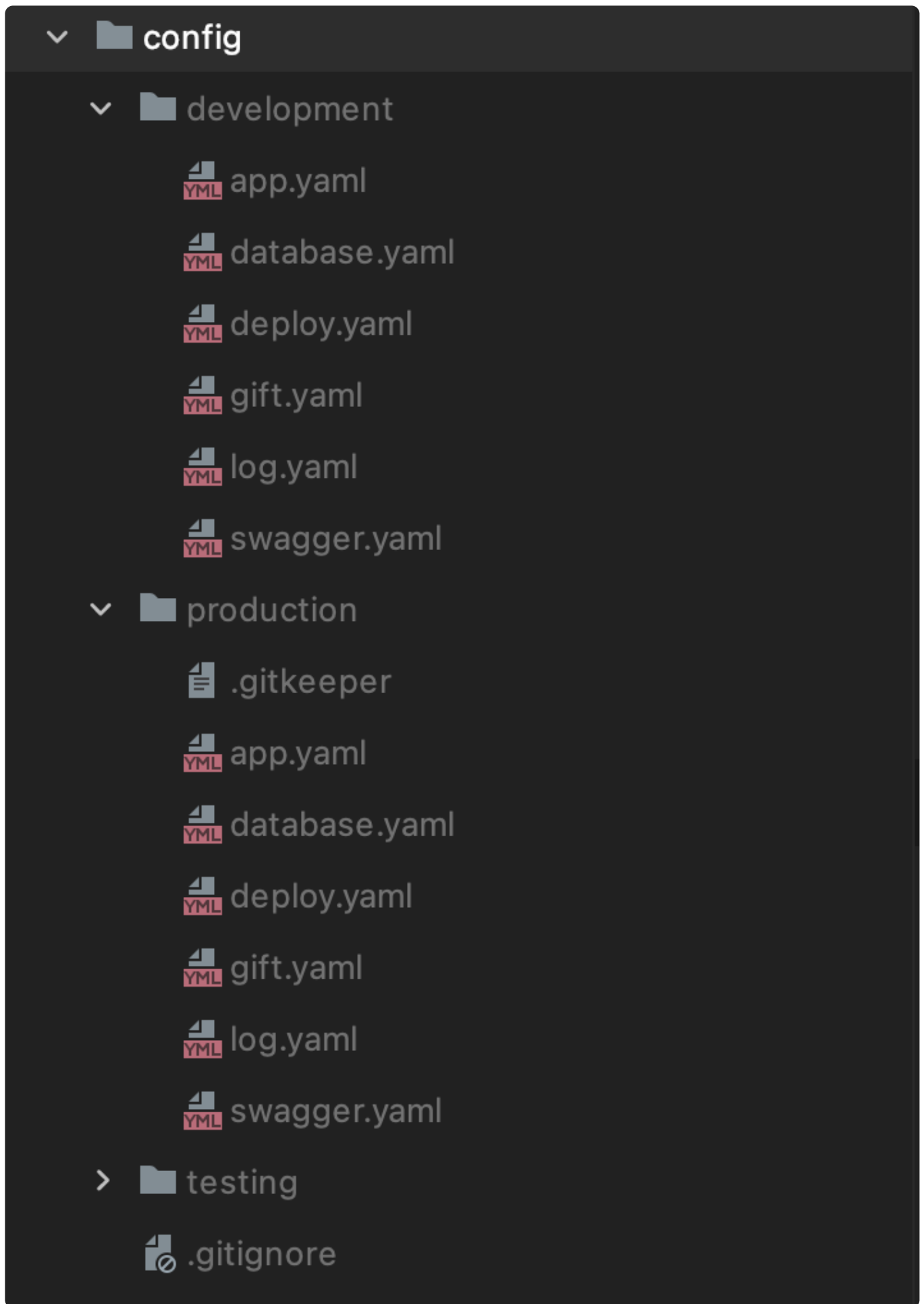
比如我们可以强制要求所有配置放在一个文件中，这样读取会非常方便。但是在实际工作中，这种实践的感受是非常糟糕的，不仅仅这个配置文件会非常冗长，而且更严重的是，**导致配置文件失去了语义化，因为每个配置文件是为了某个模块功能而设计的**，比如我们看到 log.yaml，就能知道这个配置文件是存放日志相关的配置，而 database.yaml 是存放数据库相关的配置。

所以配置文件一定是多个存在比较好。

明确了多配置文件之后，要思考下这个配置文件的存放目录。在 [第 12 节课](#) 中，我们已经确定了整体 App 的目录结构，其中存放配置文件的目录是，应用结构服务 appService 的 ConfigFolder 这个目录。但是再结合环境配置问题考虑。

之前在环境变量中设置了 APP\_ENV，表示这个应用在不同环境会有不同配置，所以如果将每个环境的配置文件存放在 ConfigFolder 下的不同文件夹下，这样配置文件结构是不是显得更为清晰呢？

我们读取文件的时候，**直接根据 APP\_ENV 的环境变量，去不同的配置文件的文件夹下获取对应的配置文件即可**，很方便。



多个配置文件+不同配置文件夹



接下来就是要从某个配置文件读取某个目标配置项了。我们可以设计成先读取文件、再读取某个字段的方式，但这样有点复杂，更简单的是，**使用点号分割的路径读取方式**来增加读取的易用性。

比如我们要读取 app.yaml 配置文件下的 data 配置项，可以直接通过 app.data 来获取这个配置项。如果 data 配置项中还有下级属性，比如 name，那可以通过 app.data.name 来获取这个 name 属性。这种根据点来区分配置路径层级的方式，能很大加速我们的读取效率。

最后，我们考虑配置文件服务的具体方法。

你应该还记得在 [第五节课](#) 的时候，我们为 Request 封装了一系列查询的方法，同样的，在这里也可以为配置文件服务设计 Get 系列的方法，根据带有点的查询路径，来获取不同类型返回值的配置项。另外再增加一个判断路径是否存在的 IsExist 方法、一个能将某个配置项解析到一个对象的 Load 方法。

我们在框架文件 framework/contract/config.go 中：

[复制代码](#)

```
1 // Config 定义了配置文件服务，读取配置文件，支持点分割的路径读取
2 // 例如：.Get("app.name") 表示从 app 文件中读取 name 属性
3 // 建议使用 yaml 属性，https://yaml.org/spec/1.2/spec.html
4 type Config interface {
5     // IsExist 检查一个属性是否存在
6     IsExist(key string) bool
7     // Get 获取一个属性值
8     Get(key string) interface{}
9     // GetBool 获取一个 bool 属性
10    GetBool(key string) bool
11    // GetInt 获取一个 int 属性
12    GetInt(key string) int
13    // GetFloat64 获取一个 float64 属性
14    GetFloat64(key string) float64
15    // GetTime 获取一个 time 属性
16    GetTime(key string) time.Time
17    // GetString 获取一个 string 属性
18    GetString(key string) string
19    // GetIntSlice 获取一个 int 数组属性
20    GetIntSlice(key string) []int
21    // GetStringSlice 获取一个 string 数组
22    GetStringSlice(key string) []string
23    // GetStringMap 获取一个 string 为 key，interface 为 val 的 map
```

```
24 GetStringMap(key string) map[string]interface{}
25 // GetStringMapString 获取一个 string 为 key, string 为 val 的 map
26 GetStringMapString(key string) map[string]string
27 // GetStringMapStringSlice 获取一个 string 为 key, 数组 string 为 val 的 map
28 GetStringMapStringSlice(key string) map[string][]string
29 // Load 加载配置到某个对象
30 Load(key string, val interface{}) error
31 }
```

到这里，配置文件服务的接口就设计完成了，下一节课我们接着讨论整个服务的具体实现。

## 小结

今天我们围绕获取配置这一个功能点，设计了环境变量服务和配置文件服务。

环境变量服务按照先读取本地默认.env 文件，再读取运行环境变量的方式来实现，并且为其设置了最关键的环境变量 APP\_ENV 来表示这个应用当前运行的环境。后续我们根据这个 APP\_ENV 来获取具体环境的本地配置文件。

有的人可能会觉得要获取一个变量，直接使用配置文件就行啊，为什么要绕这么一圈？但是，环境变量是一个应用运行环境的一些参数，在现在的容器化流行的架构设计中，一般都会选择使用环境变量来区别不同的环境和配置。所以我们为框架提供获取环境变量的方式，在实际架构，特别是微服务相关的架构中，是非常有用的。

## 思考题

关于使用环境变量作为配置项，其实有一个比较出名的服务设计经验建议了 [🔗The Twelve-Factor App](#)，这 12 条的实践标准，出自一个做 Paas 服务的 Herku 公司，是他们工程师的经验之谈。在配置文件这块，它建议所有环境的配置统一成一份，只通过环境变量进行区分。

实际上今天我们并没有完全按照实践标准来做，还是提供了将不同环境的配置分目录的方式。你不妨阅读这 12 条的实践标准，说说你的观点。

欢迎在留言区分享你的思考。感谢你的收听，如果觉得有收获，也欢迎把今天的内容分享给你身边的朋友，邀他一起学习。我们下节课见~

分享给需要的人，Ta订阅后你可得 **20 元现金奖励**

生成海报并分享

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 14 | 定时任务：如何让框架支持分布式定时脚本？

下一篇 16 | 配置和环境：配置服务中的设计思路(下)

1024 活动特惠

VIP 年卡直降 ¥2000

新课上线即解锁，享 365 天畅看全场

超值拿下 ¥999



精选留言 (1)

写留言



qinsi  
2021-10-21

12 factor的建议是配置存储在环境变量中，甚至都不建议使用配置文件。这里似乎是需要一个配置中心：启动容器的时候从配置中心拉取配置注入到容器的环境变量中，而容器中的应用只需要读取环境变量就好了。

展开

