**基于http2的微服务快速部署实践**

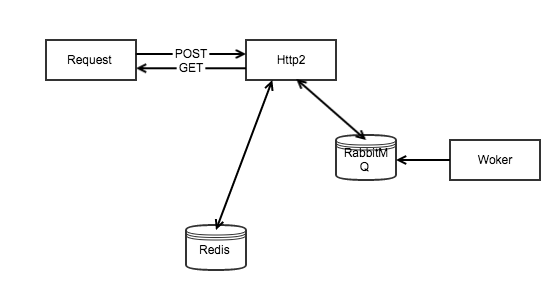
源码：<https://github.com/leitu/mse_advancedweb>

**绪论**

微服务是新兴的架构方法，微服务应用由自动化的、可独立部署、扩展和管理的服务组成，提供了所必需的可扩展、弹性、跨平台等基础。REST这样的网络服务协议已成为定义API的首选。目前HTTP2也被主流浏览器支持，如何在现代软件开发模式下快速部署RESTful微服务成为一个新的课题。

**1 需求分析**

* 1. 系统流程图



**2 工具介绍**

**2.1**[**http2**](https://en.wikipedia.org/wiki/HTTP%2F2)

2.1.1 HTTP2介绍

HTTP/2（超文本传输协议第2版，最初命名为HTTP 2.0），是HTTP协议的的第二个主要版本，使用于万维网。HTTP/2是HTTP协议自1999年HTTP 1.1发布后的首个更新，主要基于SPDY协议。它由互联网工程任务组（IETF）的Hypertext Transfer Protocol Bis（httpbis）工作小组进行开发。该组织于2014年12月将HTTP/2标准提议递交至IESG进行讨论，于2015年2月17日被批准。 HTTP2标准于2015年5月以RFC 7540正式发表。

* + 1. HTTP2的特性

HTTP2包含了大部分HTTP1.1的特性，又提供如下:

* HTTP2 采用二进制格式传输数据，已Frame作为二进制基础
* HTTP2 对消息头采用 HPACK 进行压缩传输，能够节省消息头占用的网络的流量。
* 多路复用，所有的请求都是通过一个 TCP 连接并发完成, 还支持优先级和流量控制。
* Server Push：这个是HTTP2最重要的特性，在服务端能够更快的把资源推送给客户端。例如服务端可以主动把 JS 和 CSS 文件推送给客户端。

可以根据[demo](https://http2.akamai.com/demo)看出HTTP2在性能加载上有绝对的优势。

**2.2**[**Microservices**](https://en.wikipedia.org/wiki/Microservices)

2.2.1 Microservices介绍

Microservices， 微服务是一种以业务功能为主的服务设计概念，每一个服务都具有自主运行的业务功能，对外开放不受语言限制的 API (最常用的是 HTTP)，应用程序则是由一个或多个微服务组成。

* + 1. Microservices的特性

Microservices秉承了Unix理论“Do one thing and do it well", 还包含如下:

* 每个服务都容易被取代。
* 服务是以能力来组织的。
* 服务可由不同的编程语言、数据库、硬件与软件环境实作，只在乎是否适配。
* 架构是对称而非分层。

**2.3**[**Docker**](https://en.wikipedia.org/wiki/Docker_(software))

2.3.1 Docker介绍

Docker是一个开放源代码软件专案，让应用程序布署在软件容器下的工作可以自动化进行。

* + 1. Docker的特性
* 借此在Linux操作系统上，提供一个额外的软件抽象层，以及操作系统层虚拟化的自动管理机制。
* Docker利用Linux核心中的资源分离机制，例如cgroups，以及Linux核心命名空间（name space），来建立独立的软件容器（containers）。这可以在单一Linux实体下运作，避免启动一个虚拟机器造成的额外负担。Linux核心对命名空间的支援完全隔离了工作环境中应用程序的视野，包括行程树、网络、用户ID与挂载档案系统，而核心的cgroup提供资源隔离，包括CPU、内存、block I/O与网络。

2.3.3 Docker Compose

Docker Compse的前身是Fig, 用compose file配置应用服务，快速部署多个应用程序。

**2.4**[**Go**](https://en.wikipedia.org/wiki/Go_(programming_language))

2.4.1 Go介绍

Go,又称golang，是Google开发的一种静态强类型、编译型，并发型，并具有垃圾回收功能的编程语言。

**2.5**[**Redis**](https://en.wikipedia.org/wiki/Redis)

2.5.1 Redis介绍

Redis是一个开源、支持网络、基于内存、键值对存储数据库，将全部的数据存储在内存中,支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。

**2.6**[**RabbitMQ**](https://en.wikipedia.org/wiki/RabbitMQ)

2.6.1 RabbitMQ介绍

RabbitMQ是实现了高级消息队列协议（AMQP）的开源消息代理软件，具有可伸缩性：群集服务 消息持久化：从内存持久化消息到硬盘，再从硬盘加载到内存。

2.7 JSON

**三、详细设计**

通过POST 请求，存入信息到RabbitMQ中，后端的worker 监听RabbitMQ的queue，当发现有message后，取出message中的特定项， 通过后端worker执行，并返回处理结果，redist 作为key-value数据库，记录每个POST请求。 运用docker compose 对前端，redis，RabbitMQ进行快速部署，节省开发时间开发精力。

3.1 前端RESTful

3.1.1

go语言自带的net库,可以通过几行代码就启动一个http服务，在go语言升级大版本号至1.6以后，自带的net库支持http2，通过几行代码就能够实现一个web服务器的启动。

func main() {

var srv http.Server

http2.VerboseLogs = true

srv.Addr = ":8443"

// This enables http2 support

http2.ConfigureServer(&srv, nil)

http.HandleFunc("/", func(w http.ResponseWriter, r \*http.Request) {

fmt.Fprintf(w, "你在使用HTTP2协议访问 %q\n", html.EscapeString(r.URL.Path))

ShowRequestInfoHandler(w, r)

})

}



3.1.2

基于http中定义了8种方法，在RESTful中最常用的就是GET,POST,PUT,DELETE.

在go代码中，限定/api/post 为指定接受POST方法。

代码如下。

http.HandleFunc("/api/post", func(w http.ResponseWriter, r \*http.Request) {

//get "POST" request

if r.Method == "POST" {

decoder := json.NewDecoder(r.Body)

var t jsonData

err := decoder.Decode(&t)

if err != nil {

panic(err)

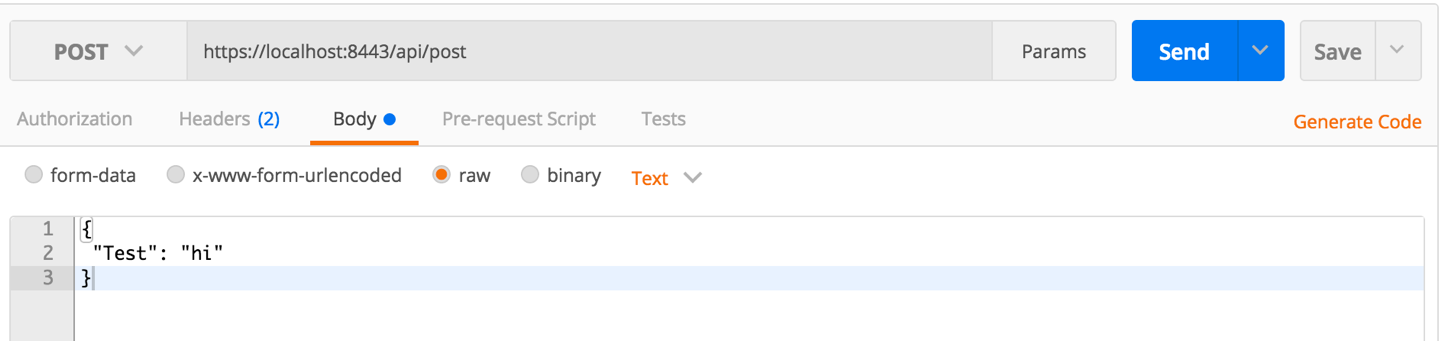
}

log.Println(t.Test)

}

})

当前端使用curl 或者 postman 将测试代码已JSON格式，POST到<https://localhost:8443/api/post>,



服务器后台的log 就会有如下日志记录

2016/05/16 15:17:06 http2: server read frame DATA flags=END\_STREAM stream=1 len=17 data="{\n \"Test\": \"hi\"\n}"

2016/05/16 15:17:06 hi

2016/05/16 15:17:06 http2: server encoding header ":status" = "200"

3.2 后端消息存储

当数据从前端的request body 进行解析后，将所需要部分传入RabbitMQ,

3.3 快速部署

**结论**

在本次实践过程中，并没有增加监控功能，微服务的架构是非常依赖与监控的，因为在生产环境中出现的失败比较不容易被发现，同样的各种微服务之间的联系节点也更加复杂，如果一个节点消失对整个环境将会造成不可预估的后果。