

hydra微服务开发入门

hydra为api接口，消息服务，定时任务，RPC服务,websocket等提供了统一的配置管理，服务注册与发现，和一致的输入与输出处理。可以很方便将hydra服务集成到自己应用中，按约定的服务实现方法，即可提供完整的服务。

开发准备：

- 1.安装zookeeper

hydra的配置信息是通过zookeeper集中管理的。同一集群内所有服务共享同一配置。配置变化后自动下发到每个server,server自动进行热更新。zookeeper还作为注册中心，用于服务注册与发现。为便于本地测试，hydra支持将本地文件系统作为注册中心，由于本地配置维护成本较高(修改配置需每台机器单独更新)，所以不推荐在生产环境中使用。

- 2. 安装hydra

```
go get github.com/micro-plat/hydra
```

1. 构建API接口服务

hydra提供的是完整的服务解决方案。包括启动，配置初始化，本地服务安装、启动，优雅关闭，日志打印等，所以需要将hydra提供的hydra.MicroApp直接集成到系统，调用Start()启动服务器。对外提供的服务则按照hydra的规范实现，并注册。

新建文件 main.go ,结构如下:

```
|---apiserver01  
|----- main.go
```

1. main.go 文件中添加如下内容:

```
package main  
  
import (  
    "github.com/micro-plat/hydra/hydra"  
)  
  
func main() {  
    app := hydra.NewApp()  
    app.Start()  
}
```

2. 使用go命令安装服务

```
go install github.com/micro-plat/hydra/quickstart/demo/apiserver01
```

3. 查看服务帮助信息

```
~/work/bin$ apiserver01 -h
```

NAME:

apiserver01 - 基于hydra的微服务应用

USAGE:

apiserver01 [global options] command [command options] [arguments...]

VERSION:

2.0.0

COMMANDS:

run	运行服务。前台运行，日志直接输出到客户端，输入ctl+c命令时退出服务
start	启动服务。后台运行服务，日志存入本地文件或日志中心。异常退出或服务器重启会自动启动
stop	停止服务。通过start启动的服务，使用此命令可停止服务
install	安装配置信息和本地服务。初次运行服务前使用，同一集群只需安装一次。配置信息变更后需再次调用
registry	安装配置信息。只安装配置信息，同一集群只需安装一次。配置信息变更后需再次调用
service	安装本地服务。初次运行服务前调用
remove	删除服务。应用启动参数发生变化后，需调用remove删除本地服务后再重新安装
status	查询服务状态
conf	查看配置信息。查看当前服务在配置中心的配置信息
help, h	Shows a list of commands or help for one command

GLOBAL OPTIONS:

--help, -h 查看帮助信息

hydra 已提供了配置安装、服务运行、后台启动、停止、删除、状态查询、配置中心参数查看 等功能

初次运行需调用 install 或 registry 进行配置安装

查看 install 命令参数：

```
~/work/bin$ apiserver01 install -h
```

NAME:

apiserver01 install - 安装配置信息和本地服务。初次运行服务前使用，同一集群只需安装一次。配置信息

USAGE:

apiserver01 install [command options] [arguments...]

OPTIONS:

--registry value, -r value	*注册中心地址, 必须项。目前支持zookeeper(zk)和本地文件系统(fs)。服务注册与发现等数据, 格式:proto://host。proto的取值有zk,fs; 如zookeeper则为ip地址(加端口号),多个ip用逗号分隔,如:zk://192.168.0.109:2181; 文件系统为本地文件路径,可以是相对路径或绝对路径,如:fs://../; 此定,也可从环境变量中获取,环境变量名为:[\$hydra_registry]
--name value, -n value	*服务全名,指服务在注册中心的完整名称,该名称是以/分隔的多级目录结构,格式: /平台名称/系统名称/服务器类型/集群名称; 平台名称为字母或下划线或数字,服务器类型则目前支持的几种服务器类型有:api,web,cli; 环境变量名为:[\$hydra_name]
--trace value, -t value	-性能跟踪,可选项。用于生成golang的pprof的性能分析数据,支持的模式有:cpu,mem,block,mutex; 该参数可从环境变量中获取,环境变量名为:[\$hydra_trace]
--rlog, -l	-启用远程日志,可选项。默认不启用。指定此参数后启动时自动从注册中心压缩后定时发送到远程服务器。该参数可从环境变量中获取,环境变量名为:[\$hydra_rlog]
--rs, -R	-启用远程服务,可选项。默认不启用。启用后本地将自动启动一个http服务,通过http://host/update/:version远程更新系统,执行该参数可从环境变量中获取,环境变量名为:[\$hydra-rs]

4. 配置信息安装:

```
~/work/bin$ sudo apiserver01 install -r zk://192.168.0.109 -n /mall/apiserver/api/test
```

-> 创建注册中心配置数据?如存在则不安装(1),如果存在则覆盖(2),删除所有配置并重建(3),退出(n|r)
创建配置: /mall/apiserver/api/test/conf

Install apiserver01: [OK]

参数说明:

-r 参数为注册中心地址。格式: proto://host ,多个host用逗号分隔(proto://host,host)。proto可选项有 zk 和 fs 。 host 为协议对应的主机名或地址。使用 fs 本地文件系统时 host 为本地文件路径如 fs://../

-n参数,由于代码中未指定 平台名称 , 系统名称 , 服务类型 , 集群名称 。故运行时需通过 -n 指定,格式为: /平台名称/系统名称/服务器类型/集群名称 。可在初始化MicroApp时指定以上参数(参见示例 apiserver02),重新执行 install -h 命令,查看参数变化。

5. 运行服务:

```
~/work/bin$ apiserver01 run -r zk://192.168.0.109 -n /mall/apiserver/api/test
[2019/06/27 10:52:36.526910][i][7c27d59a7]Connected to 192.168.0.109:2181
[2019/06/27 10:52:36.533657][i][7c27d59a7]Authenticated: id=246395503264334038, timeout=
[2019/06/27 10:52:36.533661][i][7c27d59a7]Re-submitting `0` credentials after reconnect
[2019/06/27 10:52:36.574542][i][7c27d59a7]初始化 /mall/apiserver/api/test
[2019/06/27 10:52:36.582761][i][096b62f40]开始启动[API]服务...
[2019/06/27 10:52:36.583454][d][096b62f40][未启用 header设置]
[2019/06/27 10:52:36.583437][i][096b62f40][启用 静态文件]
[2019/06/27 10:52:36.583482][d][096b62f40][未启用 metric设置]
[2019/06/27 10:52:36.583473][d][096b62f40][未启用 ajax请求限制设置]
[2019/06/27 10:52:36.583491][d][096b62f40][未启用 host设置]
[2019/06/27 10:52:36.583461][d][096b62f40][未启用 熔断设置]
[2019/06/27 10:52:36.583468][d][096b62f40][未启用 jwt设置]
[2019/06/27 10:52:37.100601][i][096b62f40]服务启动成功(API,http://192.168.4.121:8090,0)
```

由于 run 参数是前端直接运行服务，故同样需指定与 install 相同的参数。

根据日志信息系统启动过程如下：

1. 连接到zookeeper服务器 192.168.0.109:2181
2. 读取zookeeper的节点 /mall/apiserver/api/test 中的配置信息,准备初始化服务器
3. 根据配置的服务器类型为 api ，开始启动 api 服务器
4. api默认提供了 静态文件服务 ,其它 header , metric , jwt , 熔断 等参数未配置故未启动
5. 系统启动成功，地址是 http://192.168.4.121:8090 ，对外提供的服务数为 0 （未注册任何服务）

至此服务已启动成功了，由于未身MicroApp中注册任何服务，故当前服务器还不能对外提供有用的服务。

检查服务器是否启动，可以通过 telnet 命令查看：

```
~/work/bin$ telnet 192.168.4.121 8090
Trying 192.168.4.121...
Connected to 192.168.4.121.
Escape character is '^['.
```

6. 通过start命令启动

通过第4步安装显示信息 Install apiserver01：

[OK] 说

明服务已安装到本地。

ubuntu查看已安装服务的配置文件

```
~/work/bin$ cat /etc/systemd/system/apiserver01.service
[Unit]
Description=apiserver01
Requires=network.target
After=network.target

[Service]
EnvironmentFile=~/.bashrc
PIDFile=/var/run/apiserver01.pid
ExecStartPre=rm -f /var/run/apiserver01.pid
WorkingDirectory=/home/colin/work/bin
ExecStart=/home/colin/work/bin/apiserver01 run -r zk://192.168.0.109 -n /mall/apiserver/
Restart=on-failure
RestartSec=50s

[Install]
WantedBy=multi-user.target
```

可以看出在ubuntu下是安装为 systemd 服务，实际执行的命令是 run

centos, mac, windows请自行尝试

使用start启动服务

```
:~/work/bin$ sudo apiserver01 start
Starting apiserver01: [ OK ]
```

服务已启动成功，终端不再有日志输出。查看日志需进入 ../logs/ 目录查看

```
~/work/bin$ tail -f -n 100 ../logs/20190627.log
```

```
-----begin-----
```

```
[2019/06/27 11:31:45.320595][i][7c2ab0070] Connected to 192.168.0.109:2181
[2019/06/27 11:31:45.324094][i][7c2ab0070] Re-submitting `` credentials after reconnect
[2019/06/27 11:31:45.324081][i][7c2ab0070] Authenticated: id=246395503264334040, timeout
[2019/06/27 11:31:45.369671][i][7c2ab0070] 初始化 /mall/apiserver/api/test
[2019/06/27 11:31:45.377408][i][50a18ec59] 开始启动[API]服务...
[2019/06/27 11:31:45.378096][i][50a18ec59] [启用 静态文件]
[2019/06/27 11:31:45.378117][d][50a18ec59] [未启用 header设置]
[2019/06/27 11:31:45.378132][d][50a18ec59] [未启用 jwt设置]
[2019/06/27 11:31:45.378124][d][50a18ec59] [未启用 熔断设置]
[2019/06/27 11:31:45.378137][d][50a18ec59] [未启用 ajax请求限制设置]
[2019/06/27 11:31:45.378147][d][50a18ec59] [未启用 metric设置]
[2019/06/27 11:31:45.378152][d][50a18ec59] [未启用 host设置]
[2019/06/27 11:31:45.888580][i][50a18ec59] 服务启动成功(API,http://192.168.4.121:8090,0)
```

查看服务状态

```
~/work/bin$ sudo apiserver01 status
Service (pid 32677) is running...
```

停止服务

```
~/work/bin$ sudo apiserver01 stop
Stopping apiserver01: [ OK ]
```

7. 注册服务

通过以上示例清楚服务器的创建，安装，启动，停止等功能。但还不能对外提供有用的服务。
修改代码如下 (可参考示例 apiserver03):

```

package main

import (
    "github.com/micro-plat/hydra/context"
    "github.com/micro-plat/hydra/hydra"
)

func main() {
    app := hydra.NewApp()
    app.API("/hello", hello)
    app.Start()
}

func hello(ctx *context.Context) (r interface{}) {
    return "hello world"
}

```

重新编译安装:

```
go install github.com/micro-plat/hydra/quickstart/demo/apiserver01
```

注册中心配置已安装, 则直接调用 run 命令或 start 命令启动

```

~/work/bin$ apiserver01 run -r zk://192.168.0.109 -n /mall/apiserver/api/test
[2019/06/27 12:01:58.79486][i][aee04e9cf]Connected to 192.168.0.109:2181
[2019/06/27 12:01:58.83978][i][aee04e9cf]Re-submitting `0` credentials after reconnect
[2019/06/27 12:01:58.83970][i][aee04e9cf]Authenticated: id=246395503264334042, timeout=4
[2019/06/27 12:01:58.128626][i][aee04e9cf]初始化 /mall/apiserver/api/test
[2019/06/27 12:01:58.136903][i][48648a233]开始启动[API]服务...
[2019/06/27 12:01:58.137688][d][48648a233][未启用 ajax请求限制设置]
[2019/06/27 12:01:58.137704][d][48648a233][未启用 host设置]
[2019/06/27 12:01:58.137682][d][48648a233][未启用 jwt设置]
[2019/06/27 12:01:58.137666][d][48648a233][未启用 header设置]
[2019/06/27 12:01:58.137697][d][48648a233][未启用 metric设置]
[2019/06/27 12:01:58.137648][i][48648a233][启用 静态文件]
[2019/06/27 12:01:58.137675][d][48648a233][未启用 熔断设置]
[2019/06/27 12:01:58.652018][i][48648a233]服务启动成功(API,http://192.168.4.121:8090,1)

```

服务数量已变为 1 , 测试服务是否可用:

```

:~/work/bin$ curl http://192.168.4.121:8090/hello
{"data":"hello world"}

```

服务可以正常访问并返回结果。但是结果被包装为 json 格式，因为服务未指定输入格式，默认是 json ,需返回 xml , plain 等请参考其它章节

查看服务器执行日志:

```
[2019/06/27 12:24:02.566860][i][cdde16581]api.request GET /hello from 192.168.4.121  
[2019/06/27 12:24:02.567139][i][cdde16581]api.response GET /hello 200 295.959µs
```

api.request 接口请求。请求方式，请求地址，及来源IP

api.response 接口响应。请求方式，请求地址，状态码，请求处理的总时长

由于接口中未处理任何逻辑，处理时长只有0.295毫秒