检查选项

inter <delay>：设定健康状态检查的时间间隔，单位为毫秒，默认为 2000；也可以使用 fastinter 和 downinter 来根据服务器端状态优化此时间延迟；
fastinter <delay>：过渡上架、过渡下架的检查时间间隔
downinter <delay>：当后端服务器下架后，检查的时间间隔
rise <count>：设定健康状态检查中，成功检查多少次 将离线转为在线
fall <count>：确认 server 从正常状态转换为不可用状态需要检查的次数

`server  web1 192.168.80.121:80 check fastinter 10s downinter 60s rise 3 fall 3`

http 相关

keep-alive
option http-keepalive //默认选项，对于 client 带有 conntion: keep-alive 的选项，保持连接
option http-server-close //请求完成后，haproxy 与客户端保持连接，haproxy 与服务端关闭连接
option forceclose //完成整个响应以后，关闭两端连接
option http-tunnel //只有第一个请求被解析，后面的内容只是透明转发, 当在头部中发现 upgrade 字段时，也会转变成这种模式，所以 haproxy 天生支持 websocket
option httpclose //在请求和响应两端分别加上 connection: close 字段

连接复用
//对于默认情况下，client 带了 keep-alive header 时，后端连接会保留，这时候
//连接可以下次的重复使用，即多个客户端先后使用同一个服务端
http-reuse never //永远不重复利用，默认选项
http-reuse safe //client 的第一个请求新建后端连接，后续请求可以复用之前别的 client 遗留下来的连接
http-reuse aggressive //client 的第一个请求只会复用之前已经复用过一次及以上的遗留连接
http-reuse always //总是重复利用之前遗留的连接

HAproxy 健康检查的三种方式

1、通过监听端口进行健康检测 。

Haproxy 的默认的 tcp 层健康检查机制是利用 TCP 的三次握手
1、首先由 Haproxy 向代理的服务器发起 SYN 握手协商，默认是与代理的端口建立链接，比如说 8080。
2、等待代理服务器确认第一次 SYN，并响应 ACK，与发起 SYN 的第二次握手。
3、Haproxy 收到确认 ACK 之后，会向代理服务器发送 TCP 链接重置的报文，已经确认代理的服务器健康。

这种检测方式，haproxy 只会去检查后端 server 的端口，并不能保证服务的真正可用。
配置示例：

backend app

```
  balance roundrobin
  server web1 192.168.1.1:80 check
  server web2 192.168.1.2:80 check inter 500 rise 1 fall 2   #check 代表开启健康检查  inter 是检查频率，单位是毫秒，默认 2000ms

  //可能在一个 ip 上面绑定多个端口，这样健康检查可以像下面这样做，
  //haproxy 先连 110 端口，检查返回值，再连 143 端口，发送 test 字符串，再检查返回值，整个过程是一个事务，都成功，才认为这次检查的结果是健康的
  option tcp-checkoption tcp-check
  tcp-check connect port 110
  tcp-check expect string +OK\ POP3\ ready
  tcp-check connect port 143
  tcp-check send test
  tcp-check expect string *\ OK\ IMAP4\ ready
  server mail 10.0.0.1 check
-------------------------------------------------------------------------------------------------------------------------
---

#tcp 检查返回指定字符串报文
backend be_mayapp
 option tcp-check
 tcp-check send PING\r\n
 tcp-check expect string PONG
 server srv1 10.0.0.1:80 check
```

2、通过 URI 获取进行健康检测

检测方式，是用过去 GET 后端 server 的的 web 页面，基本上可以代表后端服务的可用性。
配置示例：

```
backend app
  mode http
  cookie SERVERID
  balance roundrobin
  option httpchk GET /index.html
  #http-check expect string OK   #返回内容必须包含 OK 字符串
  server web1 192.168.1.1:80 cookie server01 check
  server web2 192.168.1.2:80 cookie server02 check inter 500 rise 1 fall 2

 //返回码只有 2/3 开头才会认为检查是成功的
 http-check expect rstatus ^[23]

 //指定成功返回码
 http-check expect status 200

 //发送服务器的健康检查状态给后端
```

```
 http-check send-state
 #[HTTP_X_HAPROXY_SERVER_STATE] => UP; address=192.168.1.166; port=80; name=static/static1; node=ecs-6b57; weight=1/1; scur=0/0; qcur=0 后端会收到如下信息
 #scur=0/0 第一个数字表示当前正在进行的会话数，第二个数字表示最大会话数 qcur=0 当前排队等待处理的请求数量
 //返回 404 时，进入维修模式，请求不会转发到这个后端，可以作为 http 服务器升级使用
 http-check disable-on-404
-------------------------------------------------------------------------------------------------------------------------------

#进行健康检查并发送指定请求内容
backend be_myapp
 option httpchk
 http-check send meth HEAD uri /healthz ver HTTP/1.1 hdr Host test.local
 server srv1 192.168.1.5:80 check


backend be_myapp
 option httpchk
 http-check send meth POST uri /health hdr Content-Type "application/json;charset=UTF-8" hdr Host www.mwebsite.com body "{\"id\": 1, \"field\": \"value\"}"
 server srv1 192.168.1.5:80 check
-------------------------------------------------------------------------------------------------------------------------------

#通过 https 进行健康检查
backend be_myapp
 option httpchk
 http-check connect ssl alpn h2
 http-check send meth HEAD uri /health ver HTTP/2 hdr Host www.test.local
 server srv1 192.168.1.5:443 check
-------------------------------------------------------------------------------------------------------------------------------

#检查多个端口或者 url
backend servers
 option httpchk

 http-check connect port 8080
 http-check send meth HEAD uri /health
 http-check expect status 200

 http-check connect port 8081
```

```
 http-check send meth HEAD uri /up
 http-check expect status 200

 server server1 127.0.0.1:80 check

#Agent checks
backend servers
 server server1 192.168.0.10:80 check  weight 100  agent-check  agent-addr 192.168.0.10  agent-port 8080  agent-inter 5s  agent-send ping\n
```

go 实现简单的检查示例：

```
package main

import (
	"fmt"
	"io/ioutil"
	"net/http"
)

func checkService(url string) error {
	// 发起 GET 请求到指定 URL，这里假设是"http://localhost:80/"
	resp, err := http.Get(url)
	if err != nil {
		return err
	}
	defer resp.Body.Close()

	// 读取响应体，这里实际上可能不需要，因为我们主要看状态码
	body, _ := ioutil.ReadAll(resp.Body)

	// 检查 HTTP 状态码，200 表示成功
	if resp.StatusCode == http.StatusOK {
		fmt.Printf("Service is healthy. Response: %s\n", string(body))
		return nil
	} else {
		fmt.Printf("Service is not healthy. Status code: %d\n", resp.StatusCode)
		return fmt.Errorf("unexpected status code: %d", resp.StatusCode)
	}
}

func main() {
	// 假设我们要检查的服务地址是"http://localhost:80"
	err := checkService("http://localhost:80/")
	if err != nil {
```

```
        fmt.Println("Failed to check service:", err)
    } else {
        fmt.Println("Service check completed successfully.")
    }
}
```

3、通过 request 获取的头部信息进行匹配进行健康检测 。

这种检测方式，则是基于高级，精细的一些监测需求。通过对后端服务访问的头部信息进行匹配检测。

配置示例：

listen http_proxy 0.0.0.0:80

```
  mode http
  cookie SERVERID
  balance roundrobin
  option httpchk HEAD /index.jsp HTTP/1.1\r\nHost:\ www.xxx.com
  server web1 192.168.1.1:80 cookie server01 check
  server web2 192.168.1.2:80 cookie server02 check inter 500 rise 1 fall 2
```

4、外部健康检查

```
  //健康检查方法 3
    //外部健康检查，调用外部进程来完成健康检查，健康检查的结果通过捕获子进程退出的
    //信号，waitpid 获取子进程退出码，退出状态为 0 代表健康

    //传给子进程的参数通过注入环境变量完成
    option external-check

    //外部进程
    external-check command /bin/true

    //外部脚本
    external-check path /etc/haproxy/health_check.sh

    //外部进程的 PATH 环境变量
    external-check path "/usr/bin:/bin"

  //另外还支持多种健康检查方式，例如 redis，mysql，smtp 等
  //代理健康检查，agent，主要用于后端服务的升级
  //当后端服务要升级时，先后端服务商 listen 18080 端口，然后 haproxy 会检测这个端口打开
  //会连接上去，这时候返回 maint 字符串，haproxy 收到这个字符串以后，会把后端设置为维修模式，
  //这样，就不会有新的连接过来了，等老的连接断掉以后，把服务器升级重启，
  //然后再发送 ready 给 haproxy，haproxy 把 server 放到正常的队列里面，开始提供服务
```

```
//除了一开始发送 maint，也可以发送 up，down 来设置 server 的状态
server server1 192.168.1.3:8080 check inter 5s agent-check agent-inter 10s age
```

tips：

支持健康检查方法：httpchk、smtpchk、mysql-check、pgsql-check、ssl-hello-chk、redis-chk

pgsql-check

```
listen postgresql_haproxy
  bind *:5432
  mode tcp
  option tcplog
  option pgsql-check user haproxy_check password your_password
  server db1 192.168.1.10:5432 check
  server db2 192.168.1.11:5432 check
```

redis-chk

```
backend redis_slaves
  mode tcp
  option tcp-check
  tcp-check send INFO\ replication\r\n
  tcp-check expect string "role:slave"
  tcp-check send QUIT\r\n
  tcp-check expect string +OK
  server db1 192.168.1.10:5432 check
  server db2 192.168.1.11:5432 check
```

mysql-check

在 HAProxy 配置中对 MySQL 服务器进行健康检查，通常需要利用 HAProxy 的 TCP 模式，并发送 SQL 查询来验证数据库的响应能力。以下是一个基本的配置示例，展示了如何设置 HAProxy 以执行简单的 MySQL 健康检查：
HAProxy 不直接支持发送 SQL 命令进行健康检查，但可以通过发送一个简单的查询（如 ping 或查询数据库版本）来间接判断 MySQL 服务是否在线。由于直接发送 SQL 命令作为健康检查并不直接支持，通常采用 TCP 连接的保持和响应时间来间接评估健康状态。不过，可以模拟一个简单的逻辑，通过 TCP 检查来确保连接是可用的，虽然这不等同于数据库完全健康的直接验证。

```
backend mysql_servers
  mode tcp
  option tcp-check
  # 这里不直接发送 SQL 命令，而是依赖于 TCP 连接的成功与否
  # 如果需要更精确的健康检查，可能需要外部脚本或更复杂的逻辑
  tcp-check send PING\r\n
  tcp-check expect string PONG
```

```
server mysql1 192.168.1.10:3306 check inter 5s fall 3 rise 2
server mysql2 192.168.1.11:3306 check inter 5s fall 3 rise 2
```