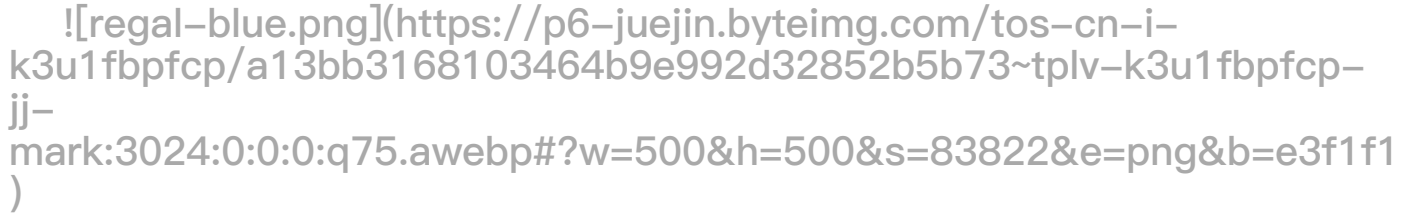
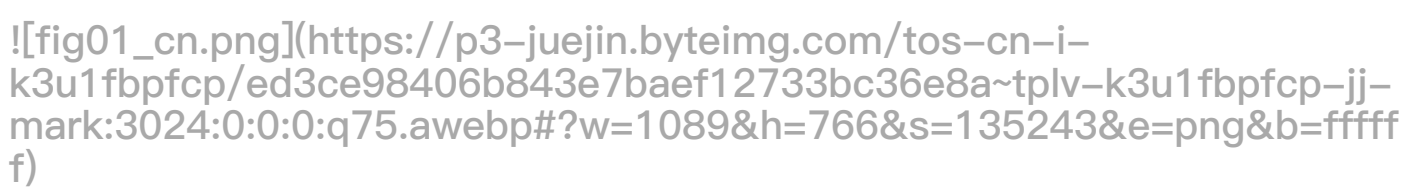
灰度分组引擎Regal-Go正式发布



用于”灰度发布”或 A/B Testing的智能分组引擎(Golang版)

Regal-Go能做什么?

举个最简单的例子，比如需要针对一个版本进行灰度发布，而这一版本对应的可能是一大堆服务器集群， 如下图:



就像图中描述的一样，无论你的服务器是多少，尤其很多中小型组织在进行灰度发布时，通常会面临分流策略在实际的技术或开发中如何去实现；

因此让Regal引擎直接介入，让它来根据你的策略进行动态地分组分流。 这里提供了两个参数:

* Combine

表示每组中的机器数量

* Schedule

作为A/B的第一组，默认为1。 可以通过使用’schedule’参数更改此行为。

功能
--

1. 提供分组策略，动态分流；
2. 支持多版本分组以及优先级可配置能力；

示例
--

详细示例可以查看“[项目GitHub](http://cxyroad.com/
”https://github.com/boylegu/regal-go”)”

示例1

```
package main

import (
”fmt”
”github.com/boylegu/regal-go”
)

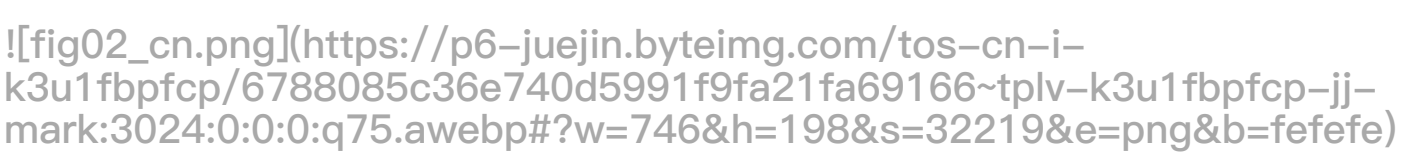
func main() {
var example1 = [][]string{
{”app-test-ver1”, ”10.1.1.1,10.1.1.2,10.1.1.3,10.1.1.4,10.1.1.5”},
}
c1 := regal.RegalEngine(example1, regal.WithCombine(2))
fmt.Println(c1.Grouping())
}
```

Output:

```
[root@gbe-pcx example]# go run main.go
[[app-test-version1.0 [[10.1.1.1] [10.1.1.2 10.1.1.3] [10.1.1.4 10.1.1.5]]]]
```

```

根据策略设置，会得到一个数据结构，这里可以观察一下：




### Example-2

```
var example2 = [][]string{
{"ver1", "10.1.1.1,10.1.1.2,10.1.1.3,10.1.1.4,10.1.1.5,10.1.1.6"},
{"ver2", "10.1.1.1,10.1.1.2,10.1.1.3,10.1.1.4,10.1.1.5"},
{"ver3", "10.1.1.1,10.1.1.2,10.1.1.3,10.1.1.4,10.1.1.5"},
}
c2 := regal.RegalEngine(
example2,
regal.WithCombine(3),
regal.WithSchedule(2),
regal.WithPriorKey("ver2"), // Set priority
)
for _, v := range c2.Grouping() {
fmt.Println(v)
}
```

Output:

```
[root@gubaoer-pcx example]# go run main.go
[ver2 [[10.1.1.1, 10.1.1.2] [10.1.1.3 10.1.1.4 10.1.1.5]]]
[ver1 [[10.1.1.1, 10.1.1.2] [10.1.1.3 10.1.1.4 10.1.1.5] [10.1.1.6]]]
[ver3 [[10.1.1.1, 10.1.1.2] [10.1.1.3 10.1.1.4 10.1.1.5]]]
```

最后
--
```

各位好，我是曾经的Regal引擎作者，时隔多年，再次推出Go语言版本的实现。尽管灰度发布还是常规的A/B Test已经是各个行业和公司进行数字化基础建设的标配，然而无论是自研还是开源工具，灰度分组的策略实现依然较为黑盒，希望这次golang版本的regal能够再一次帮助到有需要的朋友。

原文链接: https://juejin.cn/post/7379446193141530651