# Koala通用频率控制系统 - 概要介绍

黑夜路人（heiyeluren）

## 摘要

koala系统，也叫“用户行为频率控制系统”。它是用GO语言开发的高性能后端独立服务，采用redis缓存用户行为数据。初衷是支撑某互联网企业线上的多种用户行为频率控制需求，属于反作弊功能的一个组成部分。同时，它将控制策略完全配置化，koala系统本身不和业务策略直接耦合，提供http接口供业务方访问，故通用性较强，也适用于各类UGC产品的同类需求。

## 概述

通常在论坛等“用户提交类产品”中，面对各种垃圾灌水的威胁，PM会提出诸如“每个用户每天只能回帖100次”、“两次回帖至少间隔10秒”等等的反作弊需求，而最直接的做法，就是在服务端逻辑里写上“if(submit > 100) return false;”。

PM永远不会满足于当前的功能，当需求越来越多，逻辑也就越来越复杂。

当然大部分人都会想到封装，将此类功能封装到一个模块中，可维护性更高了。但是，如果没有一个比较抽象的配置文件，控制策略就会散落在代码中而不易管理；而且分布在web后端众多机器中，仅依靠同步代码来同步策略。

koala系统的引入，将这些策略统一收集到一个独立的后端服务中。所有web服务器通过访问koala服务器来完成控制功能。

并且koala系统实现了比较通用、又比较灵活的“策略配置化”能力。需要的变更，只需要对配置文件进行修改。

koala系统同时也以“高性能”为目标之一。采用GO语言实现，一方面可以有接近“脚本语言”的开发效率，另一方面有可以有接近C语言级别的性能指标。GO语言本身也是以“高并发”、“高性能”为设计目标。

并且常驻内存的demon进程，也使我们拥有了在进程地址空间内对关键数据进行缓存的能力，对性能将是有力的支撑。

有了高性能的支撑，koala不仅仅应用在“提交”操作（写操作）。比如请求量较大的“浏览”操作（读操作），也可以应用，以应对各种恶意抓取行为。

## koala结构

koala服务，是单进程，多线程，多协程（goroutine）结构模式。



图1

其总体结构如图1所示。

图中，http front、worker、logger、rule loader分别都是一个GO协程，GO语言内部实现了轻量级的协程goroutine，每个goroutine有独立的运行栈，被进程的“运行时包”调度执行，依附于OS线程运行。

httpFront负责监听并接受连接，解析http参数，并定位到特定的接口函数执行，每个请求，都起一个独立的协程，代表一个worker。

logger除了提供log相关API之外，更主要的是包含了一个log缓冲的协程。

RuleLoader是提供控制策略自动加载的独立协程。

## GO语言框架

GO语言现世较晚，实际上，没有什么现成的框架。

本次koala系统的开发，采用的GO语言http服务的模式，很容易用于以后的GO开发项目中。姑且将这些称为一个“GO框架”。

* **http框架**

采用了一个由公司内部其他团队开发并长期使用的一个http模块，清晰、实用。

此http模块，实现了“http包解析”、“http包封装”、“路由”等等必要的功能。提供了http接口的快速开发能力。

本次koala开发，基本沿用了此模块，只是补充了“获取client IP”的api。

* **log模块**

日志模块，是本项目中，新开发的模块。

主要实现了“日志分级”、“日志切割”、“日志合并写入”等功能。

有丰富的日志配置项。可以分别配置每个日志级别的“目标文件”、日志切割周期、日志输出开关等

应用GO语言的channal，我们实现了对日志的“缓存”、“合并写”；有多种策略控制日志写入文件的时机。

* **redis库**

引入了一个比较完善的redis客户端的GO语言实现，提供了redis的大部分特性，同时还实现了redis连接池，对服务本身的性能有很大的帮助。

## 控制策略配置化

我们通过一条策略，作为示例，不关注的可以忽略。

#提问，同IP，超过30次后，每次提问间隔2秒

rule : [base] [act=add\_ask;ip=+;] [base=30; time=2; count=1;] [result=2; return=224]

为了高度灵活的配置频率策略，我们制定了如上所示的格式。

用[ ]分割的几个段，分别是“方法”、“参数”、“阀值”、“返回值”4个区域。

* 方法，代表策略的类型，采用一种什么控制模式，如上面的例子，就是一种有“基数”的策略，提交超过“基数”之后才会生效。
* 参数，是满足此策略的必要条件，一般是，某个参数满足某个范围，或者等于某些特定值。这些参数通过http get参数提供给koala系统。
* 阀值，定义的特定方法模式下，需要的条件值，time，和count是基本的值，代表缓存有效期，计数上限，而base则是值。
* 返回值，代表此策略名中之后，应该采取的行为，比如“阻止提交”、“出验证码”。

通过这种多维度、较灵活的配置格式，提供的较大的配置能力。保证一段时期内，新的需求完全通过修改配置来实现。

同时，我们又实现了“配置的动态更新”。因为配置被读取到内存中长期保存，我们通过检测相关文件的MD5值变化，读取并解析到内存中，采用0/1覆盖的方法，替换内存中现有的配置数据。所以，不需要加入锁操作。

通过这些设计，我们在享受更高性能的同时，也可以像php开发一样，直接替换文件，就可以让策略生效。

## koala应用模式

* **基本模式**

既然用redis缓存用户行为数据，就意味着，每次操作，都涉及对redis中缓存数据的，读和写。因此，koala目前提供的主要服务接口，就是browse和update。

每次用户操作，首先要调用browse接口，来确认“是否允许用户的这一次操作”。

如果被“允许操作”，操作完成后，再调用update接口，将缓存计数值+n。

* **应用场景**

koala最初是为了满足线上以“提交”为主的频率控制。而实际上，koala既不是为“提交（写）”操作设计的，也不和业务融合。

基本的“提交”场景，如：一次用户“提问”操作，首先，将特征数据提供给koala，查询koala系统以确认此次“提问”的否允许执行；通过后，才进行提交内容的入库落地，成功以后，将此次行为记录。“提问”过程最终完成。

* **更多的应用场景**

比如：A产品，需要知道某个用户在某个特定的页面，是否做了某个选择（给某个帖子投票等，如果选择了，希望按钮“置灰”），cookie等方式自然就不可行了。如果使用koala，需要配置一项规则：“每个user\_id对每个帖子，X时段内，只能点1次W操作”。

再比如：

* “u用户，X时段内，Y广告弹窗，不超过1次”
* “两次Y广告弹窗，至少间隔2小时”
* “u用户，X时段内，积分增长，不超过300分”
* “某IP，X时段内，发帖超过10次，需要出验证码”

**……**

## 总结、现状与展望

koala的开发，很大一部分工作是实现对策略配置的解析，以及配置的自动load功能。而服务接口逻辑相对简单，因为很多事情，交给配置来定义了。配置语法中包含了“关键词”、“操作符”，俨然就是一个微型语言。但语法分析器之类的高级货就有点杀鸡用牛刀的意思了。简单考虑，仅用基本的条件判断if-else来实现，显得比较笨拙。

目前koala已经完成了web业务的迁移，正在为提供用户行为相关的控制。面对目前每天大约两千万次的请求调用，koala基本可以轻松应对。

从易用性的角度看，koala尚有一些不足，比如，缺少一个图形化的策略更新平台，这方面，可以补充开发一个管理后台页面，实现策略的“查看”、“上传”、“正确性验证”。

从通用性的角度来说，koala目前已经可以支持多个产品的同时应用，当然资源的隔离等方面还有必要改进。