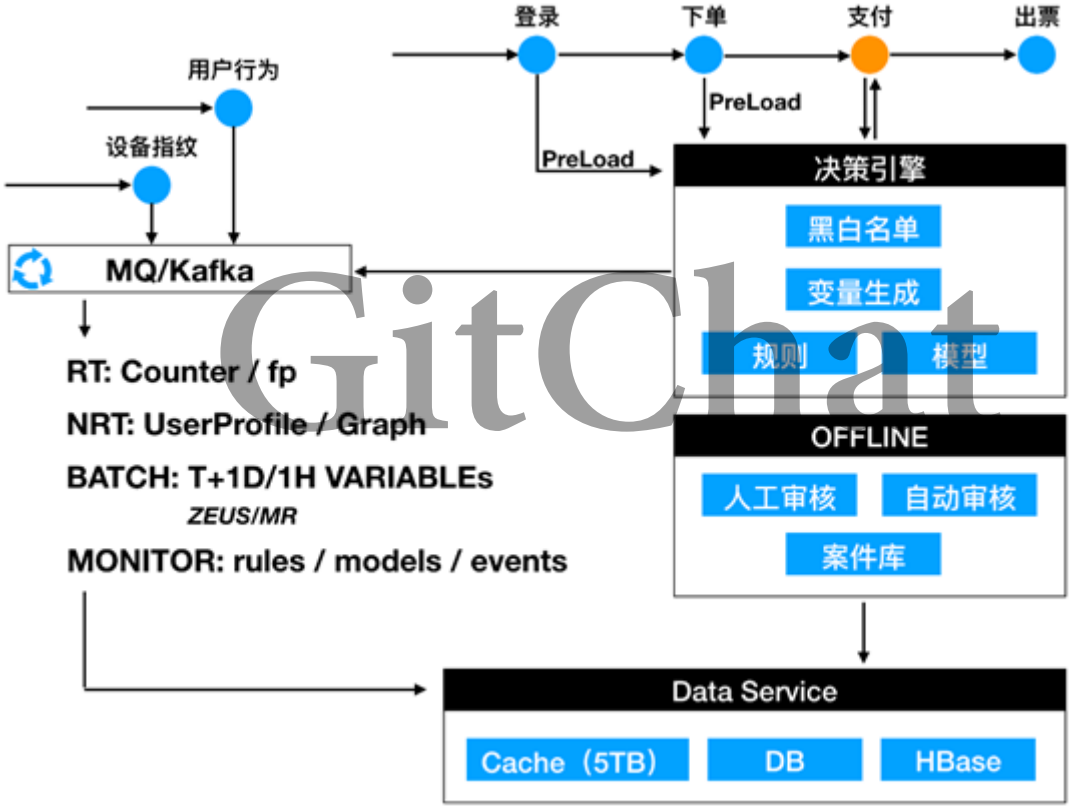
携程在线风控系统的架构解析和问题总结

为了应对日益严重的支付欺诈，携程在线风控系统2011年正式上线。现在，在线风控系统支撑了携程每日1亿+的风险事件实时处理和100亿+的准实时数据预处理；系统中运行的总规则数和总模型数分别达到了1万+和20+；风控的范围从单纯的支付风控扩展到了各种类型的业务风控（例如：恶意抢占资源、黄牛抢购、商家刷单）。

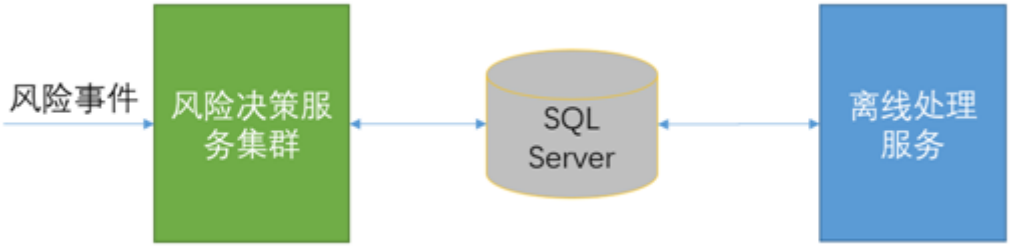
下图是当前在线风控系统的整体技术架构图 ：



当前的系统结构是比较主流的风控系统结构，包含了决策引擎、Counter、名单库、用户画像、离线处理、离线分析和监控各主要模块。携程的在线风控系统发展到这个阶段一共经过了3次重大的改版：

最初创立

2011年上线，使用了.net开发的服务，数据库使用SQL Server，简要架构图如下：



这个时候的风控服务将所有在线决策功能整合在一个系统内实现，包括规则判断、名单库、流量计算；而这些逻辑都基于数据库实现。

流量计算：通过明细表执行SQL得到（例如：SELECT COUNT(DISTINCT orderId) FROM t1 WHERE ... ）

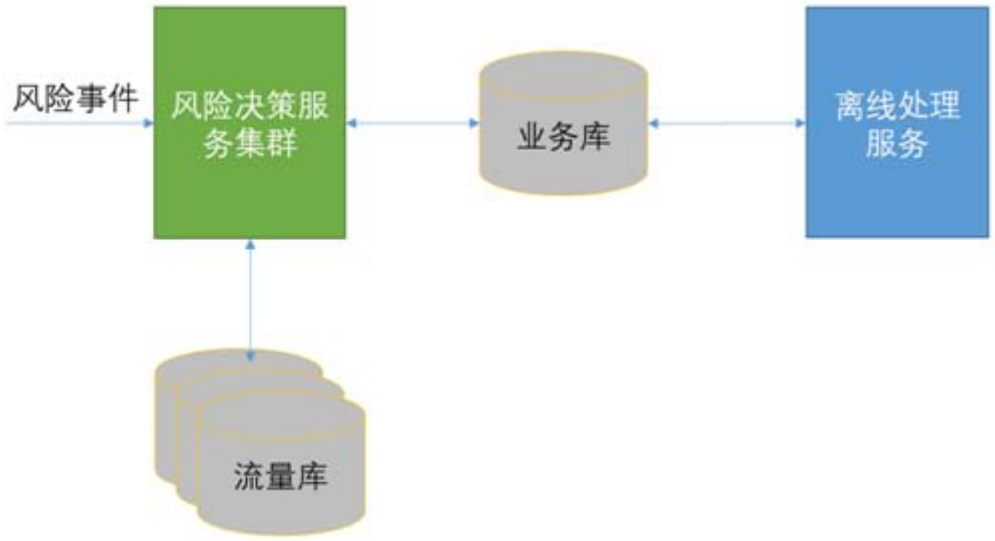
规则判断：数据库记录大于、小于、等于等判断规则，接收到风险事件后获取流量值和规则进行比较，得到最终的风险判断。

名单库判断：数据库维护黑白名单信息（属性类型、属性值、判断依据等），程序判断风险事件中的值是否命中名单。

基于当时携程对风控的需求，系统以满足功能为主。在上线运行一段时间后，随着携程业务的增长，风控系统的流量不断增加，基于SQL的流量统计耗时严重制约了系统的响应时间，因此有了第一次的性能优化改版。

流量查询性能优化改版

由于这个时候的主要性能瓶颈在于数据库实现的流量查询，这次优化主要方向就是优化流量查询的实现：在原来单个数据库的基础上，采用分库分表的方式均摊压力，以达到更快的响应时间和更高的吞吐量。架构图如下：



优化之后，流量库和业务库分离，流量库使用多个数据库实例，使用hash的方式拆分流量明细，统计的时候使用SQL。由于压力被多个数据库实例分摊，使得系统的流量查询性能得到了较好的提升。

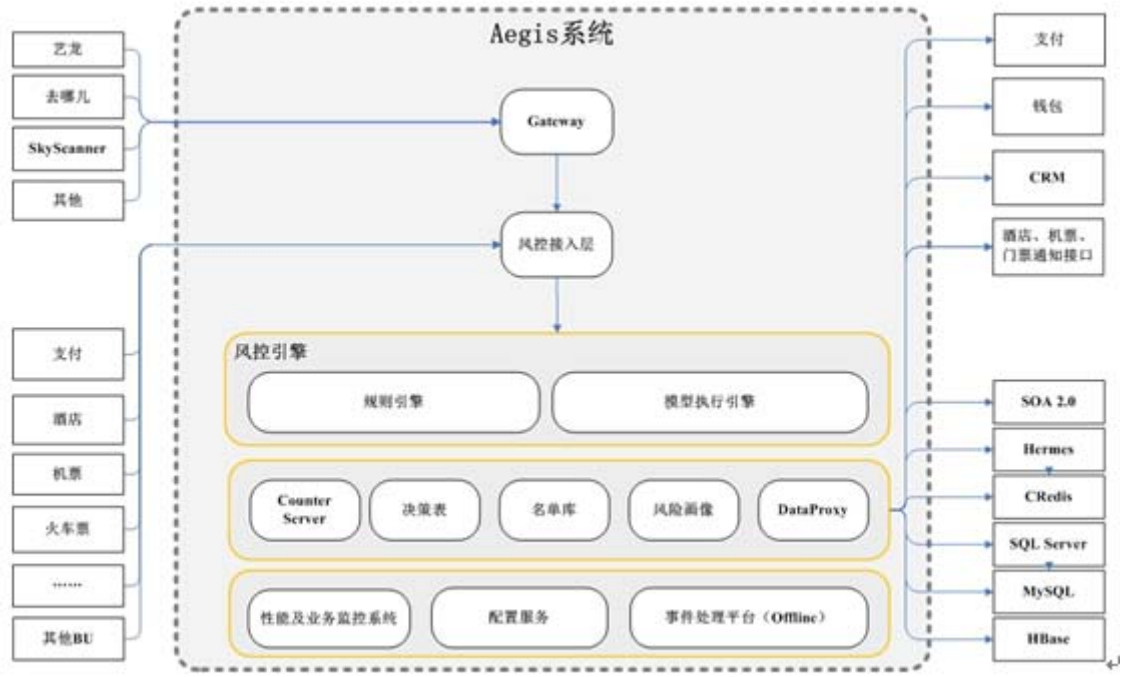
新版本上线后，携程的业务又对风控系统提出了更多的要求：

1. 更方便快捷的接入：除了支付风险，业务的风险也需要风控支持；
2. 更多的外部数据接入：用户信息、位置信息、UBT信息；
3. 更丰富的规则逻辑：支持任意变量的规则判断，支持更多的判断逻辑；
4. 更高的性能：流量10x的增长，响应时间不超过1秒；
5. 编程语言的更新：携程推动公司内.net转java。

然后，就有了奠定当前在线风控系统基础的重大改版。

风控3.0（Aegis）

从这个版本开始，风控系统全面转向Java开发，同时将核心模块独立成服务，定义了各子系统的边界以及在整个系统中的定位和作用。相比之前功能性的应用，Aegis是一个平台化的风控系统。以下是简化的系统架构图

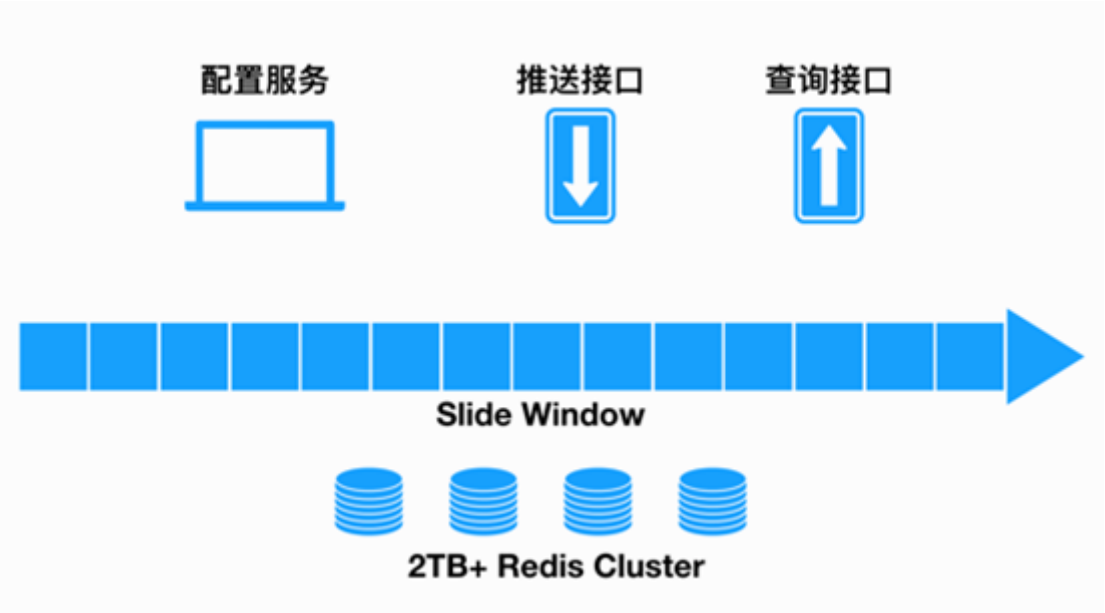


Aegis开始使用Drools脚本用于规则的编写，极大的提高了规则团队对突发事件的响应时间，紧急规则一般10分钟就能上线。

在结构上风控引擎分为同步引擎和异步引擎，同步引擎运行用于实时判断风险结果的规则和模型；异步引擎则负责验证规则/模型、数据分发、关键数据落地等逻辑。同步/异

步引擎设计成无状态的，方便随时扩容。

Aegis在流量统计上自研发了Counter Server，这是一个定制化的类TSDB服务，任意精度任意时间窗口的查询控制在5ms，同时支持高并发查询，相较于SQL的实现提升了上千倍的性能。支撑了现在风控系统内个服务每日百亿次的查询。下图是其简要的结构说明 ：



在数据预处理层面独立出了风险画像和DataProxy两个服务。

风险画像服务为引擎提供实时的用户、订单的画像（用户等级、用户行为标签、订单资源标签等）作为规则和模型的输入变量；其数据来源是实时的引擎数据、准实时的MQ数据清洗服务、离线的数据导入三部分。

DataProxy服务包装了所有对外的接口和数据库的访问，并针对数据特性的不同配置了不同的缓存策略，保证99.9%的请求在10ms内获取到所需的数据。

此外还有以下几个主要的服务：

名单库服务，支持多个独立的名单库，优化了名单判断逻辑，使得单次查询（10个维度）的响应小于10ms。

配置服务，集中系统内各个应用需要配置的功能，提供中心化的配置服务让各个应用获取响应配置。

事件处理平台，用于处理引擎无法判断或需要人工干预的事件。

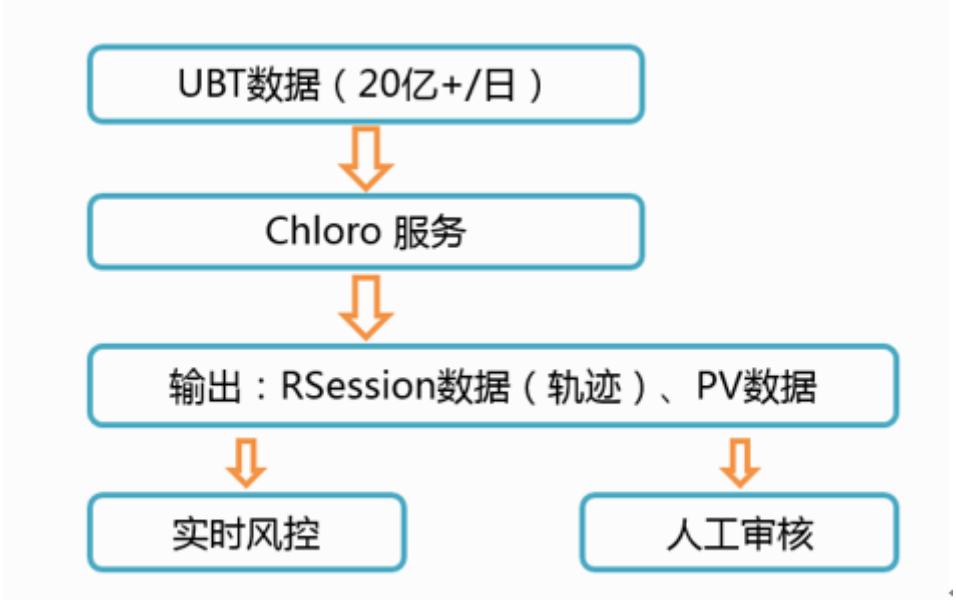
性能监控服务，监控系统内各服务的健康状态，提供预警和报警功能。

业务监控服务，监控规则模型运行情况、返回的风险结果、事件耗时等业务层面的数据，提供预警和报警功能。

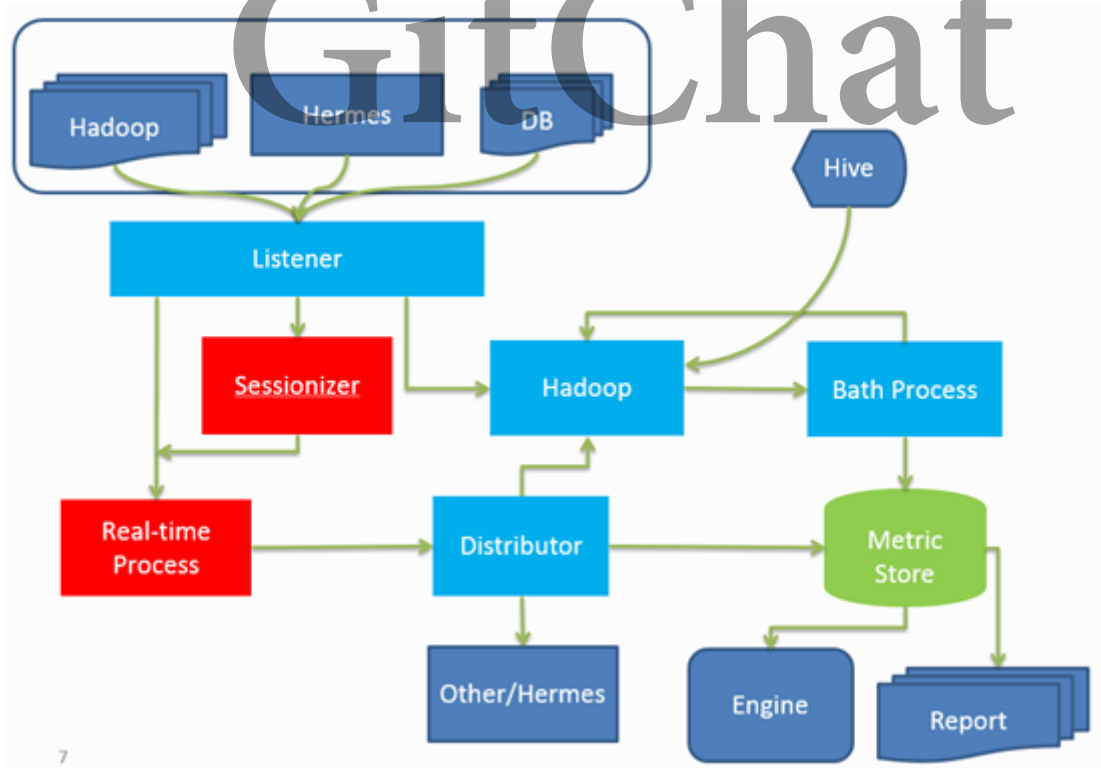
Aegis系统上线后，新风控业务接入时间缩短到一周，在10x流量增加和执行更多更复杂的规则的情况下，平均耗时控制在300+ms，比上一版本提高了1倍多。

之后Aegis家族又增加了两个重要的子系统：Sessionizer和DeviceID。这两个服务属于准实时处理应用，但是都通过预热的方式为引擎提供了实时数据。

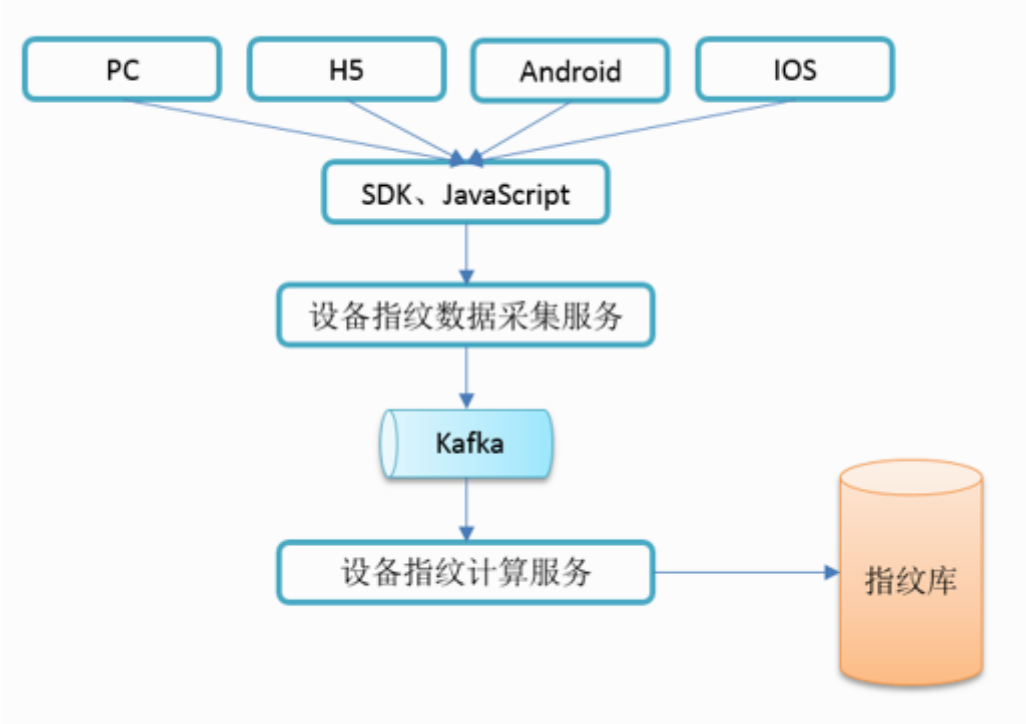
Sessionizer，归约用户的页面访问session为反应用户的操作的RiskSession，流程如下图：



其数据处理使用了自研的大数据处理系统Chloro，架构图如下：



DeviceID服务，用于指纹数据采集和指纹识别生成，从而判断设备的唯一性，同样也借助了Chloro进行数据处理，系统示意图如下：



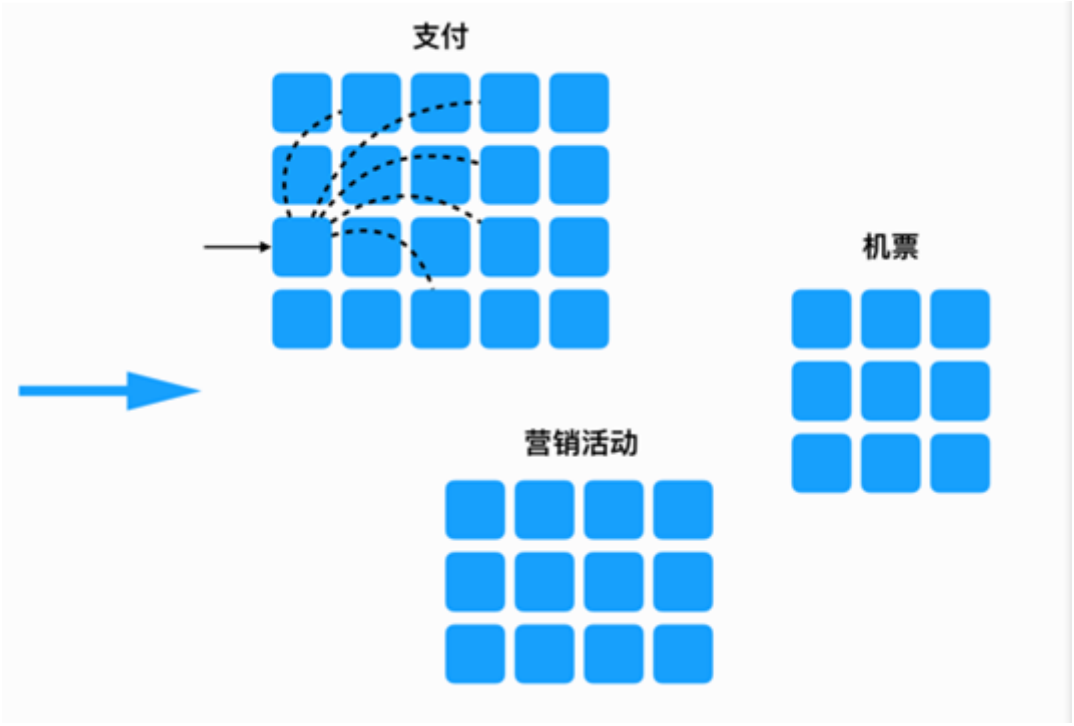
这两个服务为规则和模型以及人工处理提供了非常关键的判断数据，进一步提高了风险判断的准确性。

只此，Aegis系统的功能已比较完善，但在1年多的运行过程中发现随着流量和规则、模型量的持续增长，实时风控的响应时间出现缓慢下滑的趋势，超时（1秒）的量在千分之一上下，然后就有了之后持续的性能优化。

针对实时风控的性能优化

性能优化从1年前开始，可以分以下几个主要优化：

1. 规则分布式并行执行 ：



将单个事件需要执行的规则和模型拆分到同一逻辑组内多个服务器执行，最终合并数据。这个优化将平均耗时降到了200+ms。

2. 脚本执行引擎切换Drools到Java

   使用模版屏蔽编写规则时drools的特殊语法，之后将脚本编译成Java class执行。规则的执行性能提升1倍，整个引擎的实时平均耗时降入100ms。

3. 开发Java的模型执行引擎

   已完成随机森林和逻辑回归算法的Java版，相较使用Python，提高了一个数量级的性能。

完成上述优化后，整个系统在短期内，只需要简单的增加服务器，就可以满足容量扩张。

以上就是Aegis系统的架构和演进过程，当然演进的过程还在持续，现阶段的目标是将平台化的系统继续发展，做到服务产品化。