網易 NETEASE
www·163·com

WEB应用的亡羊补牢之道

异常情况的发现与控制

为什么说亡羊补牢

- 不可避免的异常情况
  - 无处不在的软件Bug
  - 硬件设备毫无征兆的故障
  - 潜在的性能问题
  - 突如其来的外部攻击

WHY?

- WEB应用的开发节奏飞快
- 互联网环境复杂多变
- 设备可靠性有限

为什么说亡羊补牢（cont'd）

- 出现异常后将损失最小化=亡羊补牢
  - 快速发现并响应
    - 应用级运行时状态的监控
    - 攻击与错误状态的日志检测
    - 完善的报警机制 IM + Mail + SMS

为什么说亡羊补牢（cont'd）

- 自动隔离异常控制影响范围
  - 通过统一的架构规范服务间的调用

Notice

- 大部分WEB调用服务的接口依然是同步接口，一旦服务发生问题容易产生连锁反应。
- 对于WEB应用，立即返回错误的体验往往好过漫长的等待。

监控平台

- 设计目标
  - 解决应用级监控问题
  - 支持多种语言环境
  - 提供全面的报警功能
  - 覆盖传统监控软件功能

监控平台(cont'd)

- 什么是应用级监控
  - 业务逻辑相关

举例：

博客中可能需要监控，日志发表情况、相片上传情况等

邮箱需要监控日志发送、附件上传等

  - 监控信息的判断标准随业务逻辑的变化而变化

举例：

A服务每小时只在指定时间执行1次，只需关注这次执行的情况

B服务执行非常频繁，由于依赖第三方接口，允许少量的失败

监控平台(cont'd)

- 应用级监控的困难
  - 该收集什么样的信息用于监控？
  - 根据信息如何判断应用的状态？
  - 各产品、服务愿意为监控引入多少复杂度？

监控平台(cont'd)

- 应用级监控的解决方案
  - 基于WEB应用架构的假设
    - 业务逻辑相关的监控数据始终以集群为单位处理
    - 不应存在集群中的一个或几个特定节点具有特殊的业务逻辑的情况。

推论：应用级监控无需关心集群中单个节点的监控数据

监控平台(cont'd)

- 应用级运行时数据采集
  - 应用按照一定的规范暴露运行时数据
  - 数据形式为 {timestamp, number}
    - 实践证明时间戳+数值的表现形式已能很丰富的展现运行时状态。

实际表达的信息为：{namespace, key, {timestamp, number}}

  - 采集到的数据根据监控平台中服务集群的定义对数据进行加权，变化率计算等标准处理

监控平台(cont'd)

- 数据的处理
  - 将运行时数据最终的分析逻辑交还给服务开发者
    - 开发者采用脚本语言（可以是基于JVM的任意脚本）处理平台为其经过预处理的数据。
    - 定义了API、DSL、类SQL的query接口以屏蔽监控数据的数据结构，便于开发者处理。

WHY?

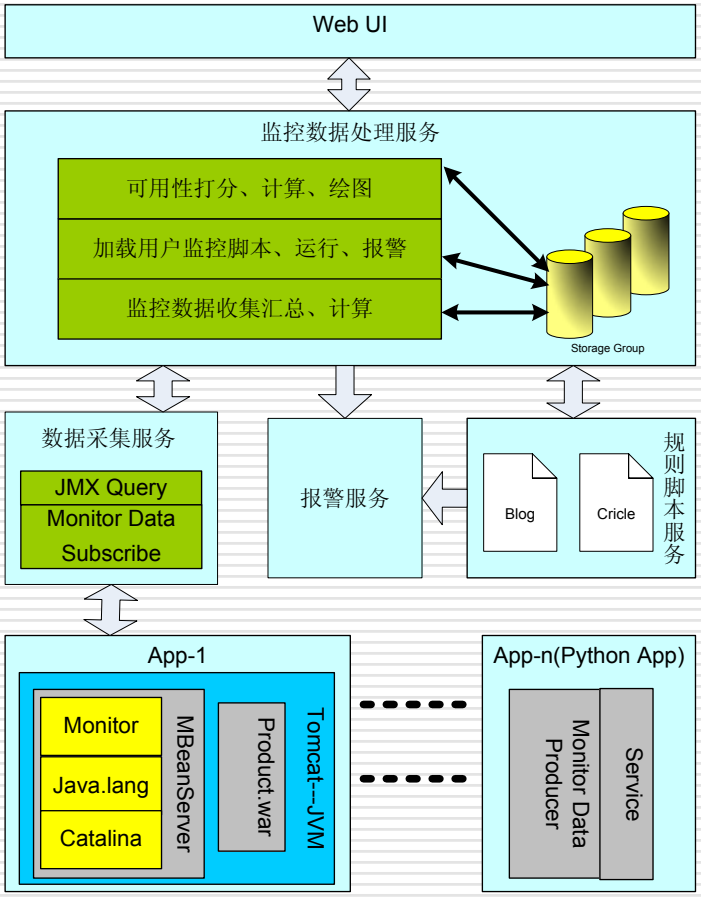
因为各类应用的逻辑千变万化，只有开发者真正理解运行时数据的意义。提供监控的基础设施（数据采集，报表呈现，报警）让开发者仅关注运行时应用状态的分析即是监控平台的设计理念。

监控平台(cont'd)

- 多语言支持
  - 支持JMX接口以支持基于java的应用
    - Dynamic MBeans
  - 订阅模式
    - 采用AMQP协议
    - 各服务以json格式播报运行时数据

监控平台(cont'd)

系统整体架构



采用ESB隔离异常

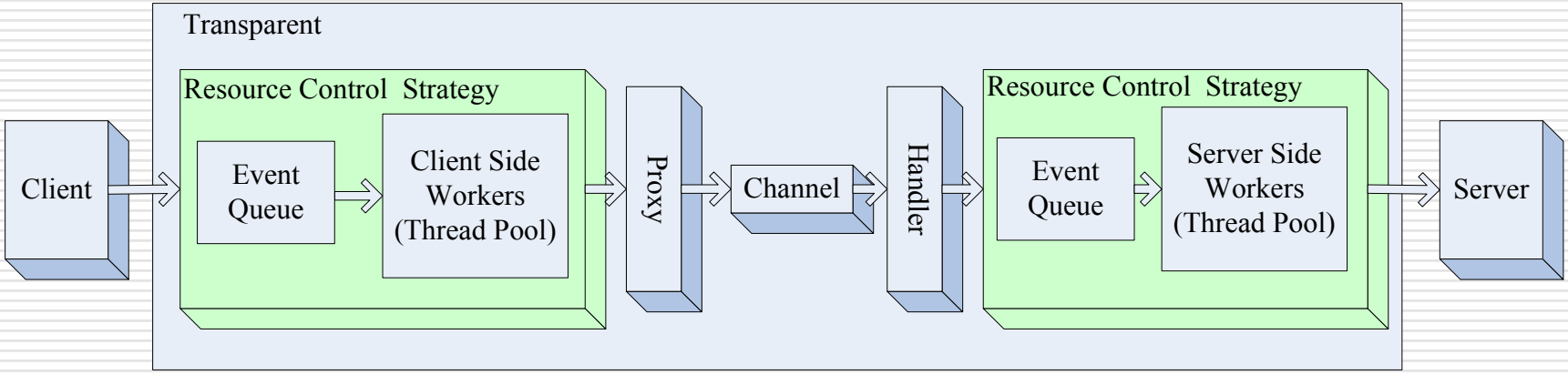
- 核心概念
  - Fail-fast & Fail-safe
  - 服务层面的SEDA模型
  - 控制Client消耗在特定服务的抽象资源上限
  - 控制Server端能够提供的抽象资源上限

抽象资源概念：

抽象资源不具体指CPU、内存、IO等具体资源， 而是对应于服务的处理能力，这点与Load的概念有些类似

采用ESB隔离异常（cont'd）

- 原理示意



通过资源控制策略来防止客户端或服务器端过载

采用ESB隔离异常（cont'd）

- 资源控制策略
  - 直接拒绝服务策略
    - 设置合适的Event Queue长度限制与Worker线程数。
  - 带优先级的拒绝服务策略
    - 应用实现优先级回调接口，根据Event内容指定事件优先级，当发生异常时优先满足高优先级事件的处理。

日志检测系统

- 设计目标
  - 海量日志实时处理
  - 检测CC DDos行为
  - 检测恶意爬虫
  - 检测HTTP错误状态
  - 根据检测结果触发各类处理行为

日志检测系统(cont'd)

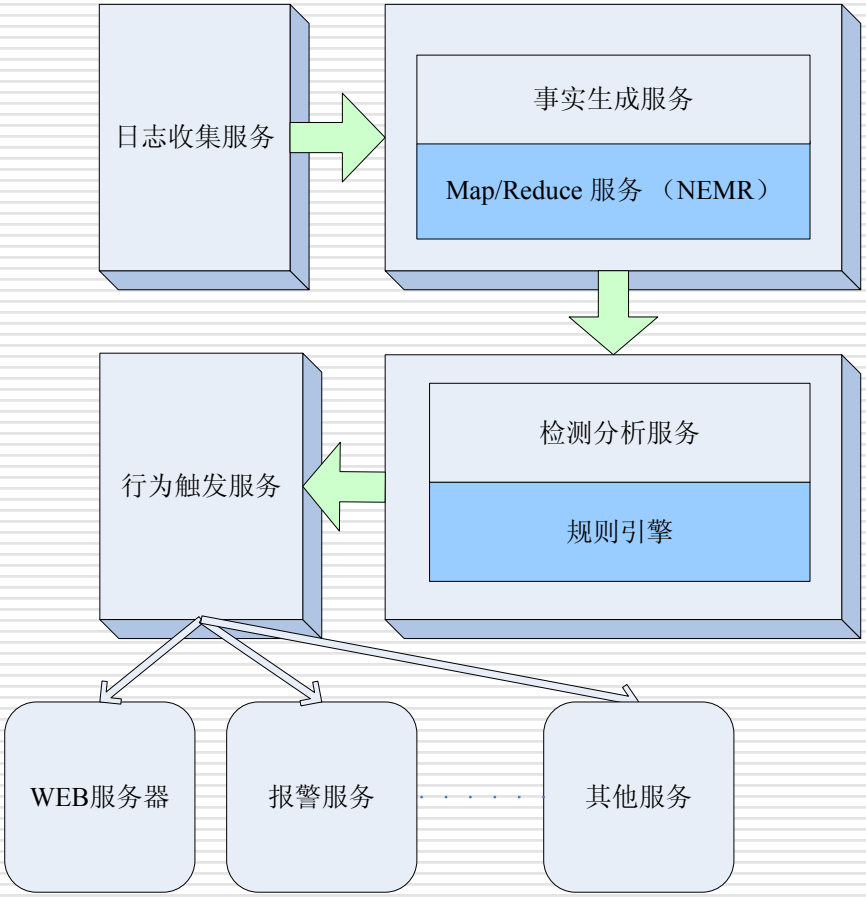
- 基于NEMR平台的海量数据实时处理
  - 利用Map/Reduce系统解决大量日志数据的实时分析。
- 利用规则引擎解决复杂的分析逻辑
  - 需要大量灵活可调整的规则实现
    事实 => 结论 的推导

举例：

利用日志包含的特征构成事实， 通过规则引擎推断出请求来源于人还是机器

日志检测系统(cont'd)

以WEB服务器访问日志检测为例



Future Works

- 监控平台
  - 更优雅的DSL
  - 远程诊断能力
  - 插件的支持

Future Works (cont'd)

- ESB
  - 与监控平台结合
  - 异常的影响域自动分析
- 日志检测系统
  - 更快的响应速度
  - Stateful

总结

- 亡羊补牢的重点在于快速发现错误
- 应用级监控的解决方法
  - 规范数据暴露方式
  - 分离关注点
- 防止错误扩散的秘密在于限制资源消耗，调用该失败时就失败。
- 日志检测同样是一项重要的监控手段，配合规则引擎能发挥更大的潜力。

谢谢！

- 陈谔
- 网易杭州研究院 前台技术部
- radiumce@gmail.com