在上一节，我带你了解了日志系统的基本原理和 ELK 是如何进行日志收集和展示的。在本节课，我将带你了解 Prometheus 是如何收集和展示统计指标数据的。

作用

我在介绍统计指标时讲过指标数据的重要性和它所能帮助我们做的事情。统计指标可以帮你了解系统当前的执行情况，并通过可视化的形式展现出来。它还可以和告警相结合，在出现异常时及时反馈。

对于监控系统而言，指标体系是其中不可缺少的一环。那么，在一个比较完整的监控系统中，指标究竟发挥了哪些作用呢？

1. **通过查看指标数据了解指标的走向，从而更好地优化系统、流量等内容。**以流量为例，假如有人在爬取你的数据，此时请求量指标可能会出现有规律的突增突降，这时你就可以结合访问日志分析出对方爬取的方式，然后结合具体的业务场景来处理。
2. **通过导出或者查询已有数据，将一段时间内的数据以 dashboard 的形式进行展示，这个也是目前主流的一个使用场景**，比如 “双十一” 淘宝的销售额。
3. **基于预先配置好的告警规则，定时检测数据是否符合规则，并对其进行告警处理。**这也是我们在指标监控中最为重要的一个目的。

原理

与讲解 ELK 时相同，这里我同样以原理为切入点，介绍目前大多数的监控系统中是如何监控的。它们底层基本都分为 4 个步骤：**数据收集**、**指标存储**、**指标查询**、**规则告警。**

数据收集

监控系统从服务器或者服务中获取数据，并将其统一收集到监控系统中。

由于统计指标是变化的，并且变化的频率较高，所以实时发送数据是不现实的，一般我们会将数据内容缓存在组件中，定期收集数据。目前主流的数据收集方式有两种。

一种是 **push，推送模式。**定期主动将数据内容发送给收集器端。

另一种是 **pull，拉取模式。**业务系统主动暴露相关的数据指标信息，收集器端利用服务发现能力感知所有可能的服务列表，并定期通过发送请求的方式获取数据内容。

两者没有好坏之分，我从以下 5 个方面对它们做出了区分，让你对在不同的场景中应该选择哪种模式有一个更好的认识。

1. 数据采集：对于推送模式来说，业务系统一旦启动，便会定期主动将数据信息上报；拉取模式则是暴露指标，由收集器决定是否获取数据。
2. 可扩展性：推送模式可以随时扩展新的节点信息，实现线性增长；拉取模式的工作负载由业务系统的数量决定，如果业务系统数量不断增加，收集器系统也只需要动态增加机器。
3. 安全性：推送模式本身就是安全的，业务系统可以防止遭受到远程攻击；拉取模式一般采用轮训协议，可能有潜在的访问攻击。
4. 灵活性：推送模式相对不灵活，因为它只能定期发送数据；拉取模式则可以随时获取数据。
5. 系统耦合：推送模式对于系统耦合性相对较高，因为它需要获取到收集器列表，并对其进行发送数据处理，如果收集系统出现问题，数据可能会丢失；拉取模式的耦合性则相对较低。

指标聚合

收集指标数据之后，收集器就可以对已有的数据内容聚合，然后保存。我在 **13 | 告警质量：如何更好地创建告警规则和质量？** 这一课时中，介绍了时序数据库的基本概念。

指标聚合也是**将数据按照主体部分，以时间维度进行按照秒或者分钟级别来聚合，然后根据不同的数据类型计算相应的数值**，比如仪表盘类型的数据只保存最近一次的值。**最后将得到的数据存储到数据库中。**

指标查询

数据存储到了相对应的数据库之后，就可以通过界面或者类似 SQL 的形式查询数据。

通常在查询时还会涉及时间范围和指标计算的工作。一个指标监控系统中最为重要的，是其**数据查询的灵活度**和**支持聚合函数的种类**，比如常见的分位值、最大值、平均值等方式的数值计算。

我们可以把这些数值与图表结合，定义自己的 dashboard，从而更快了解系统当前的运行情况。

规则告警

最后一个关键点是告警规则。**监控系统可以通过统一的配置来进行告警**，其中包含以下几个维度的数据信息。

- **检测时间**：**多长时间检测一次告警内容，时间越短说明检测的周期越密集。**通常可以指定为 1 分钟，如果业务系统对于数据比较敏感，则可以适当降低时间，比如 30 秒。
- **告警规则**：**进行告警规则的配置，通常会包含统计指标的名称、查询方式和阈值构成。**检测的时候会判断当前指标查询出的结果值是否符合告警规则，如果符合则说明达到了告警条件。
- **通知方式**：**符合告警规则后会进行相关的告警操作。**比如通过 HTTP 的回调方式告知业务系统，或者通过钉钉或者企业微信这样信息推送的方式告知业务研发人员。

通过收集数据，将数据指标聚合存储后，我们可以制定相关的规则。告警系统会查询相关的指标，判断是否符合告警的条件，如果符合，则会进行告警。能完成这样一个流程，就算是一个相对完整的指标监控系统。

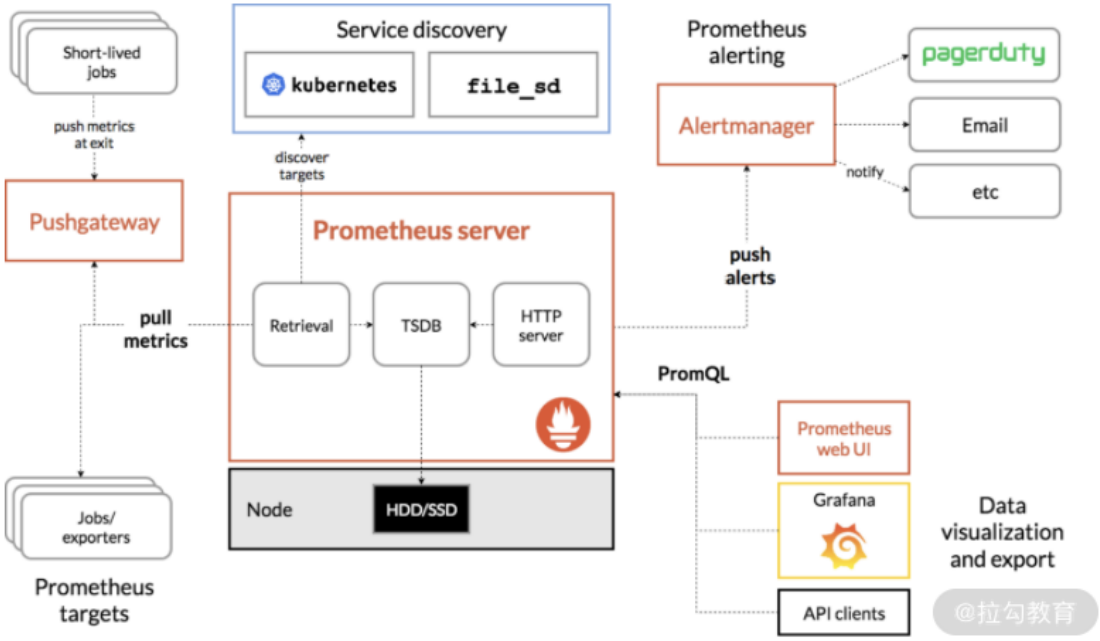
Prometheus

Prometheus 是一个开源的服务监控系统和时间序列数据库，它是一款基于统计指标的服务监控系统，可以很方便地进行统计指标的存储、查询与告警。

我会先依据我刚才讲的指标系统的原理，对 Prometheus 的系统架构做详细说明，然后讲解其中的 4 个常见功能。

系统架构

我们先来看一张 Prometheus 的架构图：



根据我在原理中讲到的数据收集、指标聚合、指标查询和规则告警，我们也可以通过这 4 个步骤来了解 Prometheus。

我们可以从图中看到，Prometheus 主要是采取拉取模式获取数据的，它提供了完善的服务发现机制，结合 K8s 的 API 可以动态感知服务的创建与销毁。通过可配置的方式，定期拉取服务列表的数据。对于一些短期存在的任务，Prometheus 同样提供了 PushGateway，让业务程序能够推送数据，再由收集系统来收集。

其次是 Prometheus 的指标聚合。Prometheus 服务器接收到数据之后，会通过 TSDB 存储引擎聚合数据，然后存储到磁盘上。TSDB 中的数据结构和我在 **13 | 告警质量：如何更好地创建告警规则和质量?** 这一课时中，讲到的时序数据库的结构一样。

在指标查询方面，Prometheus 提供了一套完整的查询语言 PromQL，通过 HTTP 服务的形式对外提供查询接口和基础的展示界面。通过接口，我们可以将数据与 Grafana 界面相结合。

Grafana 是一个强大的可视化监控指标展示工具，我们可以自定义页面中所需要展示的内容，来定制属于自己业务中的关键指标的 Dashboard。

最后，基于 Prometheus 中的配置方式来管理所有的告警内容。符合条件时，会通过 Alertmanager 来推送相应的告警内容给业务系统，比如图中的 Email、paperduty 等；也可以通过 HTTP 的方式发送给业务系统，让业务系统自行处理告警信息。

业务数据收集

介绍完 Prometheus 的系统架构之后，我再来带你了解如何将业务系统中的相关指标接入到 Prometheus 中。

一般我们会选择 Micrometer 作为接入 Prometheus 的首选 SDK。Micrometer 有点像我在日志系统中介绍的 slf4j。它通过一套统一的 API 集成多种指标系统，比如 Prometheus、Elastic、Datadog 等。我会通过一个例子来说明。

如果你的项目是基于 Java 语言和 Spring Boot 系统开发的，那么你就可以通过声明 Spring-boot-starter-actuator 和 micrometer-registry-prometheus 这两个依赖，分别引入后，会自动加入一些基础的指标内容，比如 HTTP 层调用次数，本机的 CPU、内存等资源的使用情况。

完成配置后，你的系统就可以对外暴露统计指标信息了。其中 Spring 中的 actuator 是一个完整的监控组件，除了收集的统计指标信息，你还可以上传服务相关的信息。合理利用这两部分信息，可以让你的系统更具有可观测性。

对外暴露统计指标信息后，你可以在代码中声明 **MeterRegistry** 这个对象，使用它声明自定义的统计指标，然后发送到 Prometheus 中统计。

你是否还记得我在“**04 统计指标：‘五个九’对系统稳定的意义？**”中介绍的指标类型？下面这段代码展现了比较常见的 4 种统计指标的声明方式。

```
meterRegistry.counter("counter").increment();


ThreadPoolExecutor threadPool = new ThreadPoolExecutor(0, 10, 0,
TimeUnit.MILLISECONDS, new LinkedBlockingDeque<>());


meterRegistry.gauge("thread_pool_active", threadPool, (pool) -> (double)
threadPool.getActiveCount());


final DistributionSummary summary = DistributionSummary.builder("summary")


   .serviceLevelObjectives(0, 10, 100)


   .publishPercentiles(0.90, 0.95)


   .register(meterRegistry);


summary.record(10);
```

Spring 也为我们提供了一些注解和第三方框架的适配，来辅助我们创建自己的指标内容，比如 Timed 注解。

PromQL

PromeQL 是在 Prometheus 查询指标数据中最重要的内容，通过它提供的数据内容你可以更方便地进行表格绘制、内容告警。

我们可以来看一段代码：

```
http_error_requests{job="apiserver"}[5m]
```

这段代码可以查询出 http_error_requests 这个统计指标中，代表了 HTTP 请求错误的次数，主体中 job 值等于 apiserver 的所有统计指标中，最近 5 分钟的数据值。我们可以通过这样的数据内容绘制柱状图或折线图。

我们可以使用很多的指标函数，比如下面代码就表示的是 http_error_requests 这个统计指标的当前值，和前 5 分钟相比较的增速值。

```
rate(http_error_requests[5m])
```

PromQL 还支持很多的计算函数，这里你可以参考官网的教程进行更细致的学习。

告警规则

告警规则就是定期执行 PromeQL 中的语句，当符合条件时进行告警。下面的代码就是创建了一个告警规则：当 HTTP 请求错误数据的错误数超过 20 个，并且持续了一分钟，就会告警。

```
- alert:RequestErrorAlert

# 表达式，代表触发什么样子的条件，这里代表了请求错误次数超过了 20 个

expr:http_requests_total{job="apiserver"} > 20

# 持续时间。 表示持续一分钟都出现请求错误数超过 20 个，则触发报警。

for: 1m

 # 当符合条件时进行告警通知信息

annotations:

#摘要，支持通过模板的形式定制当前统计指标中的信息

summary: "Instance {{ $labels.instance  }}请求错误数过高"

description: "{{ $labels.instance  }} 实例中的请求错误数超过 20 个(当前值：{{ $value }})"
```

上述代码声明了一个告警规则，接下来就可以通过 Alertmanager 组件进行更详细的通知方式配置。这部分内容十分简单，你可以通过官方提供的文档学习。

Grafana

最后我带你简单认识一下 Grafana，它是**一个通过可视化的形式查看指标数据的展示系统。**通过它，你可以快速构建出 dashboard。其数据源就是依赖 Prometheus，通过 PromQL 的查询实现的。如下图所示：

这张图是官方 demo 所提供的展示内容，可以点击这里访问。

Grafana 是依赖于 Prometheus 提供的数据搭建而成的。其中，**最上面一行分别展示了当前的展示模板和对应查询时间范围**，这个和我介绍的 Kibana 十分类似。**下方的每一个模块展示的是通过 PromQL 所查询出数据的结果**，这个部分可以选择不同的展示方式，例如柱状图、折线图、列表等。

Grafana 还支持导入和导出展示模板，你可以下载一些已经很成熟的模板信息来使用。·

总结

本节我向你介绍了 Prometheus 的功能和使用方法，它是在业务中统计指标时必不可少的系统。那么，你都是如何根据指标来进行告警的，在线上告警时又出现过哪些棘手的问题呢？欢迎你在留言区分享你的故事。