分布式链路追踪

一、APM系统

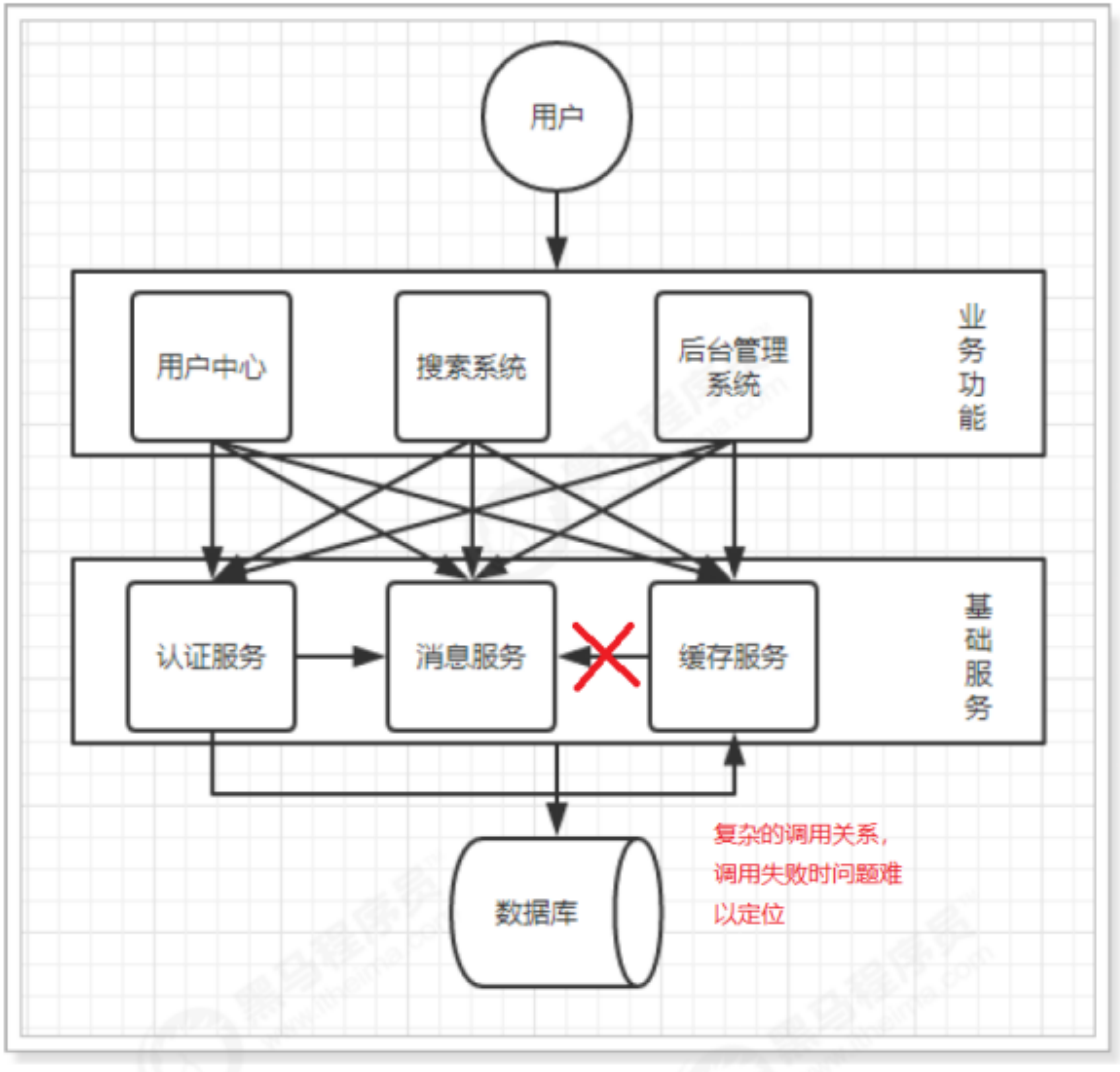
1. APM系统概述

APM (Application Performance Management) 即应用性能管理系统，是对企业系统即时监控以实现对应用程序性能管理和故障管理的系统化的解决方案。应用性能管理，主要指对企业的关键业务应用进行监测、优化，提高企业应用的可靠性和质量，保证用户得到良好的服务，降低IT总拥有成本。

APM系统是可以帮助理解系统行为、用于分析性能问题的工具，以便发生故障的时候，能够快速定位和解决问题。

2. 分布式链路追踪

随着分布式系统和微服务架构的出现，一次用户的请求会经过多个系统，不同服务之间的调用关系十分复杂，任何一个系统出错都可能影响整个请求的处理结果。以往的监控系统往往只能知道单个系统的健康状况、一次请求的成功失败，无法快速定位失败的根本原因。



除此之外，复杂的分布式系统也面临这下面这些问题：

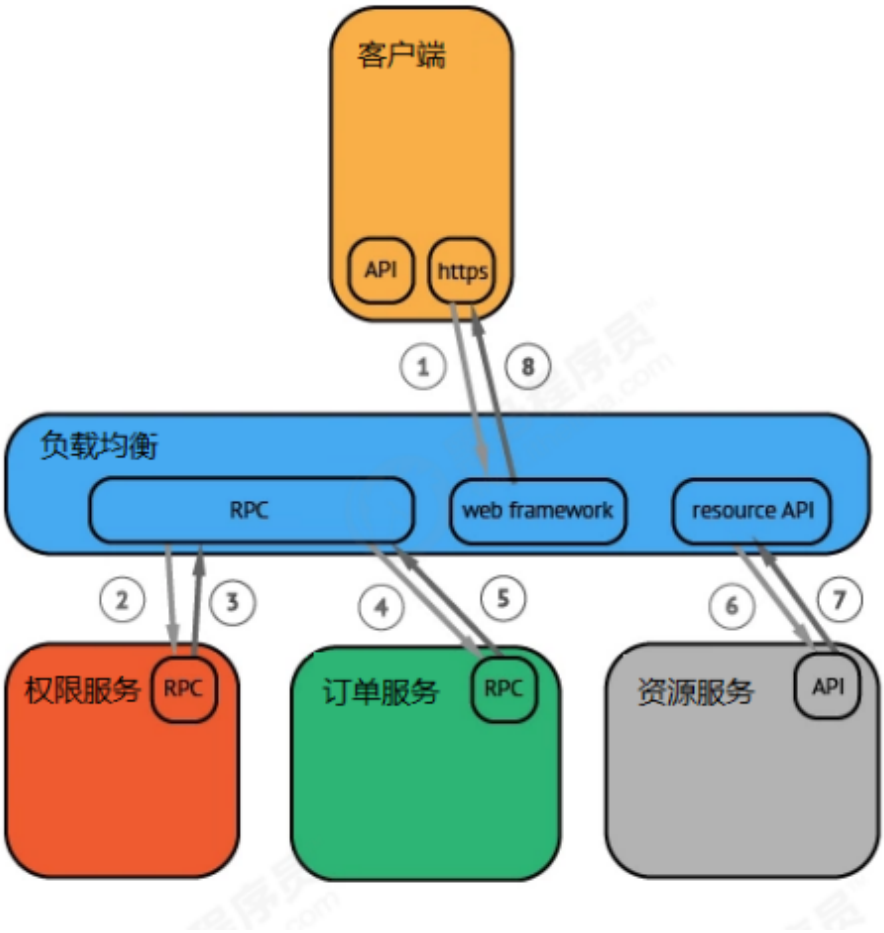
- **性能分析**：一个服务依赖很多服务，被依赖的服务也依赖了其他服务。如果某个接口耗时突然变长了，那未必是直接调用的下游服务慢了，也可能是下游的下游慢了造成的，如何快速定位耗时变长的根本原因呢？
- **链路梳理**：需求迭代很快，系统之间调用关系变化频繁，靠人工很难梳理清楚系统链路拓扑(系统之间的调用关系)。

3. OpenTrancing

（1）概述

为了解决这些问题，Google推出了一个分布式链路跟踪系统Dapper ，之后各个互联网公司都参照Dapper的思想推出了自己的分布式链路跟踪系统，而这些系统就是分布式系统下的APM系统分布式链路跟踪最先由Google在Dapper论文中提出，而OpenTracing通过提供平台无关、厂商无关的API，使得开发人员能够方便的添加（或更换）追踪系统的实现。

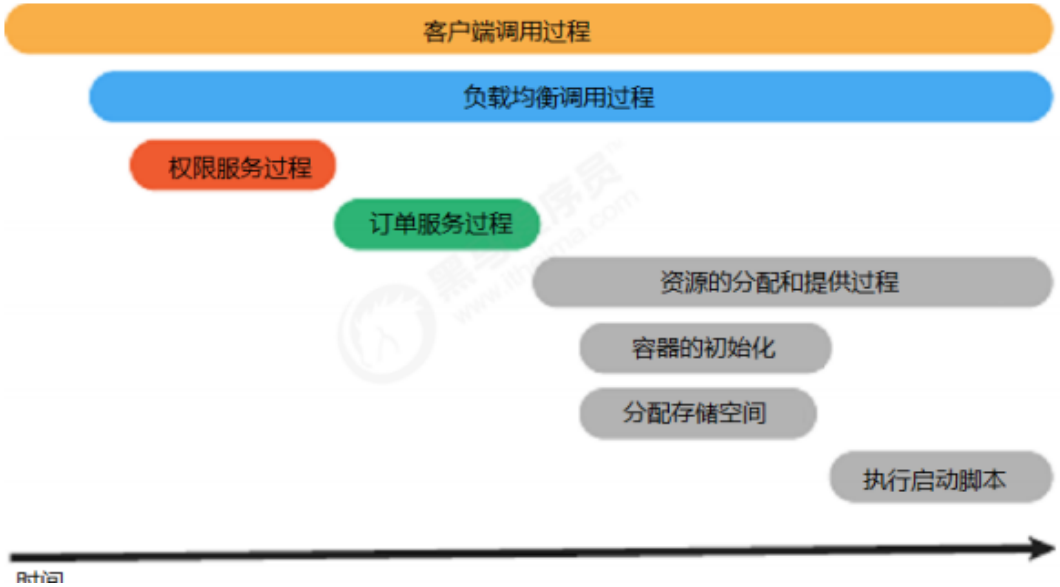
下图是一个分布式调用的例子，客户端发起请求，请求首先到达负载均衡器，接着经过认证服务，订单服务，然后请求资源，最后返回结果。



虽然这种图对于看清各组件的组合关系是很有用的，但是存在下面两个问题：

它不能很好显示组件的调用时间，是串行调用还是并行调用，如果展现更复杂的调用关系，会更加复杂，甚至无法画出这样的图。这种图也无法显示调用间的时间间隔以及是否通过定时调用来启动调用。

一种更有效的展现一个调用过程的图：



(2) Trace

客户端发起的一次请求，就可以认为是一个Trace

(3) Span

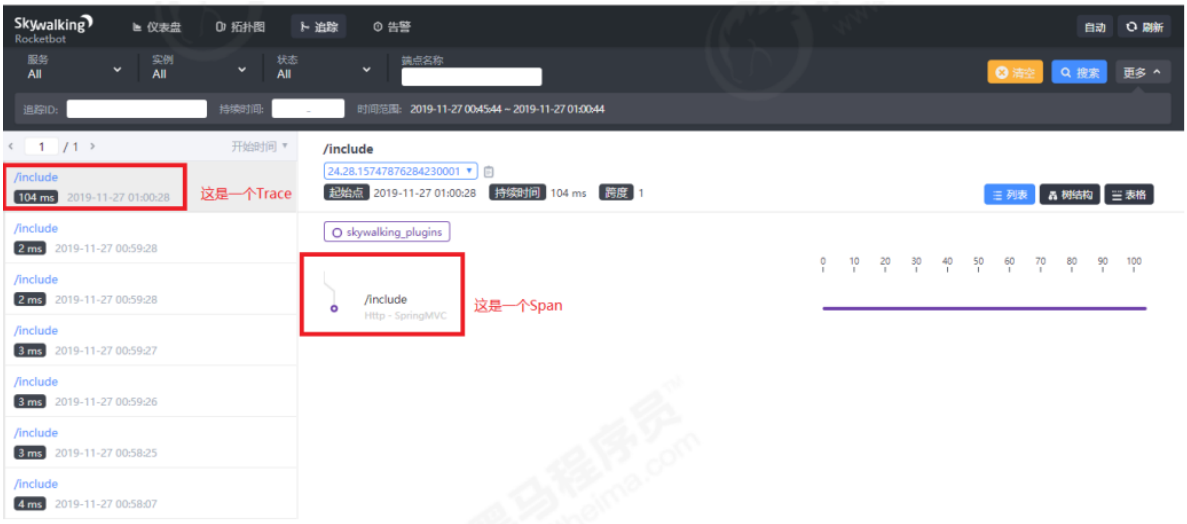
一个Span代表系统中具有开始时间和执行时长的逻辑运行单元。span之间通过嵌套或者顺序排列建立逻辑因果关系。Span里面的信息包括：操作的名字，开始时间和结束时间，可以附带多个key:value 构成的 Tags(key必须是String，value可以是 String, bool 或者数字)，还可以附带 Logs 信息(不一定所有的实现都支持)也是 key:value形式。

下面例子是一个 Trace，里面有8个 Span：

```
        [Span A]  ←←←(the root span)
            |
     +------+------+
     |             |
[Span B]      [Span C] ←←←(Span C 是 Span A 的孩子节点, ChildOf)
     |             |
[Span D]      +---+-------+
              |           |
          [Span E]    [Span F] >>> [Span G] >>> [Span H]
                                          ↑
                                          ↑
                                          ↑
                              (Span G 在 Span F 后被调用, FollowsFrom)
```

一个span可以和一个或者多个span间存在因果关系。OpenTracing定义了两种关系：ChildOf 和 FollowsFrom 。**这两种引用类型代表了子节点和父节点间的直接因果关系。**未来，OpenTracing将支持非因果关系的span引用关系。（例如：多个span被批量处理，span在同一个队列中，等等）

ChildOf 很好理解，就是父亲 Span 依赖另一个孩子 Span。比如函数调用，被调者是调用者的孩子，比如说 RPC 调用，服务端那边的Span，就是 ChildOf 客户端的。很多并发的调用，然后将结果聚合起来的操作，就构成了 ChildOf 关系。如果父亲 Span 并不依赖于孩子 Span 的返回结果，这时可以说它他构成 **FollowsFrom** 关系



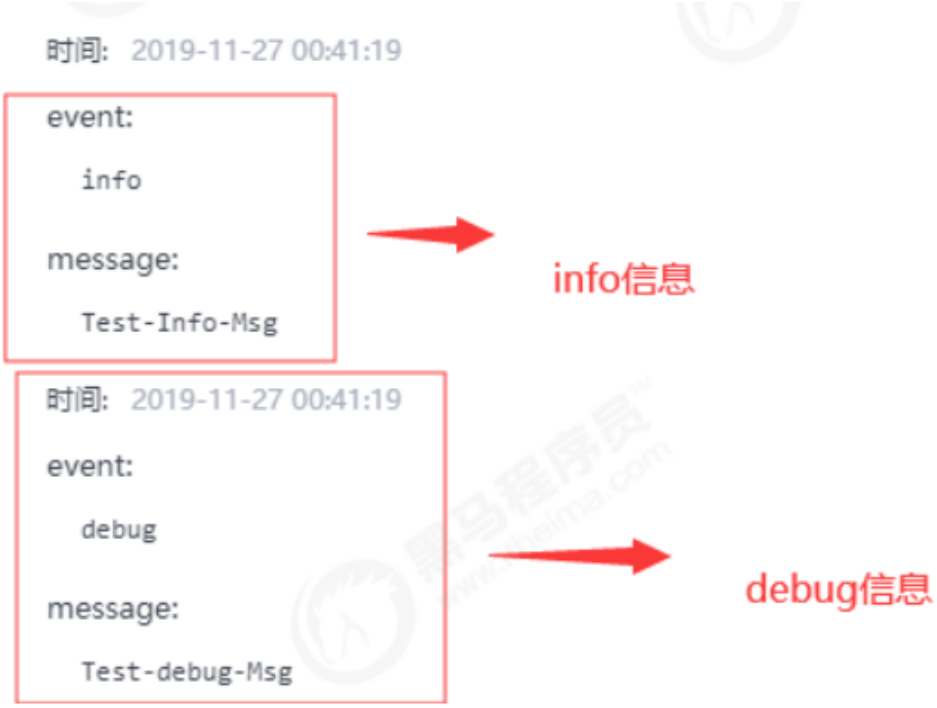
如图所示，左边的每一条追踪代表一个Trace，而右边时序图中每一个节点就是一个Span

（4）Logs

每个span可以进行多次**Logs**操作，每一次**Logs**操作，都需要一个带时间戳的时间名称，以及可选的任意大小的存储结构。

如下图是一个异常的Log：

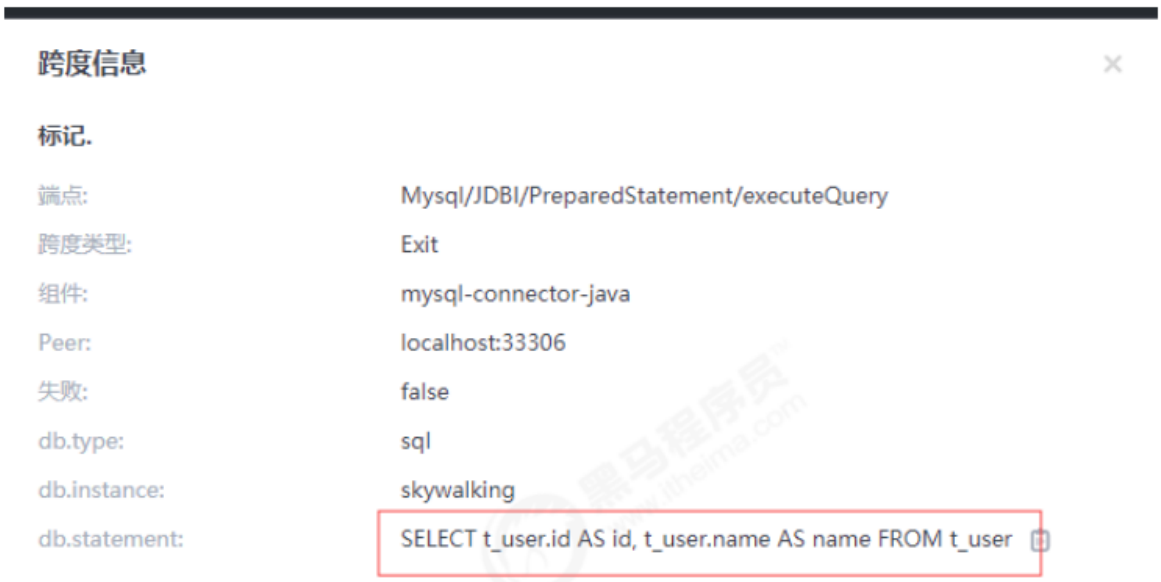
如下图是两个正常信息的Log，它们都带有时间戳和对应的事件名称、消息内容。



（5）Tags

每个span可以有多个键值对（key:value）形式的**Tags**，**Tags**是没有时间戳的，支持简单的对span进行注解和补充。

如下图就是一个Tags的详细信息，其中记录了数据库访问的SQL语句等内容。



4. 主流的开源APM产品

- SkyWalking

SkyWalking是apache基金会下面的一个开源APM项目，为微服务架构和云原生架构系统设计。它通过探针自动收集所需的指标，并进行分布式追踪。通过这些调用链路以及指标，Skywalking APM会感知应用间关系和服务间关系，并进行相应的指标统计。Skywalking支持链路追踪和监控应用组件基本涵盖主流框架和容器。

官方网站：http://skywalking.apache.org/

- ZipKin

Zipkin是由Twitter开源，是分布式链路调用监控系统，聚合各业务系统调用延迟数据，达到链路调用监控跟踪。Zipkin基于Google的Dapper论文实现，主要完成数据的收集、存储、搜索与界面展示。

官方网站：https://zipkin.io/

- 其他
    - 阿里巴巴鹰眼(EagleEye)
    - 美团CAT
    - 京东Hydra
    - Pinpoint(APM)

二、SkyWalking

1. 概述

根据官方的解释，Skywalking是一个可观测性分析平台（Observability Analysis Platform简称OAP）和应用性能管理系统（Application Performance Management简称APM）。

提供分布式链路追踪、服务网格(Service Mesh)遥测分析、度量(Metric)聚合和可视化一体化解决方案。

下面是Skywalking的几大特点：

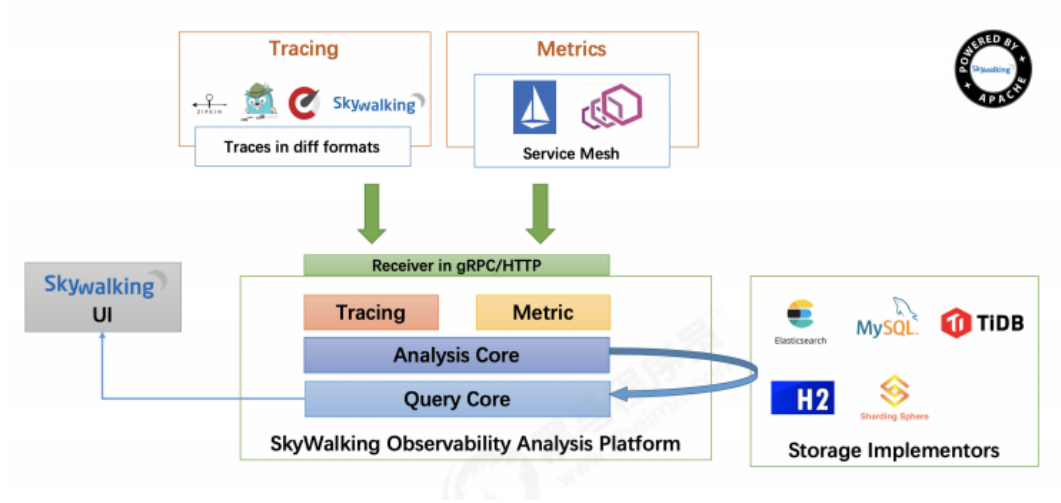
- 多语言自动探针，Java，.NET Core和Node.JS。
- 多种监控手段，语言探针和service mesh。
- 轻量高效。不需要额外搭建大数据平台。
- 模块化架构。UI、存储、集群管理多种机制可选。
- 支持告警。
- 优秀的可视化效果。

2. 优势

Skywalking相比较其他的分布式链路监控工具，具有以下特点：

- 社区相当活跃
- Skywalking支持Java，.NET Core和Node.JS语言。相对于其他平台：比如Pinpoint支持Java和PHP,具有较大的优势。
- 探针无侵入性。不修改原有项目一行代码就可以进行集成。
- 探针性能优秀。有网友对Pinpoint和Skywalking进行过测试，由于Pinpoint收集的数据过多，所以对性能损耗较大，而Skywalking探针性能十分出色。
- 支持组件较多。特别是对Rpc框架的支持，这是其他框架所不具备的。Skywalking对Dubbo、gRpc等有原生的支持，甚至连小众的motan和sofarpc都支持。

3. 整体架构



整体架构包含如下三个组成部分：

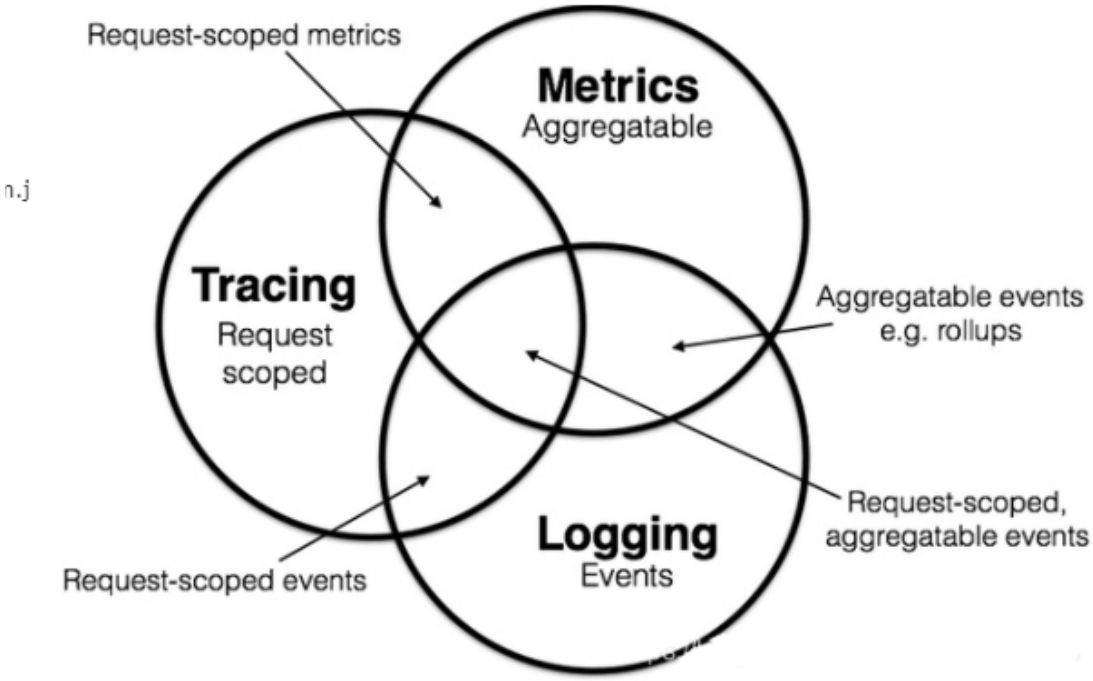
- **探针**(agent)：负责进行数据的收集，包含了Tracing和Metrics的数据，agent会被安装到服务所在的服务器上，以方便数据的获取。
- **可观测性分析平台OAP**(Observability Analysis Platform)：接收探针发送的数据，并在内存中使用分析引擎（Analysis Core)进行数据的整合运算，然后将数据存储到对应的存储介质上，比如Elasticsearch、MySQL数据库、H2数据库等。同时OAP还使用查询引擎(Query Core)提供HTTP查询接口。
- **UI**： Skywalking提供单独的UI进行数据的查看，此时UI会调用OAP提供的接口，获取对应的数据然后进行展示。

4. 跟踪、记录和度量

微服务领域很早因为追踪、日志和指标相辅相成，合力支持多维度、多形态的监控体系，三类监控各有各的类型：

- **跟踪：**它在单次请求的范围内，处理信息。任何的数据、元数据都被绑定到系统中的信息的各个事务上。例如：一次调用远程服务的RPC执行过程；一次实际的SQL查询语句；一次HTTP请求的业务性ID；
- **日志记录：**不知道大家有没有监控信息等的定义或记录。等等这类数据记录的事件，大部分情况下记录的数据很可能是错误信息，或者是记录当前的情况，或者是当前的情况的警告信息。
- **Metrics：**我们想知道 QPS 是多少服务，或者当日的用户登录次数等等，我们可能需要将某个事件进行聚合或计数，也就是说的 Metrics。特征，它们是服务内的例子（请求或者直方图）或者是元。简单的加法聚合，当持续一段时间我们又可以为直方图建模。

三者关系图



4. 环境搭建

skywalking/docker-compose.yml

```yaml
version: '3.3'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.5.0
    container_name: elasticsearch
    restart: always
    ports:
      - 9200:9200
    environment:
      - discovery.type=single-node
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms256m -Xmx256m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
  oap:
    image: apache/skywalking-oap-server:8.8.0
    container_name: oap
    depends_on:
      - elasticsearch
    links:
      - elasticsearch
    restart: always
    ports:
      - 11800:11800
      - 12800:12800
```

```
    environment:
      SW_STORAGE: elasticsearch
      SW_STORAGE_ES_CLUSTER_NODES: elasticsearch:9200
  ui:
    image: apache/skywalking-ui
    container_name: ui
    depends_on:
      - oap
    links:
      - oap
    restart: always
    ports:
      - 8080:8080
    environment:
      SW_OAP_ADDRESS: http://oap:12800
```

检查是否部署成功

Skywallking：http://192.168.200.104:8080/

ES：http://192.168.200.104:9200/

5. 对接单个.Net程序

（1）添加依赖

```
<ItemGroup>
    <PackageReference Include="SkyAPM.Agent.AspNetCore" Version="1.3.0" />
</ItemGroup>
```

（2）配置文件`skyapm.json`

```
{
  "SkyWalking": {
    "ServiceName": "SkyWalking_Order",//服务名称，必须与环境变量保持一致
    "Namespace": "",
    "HeaderVersions": [
      "sw8"
    ],
    "Sampling": {   //采样配置节点
      "SamplePer3Secs": -1,     //每3秒采样数
      "Percentage": -1.0        // 采样百分比，例如10%采样则配置为10
    },
    "Logging": {    //日志配置节点
      "Level": "Information",   //日志级别
      "FilePath": "logs\\skyapm-{Date}.log" //日志保存路径
    },
    "Transport": {  //传输配置节点
      "Interval": 3000,         //每多少毫秒刷新
      "ProtocolVersion": "v8",
```

```
      "QueueSize": 30000,
      "BatchSize": 3000,
      "gRPC": {          gRPC配置节点
        "Servers": "192.168.200.104:11800", //Skywalking服务地址，生产环境替
需替换成生产skyapm发布后的地址，多个用逗号“,”
        "Timeout": 10000,   //创建gRPC链接的超时时间，毫秒
        "ConnectTimeout": 10000,    //gRPC最长链接时间，毫秒
        "ReportTimeout": 600000,
        "Authentication": ""
      }
    }
  }
}
```

说明：

如何生成skyapm.json文件

- 安装CLI（SkyAPM.DotNet.CLI）

```
 dotnet tool install -g SkyAPM.DotNet.CLI
```

- 自动生成skyapm.json文件

```
dotnet skyapm config [service name] [server]:11800
 eg: dotnet skyapm config MySkyWalking_OrderService 192.168.3.245:11800
```

server name指的就是您刚才配置的SKYWALKING__SERVICENAME

server指的是您Skywalking的ip地址。

执行命令后，会自动生成一个skywalking.json 。

（3）启动文件配置SK`launchSettings.json`

```
"profiles": { // 项目
    "IIS Express": { // IIS部署项
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "weatherforecast",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development",
        "ASPNETCORE_HOSTINGSTARTUPASSEMBLIES": "SkyAPM.Agent.AspNetCore",
        "SKYWALKING__SERVICENAME": "SkyWalking_Order"
      }
    },
    "SkyWalkingDemo": { // Kestrel部署项
      "commandName": "Project",
      "launchBrowser": true,
```

```
      "launchUrl": "weatherforecast",
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development",
        "ASPNETCORE_HOSTINGSTARTUPASSEMBLIES": "SkyAPM.Agent.AspNetCore",
// 必须配置
        "SKYWALKING__SERVICENAME": "SkyWalking_Order" // 必须配置，在
skywalking做标识
      }
    }
  }
```

6. 对接微服务网关+后台微服务

三、Zipkin