从这篇文章开始，我们开始学习一个新的领域 APM，也是字节码技术成功应用的典型案例。将分为两大部分：APM 的基础概念和分布式跟踪的理论基础。

0x01 什么是 APM

APM 是 Application Performance Managment 的缩写，字面意思很容易理解，"应用性能管理"，它是由 Gartner 归纳抽象出的一个管理模型。近年来 APM 行业被越来越多的企业所关注，尤其是在 2014 年末，NewRelic 的成功上市，更加激发了人们对这个行业前景的无限遐想。国内崛起的听云、OneAPM，以及最近微信和 360 团队刚开源的安卓端 APM，使 APM 遍地开花。

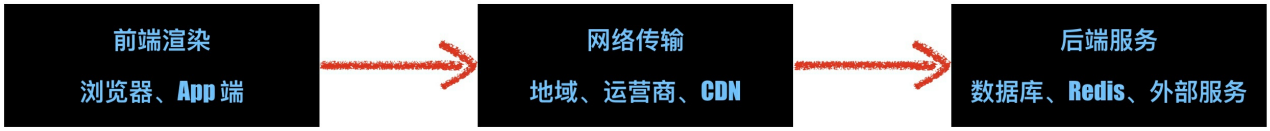
0x02 我们为什么需要 APM

影响用户体验的三大环节：

- 前端渲染
    - 页面加载时间
    - DOM 处理时间
    - 页面渲染时间
    - 首屏加载时间
- 网络传输

    - DNS 解析时间
    - TCP 建连时间
    - 网络传输时间
    - SSL 握手时间

- 后端服务
    - SQL 执行时间
    - 缓存读写时间
    - 外部服务调用时间



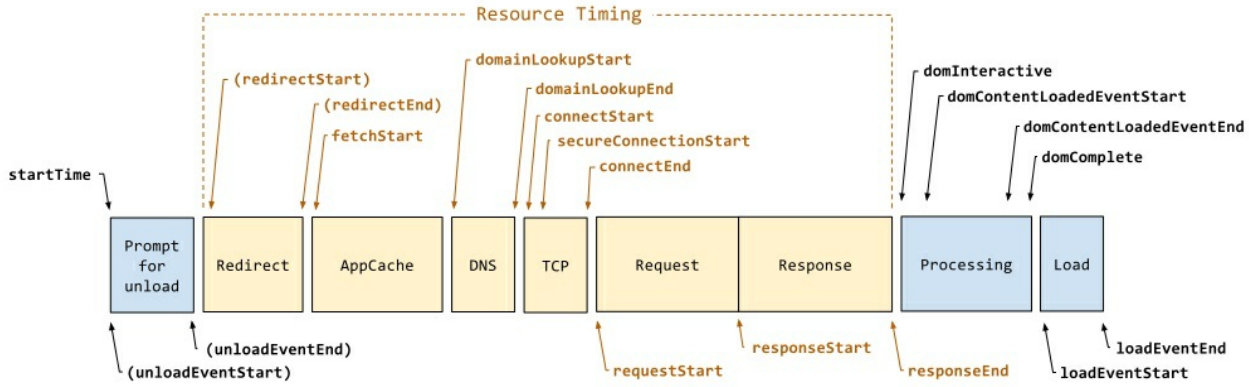
每一个环节都有可能有性能的问题，我们需要做的是把性能做到**可度量、指标化**

0x03 我们需要做什么

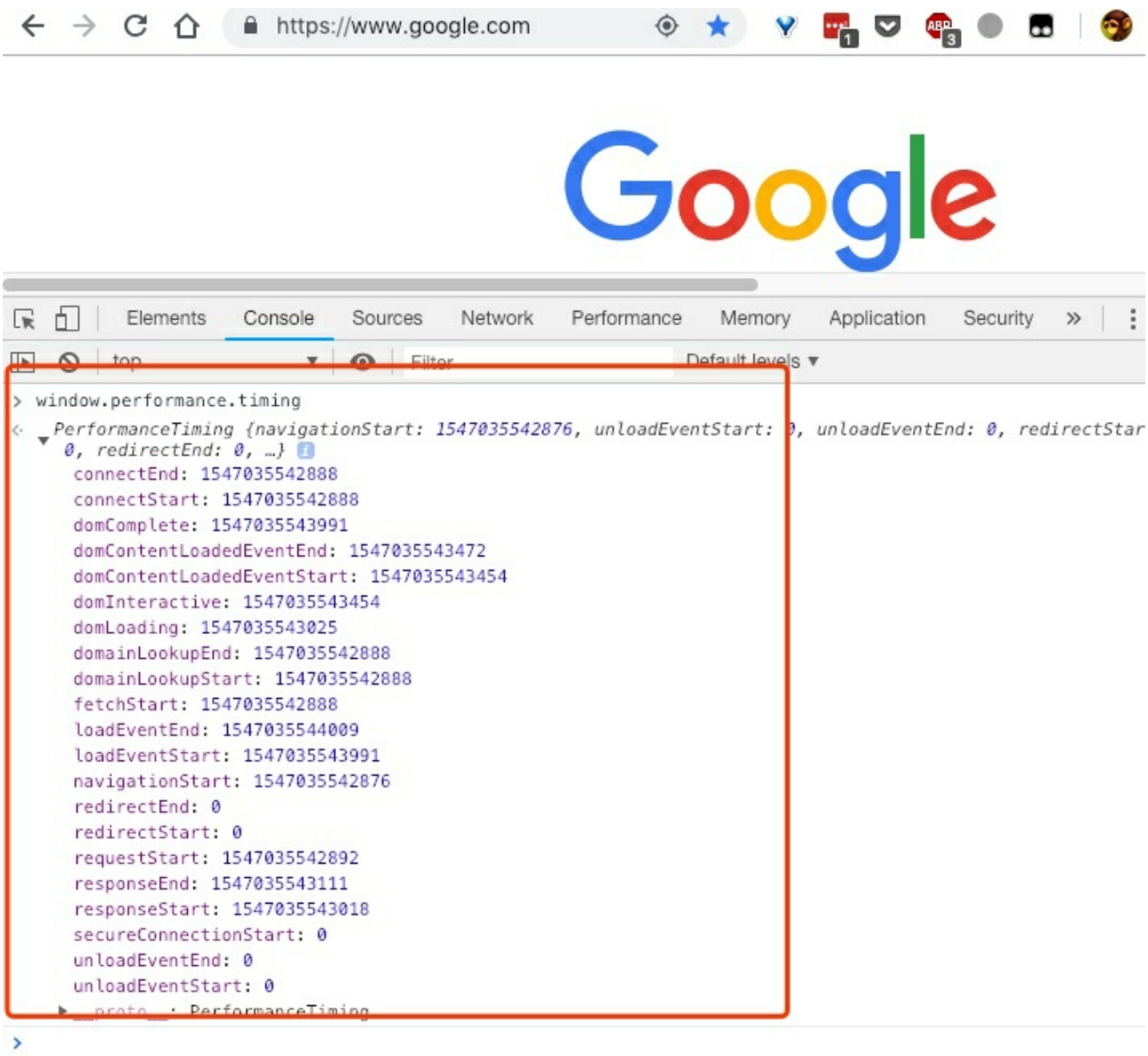
针对我们刚提到的三大环节，我们可以针对性的来看下每个环节我们可以做些什么

浏览器端

浏览器的页面加载过程如下



我们可以通过这些过程拿到非常关键的业务指标：页面加载时间、首屏时间、页面渲染时间
我们在 chrome console 里输入 `window.performance.timing` 就可以拿到详细的各阶段时间



服务端 APM

假设有这样一个函数，我们需要进行监控

```
public void saveUser() {
    doDbOperation();
    flushCache();
}
```

我们需要对它的字节码进行改写，自动注入一些代码达到监控的功能，一个最简单的模型如下面的代码所示

```
public void _saveUser() {
    // 获取开始时间
    long start = System.currentTimeMillis();
    // 记录未捕获异常
    Throwable uncaughtException = null;

    try {
        doDbOperation();
        flushCache();
    } catch (Throwable e) {
        uncaughtException = e;
        throw e;
    } finally {
        // 记录结束时间
        long end = System.currentTimeMillis();
        // 上报 spanName、开始时间、结束时间、是否有未捕获的异常
        APMUtil.report("UserService.saveUser", start, end, uncaughtException);
    }
}
```

0x04 怎么样做嵌码?

- Java 服务端：使用我们之前介绍过的 javaagent 字节码

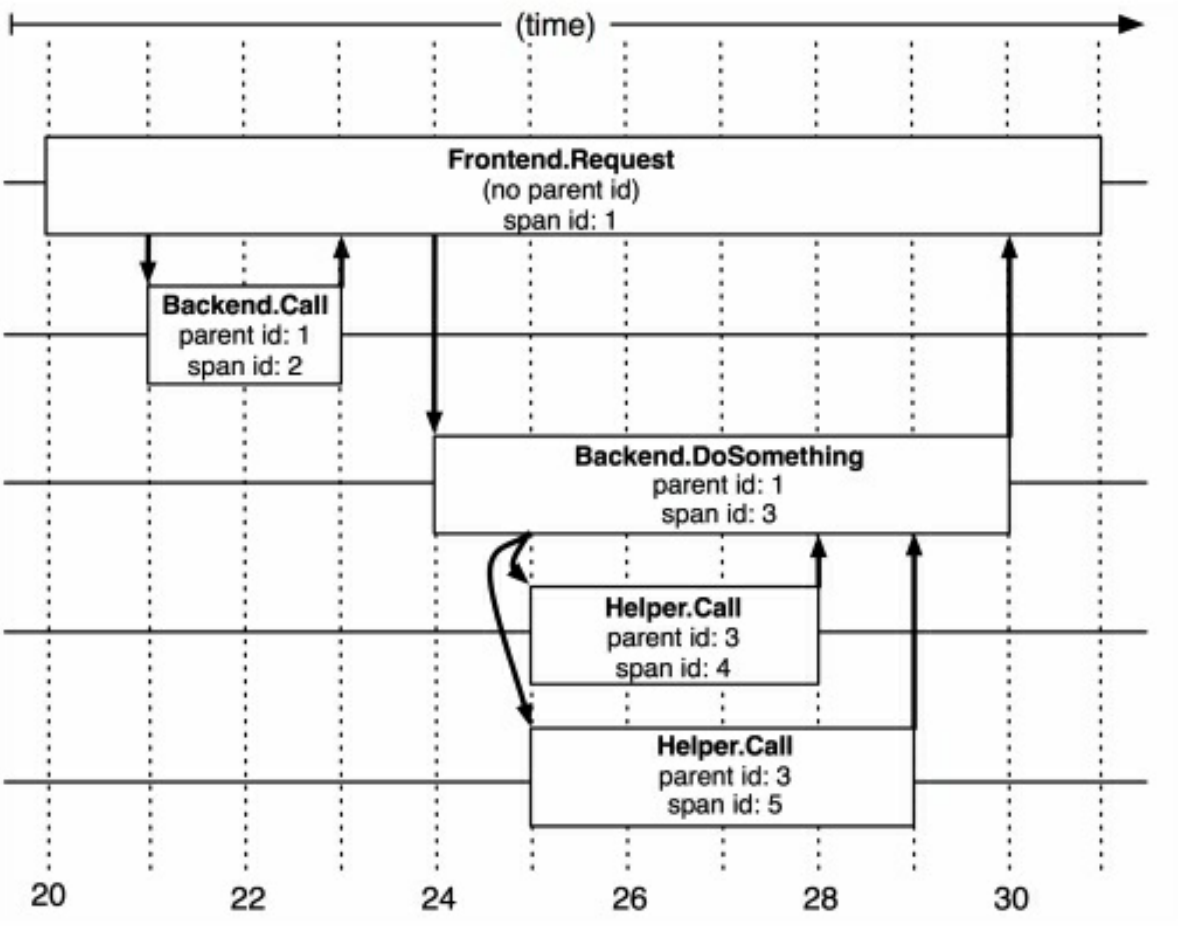
instrument 技术进行字节码改写
- Node.js 阿里有开源 pandora.js 可以参考借鉴
- 安卓：用 gradle 插件在编译期进行 hook
- iOS：Hook（Method Swizzling）

我们后面会着重介绍 Java 服务端 APM 如何来实现跨进程的调用链路跟踪监控

接下来我们将讲解分布式跟踪相关的内容，将从单 JVM 到扩进程如何实现链路调用来讲解原理与实现。

0x05 分布式跟踪理论基础

参考 Google Dapper (https://ai.google/research/pubs/pub36356) 论文实现，每个请求都生成全局唯一的 Trace ID / Span ID，端到端透传到上下游所有的节点，通过 Trace ID 将不同系统的孤立的调用日志和异常日志串联在一起，同时通过 Span ID、ParentId 表达节点的父子关系，如下图所示

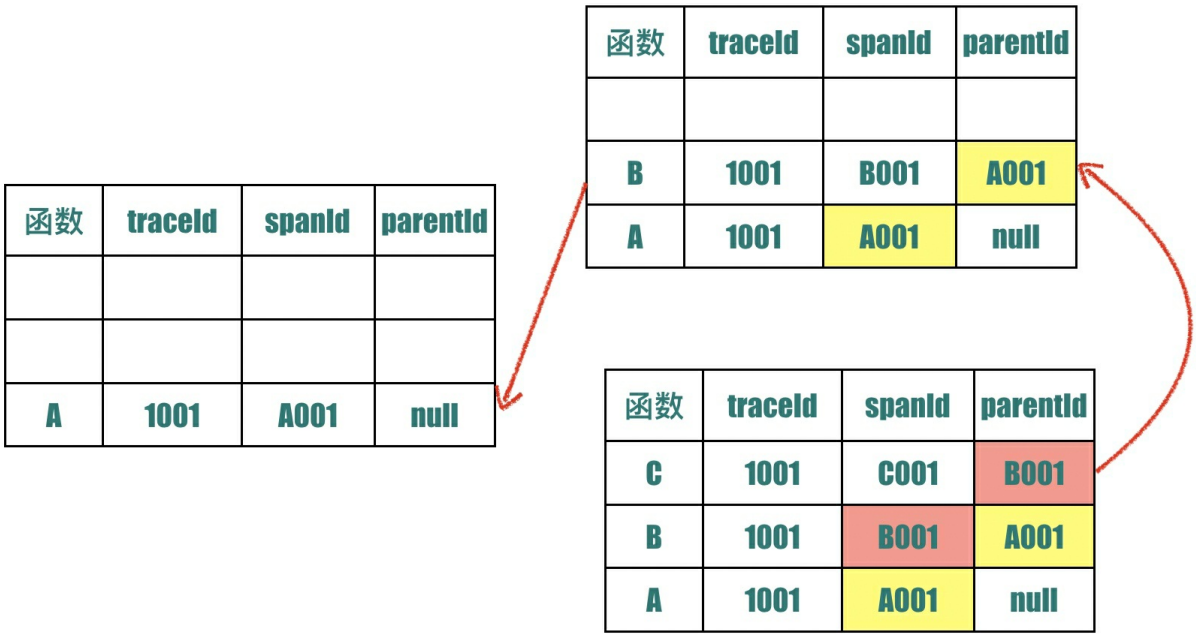


0x06 单 JVM 调用链路跟踪实现原理

在 Java 中，我们可以很方便的用 ThreadLocal 的 Stack 的实现调用栈的跟踪，比如有如下的调用关系

```
void A() {
    B();
}
void B() {
    C();
}
void C(){
}
```

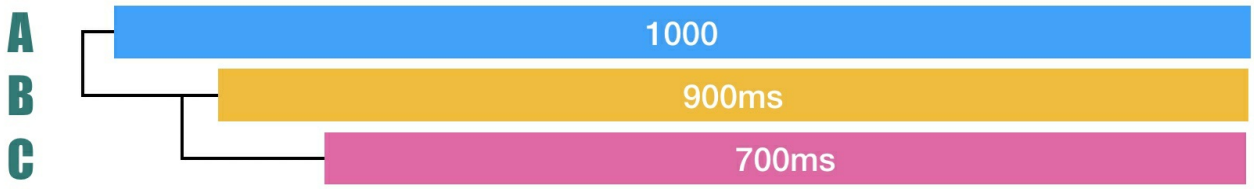
我们约定：spanId 做为当前调用id，parentId 做为父调用 id，traceId 做为整条链路 id



那么我们调用上报的 trace 信息大概如下

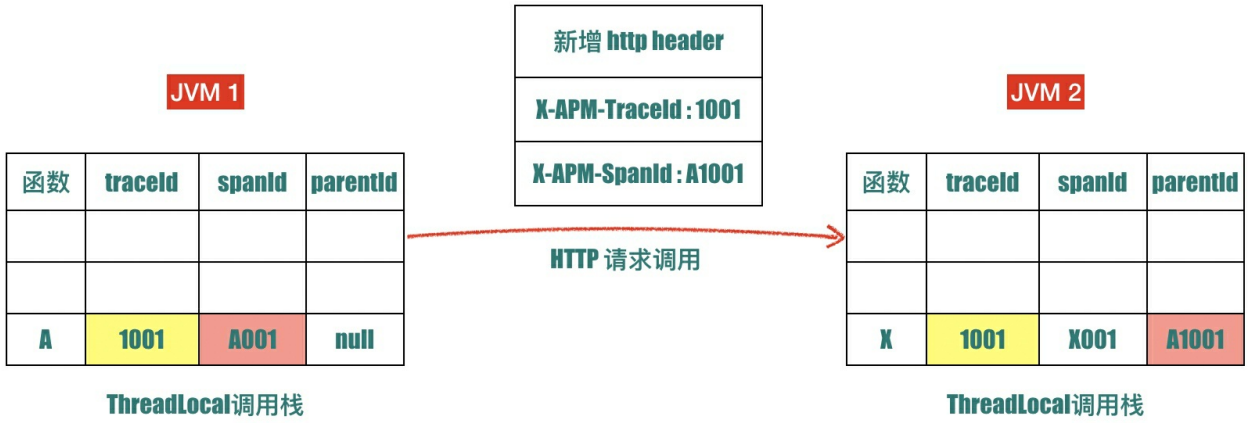
```
[
  {
    "spanName": "A()",
    "traceId": "1001",
    "spanId": "A001",
    "parentId": null,
    "timeCost": 1000,
    "startTime": 10001,
    "endTime": 11001,
    "uncaughtException": null
  },
  {
    "spanName": "B()",
    "traceId": "1001",
    "spanId": "B001",
    "parentId": "A001",
    "timeCost": 900,
    "startTime": 10001,
    "endTime": 11001,
    "uncaughtException": null
  },
  {
    "spanName": "C()",
    "traceId": "1001",
    "spanId": "C001",
    "parentId": "B001",
    "timeCost": 800,
    "startTime": 10001,
    "endTime": 11001,
    "uncaughtException":
"java.lang.RuntimeException"
  }
]
```

通过 traceId、spanId、parentId 三者的数据，我们可以很方便的构建出一个完整的调用栈



0x07 扩进程、异构系统的调用链路跟踪如何处理?

只需要把 traceId 和 spanId 传递到下一层调用就好了。比如我们采用 HTTP 调用的方式调用另外一个 JVM 的服务。
在 JVM 1 中在 HTTP 调用前调用相应 setHeader 函数新增 X-APM-TraceId 和 X-APM-SpanId 两个 header。
JVM 2 收到请求以后，会先去检查是否有这两个 header，如果没有这两个 header，说明它自己是最顶层的调用。如果有这两个 header的情况下，会把 header 中的 traceId 当做后续调用的 traceId，header 中的 spanId 做为当前调用的 parentId。如下图所示



Duboo 等 RPC 调用同理，只是参数传递的方式有所不同。

0x08 小结

这篇文章我们讲解了 APM 的基本概念，主要内容小结如下：第一，APM 的含义是"应用性能管理"，近年来 APM 行业被越来越多的企业所关注。第二，谈到影响用户体验的三大环节：前端渲染、网络传输、后端处理，以及为了提高用户体验每一步我们可以做什么使得性能可以做到**可度量**、**指标化**。第三，介绍了常见的嵌码技术，帮助以最小的接入成本进行性能监控管理。
第四，讲了基于 Google dapper 理论的分布式系统跟踪的原理，主要分了两块：第一，单进程内调用链路跟踪如何实现，第二，跨进程、异构系统的调用链路如何实现。

0x09 思考

留一道作业：

1. 你可以用 ASM 实现把上面 saveUser() 改写为 _saveUser() 吗?
2. 前面讲到 HTTP 跨进程调用可以用新增 header 的方式来注入 traceId 和 spanId，那么 Dubbo 的跨进程跟踪应该用怎么样的方式把 traceId、spanId 传给被调用方呢?

欢迎你在留言区留言，和我一起讨论。