



下载APP



## 24 | 动态监控：你的产品系统中有动态监控的能力吗？

2021-07-10 尉刚强

《性能优化高手课》

课程介绍 &gt;



讲述：尉刚强

时长 15:13 大小 13.94M



你好，我是尉刚强。今天，我们来讨论下与系统动态监控相关的能力和配置思路。

对于从事 Java 语言开发的程序员来说，应该基本上都使用过 Slf4j+Logback 来打印日志。而对于一些软件性能要求比较高的系统来说，为了减少日志打印对性能的影响，默认情况下，系统一般只会打开 WARN 或 ERROR 级别的日志。

那么，如果软件系统在执行的过程中出现了一些异常情况，需要做进一步的分析，想要打开 INFO 级别的日志，该怎么办呢？其实这个时候，你就需要在软件执行期间，实现动态修改日志打印级别的能力。



这里我们以 Logback 配置为例，其修改打印日志级别的代码如下所示：

```
1 LoggerContext loggerContext =  
2 (LoggerContext) LoggerFactory.getILoggerFactory();  
3 Logger logger = loggerContext.getLogger("root");  
4 ((ch.qos.logback.classic.Logger)logger).setLevel(Level.valueOf(logLevel));
```

当然，你也可以给软件提供一个外部接口（比如 REST 接口），然后在接口实现中去调用以上代码来修改打印级别，其实这就是一个比较简单的动态监控手段。

所以在今天这堂课上，我会从互联网应用服务领域出发，带你分析在软件系统的开发过程中，都应该扩展实现哪些动态监控的机制和能力，以及剖析在系统中实现动态监控配置的过程，从而帮助你系统化地理解和认识动态监控的手段和能力。这样，你在参与开发高性能软件系统时，就知道如何设计并实现动态监控机制来支撑对软件的性能调优了。

好，那么在开始之前，我们先来深入思考一下，为什么软件需要这种动态监控的能力呢？

## 为什么需要动态监控的能力？

在上节课，我们学习了系统的四种监控观测手段：计数器、跟踪、剖析、监视。但实际上，**在软件业务的实现过程中，你每增加一种观测手段，都会给软件执行带来额外的性能开销。**

我举个简单的例子，假设你在关键业务流程中增加了一个计数器统计，并且记录到分布式缓存 Redis 中，那么可能就会造成对应的业务处理时延增加接近 0.5ms 的时间（真实性能优化项目中的测试结果，仅供参考）；如果你在代码实现中增加了一条日志打印需求，也很有可能会给软件额外增加了一条 IO 请求操作，从而就会对业务性能造成影响（这里你其实可以将日志写入模式配置成异步模式，来减少日志打印的性能开销）。

在有些应用软件系统实现中，日志打印甚至会造成应用的性能下降 20%，但这与具体业务代码实现相关性非常大，所以数字借鉴意义不大，但也在一定程度上可以说明日志打印对应用性能的影响比较大。

也正是因为观测手段实现都会存在性能开销，所以在给一个高性能软件系统进行深度优化时，观测手段的实现与业务性能之间的冲突就会越来越大。

而这个时候，给软件添加动态监控的机制，就成为了解决这个冲突的重要手段。

接下来，我们就通过一个日志打印的观测案例，来进一步了解下采用动态监控机制的优势。

我之前设计实现过一个数据实时同步服务，业务中需要分析数据同步时出现较大时延抖动的原因。在早期的实现版本中，我主要是依赖打印日志中记录的时间值来进行分析，但是当软件处于高负载模式运行时，就会发现打印日志大量丢失，从而完全不能支撑业务分析。

所以这个时候，我采用了将数据同步的关键时延信息批量写入到数据库中的方式，从而有效支撑了高负荷模式下的性能分析工作。

实际上，这个案例主要反映了一个比较常见的现象，那就是**当系统处于高负荷模式时，很有可能导致业务添加的观测手段失效，从而影响到软件业务的性能分析**。那么针对这种情况，你可以通过动态控制和监控观测数据的获取，来规避和解决这类问题。

总之，在开发和监控软件性能的过程中，我们不能忽视因观测手段实现而引入的额外性能开销，并且还要保证软件业务在高负载模式下，实现的监控观测手段的可用性。所以，设计并实现动态监控的机制手段，就是在开发高性能软件系统时，非常重要的一项工作。

那么对于一个高性能的软件系统来说，都应该实现哪些动态监控手段呢？下面我们就具体来看看。

## 高性能系统应该实现哪些动态监控能力？

首先我们来想一下，**计数器需要动态监控吗？**

答案其实是需要的。我们都知道，计数器是一个非常重要的监控统计手段，很多与业务相关的性能指标都可以基于计数器来呈现。

在传统的应用和嵌入式系统开发过程中，计数器一般是直接记录到内存中；而在互联网分布式系统中，则通常会选择将计数器保存到数据库或缓存中。那么针对计数器记录到数据库的这种场景，有关计数器的操作开销就不能再被忽视了。

所以，你可能就需要去设计实现计数器的动态监控机制。

其实，对于很多 ToC（面向消费者业务）的互联网服务和产品来说，在针对某个用户业务进行性能分析时，很可能就需要用户级别的统计信息（计数器）。但从系统实现的角度来看，如果针对所有用户都增加统计计数器的话，势必带来的性能开销会比较大。

那么这个时候，你就可以实现仅针对特定用户的计数器统计信息，然后在软件执行的过程中动态修改监控用户，来开启对特定用户的计数器统计。

注意：在具体的业务中，你可能需要设计出大小不同的、多种粒度的计数器，来支持其动态启动和关闭。

OK，我们再来看看对于**日志打印**，是否如这节课开头所说，**只需要动态配置日志打印级别就可以了？**

其实也不是，这种跟踪分析业务流程的观测手段，其动态监控机制的设计与实现，主要还是**取决于软件系统对性能的要求是否苛刻**。

这里我就给你举几个例子，来带你分析下我之前参与的高性能软件系统中，是如何实现日志打印的。如此你就会明白，原来基于日志的动态监控还可以有那么多的变化。

首先，日志打印还可以按照模块进一步划分，这样我们就可以从**模块**和**日志打印级别**两个维度，来更精确地配置打印信息了。

不过你要注意，在很多高性能的软件系统中，使用模块和日志级别两个层级配置的打印，再打开日志后可能还是会瞬间给软件引入比较大的 IO 负载开销。因此针对这种场景，你可以控制每次打印日志的条数，比如通过代码控制每次打印确定数目的日志信息（如 100 条），或者通过代码控制只打印特定流程下的日志信息。


当然，虽然使用这种控制打印日志条数的实现方式，可以将打印日志对软件系统带来的额外开销降到比较低，但也很有可能因为打印日志信息有限，不利于有些性能问题的分析。

比方说，你通过日志打印寻找定位慢查询的请求相关信息，但是每次只能打印 100 条日志，还没有捕获到慢查询就已经打印结束了。所以，在对性能要求非常苛刻，而且又需要

打印日志信息比较丰富的场景中，你就需要定时化手段来实现日志信息的打印了，这样才能在最小性能损耗的情况下获取更多的日志打印信息。

那么具体该怎么做呢？

我们先来看一条，Logback 配置中的打印日志的模板配置，如下所示：

 复制代码

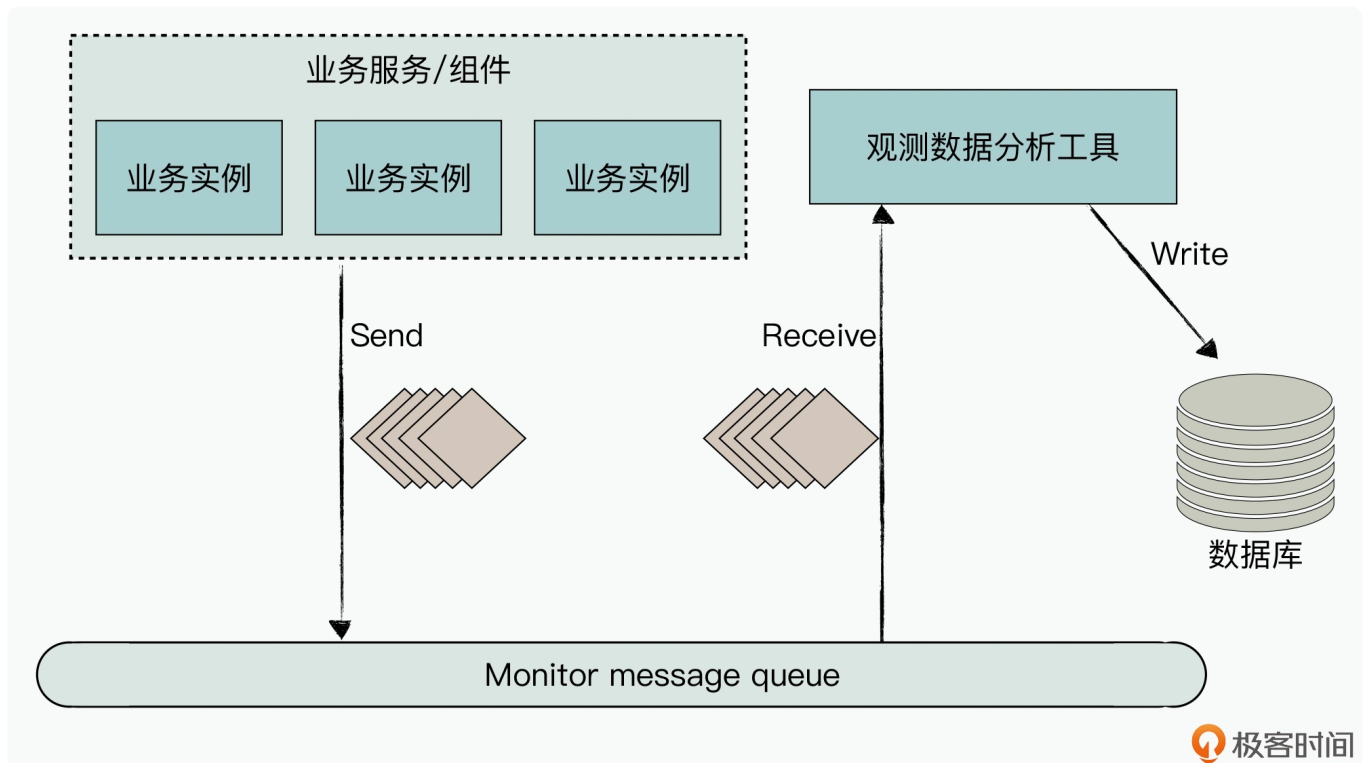
```
1 <pattern>%date %level [%thread] %logger{10} [%file:%line] %msg%n</pattern>
```

在上面的配置中，%date 代表打印时间，%level 代表日志级别。%thread 代表日志打印所在线程名，%file 代表打印日志所在文件，%line 代表打印所在行，%msg% 代表包含了参数的消息内容。

你认真思考就会发现，如果针对每条打印日志都进行统一编号，那就可以通过这个编号来获取日志级别、线程名、所在文件、代码行等多类信息了。显然，保存和传递编号的数据量会小很多。

所以说，采用标准配置模式来打印完整的日志信息，其实并不是性能最高效的实现方式。

实际上，在一些性能要求非常苛刻的业务系统中，更高效的是通过动态监控的手段来定制化实现日志打印，只把错误码、参数信息等批量记录下来，然后通过单独消息通道发送给观察分析子系统，具体如下图所示：



而且，除了打印日志之外，还有很多动态观测数据都可以通过上述消息通道的方式，来传递给外部的观察分析工具，比如在 [第 8 讲](#) 中，提到的 UDT 技术或者自定义的监控数据等。另外，我们还可以将这些观测数据持久化写入到数据库中，以支持更长时间维度上的观测数据分析。

补充：不同的软件系统对处理时延性能的敏感度是不一样的，有的系统性能优化甚至需要关注到微秒级别（1 秒 = 1000000 微秒），在这种场景下增加一条打印日志都有可能对软件性能产生影响。所以，是否可以接受监控数据对性能的影响，是与业务强相关的。

其实到这里，我只介绍了动态监控技术中的很少一部分实现策略，而且也不是说所有的高性能系统中都必须要这样去实现动态监控的机制。你要知道，**不同软件系统所需要的观测手段是存在差异的，而且它们对性能的敏感度是不一样的**，所以你还针对具体的业务场景，来实现相应的动态监控手段。

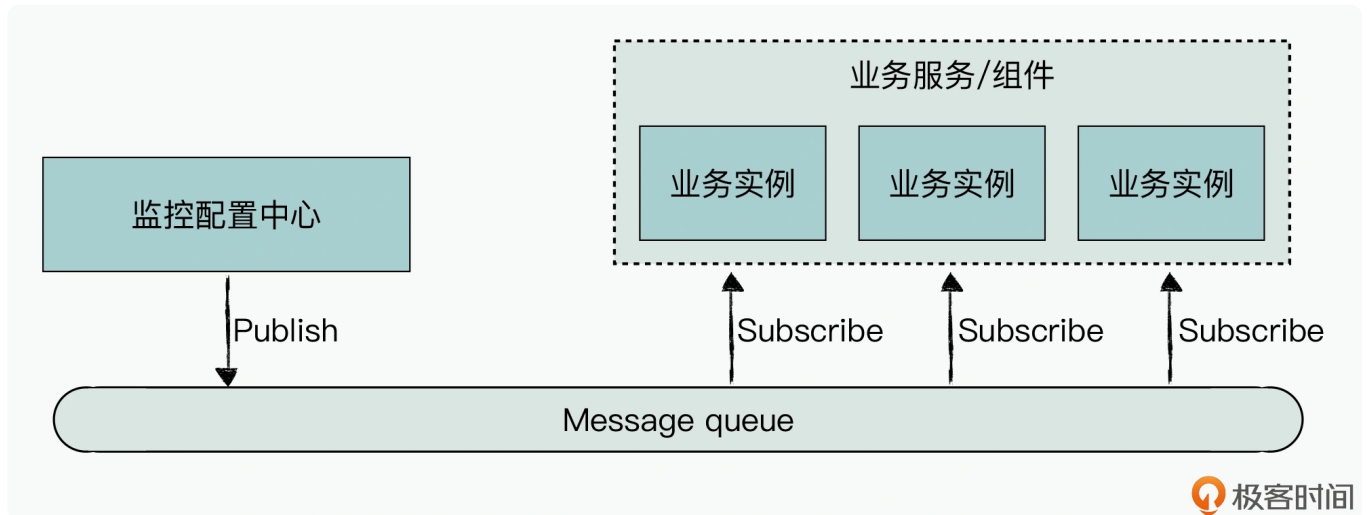
好，那么在知道了软件系统应该实现什么样的动态监控手段以后，你还需要思考的问题就是：在互联网分布式的场景下，应该如何对软件进行动态监控的配置呢？

## 如何对软件系统进行动态监控的配置？



传统的企业级应用软件，经常会使用单机部署模式，这样就只需要提供单独的监控配置接口来处理动态监控配置需求。而针对云平台上的分布式系统，软件服务通常是**多实例部署模式**，这时候依赖单独增加配置接口实现，就不能满足支持一对多的监控配置需求了。

所以，针对互联网多实例服务集群来说，比较常用的动态监控配置实现架构，就如下图所示：



也就是说，你可以从监控配置中心发布一个动态监控请求，然后业务服务实例会订阅监控配置消息，来开启动态监控功能以获取对应的监控数据。

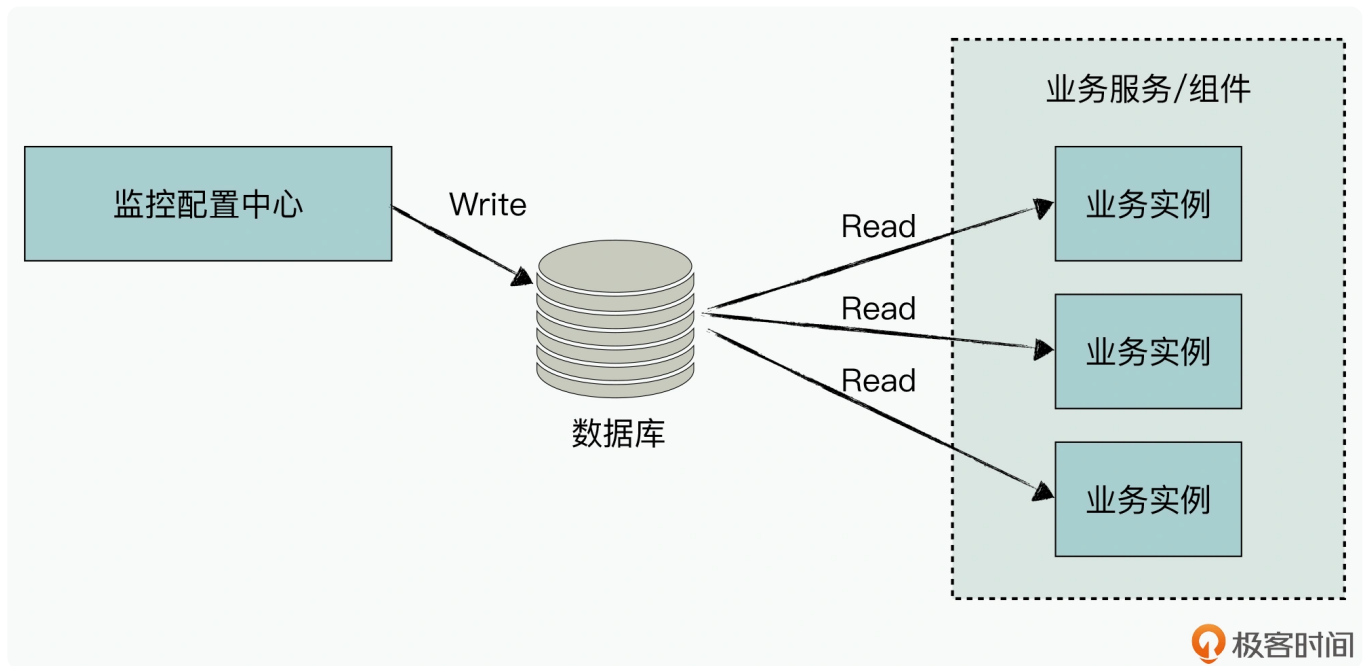
另外你也要知道，图中的监控配置中心，对于互联网分布式服务架构而言，通常是一个后台配置服务，所以你可以将其与业务特性配置管理服务集成在一起。

还有一点，就是监控配置中心与业务服务 / 组件间可以采用**发布订阅**，这样就可以很容易地实现一对多的通信（绝大多数的消息通信中间件如 RabbitMQ、Kafka 都提供这种机制和能力）。但要注意，这里使用发布订阅模式还是有一定的局限性，那就是如果业务实例重启了，对应的动态监控信息很有可能就会丢失。

所以在这种情况下，我们要怎么解决呢？

补充：在嵌入式场景下，通常会有一个假设是业务实例进程运行非常稳定，不会经常重启。所以在监控配置的过程中，你可能不需要考虑业务实例进程重启的情况。而在云平台的分布式系统中，因为业务实例的动态升级操作会非常频繁，所以在配置过程中，你就需要考虑业务实例重启场景下的配置流程是否正常。

其实，对于互联网分布式服务架构的软件系统来说，针对这种场景，你可以**将动态监控配置信息持久化到数据库中**，然后由业务实例去主动查询数据库的监控配置，具体如下图所示：



而这里你需要注意的是：具体的业务服务实例在读取数据库中监控配置时，需要保证一定的实时性，这样才能保证动态监控的启动延迟是可以接受的。

这样，当业务启动动态监控之后，你就可以按照我前面介绍的，将动态监控获取的信息通过消息通道，传递到观测数据分析工具中，从而来支持性能调优分析了（具体到监控获取的观测数据的编码设计、消息通道交互设计等环节，你就可以参考第 6 讲“[通信设计](#)”来指导实现，从而在最大程度上减少额外的性能开销）。

## 小结

我们要知道，在对软件进行性能调优时，通常只依赖操作系统 OS 或硬件芯片级别的监控信息是远远不够的。因此在软件产品设计与实现中，还需要添加一些监控手段获取业务相关的观测信息，以此来支持业务的故障定位和性能分析。

而针对一些对性能要求非常苛刻的软件系统来说，为了权衡监控观察对业务带来的额外性能开销，并且要满足极限负载下的监控数据获取要求，采用动态监控手段，就成为了一种必然的选择。



那么在今天的课程中，我就给你讲解了软件系统应该实现哪些动态监控的观测手段，以及针对互联网分布式服务架构，应该如何去实现动态监控的配置过程。你可以借鉴今天的学习内容，去分析自己当前业务场景中的监控观测机制，看看有哪些是有必要修改为动态监控模式的，然后针对软件系统，来设计实现动态监控的配置能力，从而帮助软件系统在追求高性能的同时，还可以具备足够强的观测分析手段。

## 思考题

在你参与的软件产品服务中，有没有一些与业务相关的监控手段，因为性能的原因默认是关闭的，只在特定的场景下才会打开呢？

欢迎在留言区写下你的答案，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎分享给你的朋友。我们下节课见。

分享给需要的人，Ta订阅后你可得 **20** 元现金奖励

 赞 0     提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇    23 | 监控分析：你的性能调优工具足够有效吗？

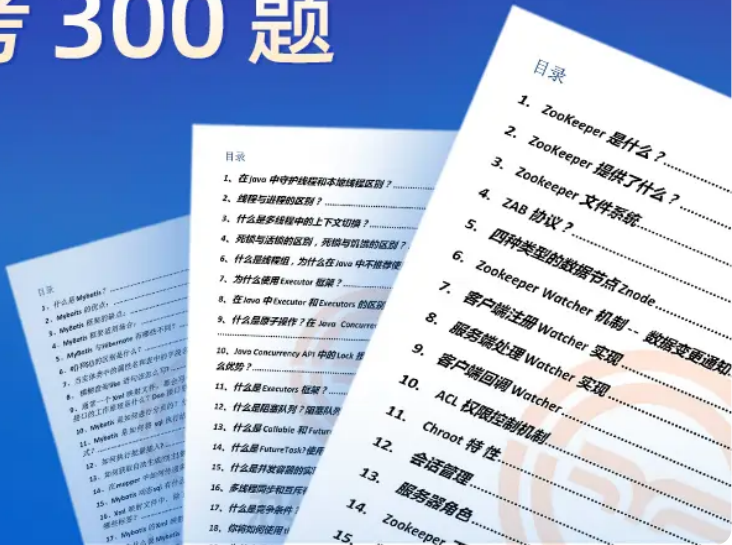
下一篇    25 | 性能调优什么时候应该停止？

更多学习推荐

# Java 面试必考 300 题

最新汇总

限时免费领取



## 精选留言

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。