

30、Skywalking的使用-异步链路追踪

 rock_fish [关注](#)

2021.07.19 16:04:57 字数 663 阅读 281

1、异步链路追踪的概述

通过对 `Callable` , `Runnable` , `Supplier` 这3种接口的实现者进行增强拦截, 将trace的上下文信息传递到子线程中, 实现了异步链路追踪。

有非常多的方式来实现 `Callable` , `Runnable` , `Supplier` 这3种接口, 那么增强就面临以下问题:

- 1. 增强所有的实现类显然不可能, 必须基于有限的约定
- 2. 不能让使用者大量修改代码, 尽可能的基于现有的实现

可能基于以上问题的考虑, skywalking提供了一种即通用又快捷的方式来规范这一现象:

- 1. 只拦截增强带有 `@TraceCrossThread` 注解的类:
- 2. 通过装饰的方式包装任务, 避免大刀阔斧的修改

原始类	提供的包装类	拦截方法	使用技巧
Callable<V>	CallableWrapper<V>	call	CallableWrapper.of(xxxCallable)
Runnable	RunnableWrapper	run	RunnableWrapper.of(xxxRunnable)
Supplier<V>	SupplierWrapper<V>	get	SupplierWrapper.of(xxxSupplier)

包装类 都有注解 `@TraceCrossThread` , skywalking内部的拦截匹配逻辑是, 标注了 `@TraceCrossThread` 的类, 拦截 其名称为 `call` 或 `run` 或 `get` , 且没有入参的方法; 对使用者来说大致分为2种方式:

- 1. 自定义类, 实现接口 `Callable` 、 `Runnable` 、 `Supplier` , 加 `@TraceCrossThread` 注解。当需要有更多的自定义属性时, 考虑这种方式; 参考 `CallableWrapper` 、 `RunnableWrapper` 、 `SupplierWrapper` 的实现方式。
- 2. 通过xxxWrapper.of 装饰的方式, 即 `CallableWrapper.of(xxxCallable)` 、 `RunnableWrapper.of(xxxRunnable)` 、 `SupplierWrapper.of(xxxSupplier)` 。大多情况下, 通过这种包装模式即可。

2、异步链路追踪的使用

需引入如下依赖 (版本限参考) :

```
1 <dependency>
2   <groupId>org.apache.skywalking</groupId>
3   <artifactId>apm-toolkit-trace</artifactId>
4   <version>8.5.0</version>
5 </dependency>
```

2.1. CallableWrapper

CallableWrapper 通过 装饰 包装



原画学习课程



 rock_fish [关注](#)

总资产6

21、Skywalking的埋点-Agent动态采样控制

阅读 219

22、skywalking的Trace数据协议

阅读 12

skywalking的日常维护1:

com.netflix.zuul.exception.ZuulExcept

阅读 41

推荐阅读

Java2021高频面试题(含答案, 适合面试前冲刺一下快速记忆)

阅读 3,530

kotlin对Java函数式API的使用

阅读 372

Spring事务什么时候会失效

阅读 495

数据绑定

阅读 205

JavaWeb开发之Filter过滤器

阅读 351



水处理净化设备





2.1.1 thread+callable

```
1 private String async_thread_callable(String way,long time11,long time22 ) throws ExecutionExce
2     FutureTask<String> futureTask = new FutureTask<String>(CallableWrapper.of(()->{
3         ActiveSpan.debug("async_Thread_Callable");
4         String str1 = service.sendMessage(way, time11, time22);
5         return str1;
6     }));
7     new Thread(futureTask).start();
8     return futureTask.get();
9 }
```

2.1.2 threadPool+callable

```
1 private String async_executorService_callable(String way,long time11,long time22 ) throws
2     Future<String> callableResult = executorService.submit(CallableWrapper.of() -> {
3         String str1 = service.sendMessage(way, time11, time22);
4         return str1;
5     });
6     return (String) callableResult.get();
7 }
```

2.2. RunnableWrapper

Skywalking 通过 RunnableWrapper 包装 Runnable



2.2.1 thread+runnable

```
1 private String async_thread_runnable(String way,long time11,long time22 ) throws Execution
2     //忽略返回值
3     FutureTask futureTask = new FutureTask(RunnableWrapper.of(() -> {
4         String str1 = service.sendMessage(way, time11, time22);
5     })), "mockRunnableResult");
6     new Thread(futureTask).start();
7     return (String) futureTask.get();
8 }
```

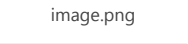


2.2.3 completableFuture + runAsync

通过RunnableWrapper.of(xxx)包装rannable即可。

2.3. SupplierWrapper

Skywalking 通过 `SupplierWrapper<V>` 包装 `Supplier<V>`



2.3.1 completableFuture + supplyAsync

```
1 private String async_completableFuture_supplyAsync(String way,long time11,long time22 ) th
2     CompletableFuture<String> stringCompletableFuture = CompletableFuture.supplyAsync(Supp
3         String str1 = service.sendMessage(way, time11, time22);
4         return str1;
5     });
6     return stringCompletableFuture.get();
7 }
```

3、异步链路追踪的内部原理

需要将trace信息，在线程之间传递，比如 线程A -调用-> 线程B 的场景：

- 线程A
 - 调用 `ContextManager.capture()` ,将trace的上下文信息保存到一个 `ContextSnapshot` 的实例，并返回，此处命名为：contextSnapshot。
 - 通过某种方式将contextSnapshot传递给线程B。
- 线程B
 - 在任务执行前，线程中B获取到contextSnapshot对象，并将其作为入参调用 `ContextManager.continued(contextSnapshot)` 。
 - 此方法中解析出trace的信息后，存储到线程B的线程上下文中。

0人点赞 >

监控



"小礼物走一走，来简书关注我"

赞赏支持

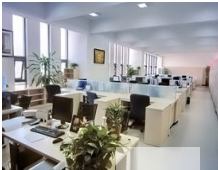
还没有人赞赏，支持一下



rock_fish
总资产6 共写了9.4W字 获得82个赞 共27个粉丝

关注

共享式办公室有什么特点




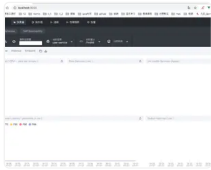
推荐阅读

更多精彩内容>

skywalking实现分布式系统链路追踪


一、背景 随着微服务的越来越流行，我们服务之间的调用关系就显得越来越复杂，我们急需一个APM工具来分析系统中存在的...

 huan1993 阅读 1,468 评论 0 赞 7

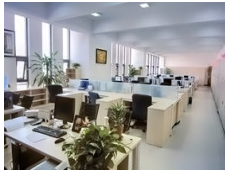


2.SkyWalking源码阅读-了解SkyWalking的几个重要概念

1. Span Span代表一个完整的调用过程，类似于方法栈的栈针，如：helloService.hello()的...

 whslowly 阅读 211 评论 0 赞 1

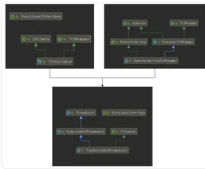
共享办公室来了,是怎样的概念



全链路追踪必备组件之 TransmittableThreadLocal 详解


我们都知道 ThreadLocal 作为一种多线程处理手段，将数据限制在当前线程中，避免多线程情况下出现错误。一...

 java梦想口服液 阅读 466 评论 0 赞 0



凯恩想在热刺和穆里尼奥建立“牢固的关系”

哈里·基恩想和新教练何塞·穆里尼奥建立一种“牢固的关系”，这将有助于托特纳姆更上一层楼。凯恩在4-2战胜奥林匹亚...

 疯狂SPORTS 阅读 4,796 评论 0 赞 5

表情管理

写下你的评论... 评论0 赞

