

# 10、Skywalking的埋点-插件开发的基本技巧

 rock\_fish 

关注

2021.06.11 09:50:51 字数 1,855 阅读 234

## 1. 概述

做中间件的埋点插件，需要通过对源码梳理，找到关键拦截点，如什么的什么方法，通过其参数、返回值、对象属性等获取构建 `Span` 的数据信息。这节内容记录一下开发插件过程中一些零碎的事项。

通过继承 `ClassInstanceMethodsEnhancePluginDefine`，在子类的实现方法中完成以下逻辑：

### 1. 指定待增强的类

```
1  @Override
2  protected ClassMatch enhanceClass() {
3      IndirectMatch hierarchyMatch = byHierarchyMatch(new String[]{ENHANCE_CLASS});
4      PrefixMatch prefixMatch = nameStartsWith(PACK_PACKAGE_PREFIX);
5      return LogicalMatchOperation.and(hierarchyMatch, prefixMatch);
6  }
```


### 2. 在构造函数中添加增强逻辑，可通过类名（构造方法同名），方法参数信息指定构造函数。

```
1  //这种方式，不做构造函数增强
2  @Override
3  public ConstructorInterceptPoint[] getConstructorsInterceptPoints() {
4      return new ConstructorInterceptPoint[0];
5  }
6
7  //指定构造方法
8  @Override
9  public ConstructorInterceptPoint[] getConstructorsInterceptPoints() {
10     return new ConstructorInterceptPoint[]{
11         new ConstructorInterceptPoint() {
12             @Override
13             public ElementMatcher<MethodDescription> getConstructorMatcher() {
14                 //第二个参数必须是String
15                 return takesArgumentWithType(2, "java.lang.String");
16             }
17
18             @Override
19             public String getConstructorInterceptor() {
20                 return INTERCEPT_CLASS;//指定类名（构造方法名）
21             }
22         }
23     };
24 }
```

### 3. 增强实例方法，通过 `InstanceMethodsInterceptPoint` 匹配将增强的方法

```
1  @Override
2  public InstanceMethodsInterceptPoint[] getInstanceMethodsInterceptPoints() {
3      return new InstanceMethodsInterceptPoint[]{
4          new InstanceMethodsInterceptPoint() {
5              @Override
6              public ElementMatcher<MethodDescription> getMethodsMatcher() {
7                  return named(ENHANCE_METHOD);
8              }
9
10             @Override
11             public String getMethodsInterceptor() {
```



 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
- [Spring MVC]HandlerMapping的初始化  
阅读 266
- AopContext.currentProxy()解决同类中调用嵌套方法AOP失效  
阅读 482
- [快速入门]通过http方式监听besu上智能合约的事件  
阅读 233
- vue3.0-基本特性和用法  
阅读 724
- JavaScript练习 - 数据监听和双向绑定  
阅读 525





```
17         return false;
18     }
19 }
20 };
21 }
```

#### 4. 增强类（静态）方法，通过 `StaticMethodsInterceptPoint` 匹配将增强的类方法

```
1  @Override
2  public StaticMethodsInterceptPoint[] getStaticMethodsInterceptPoints() {
3      return new StaticMethodsInterceptPoint[0];
4  }
```

## 2. `xxxMatch` 匹配的技巧

在以上代码中可以看到许多xxxMatch，大致是2类：

- `ClassMatch` ：用于匹配类
- `ElementMatcher` ：用于匹配方法

### 2.1 ClassMatch

在源码中可看到 `ClassMatch` 有如下这些实现类：

```
1  ClassAnnotationMatch
2  EitherInterfaceMatch
3  FailedCallbackMatch
4  HierarchyMatch
5  IndirectMatch
6  ListenableFutureCallbackMatch
7  LogicalAndMatch
8  LogicalOrMatch
9  MethodAnnotationMatch
10 MultiClassNameMatch
11 NameMatch
12 PrefixMatch
13 RegexMatch
14 SuccessCallbackMatch
```

这些匹配有按照类名字匹配、前缀匹配、继承关系、实现接口、组合与匹配、组合或匹配等等；可通过在源码中查找其应用来进一步理解其作用；另外需要注意，除了默认提供的这些，还是可以根据自己的需求来额外定制（通过实现接口 `IndirectMatch`）

### 2.2 ElementMatcher

#### 1. 系统api

在 `ElementMatchers` 中已经定义了很多用于方法匹配的方法，因其方法太多，不在此列举，自行查看

#### 2. 自定义matcher

```
1  return new ElementMatcher<MethodDescription>() {
2      @Override
3      public boolean matches(MethodDescription target) {
4          //自己编排逻辑
5          if(xxx) {
6              return true;
7          }
8          return false;
9      }
```

写下你的评论...

评论0

赞

插件埋点所需要的信息，通常是需要再多个方法中分别捕获，我们需要通过某种上下文机制将这些分散的信息搜集整合起来；从线程角度说通常分为同步请求（相同线程内完成一次执行）和异步请求（不同的线程内完成一次请求）。

### 3.1 同步请求的上下文数据

1. ThreadLocal方式，在Skywalking中已为我们提供了这种途径

```
ContextManager.getRuntimeContext();
```

- 存储数据

```
ContextManager.getRuntimeContext().put(key, data)
```

- 检查数据

```
ContextManager.getRuntimeContext().get(key) ,判断是否存在
```

- 清理数据

需要留意做善后清理数据 `ContextManager.getRuntimeContext().remove(key)`

2. Skywalking的扩展字段，Skywalking会在被增强的类中添加一个sw专用的属性.同时这个类会被修改，实现了接口 `EnhancedInstance`，通过此接口中的2个方法来读写这个扩展属性

```
1 public interface EnhancedInstance {
2     Object getSkyWalkingDynamicField();
3
4     void setSkyWalkingDynamicField(Object value);
5 }
```

这个sw专用字段，就是一个普通的Object，可根据自己的需求给其赋值。

如这样一些使用场景：

- 拦截目标类的构造方法，在构造房中new 一个自定义类，通过 `setSkyWalkingDynamicField` 赋值给扩展字段
- 在其他方法中，捕获不同的数据，填充到这个对象的扩展属性里。
- 读取扩展属性中的数据，构造、填充 `Span`

### 3.2 异步请求的上下文数据

异步请求的场景下，因为跨越了线程，所以上下文需要在多个线程中传递，那么常规的

`ThreadLocal` 方式就不可用了；通过SW扩展属性来承载数据，在多个线程中传递的方式非常方便了。

异步请求时 需要借助 `AsyncSpan#prepareForAsync` 和 `AsyncSpan#asyncFinish` 来完成span的异步关闭，这里一个问题，异步跨线程的情况下，另外的线程里如何拿到这个span实例呢？寻找在多个线程中都存在的对象，如果这个对象本身有承载额外数据的能力最好，如果没有，则需要增强这个类，借助sw的扩展字段来承载这个span，进而在异步结果处理中完成span的关闭。

异步请求的上下文的实例，通过在源码中搜索 `prepareForAsync` 方法的使用来加深理解.这里用 `ExitSpan`方式暂记一下关键逻辑和步骤：

1. 发送请求的线程a，创建ExitSpan 实例 `exitSpan`，会对`exitSpan`做一些赋值
2. 线程b中会获取当前请求的执行结果。
3. 线程a 和线程b 之间一定会有一个对象，在两个线程之间都可访问，暂叫 `externObj`;
4. 线程a，在发起异步请求之前，调用`exitSpan`的 `prepareForAsync` 方法，并把`exitSpan`装入

`externObj`中；如果`externObj`没有合适的属性来承载`exitSpan`数据，就扩展这个`externObj`的

写下你的评论...

评论0

赞

## 4. SpanStack

### 1. EntrySpan 的SpanStack

服务接收请求时，创建 `EntrySpan` ;tomcat和SpringMVC的插件都是创建 `EntrySpan` ，这样就重复创建了 `EntrySpan` ,这样没有意义，在sw中，只要最外层的 `EntrySpan` ,通过一个计数器(stackDepth)和栈的结构来实现，大致流程如下：

- 请求进入1处，创建 `EntrySpan` 暂叫tomEntrySpan，stackDepth=1,记录开始时间
- 请求进入2处，创建 `EntrySpan` 时，stackDepth+1=2，会复用上一步创建的tomEntrySpan对象，而不是new 一个新的；同时会清理此span上的layer、logs和tags；保留springMVC层的layer、logs和tags。
- 请求进入3处，执行finish方法将stackDepth-- = 1;
- 请求进入4处，执行finish方法将stackDepth--=0，关闭tomEntrySpan，记录结束时间

image.png

### 2. ExitSpan 的SpanStack

假设上图 `tomcat` 是作为调用外部请求的，创建 `ExitSpan` （这里只是借图举例）其逻辑流程却是这样：

- 请求进入1处，创建 `ExitSpan` 实例 tomExitSpan，stackDepth=1，记录开始时间;在这一层才可以记录layer、logs和tags。
- 请求进入2处，创建 `ExitSpan` 时，stackDepth+1=2，会复用上一步创建的tomExitSpan对象，而不是new 一个新的；这里layer、logs和tags设置无效。
- 请求进入3处，执行finish方法将stackDepth-- = 1;

比如仕 `ExitSpan` 场景下，A类的put1和put2方法都被拦截，并且put1方法调用了put2方法，在上述的SpanStack的机制里，由于put2方法中stackDepth=2，其内所处理的layer、logs和tags是无效的；对于这种情况，大概有2中方法处理：

- 1. 当stackDepth!=1 的时候，通过上下文记录layerlayer、logs和tags的信息，等stackDepth=1时，从上下文中取出layerlayer、logs和tags的信息赋值给span。
- 2. 控制span的创建时机，自行在上下文中增加stackDepth的计数控制，当stackDepth>1时，不在创建ExitSpan。即put1中有一个ExitSpan，put2中不创建 `ExitSpan`，那么stackDepth一直=1，layerlayer、logs和tags的信息可以随时赋值给span

0人点赞 >

监控

更多精彩内容，就在简书APP



"小礼物走一走，来简书关注我"

赞赏支持

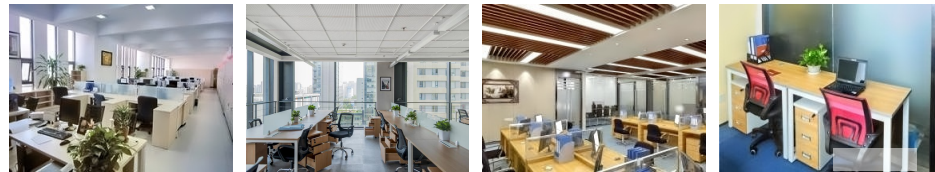
还没有人赞赏，支持一下



rock\_fish  
总资产6 共写了9.4W字 获得82个赞 共27个粉丝

关注

共享办公室来了,是怎样的概念



推荐阅读

更多精彩内容 >

9、Skywalking的埋点-Trace的基本概念

开始之前先仔细阅读skywalking创始人吴晟的一些文章资料： opentracing文档中文版吴晟[https...

rock\_fish 阅读 431 评论 0 赞 0

一个网站访问从输入URL到页面加载的过程 | WEB前端开发 前端开发者


前端开发者 | http请求 https:www.rokub.com 前言见解有限，如有描述不当之处，请帮忙指出，...

麋鹿\_720a 阅读 9,498 评论 11 赞 30




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

表情是什么，我认为表情就是表现出来的情绪。表情可以传达很多信息。高兴了当然就笑了，难过就哭了。两者是相互影响密不可...

 Persistenc\_6aea 阅读 2,199 评论 1 赞 6

Substrate的transaction-payment模块分析

Substrate的transaction-payment模块分析 transaction-payment模块提供...

 建怀 阅读 1,777 评论 0 赞 2