《31 讲带你搞懂 SkyWalking》

徐郡明 资深技术专家

— 拉勾教育出品 —

第18讲：带你揭开 toolkit-activation 工具箱的秘密

toolkit-trace 插件

SkyWalking 为了解决上述问题

提供了一个 **@Trace 注解**，只要将该注解添加到需要监控的业务方法之上

即可收集到该方法相关的 Trace 数据

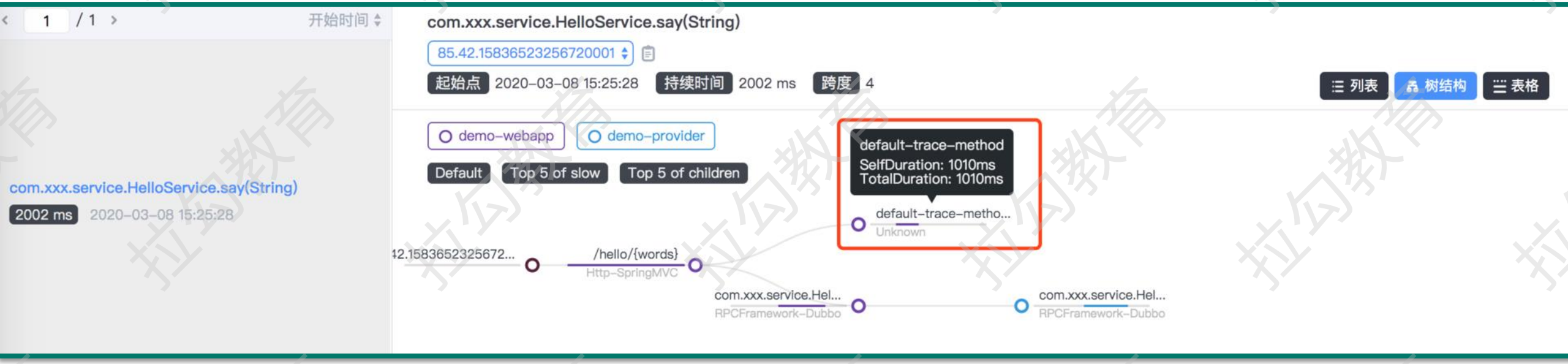
toolkit-trace 插件


```
@Service // Spring的@Service注解
public class DemoService {
    //添加@Trace注解，使用该注解需要引入apm-toolkit-trace依赖
    //在搭建demo-webapp项目时已经介绍过了，pom文件不再展示
    @Trace(operationName = "default-trace-method")
    public void traceMethod() throws Exception {
        Thread.sleep(1000);
        ActiveSpan.tag("trace-method",
            String.valueOf(System.currentTimeMillis()));
        ActiveSpan.info("traceMethod info Message");
        System.out.println(TraceContext.traceId()); // 打印Trace ID
    }
}
```

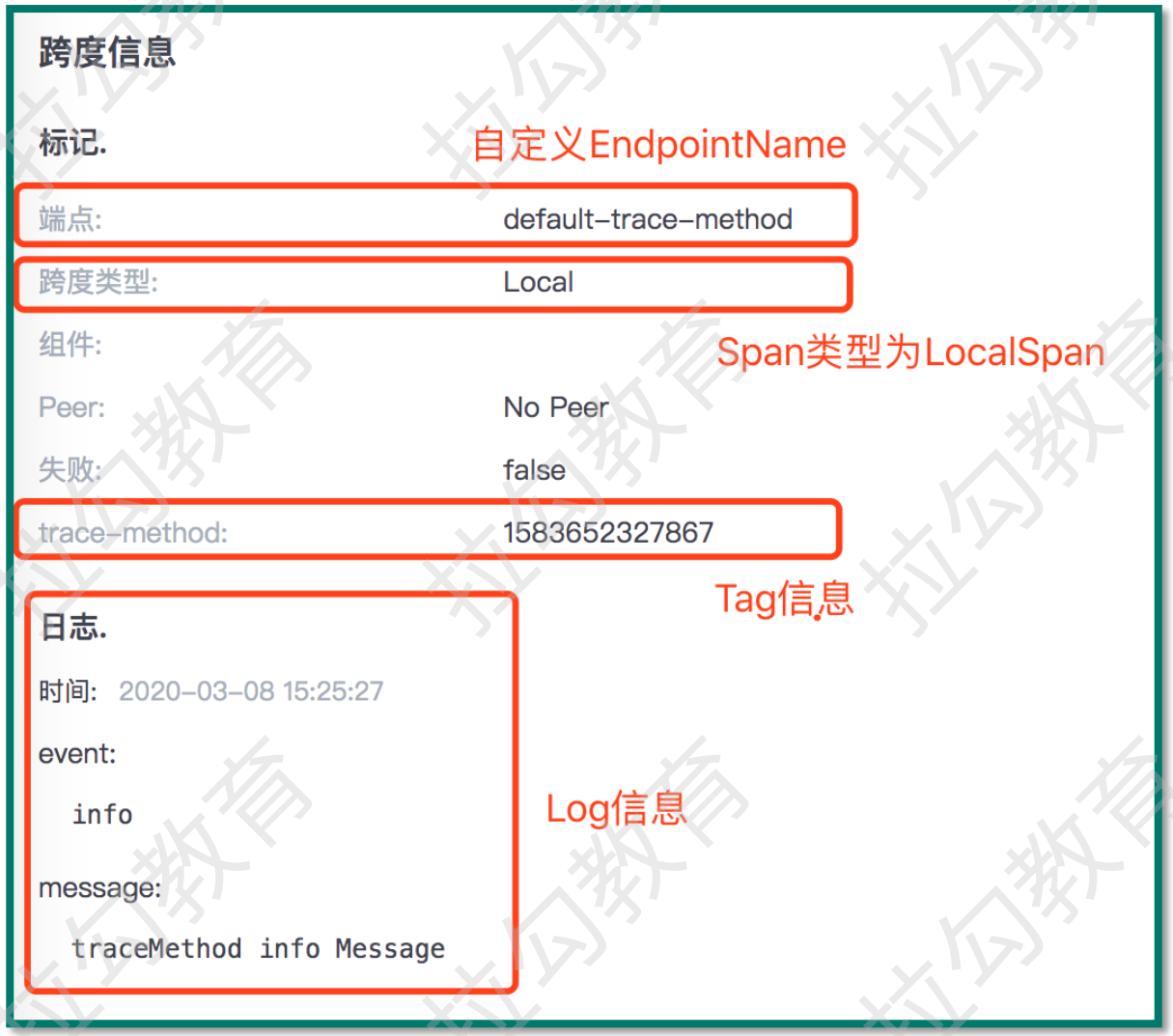
toolkit-trace 插件


```
@RestController
@RequestMapping("/")
public class HelloWorldController {
    @Autowired
    private DemoService demoService;

    @GetMapping("/hello/{words}")
    public String hello(@PathVariable("words") String words) {
        ...
        demoService.traceMethod();
        ... // 省略其他算法
    }
}
```

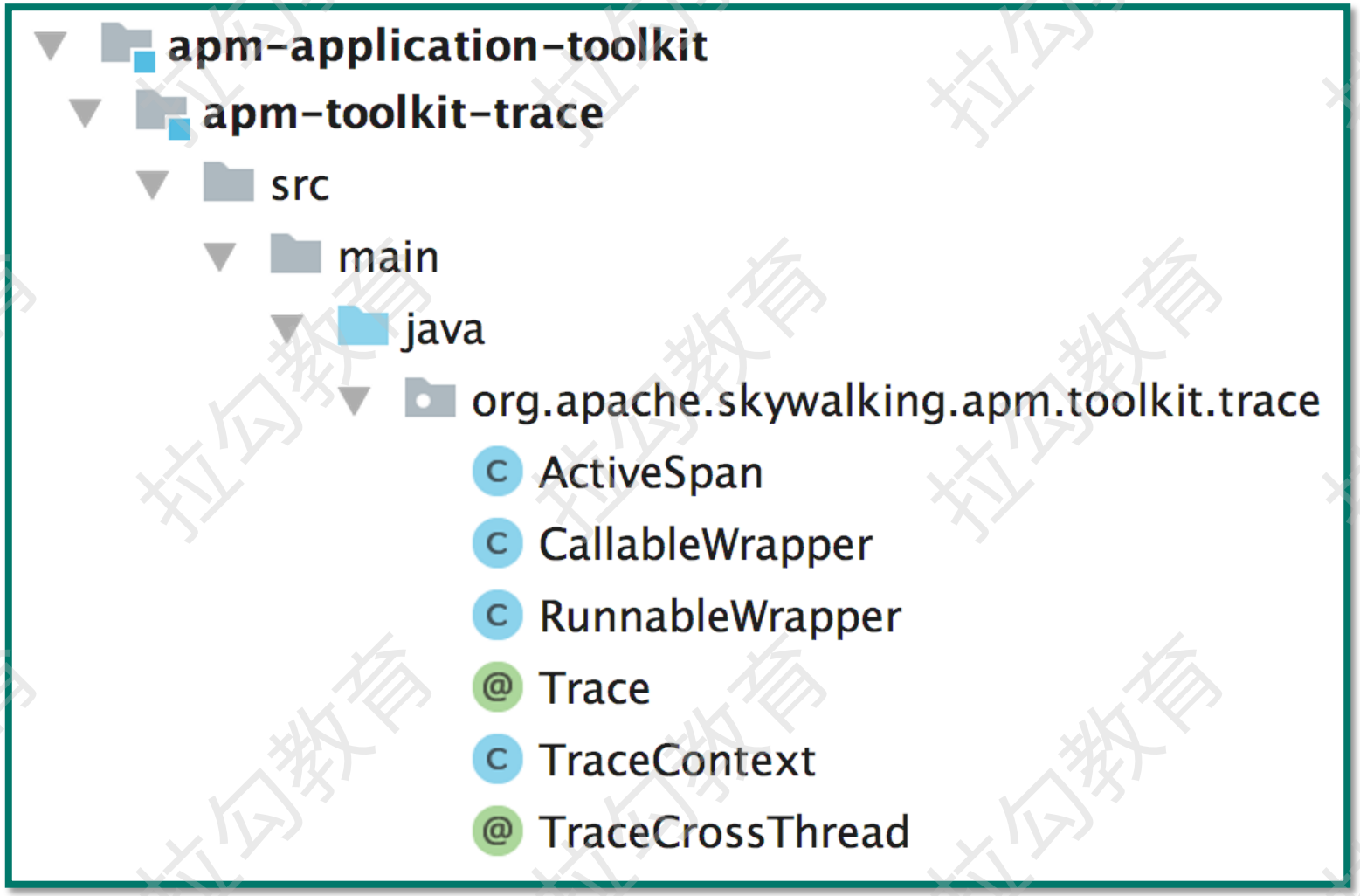
toolkit-trace 插件



toolkit-trace 插件



深入工具类原理



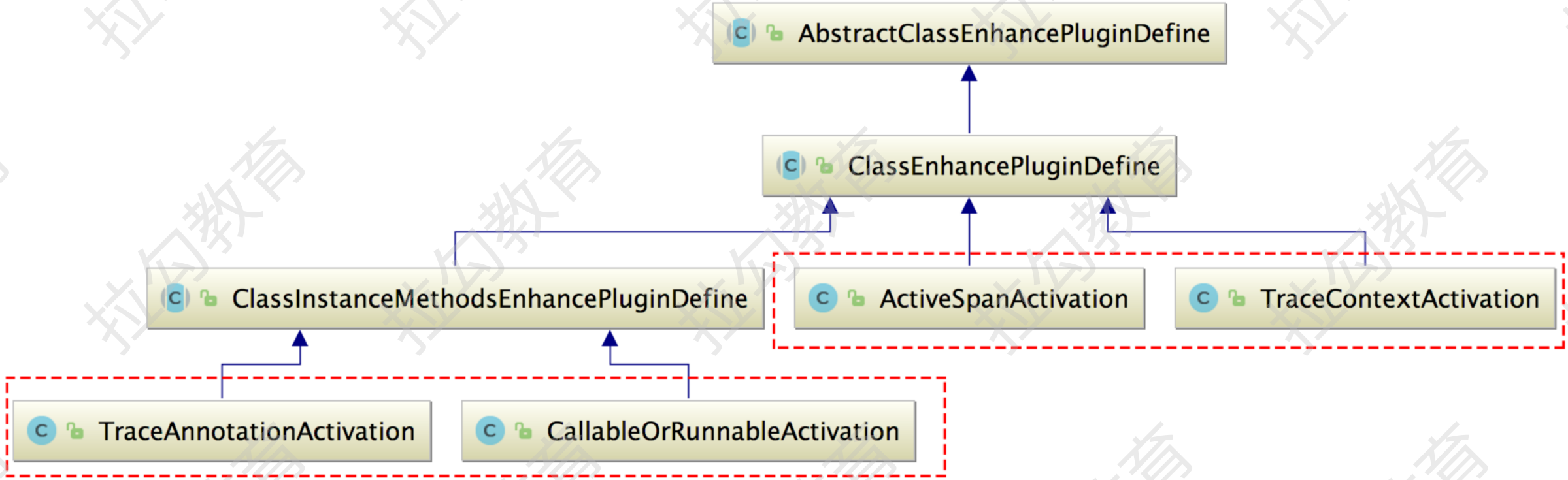
深入工具类原理

在 apm-toolkit-trace-activation 模块的 skywalking-plugin.def 文件中

定义了四个 ClassEnhancePluginDefine 实现类：

- ActiveSpanActivation
- TraceAnnotationActivation
- TraceContextActivation
- CallableOrRunnableActivation

深入工具类原理



深入工具类原理


```
static ClassMatch byMethodAnnotationMatch(String[] annotations){
    return new MethodAnnotationMatch(annotations);
}
```

深入工具类原理

TraceContextActivation 拦截的是 TraceContext.traceId() 这个 static 静态方法

具体增强逻辑在 TraceContextInterceptor 中

其 afterMethod() 方法会调用 ContextManager.getGlobalTraceId() 方法获取当前线程绑定的 Trace ID

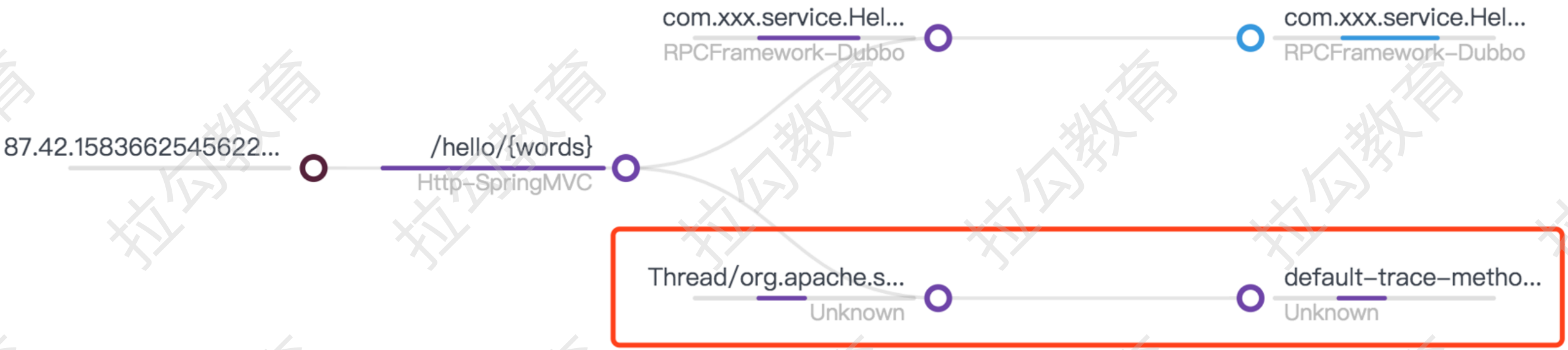
并替换 TraceContext.traceId() 方法返回的空字符串

深入工具类原理

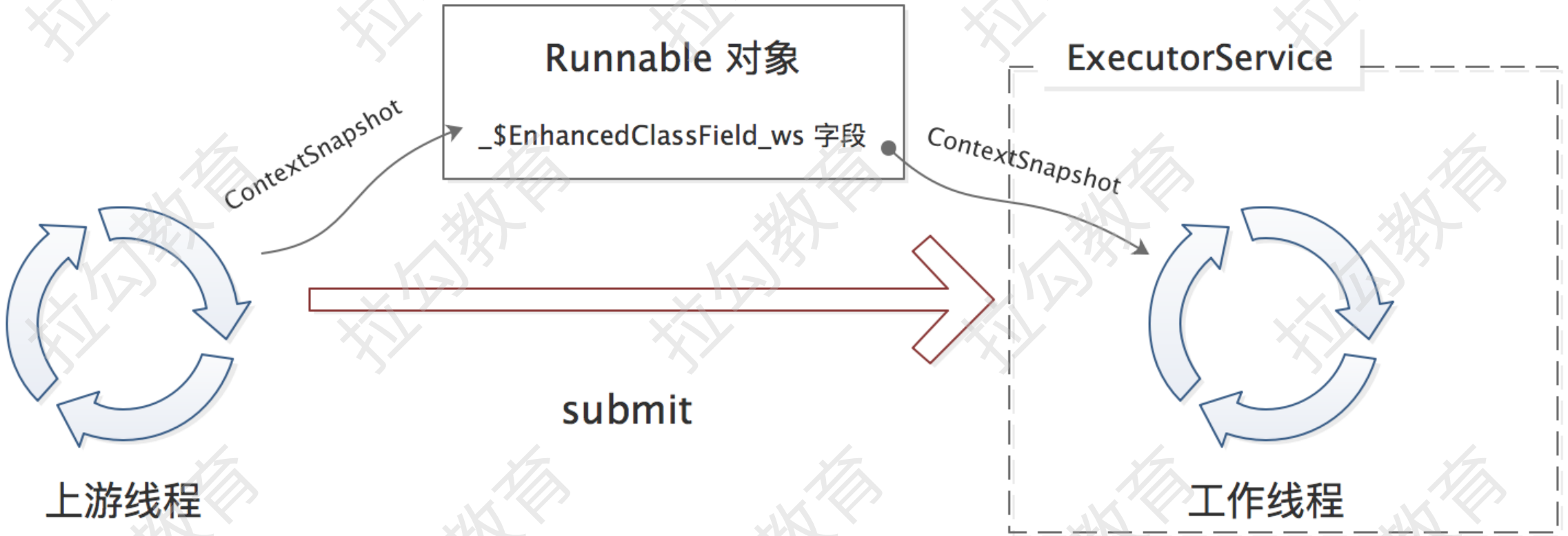
跨线程传播


```java
@RestController
@RequestMapping("/")
public class HelloWorldController {
    // 启动一个单线程的线程池
    private ExecutorService executorService =
            Executors.newSingleThreadScheduledExecutor();

    @Autowired
    private DemoService demoService;

    @GetMapping("/hello/{words}")
    public String hello(@PathVariable("words") String words){
        ... // 省略其他调用
        executorService.submit( // 省略try/catch代码块
            // 使用RunnableWrapper对Runnable进行包装，实现Trace跨线程传播
            RunnableWrapper.of(() -> demoService.traceMethod())
        );
        ...
    }
}
```

跨线程传播



跨线程传播



跨线程传播


```
public void onConstruct(EnhancedInstance objInst,
    Object[] allArguments) {
  if (ContextManager.isActive()) {
    objInst.setSkyWalkingDynamicField(ContextManager.capture());
  }
}
```

跨线程传播


```
public void beforeMethod(EnhancedInstance objInst, Method method,
    Object[] allArguments, Class<?>[] argumentsTypes,
      MethodInterceptResult result) throws Throwable {
  //该调用中会先创建TracingContext，然后创建LocalSpan
  ContextManager.createLocalSpan("Thread/" +
     objInst.getClass().getName() + "/" + method.getName());
  ContextSnapshot cachedObjects =
    (ContextSnapshot)objInst.getSkyWalkingDynamicField();
  if (cachedObjects != null) { // 恢复Trace信息
    ContextManager.continued(cachedObjects);
  }
}
```

Trace ID 与日志

为了方便将 Trace 和日志进行关联

一般会**在日志开头的固定位置打印 Trace ID**

application-toolkit 工具箱目前支持 logback、log4j-1.x、log4j-2.x 三个日志框架

下面以 logback 为例演示并分析原理

日志集成 Trace ID


```
<dependency>
  <groupId>org.apache.skywalking</groupId>
  <artifactId>apm-toolkit-logback-1.x</artifactId>
  <version>6.2.0</version>
</dependency>
```

日志集成 Trace ID

拉勾教育
互联网人实战大学

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
            <layout class="org.apache.skywalking.apm.toolkit.log.logback.v1.x.TraceIdPatternLogbackLayout">
                <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level logger_name:%logger{36} - [%tid] - message:%msg%n</pattern>
            </layout>
        </encoder>
    </appender>

    <root level="info">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

日志集成 Trace ID


```
private static final Logger LOGGER =
    LoggerFactory.getLogger(HelloWorldController.class);

@GetMapping("/hello/{words}")
public String hello(@PathVariable("words") String words)
  LOGGER.info("this is an info log,{}", words);
  ... // 省略其他代码
}
```

日志集成 Trace ID

拉勾教育
互联网人实战大学

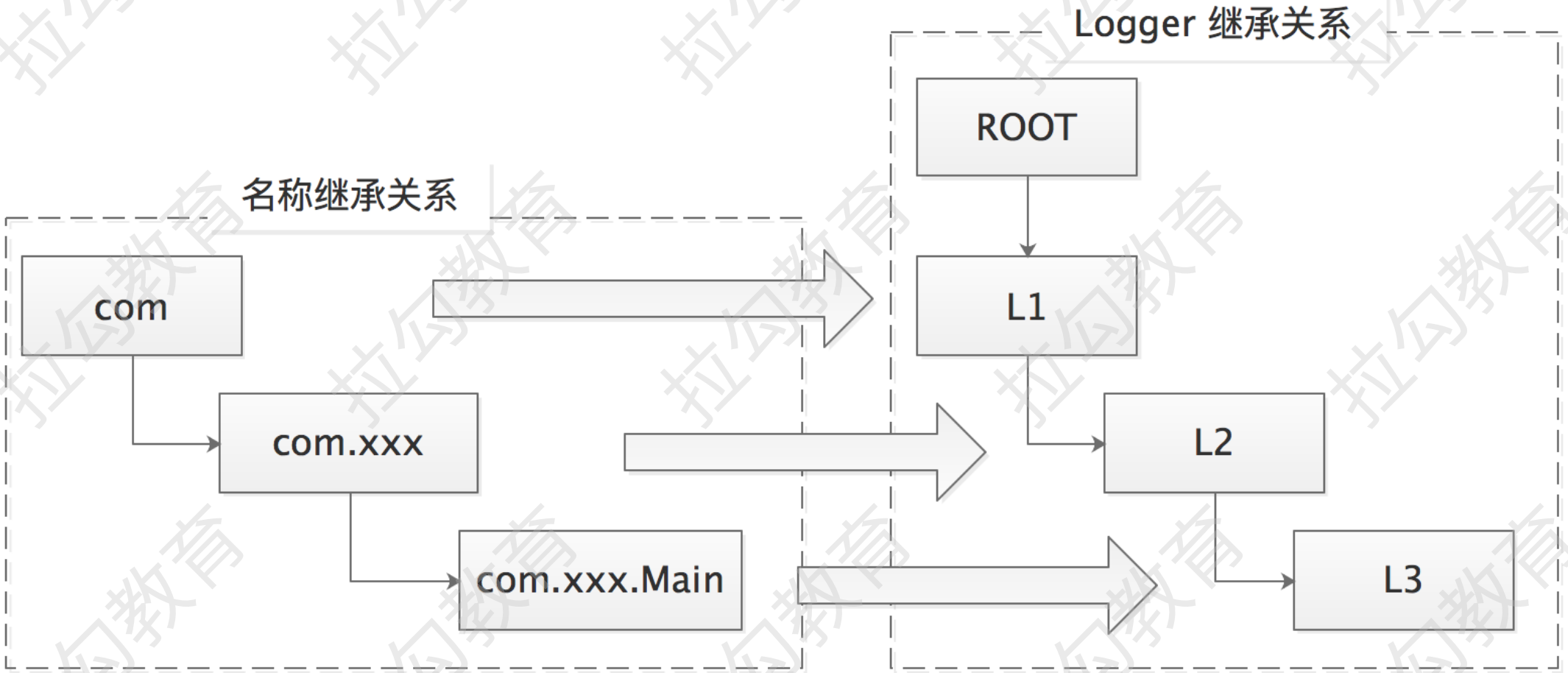
```
20:57:55.278 [main] INFO  logger_name:com.xxx.DemoWebAppApplication - [TID:N/A] - message:Started DemoWebAppApplication in 8.733 seconds (JVM running for 12.564)
20:58:03.740 [http-nio-8000-exec-1] INFO  logger_name:o.a.c.c.C.[Tomcat].[localhost].[/] - [TID:92.42.15836722837140001] - message:Initializing Spring DispatcherServlet 'dispatcherServlet'
20:58:03.740 [http-nio-8000-exec-1] INFO  logger_name:o.s.web.servlet.DispatcherServlet - [TID:92.42.15836722837140001] - message:Initializing Servlet 'dispatcherServlet'
20:58:03.772 [http-nio-8000-exec-1] INFO  logger_name:o.s.web.servlet.DispatcherServlet - [TID:92.42.15836722837140001] - message:Completed initialization in 31 ms
20:58:04.023 [http-nio-8000-exec-1] INFO  logger_name:c.x.controller.HelloWorldController - [TID:92.42.15836722837140001] - message:this is an info log,xxx
```

Logback 核心概念

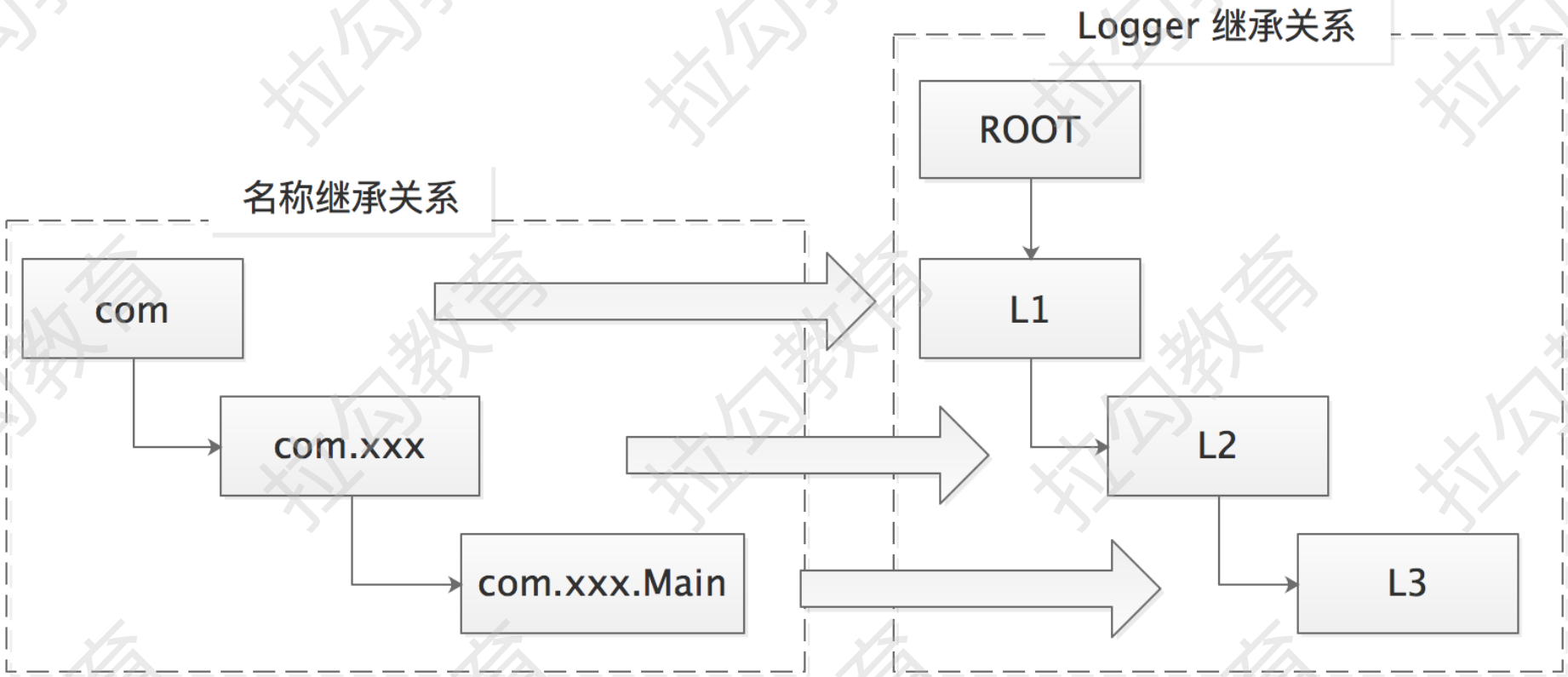
Logback 日志框架分为三个模块：**logback-core**、**logback-classic** 和 **logback-access**：

- **core** 模块是整个 logback 的核心基础
- **classic** 模块是在 core 模块上的扩展，classic 模块实现了 SLF4J API
- **access** 模块主要用于与 Servlet 容器进行集成，实现记录 access-log 的功能

Logback 核心概念



Logback 核心概念



常用的 Level 级别以及 Level 优先级

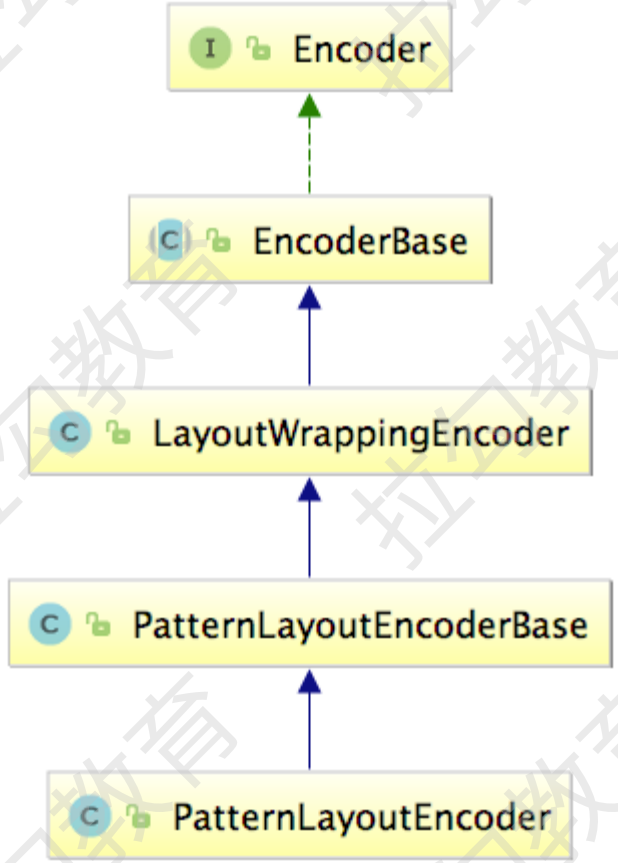
TRACE < DEBUG < INFO < WARN < ERROR

Logback 核心概念


```
private static final Logger LOGGER =
    LoggerFactory.getLogger(HelloWorldController.class);
```

Logback 核心概念

最常用的 Encoder 实现是 **PatternLayoutEncoder**



Logback 核心概念


```
public byte[] encode(E event) {
    String txt = layout.doLayout(event); //依赖Layout将日志事件转换字符串
    return convertToBytes(txt); //将字符串转换成字节数组
}
```

Logback 核心概念


```
public void start() {
    // 解析pattern字符串
    Parser<E> p = new Parser<E>(pattern);
    Node t = p.parse();
    //根据解析后的pattern创建Converter链表
    this.head = p.compile(t, getEffectiveConverterMap());
    ... ... // 省略其他代码
}
```

Logback 核心概念


```
static {
    // DateConverter处理pattern字符串中的“%d”或是“%date”占位符
    defaultConverterMap.put("d", DateConverter.class.getName());
    defaultConverterMap.put("date", DateConverter.class.getName());
    // ThreadConverter处理pattern字符串中的“%t”或是“%thread”占位符
    defaultConverterMap.put("t", ThreadConverter.class.getName());
    defaultConverterMap.put("thread",
        ThreadConverter.class.getName());
    // MessageConverter处理“%m”、“%msg”、“message”占位符
    defaultConverterMap.put("m", MessageConverter.class.getName());
    defaultConverterMap.put("msg", MessageConverter.class.getName());
    defaultConverterMap.put("message",
        MessageConverter.class.getName());
    // 省略其他占位符对应的Converter
}
```

Logback 核心概念


```
public String convert(ILoggingEvent le) {
    long timestamp = le.getTimeStamp(); // 获取日志事件
    return cachingDateFormatter.format(timestamp); // 格式化
}
```

Logback 核心概念


```
public String convert(ILoggingEvent event) {
    return event.getFormattedMessage();
}
```

toolkit-logback-1.x

拉勾教育
互联网人实战大学

```
public class TraceIdPatternLogbackLayout extends PatternLayout {
    static {
        defaultConverterMap.put("tid",
            LogbackPatternConverter.class.getName());
    }
}
```

toolkit-logback-1.x


```
public Object afterMethod(EnhancedInstance objInst, Method method,
    Object[] allArguments, Class<?>[] argumentsTypes, Object ret) {
    return "TID:" + ContextManager.getGlobalTraceId(); // 获取Trace ID
}
```

Next：第17讲《OAP 初始化流程精讲，一眼看透 SkyWalking OAP 骨架》

拉勾教育

—互联网人实战大学—

关注拉勾「教育公众号」
获取更多课程信息

L / A / G / O / U