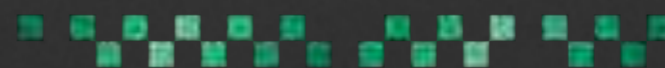


微服务实践之路

马昕曦 (小马哥)



SFDC

SegmentFault
Developer Conference

自我介绍

- 姓名

马昕曦，一个人名，没有之一。

- 职业

阿里巴巴技术专家，目前主要负责微服务技术实施和推广，重点关注云计算、微服务以及软件架构等领域。

- 工作

“手淫互联网，意淫大数据”

- 笔名

次灵均

- 花名

桃谷（非德艺双馨）



桃谷（德艺双馨）

ソレスターions専属
桃谷エリカ Erika Momotani

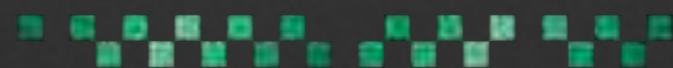
PRESTIGE Actress Special Contents プレステージ専属女優

Profile	Photo
<p>生年月日 1994.6.15 出身地 東京都 身長 165cm 3サイズ B84-W56-H82 血液型 A型</p>	<p>UP</p>  <p>2018.06.18 NEW UP</p>
<p>Movie</p> <p>UP</p>	



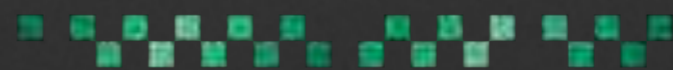
议程

- **What**
- **Why**
- **How**
- **Q&A**
- **?**



What

“是非之心，知之端也”



微服务是什么？

微服务是一种细粒度（ Fine-Grain ）的SOA架构



SOA又是什么?

**SOA = Service-Oriented
Architecture**



SOA有什么？

■ 特征

- 面向服务 (Service-Oriented)
- 松耦合 (Loose-Coupling)
- 模块化 (Modular)
- 分布式计算 (Distributed Computing)
- 平台无关性 (Independent Platform)
- 集中管理 (Center Government)

■ 技术

- Web Services
- Message Queue
- ESB

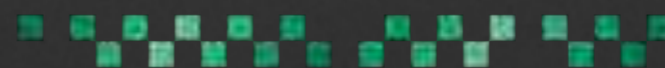


SOA不是什么?

SOA \neq Monolithic



Monolithic是什么?

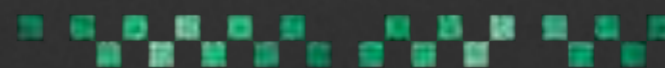


SFDC

SegmentFault
Developer Conference

Why ?

“学而不思则罔”

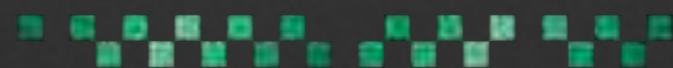


SFDC

SegmentFault
Developer Conference

为什么要微服务？

- 效率的需要
- 质量的需要
- 稳定的需要
- 运维的需要
- 成长的需要



为什么不必微服务？

- 场景单一
- 逻辑简单
- 业务渐逝
- “老成持重”
- 技术盲从



进阶阅读

- 网络

<http://microservices.io/>

<http://martinfowler.com/articles/microservices.html>

- 书籍

《Microservice Architecture》, Irakli Nadareishvili

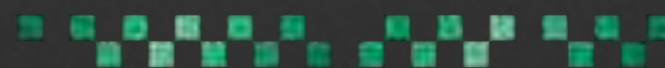
- 文稿

《2016.11.19 微服务实践之路（厦门）演讲稿》，小马哥



How

“多见阙殆，慎行其余”

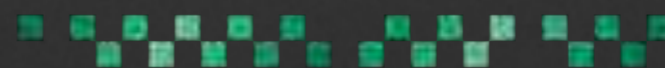


怎么实现微服务

- 心态

- 技术

- 思想



心态

- “子路有闻，未之能行，唯恐有闻”
- “不患无位，患所以立”
- “攻乎异端，斯害也已”
- “过则勿惮改”



技术

- **Docker**
- **DDD**
- **Middleware (Java)**



Docker

- **测试环境**：AliDocker + ECS（阿里云）
- **生成环境**：AliDocker + 物理机



DDD

- 有界上下文 (Bounded Context)
- 持续集成 (Continuous integration)
- ~~上下文映射 (Context Map)~~



Middleware

- **Spring**
- **Spring Boot**
- **Spring Cloud**
- **Spring Cloud Stream**
- **Spring Boot DevOps**



Spring

- Annotation驱动

```
@EnableWebMvc  
@Configuration  
public class MvcConfiguration {  
  
}
```



```
<bean class='org.springframework.web.servlet.mvc.method.annotation.RequestMappingMapping' >  
</bean>  
  
<bean class='org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter' >  
</bean>  
  
<bean class='org.springframework.web.servlet.mvc.method.annotation.ExceptionHandlerExceptionResolver' >  
</bean>  
  
<bean class='org.springframework.web.accept.ContentNegotiationManager' >  
</bean>
```



Spring Boot

- 渲染引擎

Thymeleaf (Spring 推荐)

优点：HTML结构化、UI友好，表达式功能强大

缺点：编码略微繁琐、性能一般、扩展复杂

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" xmlns="http://www.w3.org/1999/xhtml"
      th:inline="text">

<head th:include="fragments/head :: head" th:with="jeditable=true,pace=true" th:inline="text"></head>

<body>

<div id="wrapper">

    <nav th:replace="fragments/left-navbar :: navigation-bar"/>

    <div id="page-wrapper" class="gray-bg">
        <div class="row border-bottom">
            <nav th:replace="fragments/top-navbar :: top-navigation-bar"/>
        </div>
        <div th:replace="fragments/page-heading :: page-heading"></div>
        <div class="wrapper wrapper-content animated fadeInRight">
```



Spring Boot

- 渲染引擎

Velocity (广泛应用)

优点：性能良好、易于扩展、事件处理、配置灵活

缺点：HTML结构化不友好、发展停滞

```
<html>
<head>
  <title>${pageTitle}</title>
</head>
<body>
  <!-- MVC View 渲染HTML内容的渲染上下文名称 -->
  $screen_content_key

  <br />

  \${convert.toNumbers('12.6,42')} : ${convert.toNumbers('12.6,42')}

  <br />

  \${layout_key} : ${layout_key}

</body>
</html>
```

```
#macro( d )
<div>Hello,Marco</div>
#end
```

```
#set( $code = "<span>Mercy</span>" )

\${code} : ${code}
```



Spring Boot

- 渲染引擎

JSP (Java EE标准)

优点：编码灵活、兼容性好、性能优秀、多种页面结构化

缺点：限制表达式 (EL)、扩展繁琐、规约较多、Servlet强依赖

```
<c:set var="siteTemplate" value="${template.siteTemplate}"/>
<c:set var="applySites" value="${siteTemplate.applySites}"/>
<c:choose>
  <c:when test="${fn:contains(applySites,'1001')}">
    <c:set var="previewURI" value="/microshop/htdocs/preview/${template.directoryName}"/>
    <c:set var="debugURI" value="/man/microtemplate/layout/${template.directoryName}"/>
  </c:when>
  <c:otherwise>
    <c:set var="previewURI" value="/htdocs/${template.directoryName}"/>
    <c:set var="debugURI" value="/man/template/layout/${template.directoryName}"/>
  </c:otherwise>
</c:choose>
```

```
<taglib xmlns="http://www.taobao.com/taglibs/sdk"
  xmlns:cn="http://www.taobao.com/2001/01/cn"
  xmlns:xsi="http://www.taobao.com/xsi"
  xsi:schemaLocation="http://www.taobao.com/taglibs/sdk http://www.taobao.com/taglibs/sdk"
  version="2.1">

  <lib-version>1.0</lib-version>
  <short-name>sdk</short-name>
  <uri>http://tee-sdk.taobao.com/taglibs/sdk</uri>

  <function>
    <name>currentTimeMillis</name>
    <function-class>java.lang.System</function-class>
    <function-signature>long currentTimeMillis()</function-signature>
  </function>
```



Spring Cloud

- 分布式配置

```
import org.springframework.cloud.bootstrap.config.PropertySourceLocator;
import org.springframework.core.env.CompositePropertySource;
import org.springframework.core.env.Environment;
import org.springframework.core.env.PropertySource;

public class DiamondPropertySourceLocator implements PropertySourceLocator {

    @Override
    public PropertySource<?> locate(Environment environment) {
        String applicationName = environment.getProperty("spring.application.name");
        String applicationGroup = environment.getProperty("spring.application.group");

        CompositePropertySource compositePropertySource = new CompositePropertySource(DIAMOND_PROPERTY_SOURCE_NAME);

        loadGroupConfigurationRecursively(compositePropertySource, applicationGroup);

        loadApplicationConfiguration(compositePropertySource, environment, applicationGroup, applicationName);

        return compositePropertySource;
    }
}
```



Spring Cloud

- 分布式配置

```
@DiamondListener
public class DataListenerConfiguration {

    private final static Log logger = LoggerFactory.getLog(DataListenerConfiguration.class);

    @Autowired
    private UserService userService;

    import org.springframework.cloud.context.config.annotation.RefreshScope;
}
@Diamond
public class ArchimedesProperties {
    @Diamond
    @RefreshScope
    @Component
    @RefreshScope
    public class ArchimedesProperties {
        @Value("${archimedes.masterEnabled}")
        private boolean masterEnabled;
    }
}

@DiamondConfigListener(dataId = "com.aliexpress.boot:demo-application.json", ignoreMissingData = true, missingDataExceptionHandler = null)
public void receive(Map<String, Object> data) {
    logger.info(data);
}

@DiamondConfigListener(dataId = "com.aliexpress.boot:demo-application.json", ignoreMissingData = true, missingDataExceptionHandler = null)
public void receive(User user) {
    userService.update(user);
}
}
```



Spring Cloud Stream

- 消息队列 (MQ) - Source

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.stream.annotation.EnableBinding;
import org.springframework.cloud.stream.messaging.Source;
import org.springframework.context.annotation.Bean;
import org.springframework.integration.annotation.InboundChannelAdapter;
import org.springframework.integration.annotation.Poller;
import org.springframework.integration.core.MessageSource;
import org.springframework.messaging.support.GenericMessage;

@SpringBootApplication
@EnableBinding(Source.class)
public class MetaqBinderSourceDemoApp {
    public static void main(String[] args) {
        SpringApplication.run(MetaqBinderSourceDemoApp.class, args);
    }

    public MetaqBinderSourceDemoApp() {
        int x = 1;
    }

    private int i = 0;

    @Bean
    @InboundChannelAdapter(value = Source.OUTPUT, poller = @Poller(fixedDelay = "1000", maxMessagesPerPoll = "1"))
    public MessageSource<String> timerMessageSource() {
        return () -> new GenericMessage<>("Message# " + i++);
    }
}
```



Spring Cloud Stream

- 消息队列 (MQ) - Sink

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.stream.annotation.EnableBinding;
import org.springframework.cloud.stream.messaging.Sink;

@SpringBootApplication
@EnableBinding(Sink.class)
public class MetaqBinderSinkDemoApp {
    private static Logger logger = LoggerFactory.getLogger(MetaqBinderSinkDemoApp.class);

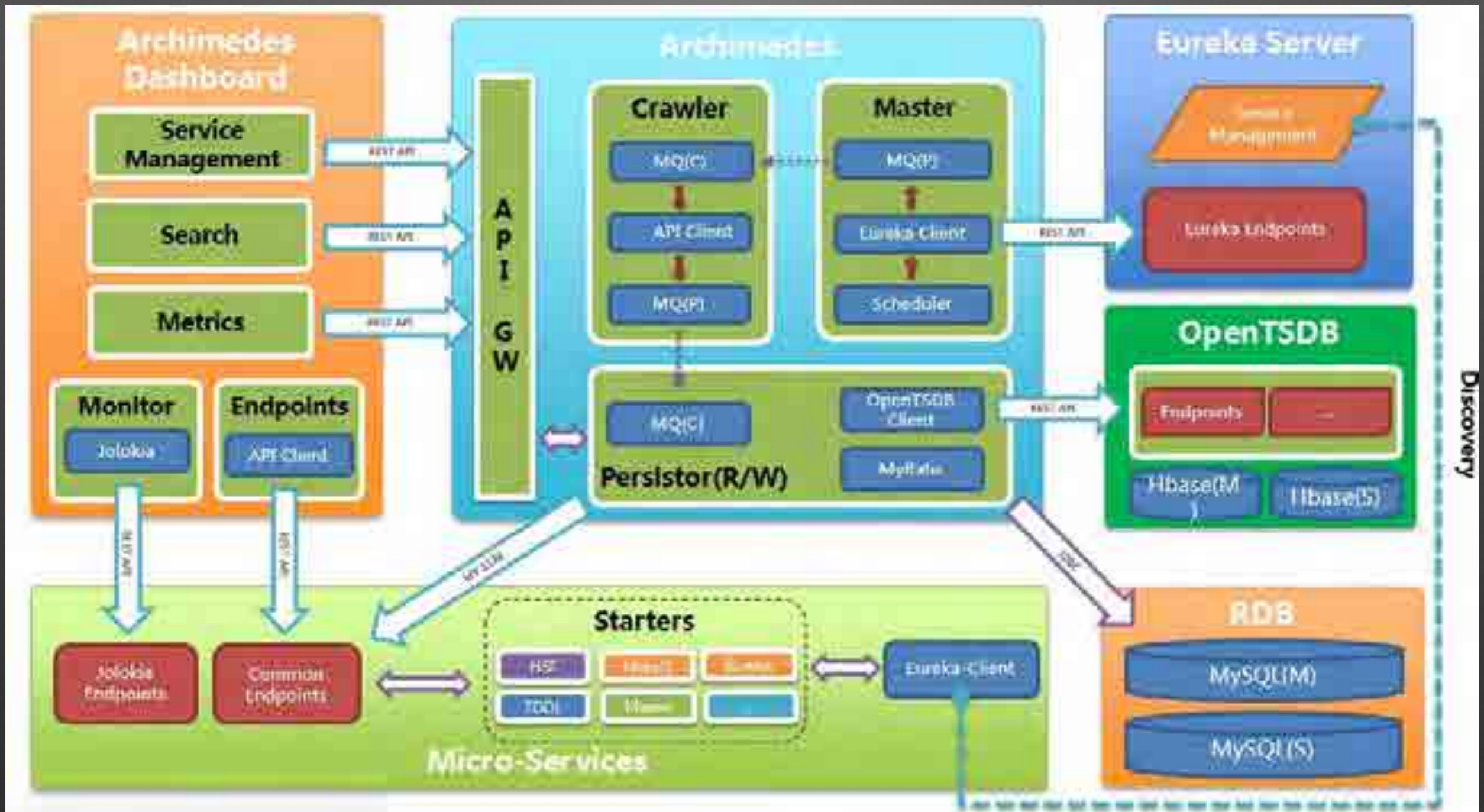
    public static void main(String[] args) {
        SpringApplication.run(MetaqBinderSinkDemoApp.class, args);
    }

    @ServiceActivator(inputChannel=Sink.INPUT)
    public void loggerSink(Object payload) {
        logger.info("Received: " + payload);
    }
}
```



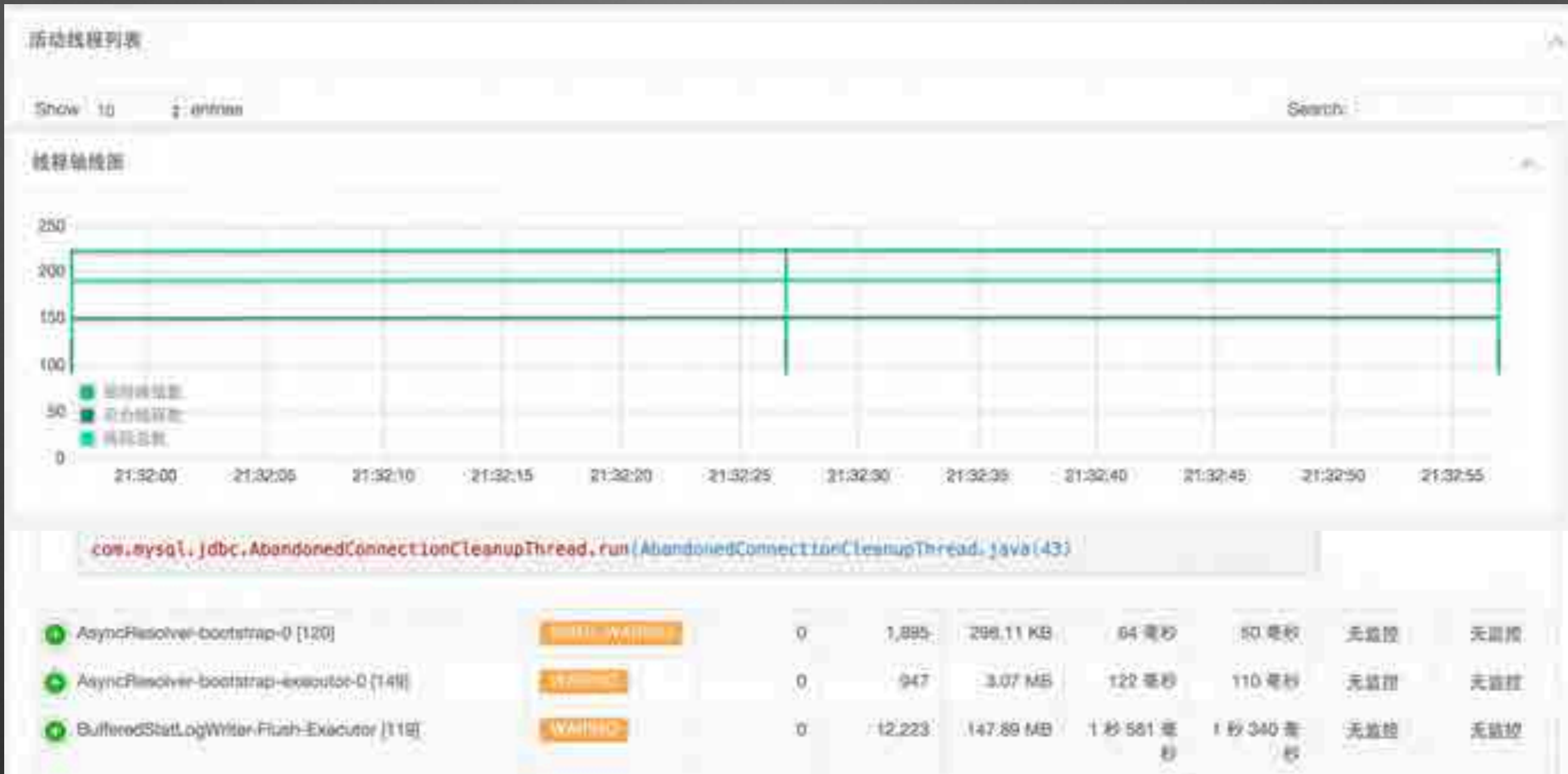
Spring Boot DevOps

- 整体架构



Spring Boot DevOps

- 线程管理



Spring Boot DevOps

- 内存管理

The screenshot displays a memory management dashboard with the following data:

内存池	内存管理	内存使用	内存使用(峰值)	内存使用(回收)
CMS Old Gen	ConcurrentMarkSweep	已使用 184 MB / 2 GB (9.20%) 初始值 2 GB 已提交 2 GB	已使用 184 MB / 2 GB (9.20%) 初始值 2 GB 已提交 2 GB	已使用 0 bytes / 2 GB (0.00%) 初始值 2 GB 已提交 0 bytes
Code Cache	CodeCache	已使用 57 MB / 245 MB (23.70%) 初始值 2 MB 已提交 57 MB	已使用 57 MB / 245 MB (23.70%) 初始值 2 MB 已提交 57 MB	
Compressed Class Space	Metaspace	已使用 107 MB / 1 GB (10.74%) 初始值 0 bytes 已提交 107 MB	已使用 107 MB / 1 GB (10.74%) 初始值 0 bytes 已提交 107 MB	
Metaspace	Metaspace	已使用 65 MB / 256 MB (25.57%) 初始值 0 bytes 已提交 65 MB	已使用 65 MB / 256 MB (25.57%) 初始值 0 bytes 已提交 65 MB	
Par Eden Space	ConcurrentMarkSweep	已使用 1 GB / 2 GB (50.00%) 初始值 2 GB 已提交 2 GB	已使用 2 GB / 2 GB (100.00%) 初始值 2 GB 已提交 2 GB	已使用 0 bytes / 2 GB (0.00%) 初始值 2 GB 已提交 2 GB



Spring Boot DevOps

- 日志管理

The screenshot displays the logging management interface for Spring Boot DevOps. It features two informational boxes at the top: a green one for '描述信息' (Description) and a blue one for '操作' (Operations). Below these are tabs for 'Java Logging', 'Log4j', and 'Logback', with 'Logback' selected. A 'Show 10 entries' control is present. The main area is a table with columns for 'Logger名称', '级别 (原始)', '级别 (Current)', and '操作'. The table lists four loggers, all with 'INFO' levels and 'INFO' current levels. Each row has an orange '重置' (Reset) button.

描述信息
智能识别当前系统引入的日志框架，目前支持：Java Logging，Log4j，Log4j2 以及 Logback，如果当前系统使用了多种日志框架时，下面的Tab页面会出现多个

操作
重置操作：当日志级别修改后，重置操作帮助日志级别恢复到原始级别

Java Logging Log4j **Logback**

Show 10 entries

Logger名称	<input type="checkbox"/> 级别 (原始)	级别 (Current)	操作
com	INFO	INFO	重置
com.alibaba	INFO	INFO	重置
com.alibaba.alimonitor	INFO	INFO	重置
com.alibaba.alimonitor.jmonitor	INFO	INFO	重置



思想

- 少谈“敏捷”
- 推崇“简洁”
- 学习“狄仁杰”



Q&A

“敏而好学，不耻下问”

