# 最新后端和微服务面试题

说明：最新的面试题每个月会定期收集、更新，请关注公众号“疯狂图书”联系助手获取。

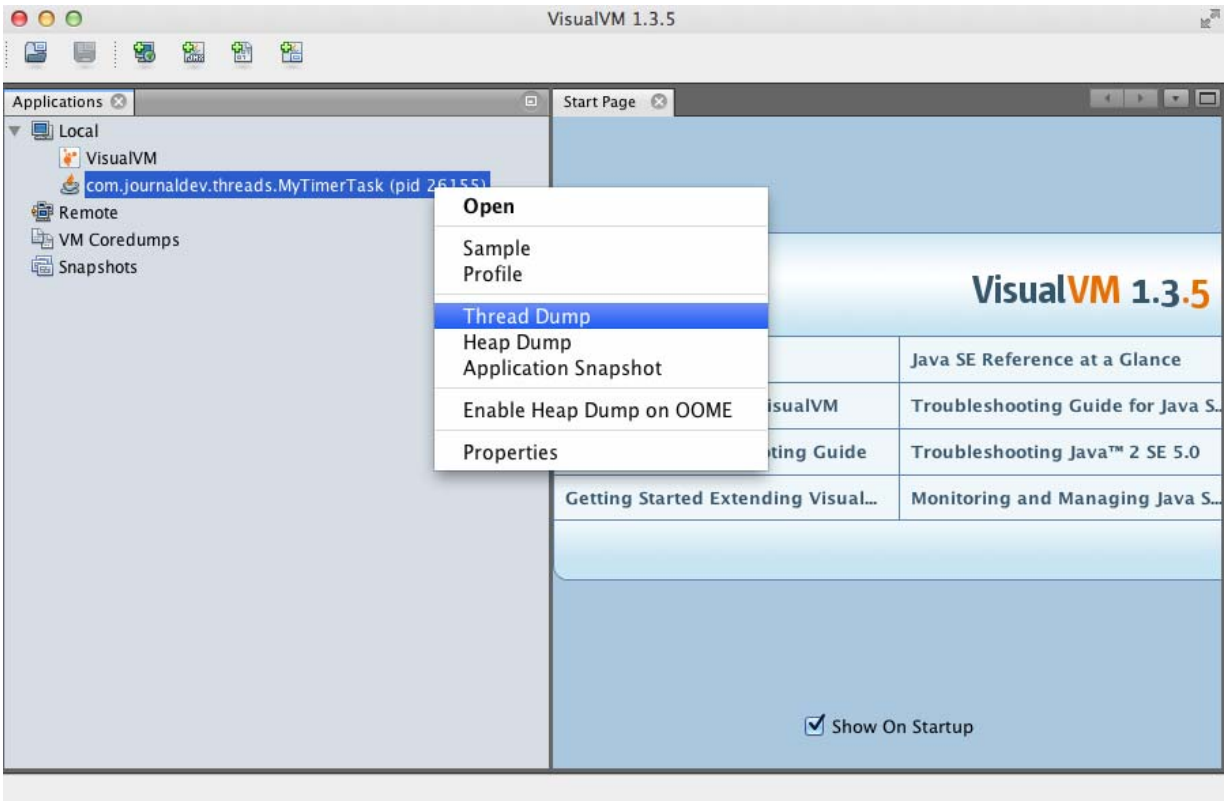
# 多线程、并发编程

## Q1、同步方法还是同步代码块，哪个更可取的方式？

同步代码块是更可取的方式，因为它不需要对整个对象加锁，它可以选择任意资源作为同步监视器来加锁。对于同步方式而言，它总是以当前对象或当前类（对于 static 方法）作为同步监视器，这样就不够灵活。如果类中有多个同步代码块，即使它们不相关，只要为它们选择相同的同步监视器，它也会停止它们的执行，并将它们置于等待状态以获得对象上的锁。
如果需要获取了解同步代码块和同步方法的详细用法，可参考《疯狂 Java 讲义》的 16.5 节。

## Q2、什么是 Java Thread Dump，怎样才能得到一个程序的 Java Thread Dump？

Java Thread Dump 是 JVM 中所有活动线程的列表，线程转储对于分析应用程序中的瓶颈和死锁情况非常有用。我们可以使用很多方法来生成线程转储，比如使用 Profiler、kill-3 pid 命令（对 Linux 才有用）、jstack 工具等。
通常来说，如果只需要简单地生成程序的线程转储，可以考虑使用 jstack，这个工具的优点是简单、易用，而且它是 JDK 自带。缺点是它只是一个基于命令行的工具，对于初学者似乎不太友好，但有时这也是它的优点，我们可以创建一个脚本来来定期生成 Java Thread Dump，以便以后对其进行分析。
如果需要更强大的分析工具，VisualVM Profiler 是一个不错的选择，当你正在分析一个程序为何太慢时，使用 VisualVM Profiler 是一个不错的选择，我们可以非常简单地使用 VisualVM Profiler 来生成 Java Thread Dump，你只要在任何运行的线程上右击鼠标，然后单击“Thread Dump”就可生成 Thread Dump。




# Q3、什么是 BlockingQueue?是否可通过 BlockingQueue 来解决生产者-消费者问题？

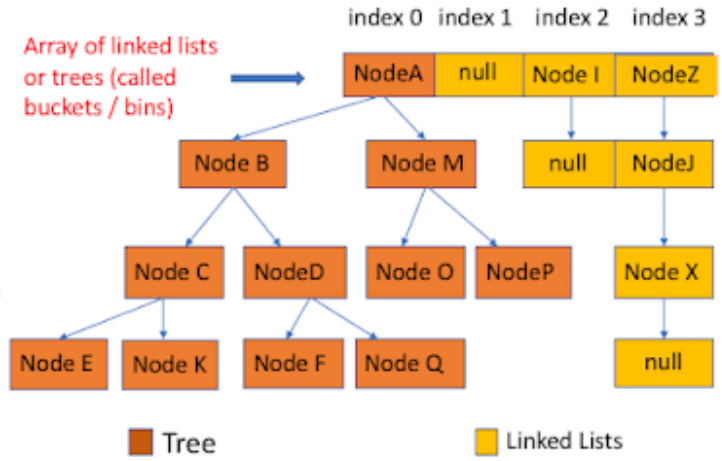
java.util.concurrent.BlockingQueue 是一个队列，它要求在获取和删除元素时队列必须非空，否则就会进入等待；在添加元素时 BlockingQueue 要求队列的空间可用、否则也会进行等待。
如果尝试在队列中存储 null，BlockingQueue 不接受 null 值，并抛出 NullPointerException。
阻塞队列实现是线程安全的。所有的排队方法都是原子的，使用内部锁或其他形式的并发控制。
BlockingQueue 接口是 Java 集合框架的一部分，它主要用于实现生产者-消费者问题。
如果要获得使用 BlockingQueue 解决生产者-消费者的问题，可参考《疯狂 Java 讲义》16.5 节。

# Q4、什么是 FutureTask？

FutureTask 是 Future 接口的基本实现类，通常可和 Executors 一起用于异步编程。大多数情况下，我们不需要使用 FutureTask 类，直接使用 Future 接口即可。但如果我们想覆盖 Future 接口的一些方法，并且想保留大部分的基本实现，FutureTask 就会非常方便。我们可以根据需要继承这个类并重写某些方法。
如果需要参考 Future 的具体编程代码，可参考《疯狂 Java 讲义》16.2 节。

# Q5、你知道 HashMap 中链表转换成树的 threshold 值吗？

在 Java 8 的 HashMap 中可以找到如下一行代码：
static final int TREEIFY_THRESHOLD = 8;
这意味着只要任意一个 bucket 上链表中的元素超过 8 个，这个链表将会被转换为红黑树。
下图显示了 HashMap 底层各 bucket 的节点的示意图，在 index 为 0 的 bucket 上，由于它包含了 11 个节点，因此该 bucket 上的节点变成了红黑树结构；而 index 2 对应 bucket 上只有个节点，因此该节点依然保持链表结构；index 3 代表的 bucket 上只有 3 个节点，因此该节点依然保持链表结构。




# Q6 你知道 HashMap 中树转换成链表的 threshold 值吗？

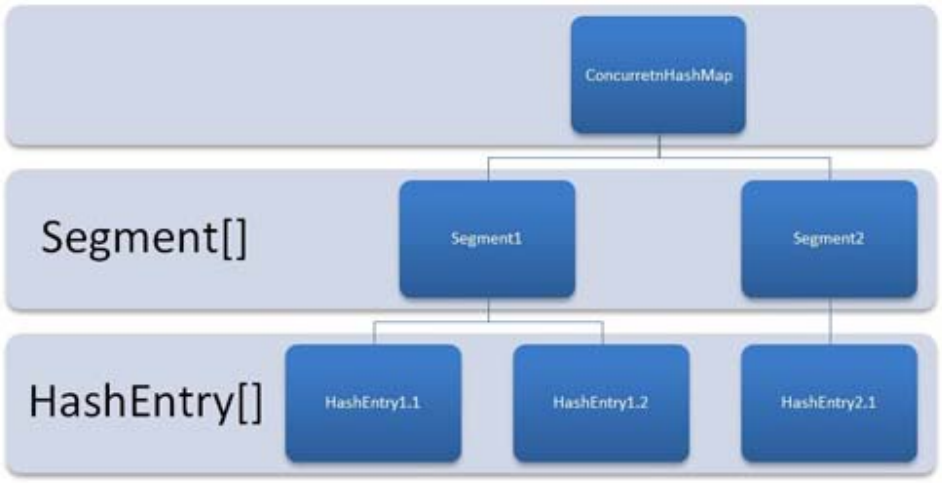
在 Java 8 的 HashMap 中可以找到如下一行代码：
static final int UNTREEIFY_THRESHOLD = 6;
这意味着只要任意一个 bucket 上的红黑树中包含的节点数量小于 6 个，该红黑树就会被转换成链表。
如果需要学习 Java 8 中 HashMapFuture 的源代码实现，可参考《疯狂 Java 面试讲义》3.1 节。

# Q7、HashMap 和 ConcurrentHashMap 的区别

为了更形象地理解 ConcurrentHashMap，可以把 ConcurrentHashMap 当成一组 HashMap 来考虑。当程序要从 HashMap 中获取和放置 key-value 对时，需要先计算 key 对应的 hashCode，然后根据 hashCode 去计算该 key-value 所在的 bucket 位置。

在 ConcurrentHashMap 中，区别在于存储这些键值对的内部结构。ConcurrentHashMap 增加了 segment 的概念。基本上你可以把每个 Segment 近似地当成一个独立的 HashMap，ConcurrentHashMap 在初始化时被划分为若干个 Segment(默认是 16 个)，这样 ConcurrentHashMap 最多允许与 Segment 相同数量(16)的线程同时访问这些 Segment，以便在高并发性期间每个线程在特定的 Segment 上工作。

在这种方式下，假如某个线程要访问 key-value 对保存在 Segment 10th 上，程序只需要对该 Segment 加锁，无需锁定剩余的 15 个 Segment。




ConcurrentHashMap 内部结构

换句话说，ConcurrentHashMap 使用多个锁，每个锁控制一个 Map 的一个 Segment，当在特定的 Segment 中设置 key-value 时，只需对该 Segment 进行锁定，段中设置数据时，将获得该段的锁定。所以基本上更新操作都是同步的。

在获取数据时，只需要使用 volatile 读取，而不需要使用任何同步，如果 volatile 读取导致未命中，则获得该 Segment 的锁，并在 synchronized 代码块中再次搜索该 key-value 对。

## Q8 、 ConcurrentHashMap 与 Collections.synchronizedMap(HashMap)的差别？

这两个 Map 都是线程安全的版本，其区别主要在功能上和内部结构上。

ConcurrentHashMap 内部由一系列的 Segment 组成，每个 Segment 可被视为独立的 HashMap，因此每个 Segment 都可以被单独的线程独立锁定、从而提供高并发执行，在这种方式下，最多允许与 Segment 数量相同的线程来同时读、写 key-value 对，这些线程无需互相阻塞、等待。

对于 Collections.synchronizedMap()，它只是 HashMap 简单的同步版本，因此它使用的也是最简单的同步行为，因此在这种方式下，每次最多只允许一个线程来 Collections.synchronizedMap()进行读、写，其他线程处于阻塞、等待状态。

## Q9、什么是反应式编程

反应式编程是非阻塞的、事件驱动的编程方式，它可用少量线程提供具有伸缩性的扩展，而背压（back pressure）是一个关键因素，用于确保生产者不会压倒消费者。

反应式编程的主要好处包括：

- 提高多核和多 CPU 硬件上计算资源的利用率
- 并通过减少序列化来提高性能

反应式编程通常是事件驱动的，而反应式系统通常是消息驱动的，因此反应式编程并不意味着一定会构建反应式系统，反应式系统是一个架构风格。
通常来说，反应式编程的方式经常用于构建反应式系统。
反应式系统通常具有如下重要特征：

- 响应：系统应及时响应。
- 容错：如果系统面临任何故障，它应该保持响应
- 弹性：反应式系统可以对变化做出反应，并在不同的工作负载下保持响应。
- 消息驱动：反应式系统需要依靠异步消息传递在各组件之间进行交互。

# 后端开发

## Q10、你知道什么是 Spring WebFlux 吗？

Spring WebFlux 是 Spring 的反应式 Web 框架，它可以代替传统的 Spring MVC。
由于 Spring Web Flux 是反应式框架，这种反应式模型具有高度的可伸缩性，且都是非阻塞的，因此 Spring WebFlux 的最大优势在于：能以较小的、固定数量的线程和更少的内存来处理更多的并发请求，因此 Spring WebFlux 可以在高负载的情况具有更好的可伸缩性——因为无需显著增加线程和内存。
通常来说，Spring MVC 适用于同步处理的场景，Spring WebFlux 适用于异步处理的场景，
关于 Spring WebFlux 编程的详细细节可参考《疯狂 Spring Boot 终极讲义》3.6 节。

## Q11、你知道什么是 Mono 和 Flux 吗？

Mono 和 Flux 是 Reactor 框架（Spring WebFlux 依赖该框架）的两个消息发布者接口，因此它们都实现了 CorePublisher<T>接口，它们的区别在于：

- Mono 代表 0～1 个非阻塞数据；而 Flux 则代表 1 个非阻塞序列。
- Mono 相当于只是一个 Optional 值；而 Flux 才是 Stream。

简单来说，Mono 包含多个数据项，而 Flux 能包含多个数据项。Spring WebFlux 一样也要用 Mono 和 Flux 这两个接口。

## Q12、简单介绍一下 WebClient 和 WebTestClient？

WebClient 是 Spring WebFlux 框架中的一个组件，它可以充当执行非阻塞 HTTP 请求的反应式客户端。作为一个反应式客户端，它可以处理带有背压的反应流，并且可以充分利用 Java 8 Lambda 表达式的优势，它还可以处理同步和异步场景。
WebTestClient 也是一个类似的类，我们可以在测试中使用它。基本上，WebTestClient 只是 WebClient 的简单包装，它可以通过 HTTP 连接连接到任何服务器。它还可以使用 Mock request 和 Mock Response 直接绑定到 WebFlux 应用程序，而不需要 HTTP 服务器。

# Q13、你觉得使用反应式编程有哪些缺点。

使用反应流的主要缺点是：

- 调试反应式应用程序时会有一些困难。
- 目前对底层的反应式数据存储的支持有限，传统的关系数据库官方目前还不支持反应式编程范式，但已有一些第三扩展来可以让传统关系数据库支持反应式编程。

如需掌握数据库反应式编程的 R2DBC 方法，可参考《疯狂 Spring Boot 终极讲义》的 5.6 节。

# Q14、Spring 5 是否兼容更早版本的 Java？

为了充分利用 Java 8 的优势，Spring 的代码库已经做了升级，这意味着更早版本的 Java 不再被支持，Spring 5 必须在 Java 8+以上的版本。

# Q15、介绍一下领域驱动设计（Domain Driven Design、DDD）

领域驱动设计主要关注领域对象的核心逻辑，领域启动设计需要不断地与领域专家协作，以解决与领域相关的问题并改进应用程序的领域模型。

在回答微服务面试问题时，还需要提到 DDD 的核心基础知识，包括：

- DDD 主要关注领域逻辑和领域域本身。
- 复杂的设计完全基于领域对象的模型。
- 为了改进领域模型的设计并解决任何新出现的问题，DDD 需要不断与领域专家合作。

# Q16、什么是耦合性和内聚性？

耦合性用于衡量各组件之间的依赖强度，内聚性则用于衡量模块内部各元素保持聚合在一起的程度，一个好的微服务应用程序设计总是由低耦合和高内聚组成。

如果你要参加微服务面试，那么必须记住，设计微服务的一个重要关键是低耦合和高内聚的组合。当松耦合时，服务对其他服务知之甚少。这样可以保持服务完好无损。在高内聚性中，可以将所有相关逻辑保留在服务中，否则，服务将尝试相互通信，从而影响整体性能。

# Q17、什么是 OAuth？

开放授权协议（open authorization protocol，又称 OAuth）帮助使用第三方协议（如微信、支付宝等）通过 HTTPS 访问客户端应用程序。使用 OAuth 还可以在不同的站点之间共享资源，而不需要凭据（credentials）。

OAuth 允许终端用户的帐户信息被第三方（如微信）使用，同时保持其安全性（不需要使用或公开用户的密码）。它就像是代表用户的中介，同时向服务器提供令牌来访问所需的信息。

# Q18、何时需要使用@WebMvCTest 注解？

@WebMvcTest 用于对 Spring MVC 应用做单元测试，顾名思义，它只关注 SpringMvc 组件。例如如下注解，
@WebMvcTest(value = ToTestController.class, secure = false)
该注解只启动 ToTestController 控制器，在该控制器完成单元测试之前，不会启动其他映射和控制器。

# Q19、介绍一下关于@RequestMapping 注解的理解？

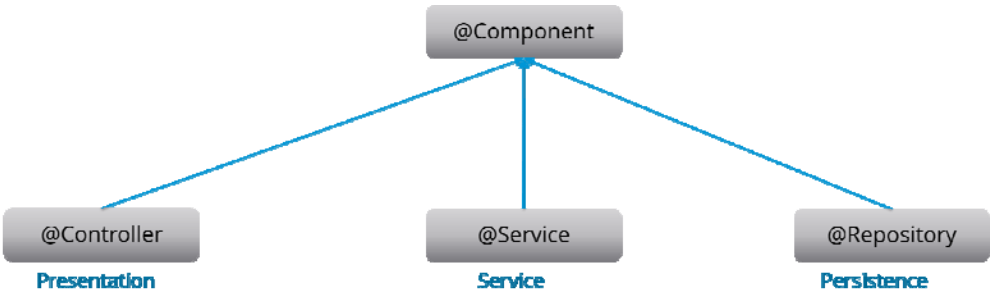
@RequestMapping 注解用于将指定 HTTP 请求的 URL 映射到对应控制器类或方法，该类或方法将处理对应的请求。
该注解可应用于两个级别：

- 类级别：映射请求的 URL。
- 方法级别：映射请求的 URL 和处理 HTTP 请求的处理方法。

简单的来说，被@RequestMapping 修饰的方法就是控制器的处理方法，该方法将负责处理该 URL 对应的 HTTP 请求；如果@RequestMapping 修饰的处理方法又位于被@RequestMapping 修饰的控制器类中，该处理方法将负责所处理的类级别上@RequestMapping 注解指定的 URL 和方法级别上@RequestMapping 注解所指定的 URL 的总和。

# Q20、介绍一下@Component、@Controller、@Repository 和@Service 注解之间的区别？

@Component：该注解修饰的类将作为 Spring 容器中的 Bean。Spring 的组件扫描机制可以自动扫描并识别该注解，并将将注解修饰的类配置成容器中的 Bean。
@Controller、@Service、@Repository 其实都是@Component 的的子注解，因此它们的功能基本相似。




@Controller：该注解修饰的类被扫描成 Spring 容器中的 Bean，主要是作为 Spring MVC 的控制器。
@Service：该注解修饰的类将被扫描成 Spring 容器中的 Bean，主要是作为 Service 组件。
@Repository::该注解修饰的类将被扫描成 Spring 容器中的 Bean，主要是作为 DAO 组件。

## Q21、双重授权有哪些不同类型？

有三种类型的凭证可用于完成双重授权

- 任何客户能记住的东西，比如密码或屏幕解锁图案。
- 任何客户拥有的物理凭证，比如 OTP 或 ATM 卡。换而言之，它可以是外部或第三方设备中拥有的任何类型的凭据。
- 客户的身体特征，比如语音识别或生物特征，如指纹或眼睛扫描仪（eye scanner）

## Q22、什么是客户端证书

客户机证书是一种数字证书，客户机系统使用它向远程服务器发出经过身份验证的请求。它在许多相互认证设计中起着关键作用，为请求者的身份提供了强有力的保证，当然也需要配置一个完全的后端服务来验证客户的客户端证书。

# 微服务

## Q23、如何覆盖 Spring Boot 项目的默认属性？

可以通过在 application.properties 文件中指定属性来完成配置。
例如，在 Spring MVC 应用程序中，必须指定后缀和前缀。这可以通过在 application.properties 中输入如下属性来完成配置.
配置后缀 – spring.mvc.view.suffix: .jsp
配置前缀 – spring.mvc.view.prefix: /WEB-INF/

## Q24、介绍一下 Spring Boot 中 Actuator 的角色？

Actuator 是 Spring Boot 最重要的特性之一，可以帮助访问在生产环境中运行的应用程序的当前状态，它可以使用多个度量（Metrics）来检查当前状态。它们还提供 RESTful Web Service 接口，可用于简单地检查不同的度量（Metrics）。

## Q25、Spring Boot 支持哪些嵌入式容器？

Java 应用程序可通过两种方法进行部署：

- 通过使用外部的容器。
- 使用嵌入到 JAR 包中的容器。

SpringBoot 包含 Tomcat、Jetty、undertow 和 Reactor Netty 服务器，所有这些服务器都是嵌入式的。
Jetty：用于许多项目中，Eclipse Jetty 可以嵌入到框架和应用服务器、工具以及集群中。
Tomcat：ApacheTomcat 是一个开源的 Web 服务器，可以很好地与嵌入式系统配合使用。

Undertow：一个灵活而突出的 Web 服务器，它使用小型单一处理程序来开发 Web 服务器。
Reactor Netty：Spring WebFlux 默认使用 Reactor Netty 作为嵌入式服务器。

# Q26、介绍一些微服务中的端对端测试？

端对端测试会验证工作流中的所有流程，以检查是否一切都按预期正常工作。它还确保系统以统一的方式工作，从而满足业务需求。
对微服务应用程序进行端对端测试可以确保每个进程都正常运行，这验证了整个系统是否正常工作。由于微服务应用程序是给予动态编排的多个模块构建的，因此端对端测试必须覆盖所有服务之间的通信。
端对端的微服务测试的步骤包括：

1. 定义对 e2e 测试的期望。
2. 定义要测试的系统的范围。
3. 在测试环境中执行身份验证。
4. 选择一个能解决绝大部分问题的测试框架。
5. 测试异步流
6. 自动化测试

# Q27、什么是语义监控？

语义监控可将整个应用程序的监视与自动化测试结合起来。语义监控的主要作用就是帮助找出对业务更有利的因素。
语义监控和服务层监控从业务的角度对微服务进行监控。一旦发现问题，它们允许执行更快的隔离和错误分类，从而减少修复所需的时间。它对服务层和事务进行分类，以便找出受可用性或较差性能影响的事务。

# Q28、如何设置服务发现

有多种方法可用于设置服务发现。通常会选择一个最简单、有效方式：使用 Netflix Eureka 做服务发现，这是一个简单的过程，对应用程序没有太大影响。此外，它还支持多种类型的 Web 应用程序。
Eureka 配置包括两个步骤：客户端配置和服务器配置。
客户机配置可以通过使用属性文件轻松完成，Eureka 会自动搜索类路径中的 eureka-client.properties 文件. 它还会搜索针对特定于环境的属性文件对默认 eureka-client.properties 文件进行覆盖。
对于服务器配置，必须首先完成客户端配置。客户端配置完成后，服务器启动一个用于查找其他服务器的客户机。默认情况下，Eureka 服务器使用客户机配置来查找对等服务器。

# Q29、为什么选择微服务架构

微服务体系结构提供了很多优点。以下是其中一些：

- 微服务可以很容易地适应其他框架或技术。

- 单个进程的失败不会影响整个系统。
- 为大企业和小团队都提供了支持。
- 可以在相对较少的时间内完成独立部署。

## Q30、在微服务中为什么需要 Report 和 Dashboard？

Report 和 Dashboard 主要用于监视和维护微服务。有多种工具可以帮助实现这一目的。
Report 和 Dashboard 可用于：

- 找出哪个微服务暴露了哪些资源。
- 找出在一个组件在发生更改时所影响的服务。
- 提供一个方便点，在需要文档时可以访问该点。
- 查看已部署组件的版本。
- 获取组件的成熟度和遵从度。

## Q31、PACT 是怎么工作的？

PACT 是一个开源工具。它用于帮助测试消费者和服务提供者之间的交互。
消费者程序的开发人员首先编写一个测试，定义消费者程序与服务提供者之间的交互模式。测试包括服务提供者的状态、请求主体和预期的响应。在此基础上，PACT 创建一个存根来执行测试，测试输出被存储在 JSON 文件中。

## Q32、为什么需要用于微服务的容器？

要管理基于微服务的应用程序，容器是最简单的选择。它可以帮助用户单独部署和开发。您还可以使用 Docker 将微服务封装在容器的映像中。微服务就可以使用这些元素几乎不需要管理任何额外的依赖关系。

## Q33、如何访问 RESTful 微服务？

考虑到微服务架构的概念，每个微服务都需要一个访问接口。基于 Web 开放网络的原则，RESTful API 为在微服务体系结构的各个组件之间构建接口提供了最合理的模型。RESTful API 可以通过如下方式访问：

- 使用负载平衡的 RestTemplate。
- 使用 WebFlux 的 WebClient。
- 使用 Feign。

## Q34、你觉得微服务测试的主要障碍有哪些？

这个问题是也是微服务所面对的另一个挑战。微服务测试主要面临以下几点挑战

- 测试人员在开始编写集成测试的测试用例之前，应该对所有 inbound 和 outbound 过程有一个透彻的了解。
- 当独立的团队处理不同的功能时，协作可能是一项非常困难的任务。很难找到一个空闲时间窗口来执行一轮完整的回归测试。
- 随着微服务数量的增加，系统的复杂性也随之增加。
- 从单体架构向微服务架构过渡的过程中，测试人员必须确保组件之间的内部通信没有中断。

# Q35、微服务设计的基础是什么？

这应该是微服务面试中最容易遇到的问题之一。回答该问题应该紧紧把握以下几点原则：

- 定义范围
- 将松耦合与高内聚结合起来
- 创建一个唯一的服务，它将充当标识源，就像数据库表中的唯一键一样
- 创建正确的 API 并在整合过程中特别小心。
- 限制对数据的访问、并将其限制到所需的级别
- 保持请求和响应之间的流畅
- 自动化大多数流程以降低时间复杂性
- 将表的数量保持在最小水平以降低空间复杂性
- 持续监控体系结构，一旦检测到任何缺陷，及即进行修复。
- 每个微服务的数据都应该分开存储。
- 对于每个微服务，都应该有一个独立的构建。
- 将微服务部署到容器中。
- 服务器应被视为无状态的。

# Q36、谈谈你对 Bounded Context 的理解？

Bounded Context 是领域驱动设计中常见的一种中心模式，Bounded Context 是 DDD 策略设计部分的主要关注点。这都是和大型团队和大型模型的有关的内容。DDD 通过将大型模型分解为多个 Bounded Context 来处理它们，这样也能明确地解释了它们之间的关系。

# Q37、Melvin Conway 法则暗示了什么？

Melvin Conway 法则规定“设计系统的组织必须生产这些组织的通信结构的复制品。”
面试官可能会问一个反微服务面试问题，比如 Melvin Conway 法则与微服务有什么关系。一些松耦合的 API 构成了微服务的体系结构。这种结构非常适合于小团队如何实现自主的组件。这种架构使组织在重组其工作流程时更加灵活。

# Q38、怎么配置 Spring Boot 应用的日志？

Spring Boot 自带了对 Log4J2、JUL 日志和 LogBack 的支持，它的日志配置默认为控制台输出。
Spring Boot 日志的默认输出级别是 INOF，也可通过指定在 application properties 文件中配置 logging.level 属性来配置日志的输出级别
logging.level.spring.framework=debug
关于 Spring Boot 日志的更多详细配置可参考《疯狂 Spring Boot 终极讲义》2.6 节。

# Q39、微服务的主要特点有哪些？

微服务有如下主要特点：
- 微服务包含多个可独立部署的组件。
- 各服务是基于业务功能分开。
- 分散的数据管理。
- DevOps 实现。
- 技术独立性。
- 隐藏组件复杂性以避免不必要的微服务依赖。

# Q40、微服务包含哪些主要组件？

微服务架构包含如下主要组件：
- 容器、集群和编排
- IaC（基础设施作为代码概念）
- 云基础设施
- API 网关
- 企业服务总线（ESB）
- 服务交付

# Q41、微服务架构是如何工作的？

在微服务架构下，应用程序被简化为多个模块，这些模块独立执行单个精确的独立任务：
被分割成松耦合的不同模块，每个模块执行不同的功能。
它可以分布在云和数据中心中。
将每个模块实现为一个独立的服务/流程，可以在不中断应用程序其余部分的情况下进行替换、更新或删除。
在微服务架构下，应用程序可以随着其需求而增长。

# Q42、微服务设计的基本特征是什么？

服务围绕业务功能进行拆分和组织。
由不同的开发团队处理和拥有的独立模块。
分散的框架。
由各自的开发团队维护各自的模块。
不同的数据库可以维护不同的模块。
微服务架构中的模块可以单独部署。它们可以在不中断整个体系结构的情况下进行更新、增强或删除。
实时监控应用程序。

# Q43、微服务部署时有哪些挑战？

微服务部署的挑战既有技术上的，也有功能上的。
从商业角度来看，主要挑战包括：

- 需要大量投资
- 需要大量基础设施的设置
- 管理运营开销的过度规划

人员选择和维护。

- 从技术角度来看，主要挑战包括：
- 应用程序中不同微服务之间的通信。
- 组件自动化
- 应用程序维护
- 配置管理
- 繁重的操作开销
- 部署挑战
- 测试和调试挑战

# Q44、微服务有那些优点？有哪些缺点？

优点：

- 提高的的可扩展性。
- 故障隔离。
- 复杂性被限制在模块内部。
- 提高了敏捷性
- 简化调试和维护
- 为开发人员与业务用户之间提供了更好的响应。
- 可将大型项目分解成多个小型开发团队
- 更好的技术升级空间。

缺点：

- 整体上很复杂。

- 需要准确的预先规划
- 模块依赖关系很难计算。
- 降低了对第三方应用程序的控制
- 模块之间的相互依赖很难被跟踪。
- 被恶意入侵的机会变得更多。
- 完整的端到端测试变得更困难。
- 部署难度增大。

## Q45、微服务有哪些不同的部署策略？

微服务常见的有如下部署策略：

- 一个主机运行多个服务实例：在单个物理或虚拟主机上运行应用程序的单个或多个服务实例。
- 每个主机运行一个服务实例：每个主机运行一个服务实例。
- 每个容器运行一个服务实例：在其各自的容器中单独运行每个服务实例。
- 无服务器部署：将服务打包为 ZIP 文件并将其上载给 Lambda 函数。Lambda 函数是一个无状态服务，它自动运行所有微服务来处理所有请求。

## Q46、简单说明一下单体应用、SOA 和微服务体系架构之间的区别。

在单体应用结构中，应用程序的所有组件都是紧密组装和打包在一起的。
SOA（面向服务的体系结构）是通过简单的数据传递或活动协调来相互通信的服务集合。
微服务体系结构是多个小型功能模块的集合。这些功能模块是可独立部署的、可扩展的、针对特定业务目标的，并通过标准协议相互通信。

## Q47、列出单体应用、SOA 和微服务体系架构之间的区别？

- 单体结构：在这种体系结构中，应用程序的不同组件（如 UI、业务逻辑组件、数据访问组件）被组合到单独的一个应用或平台中。
- SOA（面向服务架构的体系结构）：在这个体系结构中，各个组件是松散耦合的，并执行一个各自的功能。SOA 架构主要有两个主要角色：服务提供者和服务使用者。在 SOA 架构中，模块可以被整合和复用，从而使得 SOA 架构灵活可靠。
- 微服务架构（Microservices Architecture）：它是一种 SOA，其中构建并组合了一系列自主组件以生成应用程序。这些组件通过使用 API 来进行整合。这种方式侧重于业务优先级和功能，并提供高度的灵活性，应用程序的每个组件都可以独立于其他组件构建。

## Q48、介绍以下 Spring Cloud 所解决的问题？

Spring Cloud 可以解决以下问题：

- 网络问题、延迟开销、带宽问题、安全问题以及分布式系统的其他问题
- 分布式系统中出现的冗余问题。
- 平衡各种资源（如网络链路、CPU、集群等）之间的负载分配。
- 操作开销导致的性能问题。
- 服务发现问题，以确保群集中服务之间的平滑通信。

# Q49、什么是分布式事务？

分布式事务有两个或多个参与的网络主机，事务由负责开发和处理事务的事务管理器处理，如果事务涉及多个对等的事务资源，则每个事务资源方的事务管理器使用下级或上级关系相互通信。

同样，资源由资源管理器处理，它还与分布式事务协调器协调事务原子性和隔离性。

# Q50、什么是 Mike Cohn 测试金字塔？

Mike Cohn 测试金字塔助于最大限度地自动化所有级别的测试，即单元测试、Service 级别测试、UI 测试。金字塔表明，虽然单元测试更快、更孤立，但处于最高级别的 UI 测试需要时间并专注于整合。
Mike Cohn 测试金字塔描述了软件开发所需的自动化测试类型。测试金字塔只是一个隐喻，它暗示了一组基于粒度的测试。测试金字塔告诉我们应该对金字塔的不同层次应用哪种测试。

Mike Cohn 测试金字塔由三层组成，每层都包含对应的测试套件（test suite）：
- 单元测试
- Service 测试
- UI 测试。

通过 Mike Cohn 测试金字塔可以得到两点：
- 定义不同粒度的测试
- 组件所在级别越高，应该做的测试就越少。

# Q51、在 Spring Boot 应用中怎么实现 Spring Security？

先在 pom.xml 文件中添加 spring-boot-starter-security。
然后创建一个继承 WebSecurityConfigurerAdapter 的 Spring Config 类，该配置类重写所需 configure()方法，通过该方法中的 AuthenticationManagerBuilder 参数设置授权所使用 AuthenticationProvider，如果用户信息存储在数据库，通常要使用 DaoAuthenticationProvider 实现类。

# Q52、什么是契约测试？

契约测试确保微服务体系结构的显式和隐式契约都能按预期工作。契约测试有个角度——服务消费者和

服务提供者。服务消费者是使用微服务的程序组件，服务提供者是提供微服务的程序组件。这些服务组件应该在预定义的规范下工作，而契约测试则用于确保了这一点。

## Q53、谈谈你对 Conway 定律的理解？

Melvin Conway（梅林.康维）在 20 世纪 60 年代末提出了这个想法。康维定律的原文是：
Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations. - Melvin Conway
直接翻译过来就就是：设置系统的组件受限于他们所做到设计，而这些设计就是这些组织的通信结构的翻版。
通俗来说，就是说：设计软件的团队最终将按照他们的观点进行设计。说得再通俗一点就是：什么样的团队搞出什么样的系统设计。反过来也是成立，如果一个软件系统被设计得非常糟糕，那么这个团队内部的通信结构也是混乱的，反而亦然。

## Q54、Mock 和 Stub 有什么区别？

Mock 通常是一个模拟对象，该模拟对象在初始化时就设置了某些特征。它的行为主要取决于这些特征，Mock 对象通常用于模拟某些还未开发出来的或未经测试的程序组件。
存根是帮助运行测试的对象。它在一定条件下以固定的方式工作。通常来说，存根程序可使用 Mock 对象来充当。

## Q55、请问 Docker 对微服务有什么帮助？

正如我们所知，微服务是自包含的、独立的单元，每个独立的单元只完成一个业务功能，以至于每个单元都可以被视为一个独立的应用程序。
应用程序开发环境和部署环境必然在许多方面有所不同，这会导致很多部署方面问题，Docker 为应用程序运行提供了一个静态背景，从而避免了部署问题。
事实上，Docker 是一种类似集装箱的工具，它减少了开销，并在同一台服务器上部署了数千个微服务。Docker 确保应用程序微服务将在其自己的环境中运行，并且完全独立于其操作系统。

## Q56、什么是 Canary Releasing?

Canary releasing 是一种新的软件版本发布技术，这种发布技术就是先软件的更新版本发布给一小部分用户、要求这部分用户进行测试，然后再将其发布到整个基础设施并使其对所有人都可用。这项技术之所以被称为“Canary”，是因为它是基于煤矿中的 canary releases，当有毒气体达到危险水平时向矿工发出警报。

# Q57、什么是消费者驱动契约（CDC，Consumer Driven Contract）

消费者驱动的契约是逐步形成服务的模式。在这里，每个消费者在单独的契约中捕获它们的服务提供者。然后，所有这些契约都与服务提供者共享，这有助于它们深入了解它们必须为每个客户提供的功能。

# Q58、什么是不确定性测试，你怎么才能消除它们？

不确定性测试是不可靠的试验。这种测试有时通过，有时失败。当这些测试失败时，它们将重新运行以给出结果。
测试中的不确定性可以通过以下方式消除：

- 异步
- 远程服务
- 隔离
- 时间
- 资源泄漏

# Q59、什么是微服务中的反应式扩展？

反应式扩展是一种设计方法，通过调用多个服务来收集结果，然后将多个结果组合成最终响应，也称为 Rx，这些调用可以是同步的，也可以是异步的。

# Q60、RESTful API 在微服务中的作用是什么？

微服务是基于这样一个概念，即它的所有组件服务都需要彼此交互才能完成业务功能，这要求每个微服务都需要对外提供一个接口，而 RESTful API 则构建这些接口提供了逻辑模型，它基于 Web 的开放网络原则，因此，它是微服务最关键因素。

# Q61、Eureka 在微服务中起什么作用？

Eureka 是 Netflix 开发的服务发现框架，一个 Eureka 可分为 eureka server 和 eureka client。其中 eureka server 是作为服务的注册与发现中心。eureka client 既可以作为服务生产者，也可以作为服务消费者。
微服务系统使用 Eureka 来注册、发现整个系统中各服务组件。

# Q62、如何利用 Spring Cloud 在服务端做负载均衡？

服务器端负载平衡可以通过 Netflix Zuul 来完成，它也被称为基于 JVM 的路由器。

# Q63、你认为什么时候适合使用 Hystrix?

Netflix Hystrix 是一个容错和熔断保护中间件，Netflix Hystrix 主要用于隔离接入点，它还可确保所有第三方库和服务受到限制。因此，可以使用 Hystrix 来确保应用程序高效运行，并避免在分布式系统中发生的故障。

# Q64、什么是 Spring Batch Framework?

Spring Batch 是一个用于批处理的开源框架——它可用于执行一系列作业。spring Batch 提供了系列类和 API，用于读/写资源、事务管理、作业处理统计、作业重启、以及用于处理大量数据的分区技术。

# Q65、什么是 Tasklet？什么是 Chunk？

Tasklet 是一个简单的、只有一个方法可执行的接口，Tasklet 可用于执行单个任务，如运行查询、删除文件等。在 Spring Batch 中，Tasklet 是一个接口，可用于在任何步骤执行之前或之后执行一个唯一的的任务，如初始化或清理资源。
Spring Batch 在其最常见的实现中使用了“面向块”的处理风格。面向块的处理是指一次读取一个数据块，并在事务边界内创建将被写出的块。

# Q66、怎么在微服务中进行异常处理？

如果在处理 HTTP 请求时发生异常，则需要在控制器或 Service 组件中捕获异常，并手动返回对应的 ResponseEntity。
下面是一些异常处理的经验法则。

- 为自定义的异常类添加@ResponseStatus 注解。
- 对于所有控制器都可能出现的通用异常，请在@ControllerAdvice 类或@RestControllerAdvice 类中使用@ExceptionHandler 注解修饰异常处理方法或使用传统的 SimpleMappingExceptionResolver 进行异常处理。
- 对于指定控制器特定的的异常，将@ExceptionHandler 修饰的异常处理方法添加到控制器中。
- 需要注意的是，控制器中的@ExceptionHandler 方法总是在任何@ControllerAdvice 实例中的@ExceptionHandler 方法之前执行。

如果希望详细学习@ResponseStatus、@ControllerAdvice、@RestControllerAdvice、@ExceptionHandler 等注解的用法，可参考《轻量级 Java Web 企业应用实战》7.2 节。

# Q67、如何访问 RESTful 微服务？

考虑到微服务架构的概念，每个微服务都需要一个访问接口。基于 Web 开放网络的原则，RESTful API 为在微服务体系结构的各个组件之间构建接口提供了最合理的模型。RESTful API 可以通过如下方式访问：

- 使用负载平衡的 RestTemplate。
- 使用 WebFlux 的 WebClient。
- 使用 Feign。

## Q68、独立的微服务之间彼此如何通信？

微服务可通过以下方式相互通信：

- 对于传统的请求-响应可使用 HTTP 进行通信。
- 对于实时性要求较高的数据流可使用 WebSocket
- 使用运行高级路由算法的服务器程序或代理（Broker）
- 对于消息代理，则可以使用 RabbitMQ、Kafka 等，它们都是为特定的消息语义而构建的。

## Q69、如何对微服务进行安全测试？

Microservices 应用程序是系列较小的、独立的、功能模块的集合，这些模块可以用不同的编程语言开发，具有不同的数据源，并运行在不同的操作系统上。这使得将微服务应用程序当成整体进行测试变得非常困难。因此不同的部件/模块需要进行单独测试，这有三种常见的测试过程：

- 代码扫描：为了确保每一行代码都没有 bug 并且可以复制。
- 伸缩性：安全协议应该根据每个系统要求具有一定的伸缩性。
- 适应性：安全协议应该能适应恶意入侵。

## Q70、介绍下你对幂等性（Idempotence）的理解，并简单说下如何使用它？

幂等性指的是对一项任务进行多次重复执行，而最终结果总保持不变。它主要用作数据源或远程服务，当它多次接收指令时，只处理一次指令。

## Q71、Report 和 Dashboard 在微服务环境中有什么作用？

Report 和 Dashboard 通常用于监视系统，对于微服务环境，Report 和 Dashboard 主要提供如下功能：

- 找到哪个微服务支持哪个资源。
- 找出在组件发生更改时受影响的服务。
- 为文档目的提供一个方便的访问点。
- 查看已部署组件的版本。
- 从组件获得合规性。

# Q72、管理微服务架构可以考虑哪些工具？

可用于构建/管理微服务体系结构的主要工具有：
MongoDB：它是一个基于文档的开源分布式数据库。在这里，数据以 JSON 格式存储，不同的文档具有不同的结构。它还支持许多编程语言，如 C、C++、C 语言、Perl、PHP、Python、java、Ruby、Scala 等。
Elasticsearch：它是一个基于 Lucene 的全文搜索引擎。
Kafka：它是一个开源的、分布式事件流系统。所有事务都通过事件队列进行处理，从而避免了不同服务之间类似 Web 的随机交互。Kafka 呈现了一个健壮而干净的微服务架构。
Jenkins：它是一个自动化工具，支持持续集成和持续开发。它支持许多插件，并且可以轻松地与几乎所有工具集成。
Docker：应用程序开发环境和应用程序部署环境在许多方面必然会有所不同，这就会导致应用部署方面的问题。而 Docker 为应用程序运行提供了一个静态背景，从而避免了各种部署问题。
K8S：当一个应用程序中运行着数千个服务，K8S 可作为一个引擎来协调整个过程。
Jaeger：它是一个开源的端到端分布式跟踪工具。Jaeger 监视分布式事务，帮助优化性能，并查找服务之间的依赖关系。并进行了根本原因分析。
Fluent：在多服务体系结构中，所有不同的系统都通过不同的编程语言、不同的数据库进行管理，并在不同的操作系统中运行，登录和跟踪它是一个重要的问题。Fluent 提供了一个日志记录层，简化了这个问题。还可以收集日志并将其聚合到数据源中。
PROMETHEUS：它是一个监控工具，可以帮助检查应用程序部署时是否所有服务都正常工作。它是一个时间序列数据存储。它从应用程序收集度量并以图形格式显示。
grafana：它提供分析和监控到不同的可视化格式，如图形、图表、表格等。
Nginx：它充当反向代理。它充当一个单点入口，所有 API 调用都是通过这个入口进行的。