趋势报告框架

第一部分：Java的技术采用生命周期

这部分采用与英文站同样的标准划分：

- 创新者
- 早期采用者
- 早期大众
- 晚期大众

技术采用生命周期是美国高科技营销大师杰弗里·摩尔在自己的书《跨越鸿沟》里提出的概念。技术采用生命周期是一个用来衡量用户对某项新技术接受程度的模型，它认为一个新的技术，从一开始出现到最后走向成熟，必然会经历创新者、早期采用者、早期大众、晚期大众的阶段。

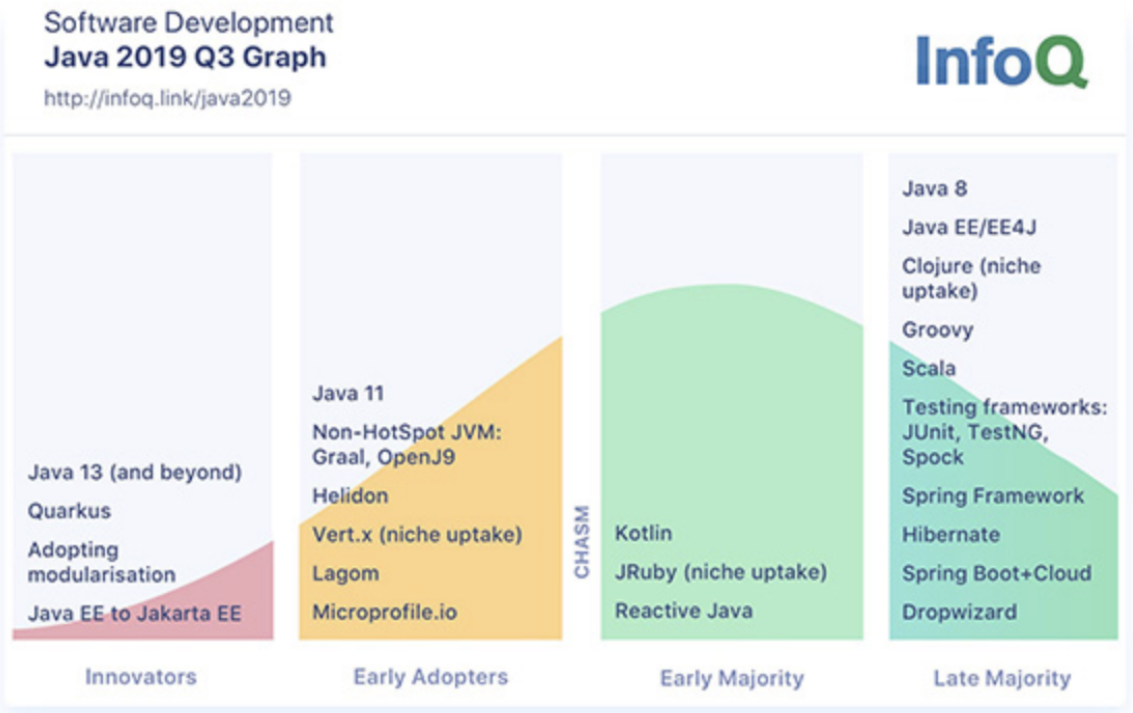
虽然每个人群间都会有裂缝，但是早期采用者和早期大众之间的那条裂缝最大，这条裂缝就是传说中的“鸿沟”，只有跨越过这条鸿沟，渗透到早期大众这个人群，产品才等于是进入了主流市场。

希望您结合国内使用和发展情况，把以下技术对应到技术采用生命周期相应的不同阶段中：

- Java/JVM
    - Java版本（8～13）；
    - OpenJDK定制版或者公开发行版，Oracle JDK, OpenJDK by Oracle/Redhat/Azul/Alibaba/Amazon, 或者其他；
    - 非Hotspot JDK生产实践，如GraalVM、IBM OpenJ9；
    - 语法与特性，例如：Lambda /Stream、Vector API等（可以从是否有哪些特性带来了突出甚至不可替代的生产价值的角度，评判他们在技术采用生命周期中的位置）；
    - JVM语言，Kotlin、Scala、Groovy、其他；
- 不同层次的主流框架：Java EE（Jakarta EE）、Spring Framework、RxJava 、Vert.x、Netty；
- 微服务领域：Spring Boot/Cloud、Dubbo、TarsJava、ServiceComb、其他

> 小马哥（@mercyblitz）：

创新者	早期采用者	早期大众	晚期大众
Java 13	Java 11	OpenJDK	Java 8
Jakarta EE	GraalVM	Reactive Streams	Lambda/Stream
Apache Dubbo (ECO System)	Vert.x	Kotlin	Scala、Groovy
	TarsJava	RxJava/Reactor	Java EE（Jakarta EE）、Netty
	ServiceComb		Spring Framework
			Spring Boot/Cloud
			Apache Dubbo



InfoQ英文站结果供参考

第二部分：Java趋势点评

对第一部分中您所归类的处于不同阶段的技术，请您逐一做出如下点评问题包括：

- 某个技术为什么要被划在这个技术采用生命周期内？这个技术在国内的发展情况以及机遇和挑战是什么？

> 小马哥（@mercyblitz）：
>
> - Java / JVM 语言 - Java 8 已被业界普遍接受，无论像 Spring Framework、Spring Boot，以

及 Spring Cloud 这样的现代 Java 框架，还是类似于 Vert.x、RxJava 或 Reactor 这类小众框架，均已构建在 Java 8 以及更高的版本之上。同时，Lambda 语法以及 Stream API 也在开发人员的日常工作中广泛地运用，并且没有看到语法回退的趋势。因此，Java 8 和 Lambda/Stream 可归类为“**晚期大众**”。 JVM 语言 Scala 和 Groovy 已快成为明日黄花，往昔的光芒逐渐地被后期之秀 Kotlin 替代，故 Scala 和 Groovy 属于“**晚期大众**”，而 Kotlin 则纳入”**早期大众**“ 之流。然而 Java 9 的被接受程度则没有那么幸运，尽管我们等待它的到来已有数年。 Java 模块化作为 Java 9 核心的特性，就我个人而言，完全能够理解和接受它的设计。虽然模块化增强了模块的隔离性，减少了内存的 Footprint，然而，它更强的封装性无形之中增加了管理依赖的成本。所谓曲高和寡，夸张地说，模块化形同虚设。因此，应用升级 Java 9 的效果相当于 Java API 以及 JVM 的更新。同时，Oracle 宣布从 Java 9 开始，每半年将更新一个 Java 大版本。那么，更多的人会选择 Java 11 这样的长期支持（Long-Term - Support, LTS）版本，换言之，Java 9/10 则成了过渡版本（non-LTS）[1]。因此，Java 11 将是未来 Java 用户的最可能选项，将其列为“早期采用者”。至于 Java 13，最近有注意到该本版在新 GC 算法的提升以及 Socket 实现上的变化[2]，还是非常令人期待的，故排在“**创新者**” 之列。除了模块化之外，Java 9 还有大量的 API 更新，在偏开发侧的部分，Flow API 可能是最有吸引力的，提供了 Reactive Streams[3] 标准接口以及实现，并且内建了 HTTP Client Reactive 实现。尽管 Spring 和 Eclipse 社区大力的推广，并且 Java Lambda 以及 Stream API 也流行开来，不过对开发者而言，Reactive Streams 技术还是相对陌生，将其放在 “**早期大众**“。同理，Spring 引入的 Reactor 框架也属于“**早期大众**” 阶段。虽然 RxJava 并非 Reactive Streams 的实现，但是相较于前两者而言，它在 Reactive 编程中的地位必然是有过之而无不及的，故也在 “**早期大众**“ 之中。

- OpenJDK - 由于 Oracle 宣布 2019 年伊始，Oracle JDK 8 以及更高版本在服务器端部署不再免费，OpenJDK 则成为大多数 Java 用户的选项。尽管 Oracle JDK 与 OpenJDK 几乎出至于同一家之手，不过OpenJDK 很可能被认为是一种退而求其次的选择。对于具备自主研发的企业，它们可能选择在 OpenJDK 的基础上，自定义分支继续开发。在一定程度上，Java 的发展方向出现了裂痕，所以未来仍存在着不确定性，故将其放置于 “**早期大众**“
- 非Hotspot JDK生产实践 - 按照 GraalVM 官方的描述，GraalVM 将会是下一代的 JVM 基础设施，也是 Oracle 的重点项目，能够将传统的 JVM 进程 native 化，那么未来 Java 的性能提升以及快速启停则不再遥远，不够目前还尚不可知其兼容性情况以及明确的商业化条款，列为“**早期采用者** 应该是合理的

- 不同层次的主流框架
    - Java EE - 在 Java 生态中，绝大多数应用直接或间接地使用了 Spring Framework，这个曾经以轻量级著称的框架，目前显然“名不副实”，不过巨大的用户基础，早已进入“**晚期大众**”的行列。相反，Java EE 规范或者重组后的 Jakarta EE[4] 则寂寥许多。实际上，作为 J2EE 或 Java EE 的模仿者，尽管 Spring Framework 多半的特性和实现向 JSR[5] 参考。一定程度上，Spring Framework 的流行“压缩”了 Java EE 以及 JSR 的认知空间，因此不少的开发人员不知道 Java EE API 或者 JSR 的存在，尤其是年轻的国内从业人员。当然，Spring 也反哺了少量标准给 Java EE，比如 JSR 330[6]，因此，将 Java EE 列为 **“晚期大众**” 是毋容置疑的。值得一提的是，Jakarta EE 在 Eclipse 基金会[7]的带领下能否再次“伟大” 值得期待，后续未来 Jakarta EE 将是“**创新者**”成员
    - 网络框架 - Java 的网络框架只有两种，其一是 Netty，其二则是其他。如此描述也丝毫不夸张，毕竟大多数与网络相关的框架或中间件都和 Netty 多少存在关联，如 Apache Dubbo、Spring 5 Web Server、Jersey 等，所以**“晚期大众**”的排行众望所归。
    - 微服务框架 - Java 微服务框架的王者非 Spring Boot 和 Spring Cloud 莫属，进过数年

> 的生产使用，两者早已属于“**晚期大众**” 的技术栈。相对应地，Apache Dubbo 出现和开源的时间比 Spring Cloud 早不少，无论是性能还是稳定性，相对于后者要优秀不少，因此，Apache Dubbo 也属于 “**晚期大众**” 框架。不过，最新的 Apache Dubbo ECO System（生态系统）则基于 Apache Dubbo 衍进的 Cloud Native 解决方案，目前尚未枝叶茂盛，处于 “**创新者**” 阵营。而小众 的 Vert.x 则由于编程模型以及 API 熟悉度等客观条件所限，它不得不仍处于 ”**早期采用者**”。类似，TarsJava 以及 ServiceComb 最近才出现，用户的认可度和稳定性稍微成熟，同样处于 ”**早期采用者**”。

- 国内是否在某个相对完整的领域，形成甚至开始引领技术趋势?

> 小马哥（@mercyblitz）：在国内的开源软件中，Apache Dubbo（后文简称 Dubbo）常年受到业界的青睐，并荣获多次殊荣。个人认为 Dubbo 有机会引领技术趋势。Dubbo 从过去的高性能 RPC 框架，正在走向 Cloud Native 的生态系统（后文简称 Dubbo ECO System）。具体执行的步骤也是 Dubbo 社区对 Cloud Native 的发展趋势研判，首单其冲的是 Dubbo 在 Spring Cloud 场景下的整合。这部分工作已在 Spring Cloud Alibaba 项目中完成，作为项目中的核心成员，Dubbo Spring Cloud 可无缝地替换过传统 Spring Cloud OpenFeign，不仅提供更好的性能，并且具备更多的负载均衡策略和熔断等特性。同时，灵活的扩展点和内建实现帮助 Dubbo 适应不同语言、环境和基础设施。随着 Dubbo 内核即将发布 Cloud Native（即将发布）特性 ，能够将 Dubbo 帮助独立于任意的基础设施，实现 Dubbo（原生）与 Spring Cloud（原生）调用互通，甚至在 K8S 场景下。无论 Dubbo 身处何处，统一的编程模型帮助开发人员快速和高效地实现业务逻辑，达成 “**Write Once, Run Anywhere**” 的目的。后续，Dubbo 在多语言、Mesh 化 以及 Istio 的建树将逐一呈现。

第三部分：应用实践

请您尽可能详细地回答以下问题，这些问题的回答除了作为趋势报告的一部分之外，还可以作为您所在企业的技术实践，单独成文（InfoQ记者将针对每位嘉宾所在企业和技术实践，补充个性化问题）：

Java相关：

- 您的企业使用的JDK版本情况，是否采用了某个OpenJDK 发行版？您如何看待OpenJDK 在国内的发展？（如果没有采用，原因以及后续计划？）

> 小马哥（@mercyblitz）：在开源方面， OpenJDK 的选择仅为 Oracle 官方提供。如果是公司内部的话，则是 OpenJDK Alibaba 的分支。OpenJDK 在国内直接拿去使用的多，有能力围之扩展的公司凤毛菱角。可能随着 OpenJDK 不同分支的成熟，未来国内公司选择面将会更广

- 您的企业目前在支持Java技术栈方面的策略是什么？计划和目标是什么？相关的核心痛点或者业务需求是什么?

> 小马哥（@mercyblitz）：关于公司在战略层面的设定，个人是不是非常清楚的。不过，作为一名基础设施的研发人员，自身遇到和周遭听到同事抱怨的痛点还是存在的。比如，Java 9 之后的版本更新问题。 Reactive Streams 推广和落地时的阻碍，毕竟大多数开发人员从事业务开发，业务系统的稳定性是他们核心 KPI 之一，他们缺少足够的勇气和动力蒸腾新技术的引进、也没有时间和精力关注其中细节，尤其当没有明显特性变化的基础设施 和 API 升级

> 时。对于本人而言，我比较关心 JVM 中的变化，包括 GC 算法和性能提升，比如 C1 和 C2 编译所导致的延迟和资源开销，如何帮助 JVM 快速启停，因为这些会限制 Java 作为编程语言在云原生时代的想象空间

- 对于当前Java的整体发展情况，您有什么感想?

> 小马哥（@mercyblitz）：Java 目前仍具在编程语言排行榜上夺魁，不过在整体比重上微幅下滑。个人看来，未来这个趋势还将持续。究其原因，一方面是由于新语种出现的中短期效应，一方面是 Java 的编程复杂度并没有明显的降低，比如 I/O 处理、并发/并行计算，以及类加载等等。再者是 Java 与操作系统之间的交互仍不够充分，尽管 Java 9 开始提供了不少的 API，然而了解和使用的群体不足。Java 在这方面明显不及 GO 语言。
>
> 从语言层面来看，Java 正在向主流非 Java 语言融合，解决其中鸿沟的起手式就是语法的变化，比如 Java 8 的 Lambda 表达式 和 Java 10 的局部变量类型（`var`）等。个人认为这是一件好事，未来前后端不分家，相互渗透，对于彼此语言都是良性发展。
>
> 除此之外，个人比较期待的是 GraalVM 对 Java 的改变，传统 Java 应用必须依赖 JVM 进程加载字节码后解释执行，无法保证所有的代码能够在运行期编程完成，不免有运行时编译所带来的性能开销，从而影响 JVM 的启停时间，简单地说，这种方式不够 Native，对于云原生或许不够友好。如果未来 GraalVM 的社区版也能够像 OpenJDK 那般“亲民”，那么，Java 的变化将是颠覆性的。

微服务相关：

- 请介绍您的企业是否进行了微服务实践？如果是，在整体系统架构中的比例是多少？如果不是，是否有相关计划?
- 您所采用的主要微服务框架是什么？如何判断国内该领域的技术发展情况？您认为微服务主流框架的争夺是否尘埃落定?
- 您如何看待Service Mesh在国内的发展现状和发展前景?

参与专家：

小马哥(@mercyblitz)，《Spring Boot 编程思想》作者、Apache Dubbo PMC 和 Spring-Cloud-Alibaba Architect

---

1. Oracle Java SE Support Roadmap - https://www.oracle.com/technetwork/java/java-se-support-roadmap.html↩
2. Java Performance Tuning News August 2019 - http://www.javaperformancetuning.com/news/news225.shtml↩
3. Reactive Streams JVM - https://github.com/reactive-streams/reactive-streams-jvm↩
4. Jakarta EE - https://jakarta.ee/↩
5. Java Community Process - https://jcp.org/en/home/index↩
6. JSR 330: Dependency Injection for Java - https://jcp.org/en/jsr/detail?id=330↩
7. Eclipse Foundation - https://www.eclipse.org/org/foundation/↩