ArchSummit
全球架构师峰会

10th

全球架构师峰会·深圳站

ArchSummit 10th Anniversary

主办方：InfoQ

从软件的历史看架构的未来

周志明 / 华为 SaaS 首席软件教练

- **程序员**

  华为 SaaS 首席软件教练，高级技术专家。工作中主要从事 To B 端、SaaS 企业级软件的研发，兴趣是当一名纯粹的程序员，业余里对计算机科学相关的多个领域都有持续跟进。

  • https://icyfenix.cn　　• https://github.com/fenixsoft

- **研究员**

  理学博士，研究方向为机器学习自动化特征选择。曾任远光软件研究院院长，澳门科大-远光人工智能联合实验室主任。

- **技术布道师**

  开源技术的积极倡导者和推动者，对计算机科学相关的多个领域都有持续跟进。曾担任过华为云、阿里云、腾讯云三家中国主流云厂商的最有价值技术专家。

- **计算机技术作家**

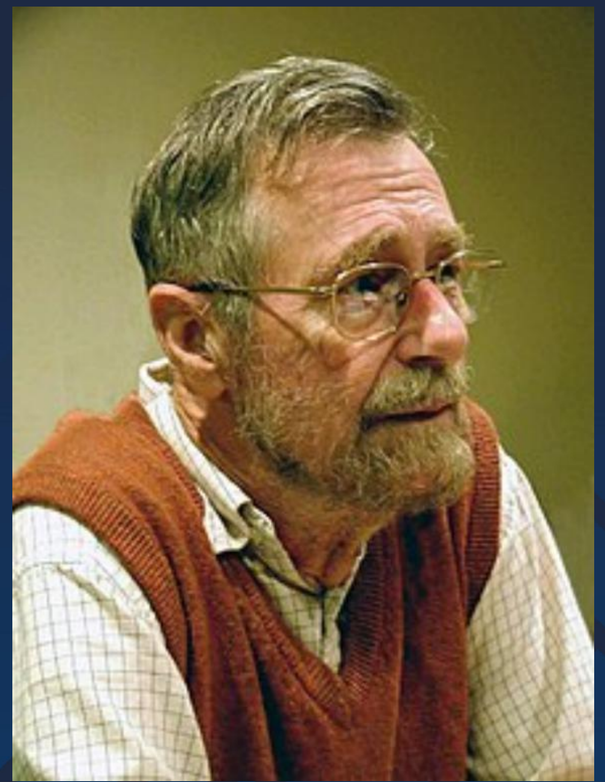
  出版过 8 部计算机技术书籍，撰写过两部开源文档，口碑和销量均得到读者的认可。其中 5 本书在豆瓣上获得了 9.0 分或以上的评价“深入理解 Java 虚拟机”系列重印超过 45 次，总销量逾 35 万册。

  • 2021年《凤凰架构：构建可靠的大型分布式系统》（豆瓣 9.1）
  • 2020年《软件架构探索：The Fenix Project》（开源文档）
  • 2019年《深入理解 Java 虚拟机：JVM 高级特性与最佳实践（第三版）》（豆瓣 9.6）
  • 2018年《智慧的疆界：从图灵机到人工智能》（豆瓣 9.1）
  • 2016年《深入理解 Java 虚拟机：JVM高级特性与最佳实践（第二版）》（豆瓣 9.0）
  • 2015年《Java 虚拟机规范 （Java SE 8 中文版）》（官方授权翻译，豆瓣 8.0）
  • 2014年《Java 虚拟机规范 （Java SE 7 中文版）》（官方授权翻译，豆瓣 9.0）
  • 2013年《深入理解 OSGi：Equinox 原理、应用与最佳实践》（豆瓣 7.7）
  • 2011年《深入理解 Java 虚拟机：JVM 高级特性与最佳实践（第一版）》（豆瓣 8.6）
  • 2011年《Java 虚拟机规范 （Java SE 7 中文版）》（开源文档）

As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

在没有计算机的时候，不存在编程问题；当我们有了简单的计算机，编程只是个小问题；而现在我们有了算力规模庞大的计算机，那编程也就成为了一个同样巨大的问题了。

—— Edsger Dijkstra, Communications of the ACM, 1972



第一次软件危机：

机器算力规模超过了人类个体的生理极限

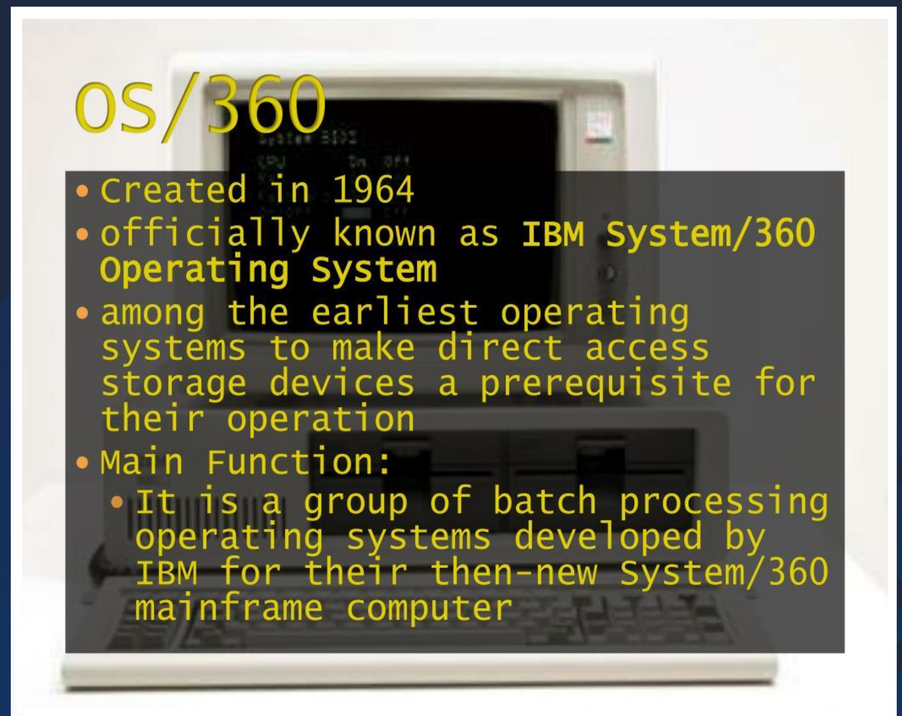
第一次软件危机：

机器算力规模超过了人类个体的生理极限

- **开发方式：**程序员的大脑就足够记住数据在几 KB 内存中的分布细节，记住每项操作在电路中的运行逻辑。软件架构与硬件架构是直接物理对齐。
- **部署形式：**大型机。
- **主要矛盾：**当世界上最聪明的人都无法为机器编写出合理的程序了，计算机科学还能继续发展吗？
- **应对方案：**结构化编程。让软件的每个局部都可以设计独立的算法和数据结构，在整体上控制住了软件开发的复杂度。

第二次软件危机：

机器算力规模超过了人类群体的沟通极限



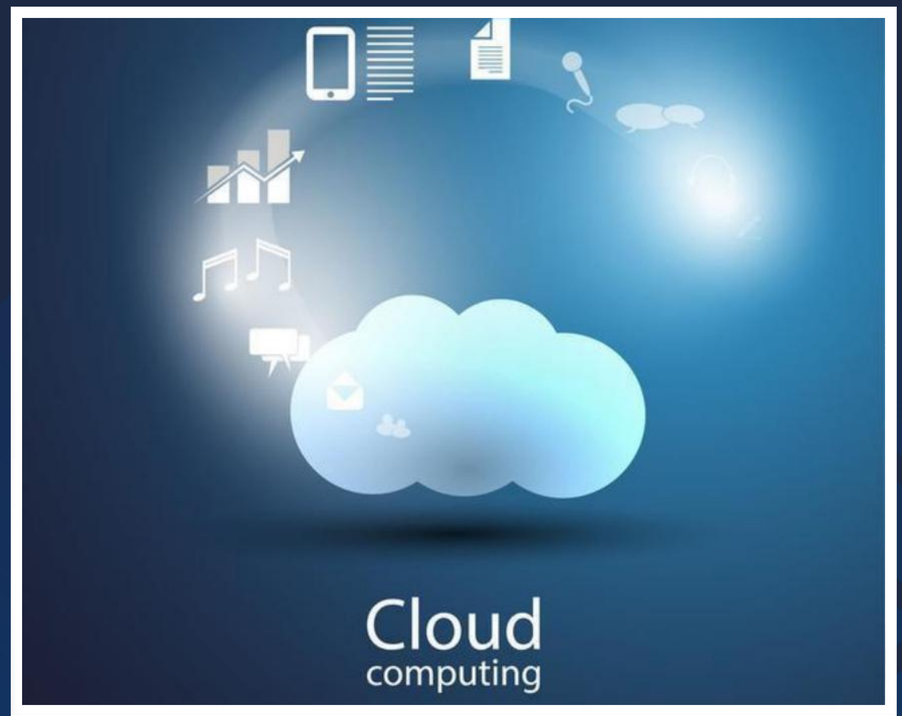
第二次软件危机：

机器算力规模超过了人类群体的沟通极限

- **开发方式：**每个局部模块的程序员与对接的上下游之间沟通，约定好接口细节，由一个个模块编译组装成大型软件。
- **部署形式：**服务器 - 客户端。
- **主要矛盾：**每个人都有自己的理解与认知，如何让每个模块都能准确的协同工作就成了一场灾难。
- **应对方案：**面向对象的编程、软件工程学。追求最符合人类思维的视角来抽象问题取代了追求最符合机器运行特征的算法与数据结构成为软件架构的最优先原则。

今天，云计算与分布式时代：

人驾驭机器的基本矛盾是什么？



今天，云计算与分布式时代：

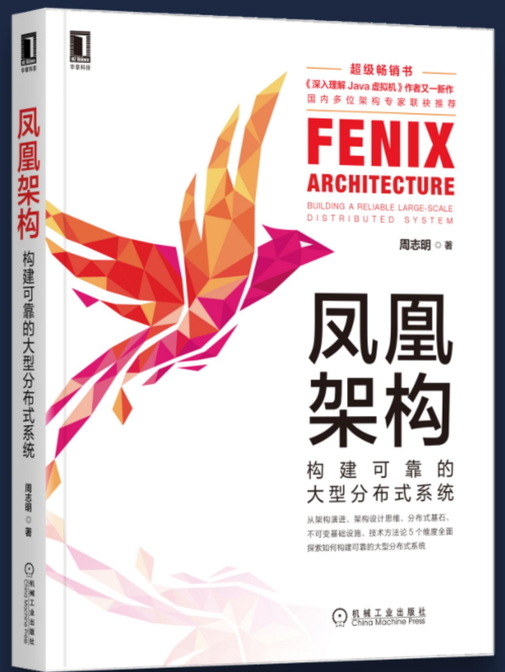
机器算力规模逼近了人类协作的工程极限

- **开发方式：**何以系统性、规范化、可定量的方法去高质量地开发和维护软件成为一门独立的科学。程序员适合的软件工程措施和管理措施相互协作。
- **部署形式：**云计算数据中心。
- **主要矛盾：**只要时间足够长，环境足够复杂，就肯定会有人疏忽犯错，会有代码携带缺陷，会有电脑宕机崩溃，会有网络堵塞中断。
- **应对方案：**以微服务为主要形式的分布式架构，整体与部分有物理层面的隔离。流水不腐，面向错误设计而不追求7*24的完美稳定，正视局部的消亡与重生。

几乎所有分布式的研究都围绕着“错误”而展开

面向错误设计是分布式的最核心价值

- **机器宕机/恢复**：如何应对分布式系统中节点非优雅下线的影响，和上线后的恢复处理。
- **网络分区容忍**：如何在不可靠网络中支撑分区容忍性。
- **分布式三态**：如何对分布式系统通讯中成功、失败、未知三种状态的处理。
- **存储数据丢失**：对于节点的状态丢失后，如何从其他节点读取、恢复存储的状态。
- **异常处理原则**：任何在设计阶段考虑到的异常情况一定会在系统实际运行中发生，但在系统实际运行遇到的异常却很有可能在设计时未能考虑。
- ……



主旨观点：

流水不腐，有老朽，有消亡，有重生，有更迭才是生态运行的合理规律，如何采用不可靠的部件来构造出一个可靠的系统，是软件架构适配云与分布式算力发展的关键所在。

如果系统中局部能拥有独立的生命周期，在整体架构上有物理隔离的设计，那即便采用了不可靠部件，在系统外观察，整体上仍然有可能表现出稳定健壮的服务能力。

人驾驭机器的下一个关键矛盾是什么？

机器算力规模超过人应有合理知识的极限

- **大型机时代**

  机器算力规模超过了人类个体的生理极限
- **服务器 – 客户端时代**

  机器算力规模超过了人类群体的沟通极限
- **云计算时代**

  机器算力规模超过了人类协作的工程极限

机器算力规模超过人应有合理知识的极限

云与分布式时代的知识膨胀

- **工程与设计理论众多**，DevSecOps、Scrum、OOAD、SCA、DDD、12-Factors，等等。
- **框架与工具众多**，Spring Cloud 全家桶、Netflix OSS 全家桶、Kubernetes、Istio、Envoy，等等。
- **软件架构模式众多**，SOA、Microservice、Service Mesh、Serverless FaaS，等等。
- **分布式的技巧众多**，光一个远程服务，就须考虑注册、发现、调用、弹性、熔断、限流、负载均衡等问题。
- **云厂商的服务众多**，光 AWS 就有 200 余项云计算服务，其他云厂商也在持续跟进之中。

机器算力规模超过人应有合理知识的极限

云与分布式时代的知识膨胀

- **哲学意义上的“知识膨胀”**：人类科学的前沿在不断拓展，触及到前沿所需的基础知识也不断增加，是否会陷入后来者终其一生都无法攒下足够基础，导致人类知识陷入止步不前的危机之中。
- **计算机科学上的“知识膨胀”**：从毕业到“35岁退休”（梗）之前，多数程序员恐怕都不具备设计分布式架构所需的全面知识。

这次关键矛盾的应对方案是什么？

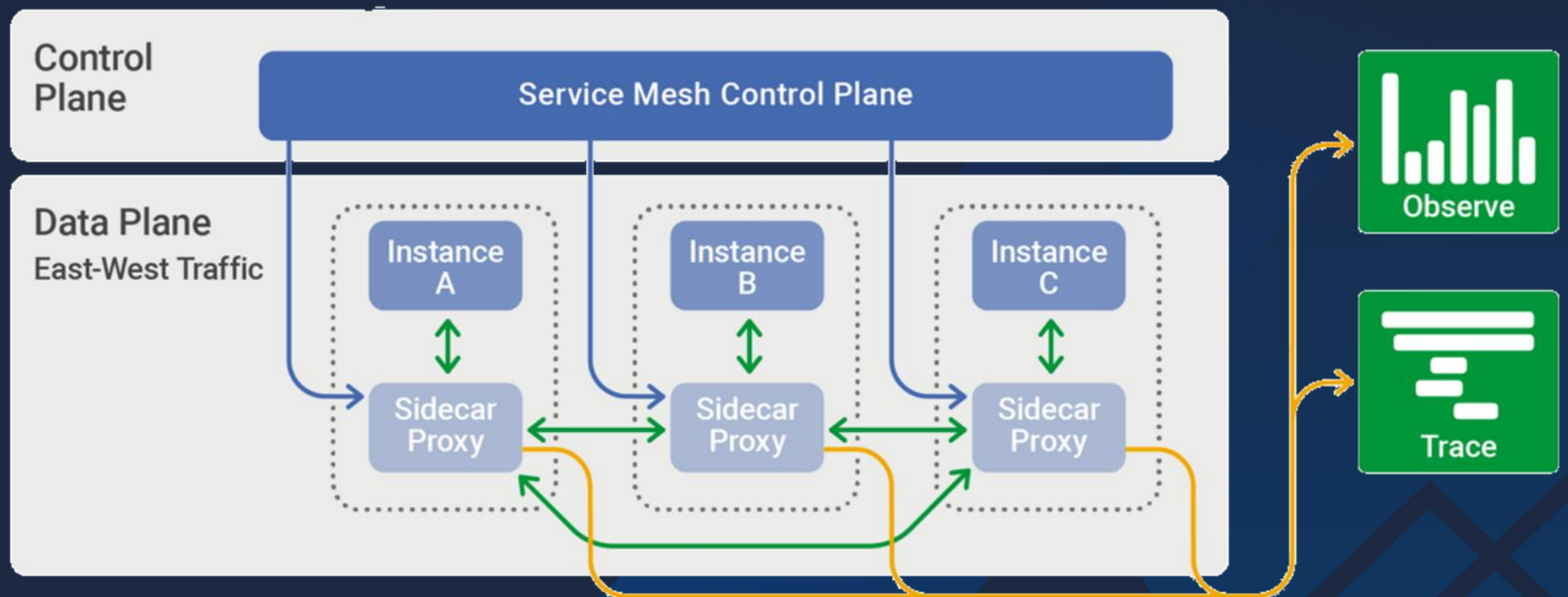
云与分布式时代，又到了该为知识“打个结”的时候

一种可外挂的软件架构

云服务与分布式能力在程序逻辑之外、在开发阶段之后附加

Service Mesh Architecture

brings standard, universal traffic management, telemetry, and security to complex service interaction



为知识“打个结”

分离软件的非功能属性并标准化

- Service Mesh：把服务调用过程中的安全、路由、治理、观测等能力标准化、独立管理。
- Database Mesh：把数据库访问过程中的数据库发现、数据分片、读写分离、负载均衡等能力标准化、独立管理。
- MessageQueue Mesh：把消息发布和消费过程中的死信管理、ACK、可靠投递、限流、追踪等能力标准化、独立管理。
- Cache Mesh：把缓存处理过程中淘汰策略、失效通知、并发级别管理、容量控制、持久化等能力标准化、独立管理。
- ……

当前的编程

```
Set<HostAndPort> nodes = new LinkedHashSet<HostAndPort>();
nodes.add(new HostAndPort("192.168.1.1", 6379));
nodes.add(new HostAndPort("192.168.1.2", 6379));
JedisPoolConfig config = new JedisPoolConfig();
config.setMaxTotal(1);
config.setMaxIdle(1);
try (Jedis jedis = new JedisCluster(nodes, config)) {
    String result = jedis.set("icyfenix", "{\"name\":\"zzm\", \"email\":\"icyfenix@gmail.com\"}");
    return ok(result);
} catch (Exception e) {
    log.error("Redis error:{}", ExceptionTools.getExceptionStackTrace(e));
    return false;
}
```

当前的编程

```
Set<HostAndPort> nodes = new LinkedHashSet<HostAndPort>();
nodes.add(new HostAndPort("192.168.1.1", 6379));
nodes.add(new HostAndPort("192.168.1.2", 6379));
JedisPoolConfig config = new JedisPoolConfig();
config.setMaxTotal(1);
config.setMaxIdle(1);
try (Jedis jedis = new JedisCluster(nodes, config)) {
    String result = jedis.set("icyfenix", "{\"name\":\"zzm\", \"email\":\"icyfenix@gmail.com\"}");
    return ok(result);
} catch (Exception e) {
    log.error("Redis error:{}", ExceptionTools.getExceptionStackTrace(e));
    return false;
}
```

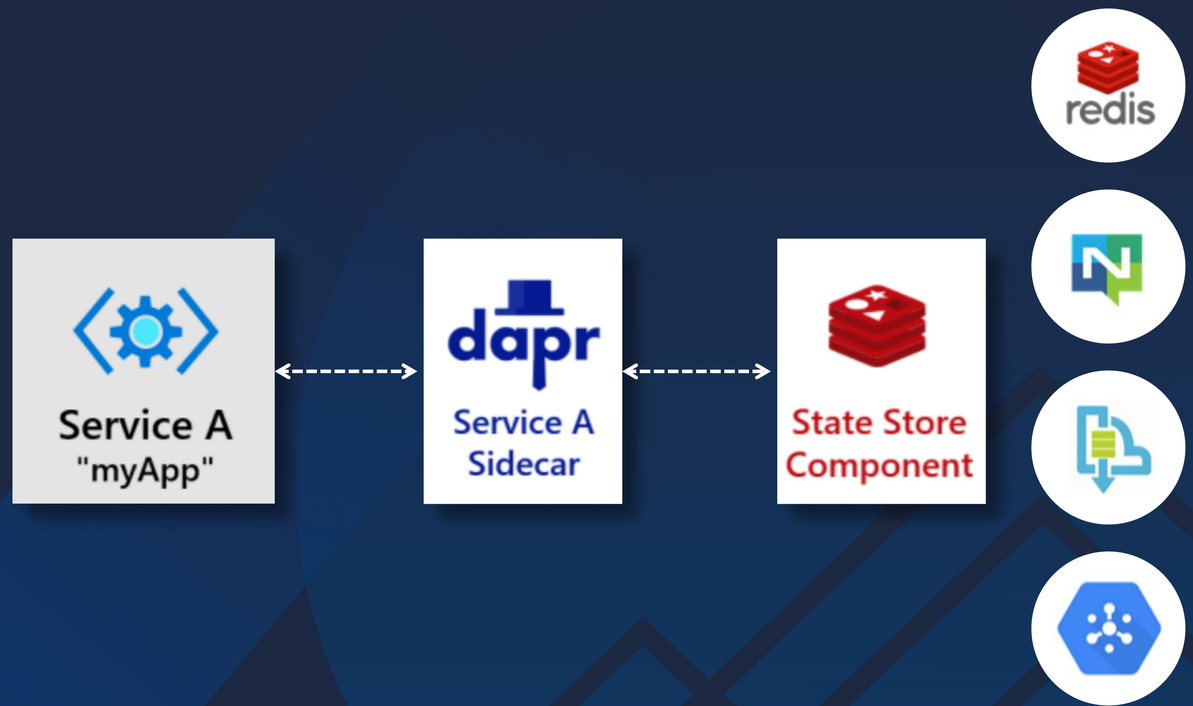
- 需要了解 Redis 的知识，不说实现原理，起码要知道它的 API 该如何使用，程序代码也必须引入 Redis 的客户端 SDK 作为依赖项。
- 需要知道 Redis 的服务位置、部署方式、链接信息，这些其实应该是 SRE 而不是 SDE 的职责。
- 可能还需要考虑额外的非功能属性：要不要启用连接池？并发策略是 first-write-wins 还是 last-write-wins ？是否需要支持事务？数据能保证什么级别的一致性？要批量操作该怎么办？假若这些非功能属性都反应到代码上，结果肯定要比现在看到的复杂上不少
- 有一些需求甚至仅凭应用代码是无法解决的。譬如要支持事务，用 Redis 可以，用 Memcached/Cassandra 就不行；要支持强一致性，用 Etcd/ZooKeeper 可以，用 Redis 就不行。

设想的编程

```
curl -X POST http://localhost:3500/v1.0/state/users \
  -H "Content-Type: application/json" \
  -d '[
        {
          "key": "icyfenix",
          "value": {"name":"zzm", "email":"icyfenix@gmail.com"}
        }
      ]'

curl http://localhost:3500/v1.0/state/users/icyfenix \
  -H "Content-Type: application/json"

{"name":"zzm", "email":"icyfenix@gmail.com"}
```

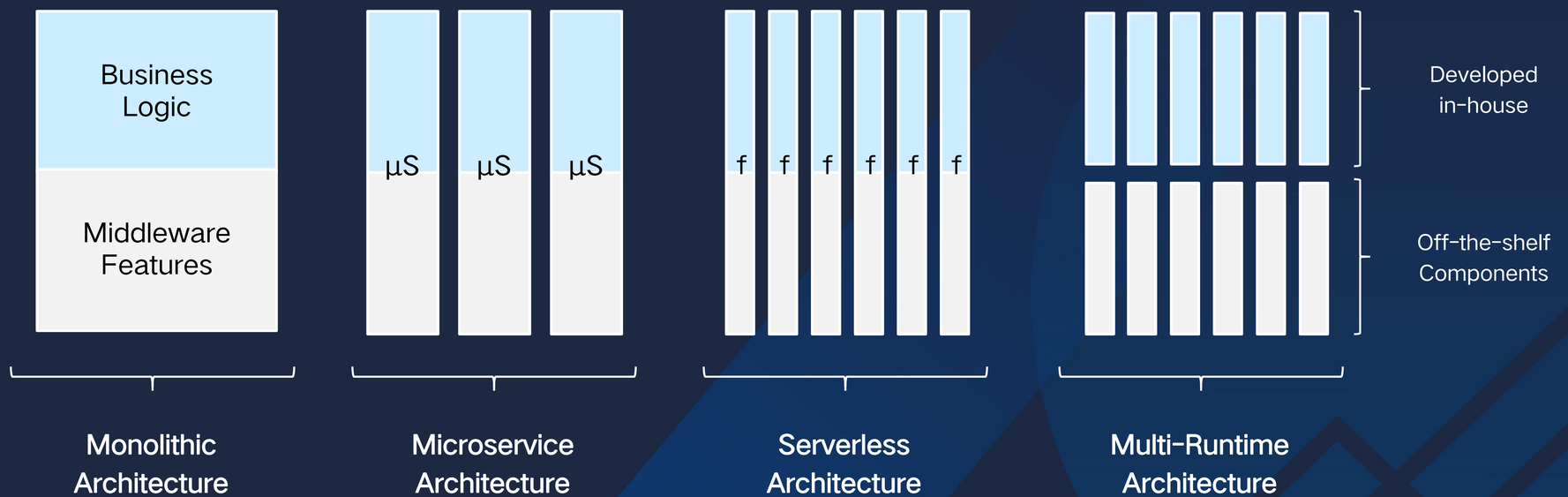
设想的编程

```
curl -X POST http://localhost:3500/v1.0/state/users \
  -H "Content-Type: application/json" \
  -d '[
        {
          "key": "icyfenix",
          "value": {"name":"zzm", "email":"icyfenix@gmail.com"}
        }
      ]'

curl http://localhost:3500/v1.0/state/users/icyfenix \
  -H "Content-Type: application/json"

{"name":"zzm", "email":"icyfenix@gmail.com"}
```



Multi-Runtime Architecture

loose coupling between the business logic and the increasing list of distributed systems concerns

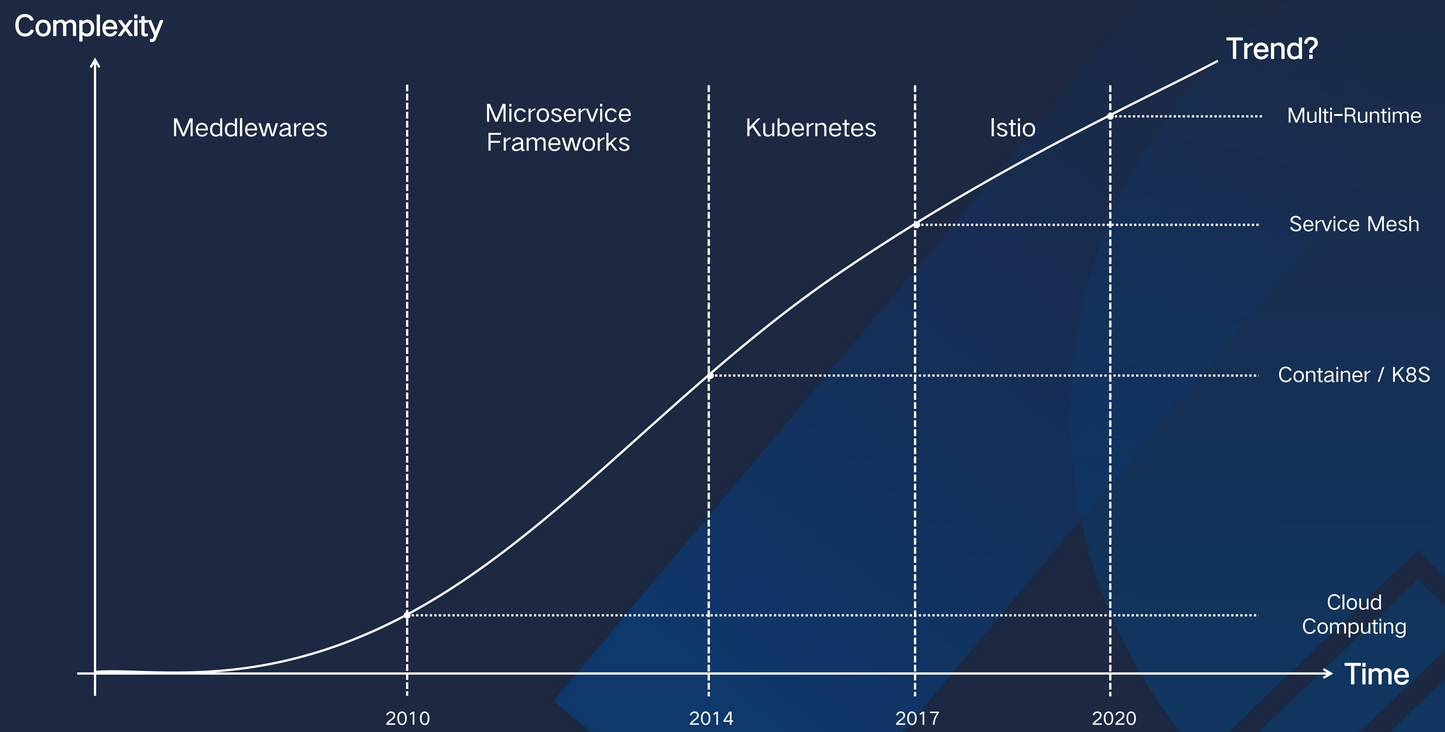


Dapr Building Blocks

modular best practices accessible over standard HTTP or gRPC APIs

Complexity of Programming

now we have gigantic computers, programming has become an equally gigantic problem



生产关系适应生产力的发展

软件架构适应机器算力的发展

- **云计算从竞争优势变为基石**：云计算服务会逐渐下沉，系统与云服务商不会像今天这样有强烈的绑定关系，从云原生到云不可知（Cloud Agnostic）是云计算的成熟阶段。
- **开发变得更加复杂，又更加简单**：系统的功能与非功能特性由不同的团队去维护，软件系统由少量的负责架构和质量属性的专家，与大量的负责功能属性的业务程序员去协作完成。
- **程序员群体同样出现两极分化**：更先进的软件架构已经允许更平庸的开发者也同样能写出可运行、可用于生产的软件产品，同时又对精英开发者提出更多、更复杂的技术要求。

周志明（IcyFenix）

- icyfenix@gmail.com
- https://github.com/fenixsoft

https://github.com/fenixsoft

《凤凰架构：构建可靠的大型分布式系统》

THANKS

全球架构师峰会·深圳站

ArchSummit 10th Anniversary

主办方：InfoQ