微服务架构设计与实践

服务注册与服务发现篇

孙玄@58集团

关于我

- 58集团技 QCon 全球软件开发大会 主席 DTCC 2016中国数据库技术大会 DATABASE TECHNOLOGY CONFERENCE CHINA 2016
- 58集团高 SDCC 中国软件开发者大会 Software Developer Conference China 构师 TOP1 技术型企业案例研究智库
- 转转架构 PROGRAMMER 程序员 责人 ArchSummit 全球架构师峰会
- 百度高级工程师
- 毕业于浙江大学
- 代表公司多次对外分享
- 企业内训&公开课

关于我

企业内训

- ✓ 华为
- ✓ 中航信
- ✓ 平安
- ✓ 银联
- ✓ 华泰证券
- ✓ 思科
- ✓ 云南电力
- ✓ 深信服
- ✓ 新华社
- ✓ 民生银行
- ✓ 招商银行
- ✓ ……

公开课

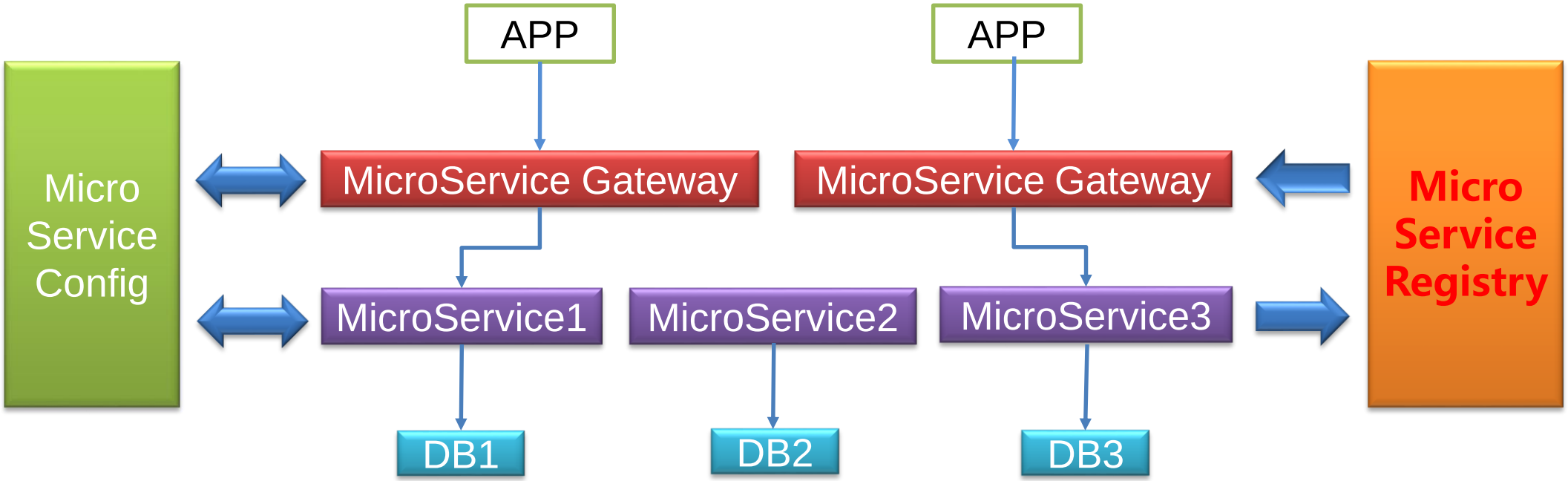
- ✓ 北京
- ✓ 上海
- ✓ 深圳
- ✓ 广州
- ✓ 成都
- ✓ ……

分享要点

微服务架构

1. ZooKeeper是什么
2. ZooKeeper使用
3. 服务注册设计
4. 服务发现设计
5. Tips
6. 案例实践

ZooKeeper是什么



ZooKeeper是什么

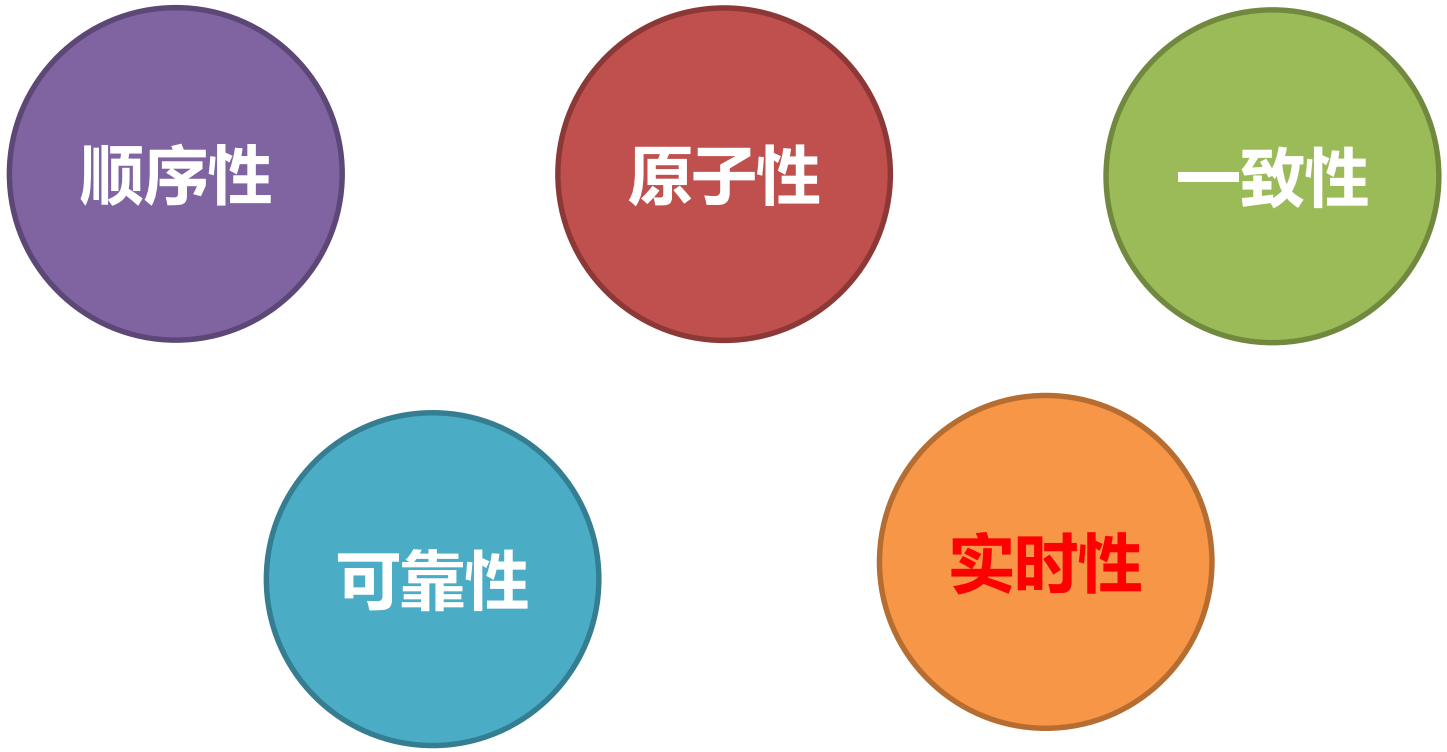
- **注册中心**
  - Service Registry
    - 微服务相关配置信息注册
  - Service Discovery
    - 获取微服务配置信息
  - 能力
    - 分布式数据一致性
    - 高可用
    - 高性能

ZooKeeper是什么

- 分布式环境下协调服务
- 基于Google Chubby思想
- Yahoo Java
- Hadoop、Hbase、Kafka
- 基于ZK实现
  - 分布式协调、集群管理、分布式锁、分布式队列、负载均衡等
- 竞品
  - Etcd、Eureka、Consul等等

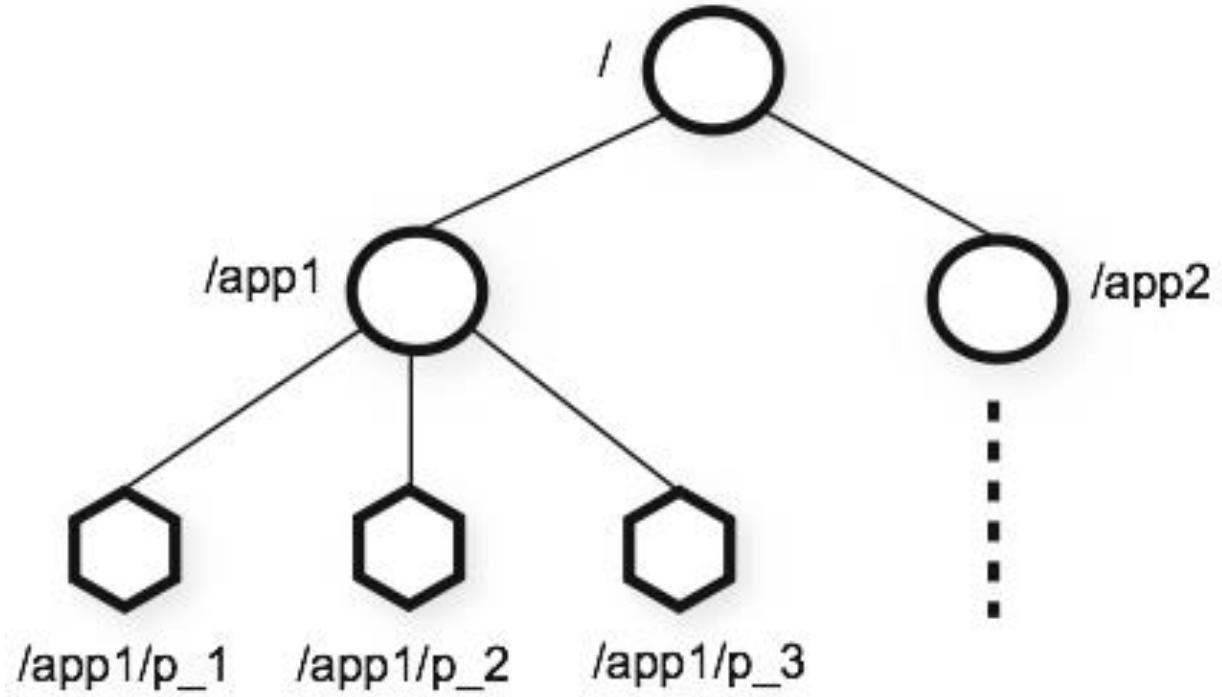
ZooKeeper是什么

- 集群特性



ZooKeeper是什么

- 数据模型
  - 树状层次结构
  - 类似文件系统
  - 树中的节点称为znode
    - 存储数据
    - 关联ACL（Access Control List）
  - 每个znode文件或者目录
  - 每个znode存储数据不超过1M
  - 文件系统路径是绝对路径，以“/”开头，必须是标准



ZooKeeper是什么

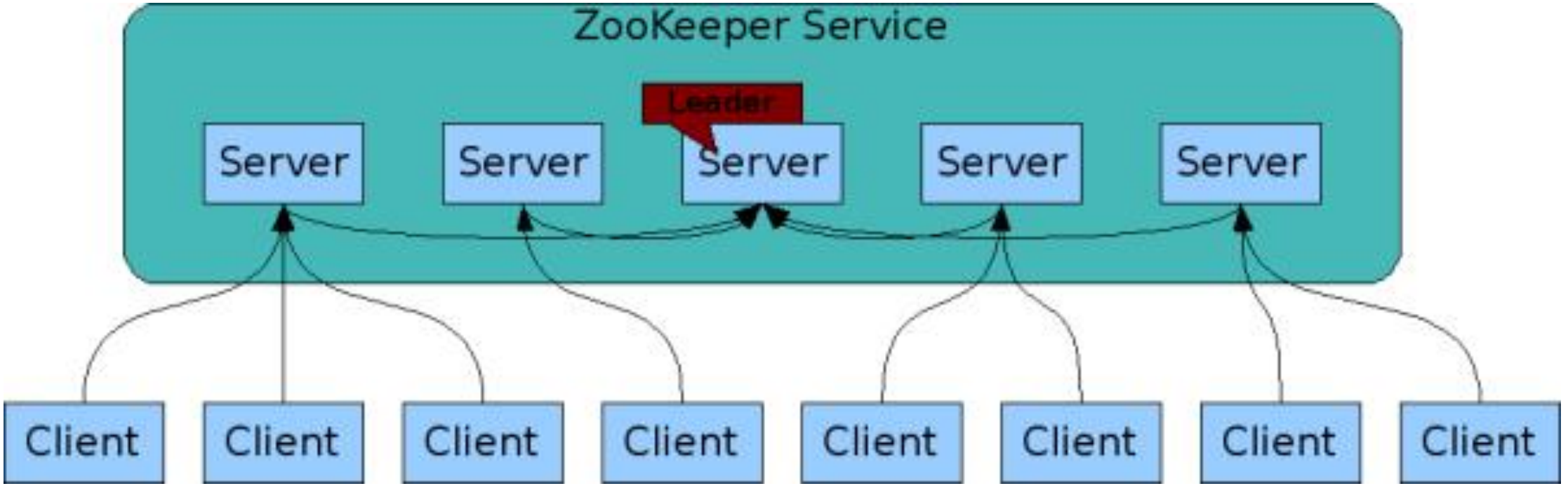
- 数据模型
  - znode两种类型
    - 短暂节点、持久节点
      - 类型在创建时确定，之后不能修改
      - 短暂znode不能有子节点
    - 顺序号
      - 创建设置顺序标识，zonde名称后面值（单调递增）顺序号，父节点维护
      - 应用（共享锁）
    - 观察（watch）
      - 当znode以某种方式发生改变时，这种机制让客户端得到通知。（配置管理）

ZooKeeper是什么

- **部署**

ZooKeeper是什么

- **集群部署**
  - Leader & Follower



ZooKeeper是什么

- **集群架构**
  - 集群中半数以上的机器处于可用状态，就能提供服务，机器间互相通信
  - 确保znode树中的每一个修改都会被复制到集群中半数以上的机器。确保在可用状态下，至少有一个机器保存最新状态，其余副本会更新到最新状态
  - 如何实现
    - Zab(ZooKeeper Atomic Broadcase)原子广播协议
      - 参考Paxos协议
    - 阶段一：领导者选举（Leader Election）
    - 阶段二：原子广播（Atomic Broadcast）
  - 3~5节点（奇数）

ZooKeeper使用

- **环境准备**
  - JDK 1.6及以上
    - Oracle JDK
  - ZooKeeper
    - http://zookeeper.apache.org/releases.html
    - stable版本
      - 3.4.9
      - 解压即可使用

ZooKeeper使用

- **配置文件**
  - cp conf/zoo_sample.cfg conf/zoo.cfg
    - tickTime=3000
      - 嘀嗒时间
      - 最小时间单元长度
    - initLimit=10
      - Leader等待Follower启动并完成数据同步时间
      - 10*tickTime

ZooKeeper使用

- **配置文件**
  - syncLimit=10
    - Leader和Follower间心跳检测最大延迟
    - 10*tickTime
  - dataDir=/opt/zookeeper
    - myid（集群节点的唯一ID）
  - clientPort=2181
    - ZooKeeper服务器对外暴露端口

ZooKeeper使用

- **配置文件**
  - server.1=127.0.0.1:2888:3888
    - server.<id>=<ip>:<port1>:<port2>
    - id节点编号，取值1~255
      - 放在myid中
    - ip节点所在ip地址
    - port1表示Leader节点和Follower节点心跳与数据同步端口
    - port2表示领导者选举过程，投票通信端口

ZooKeeper使用

- **启动ZooKeeper**
  - bin/zkServer.sh start
  - bin/zkServer.sh start-foreground
- 验证ZooKeeper
  - bin/zkServer.sh status
  - telnet ip port

ZooKeeper使用

- **ZooKeeper基本操作**

操作	描述
create	创建一个znode(必须要有父节点，创建时可以设置数据)
delete	删除一个znode(该znode不能有任何子节点)
exists	测试一个znode是否存在并且查询它的元数据
getACL,setACL	获取/设置一个znode的ACL
getChildren	获取一个znode的子节点列表
getData,setData	获取/设置一个znode所保存的数据
sync	将客户端的znode试图与ZooKeeper同步

ZooKeeper使用

- **ZooKeeper观察触发器**
  - 读操作 exists、getChildren和getData上可以设置观察
  - 这些观察可以被写操作create、delete和setData触发

设置观察的操作	观察触发器				
	create		delete		setData
	znode	子节点	znode	子节点	
exists	NodeCreated		NodeDeleted		NodeDataChanged
getData			NodeDeleted		NodeDataChanged
getChildren		NodeChildrenChanged	NodeDeleted	NodeChildrenChanged	

服务注册设计

- **基于ZooKeeper服务注册表数据结构设计**
  - 把多个微服务IP和Port信息注册到ZooKeeper集群
    - 根节点
    - 微服务节点
    - 地址节点

服务注册设计

- ✓ “/root”
- ✓ 持久化节点
- ✓ 一个

服务注册设计

微服务节点

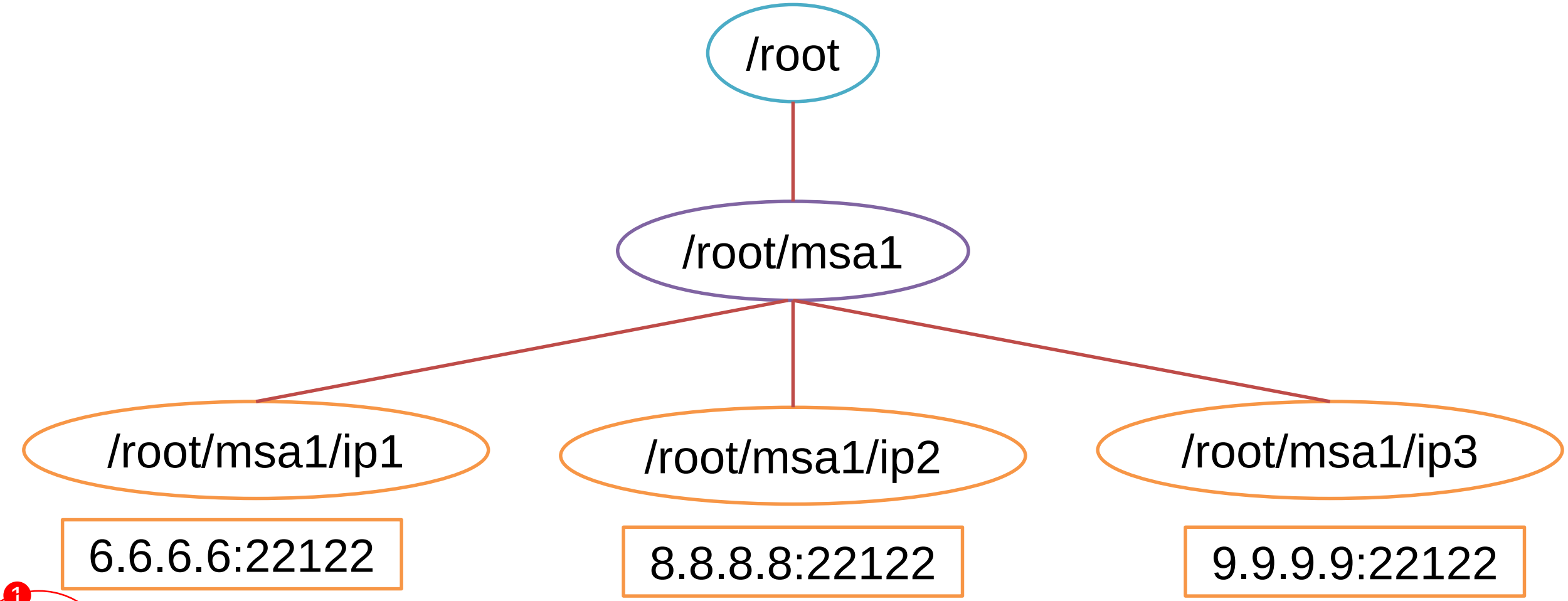
- ✓ “/root/msa”
- ✓ 持久化节点
- ✓ 微服务名称（高可用部署多个）
- ✓ 一个（高可用部署多个）

服务注册设计

地址节点

- ✓ “/root/ms/ip1”
- ✓ 临时顺序节点
- ✓ 具有数据
- ✓ 多个节点

服务注册设计

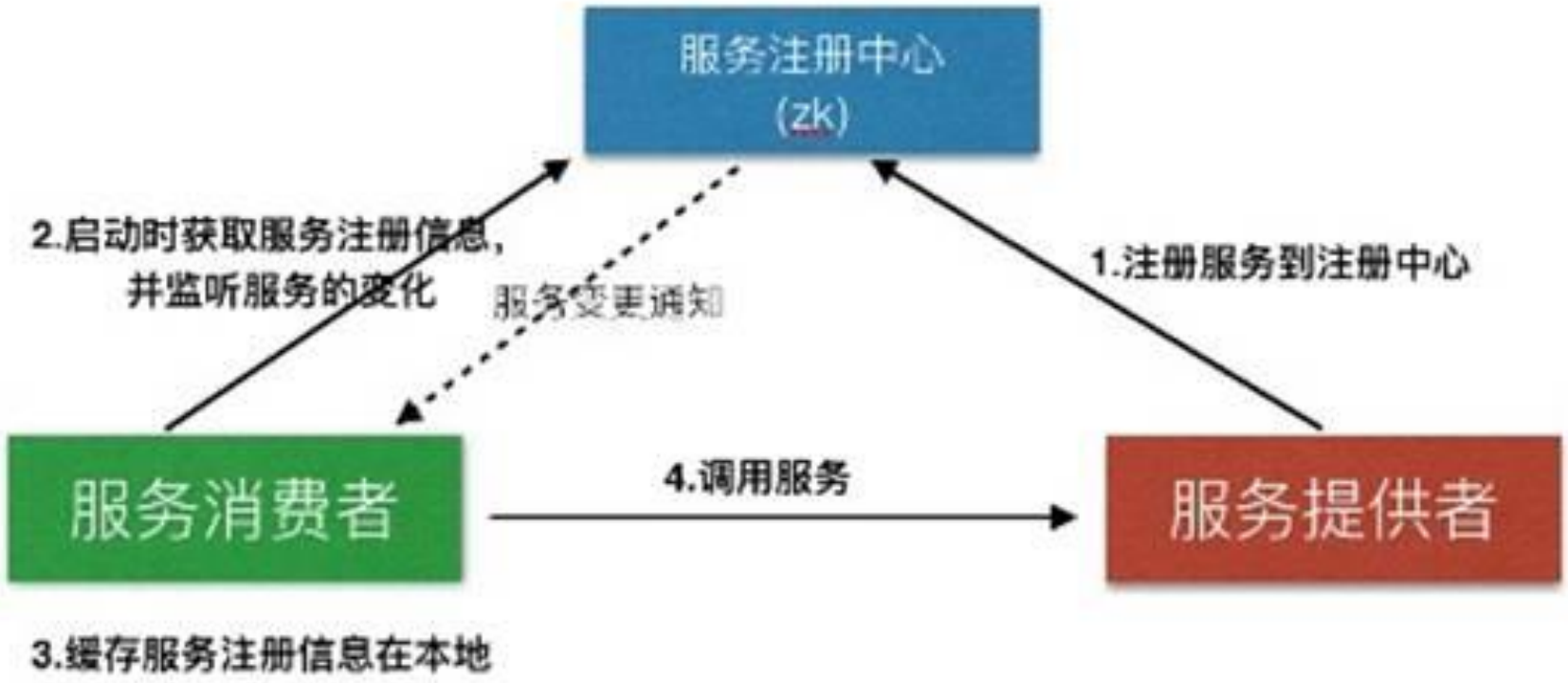

/root
/root/msa1
/root/msa1/ip1
/root/msa1/ip2
/root/msa1/ip3
6.6.6.6:22122
8.8.8.8:22122
9.9.9.9:22122

微服务注册设计

- **注册步骤**
  - 微服务引入ZooKeeper客户端
  - 微服务建立和ZooKeeper的连接
    - 所有节点
  - 微服务Create根节点、服务节点、地址节点
  - 微服务和ZooKeeper心跳检测
    - ZooKeeper znode节点清理
  - 微服务异常处理
    - session失效重连

服务发现设计

- 服务发现
  - 获取可用微服务配置过程



服务发现设计

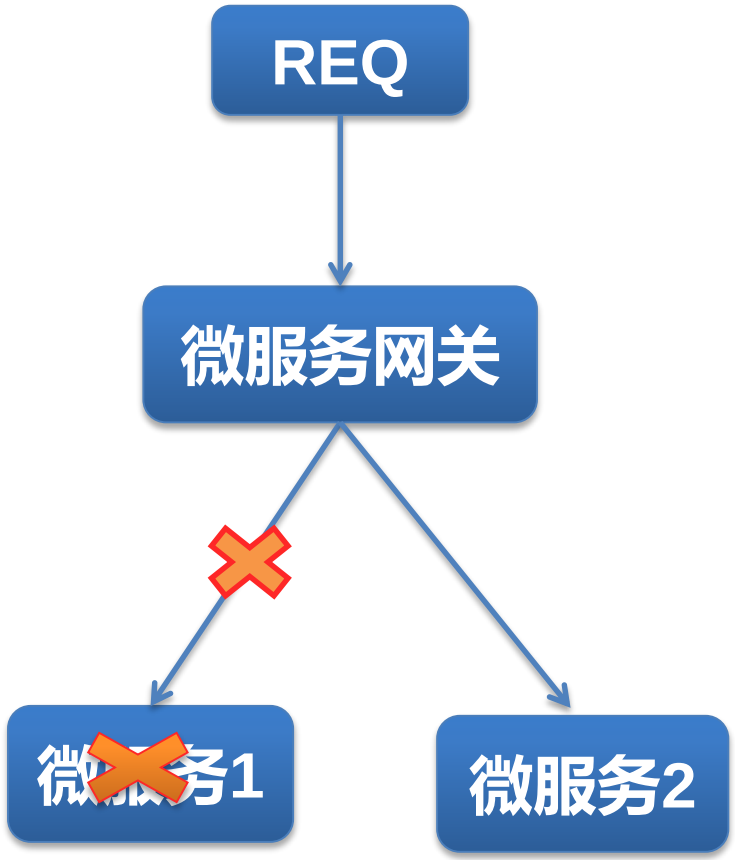
- 基于ZooKeeper服务发现步骤
  - 服务消费者和ZooKeeper建立连接
  - 获取地址节点数据
  - 负载均衡选取IP访问

服务发现设计

- 基于ZooKeeper服务发现步骤
  - 服务消费者在启动时从服务注册中心获取需要的服务注册信息
  - 将服务注册信息缓存在本地
  - 监听服务注册信息的变更，如接收到服务注册中心的服务变更通知，则在本地缓存中更新服务的注册信息
  - 根据本地缓存中的服务注册信息构建服务调用请求，并根据负载均衡策略(随机负载均衡，Round-Robin负载均衡等)来转发请求求
  - 对服务提供方的存活进行检测，如果出现服务不可用的服务提供方，将从本地缓存中剔除
  - 服务消费者只在自己初始化以及服务变更时会依赖服务注册中心，在此阶段的单点故障通过Zookeeper集群来进行保障。在整个服务调用过程，不依赖于任何第三方服务。

Tips

- **请求不丢失**
  - 服务发现
  - 请求Retry



案例实践

- **58帮帮**
- **58转转**
  - 连接ZooKeeper集群
  - 服务发现IP/Port本地缓存
  - 微服务高可用

要点回顾

微服务架构

1. ZooKeeper是什么
2. ZooKeeper使用
3. 服务注册设计
4. 服务发现设计
5. Tips
6. 案例实践

欢迎关注本人公众号 “架构之美”

Thanks!

让 生 活 更 简 单