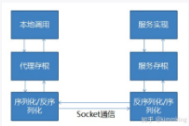
RPC与微服务

RPC

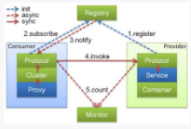
原理



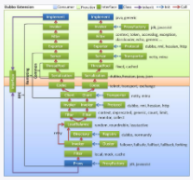
- RPC是基于接口的远程服务调用，共享：POJO实体类定义，接口定义
- 代理，可以选择动态代理，或者AOP实现
- 序列化
  - 语言原生的序列化，RMI，Remoting
  - 二进制平台无关，Hessian，probuffer，avro，fst
  - 文本，JSON、XML
- 网络传输
  - TCP/SSL
  - HTTP/HTTPS
- 查找实现类
  - 通过接口查找服务端的实现类

dubbo

- 核心能力
  - 面向接口代理的高性能RPC调用
  - 智能容错和负载均衡
  - 服务自动注册和发现
  - 高度可拓展能力
  - 运行期流量调度
  - 可视化的服务治理与运维
- 主要功能
  - RPC调用
  - 多协议（序列化，传输，RPC）
  - 服务注册发现
  - 配置、元数据管理
  - 集群，负载均衡
  - 治理，路由
  - 控制台，管理与监控



- 框架



- 建议
  - 建议将服务接口、服务模型、服务异常等均放在API包中，因为服务模型和异常也是API的一部分，这样做也符合分包原则：重用发布等价原则，共同重用原则
  - 服务接口尽量大粒度，每个服务方法应代表一个功能，而不是某个功能的一个步骤，否则将面临分布式事务问题，Dubbo暂未提供分布式事务支持
  - 服务接口建议以业务场景为单位划分，并对相近业务做抽象，防止接口数量爆炸
  - 不建议使用过于抽象的通用接口
  - 多环境隔离
    - 多注册中心机制
    - group机制
    - 版本机制
  - 参数配置
    - 通用参数以concumer端为准，如果consumer端没有设置，使用provider数值
    - 建议在Provider端配置的Consumer端属性
      - timeout：方法调用超时时间
      - retries：失败重试次数，缺省是2
      - loadbalance：负载均衡算法，缺省是随机
      - actives：消费端最大并发调用限制
    - 建议在Provider端配置的Provider端属性
      - threads：服务线程次大小
      - executes：一个服务提供者并行执行请求的上限，即当Provider对一个服务的并发调用达到上限后，新调用会阻塞，此时Consumer可能会超时


容器化部署
docker使用宿主机网络
docker参数指定注册的IP和端口
运维与监控
APM
Promethus + Grafana
分布式事务
柔性事务：SAGA、TCC、AT
Seata
hmily
重试与幂等
去重
类似乐观锁机制

分布式服务

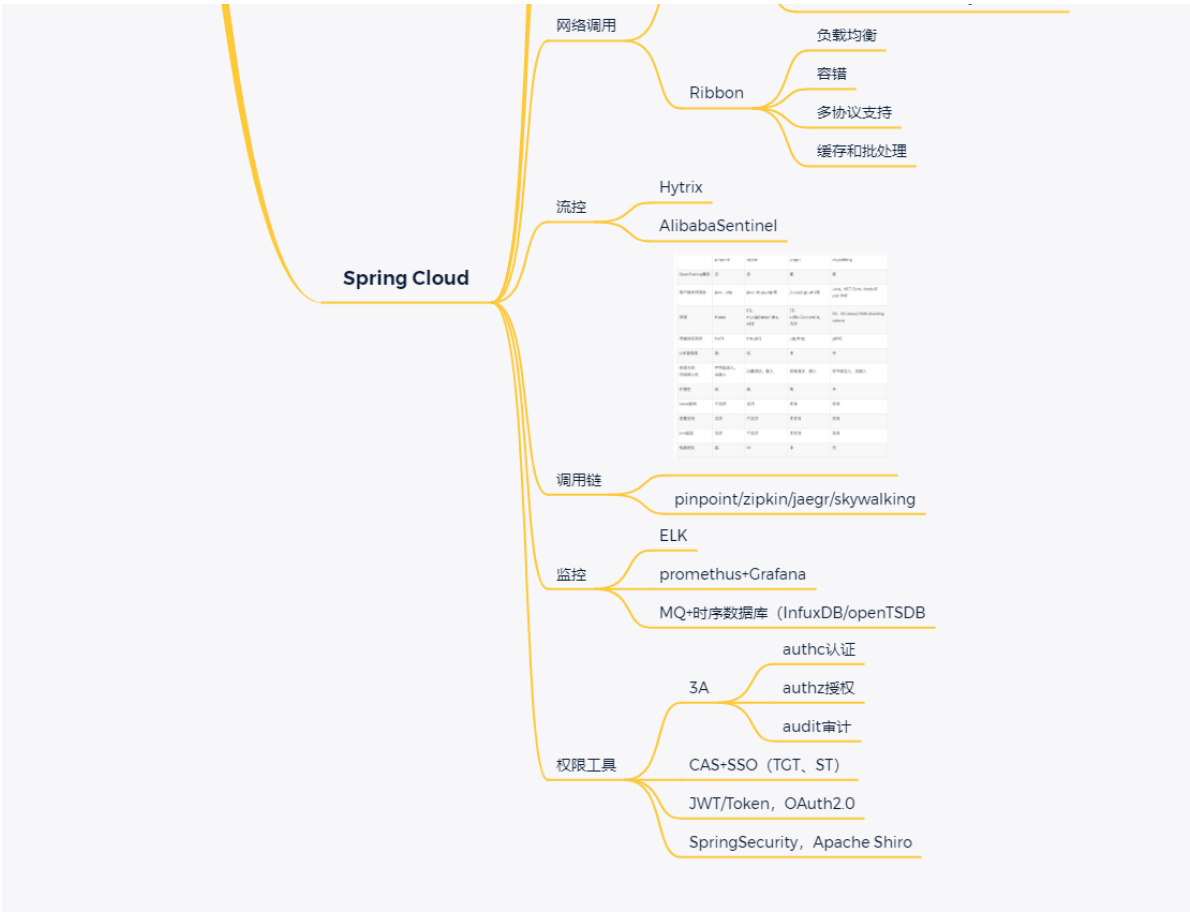
非功能性

- 配置中心
  - 大规模集群下，管理配置信息
  - zookeeper、etcd、nacos、apollo
- 注册中心
  - 让消费者能动态知道生产者集群的状态变化
  - nacos、eureka、consul、zookeeper
- 元数据中心
  - 定义所有业务服务的模型
- 服务集群与路由
  - 服务路由
    - 基于IP过滤
    - 服务带上tag，用tag匹配调用范围
  - 服务负载均衡
    - Random
    - RoundRobin
    - LeastActive
    - ConsistentHashLoadBalance
- 服务过滤与流控
  - 过滤
    - 管道+过滤器
  - 流控
    - 限流
    - 服务降级
    - 过载保护

微服务

- 发展历程
  - 响应式微服务
    - 自治性
    - 异步性
    - 伸缩性
    - 弹性
  - 服务网格与云原生
    - 将服务间的网络通信层及其控制策略下沉到基础设施，就形成所谓的"服务网格"技术
  - 数据库网格
  - 单元化架构
    - 每个单元是一个五脏俱全的缩小版整站，它是全能，但是数据不是全部
- 最佳实践
  - 遗留系统改造
    - 功能剥离、数据解耦
    - 自然演进，逐步拆分
    - 小步快跑，快速迭代
    - 灰度发布，谨慎试错
    - 提质量线，还技术债
  - 自动化管理
    - 自动化测试
    - 自动化部署
    - 自动化运维
  - 恰当粒度拆分
    - 高内聚低耦合
    - 不同阶段拆分要点不同
  - 分布式事务
    - 柔性事务：TCC、AT
    - 幂等、去重、补偿
  - 拓展立方体
    - 水平复制：复制系统
    - 功能解耦：拆分业务
    - 数据分区：切分数据
  - 完事监控系统
    - 业务监控
    - 系统监控
    - 容量规划
    - 报警预警
    - 运维流程
    - 故障处理

- 注册中心
  - Config/Eureka/Consul
- 网关
  - Zuul/Zuul2/Sprng Cloud Gateway
- Feign
  - 全部基于注解，简单方便
  - 和XXTemplate一样，内置了简化操作，OOP
  - 跟其他组件，ribbon、hytrix联合使用



目前互联网行业面对业务越来越复杂，单体模式进过分层、SOA演化，最终演化成微服务。

- rpc & dubbo

  rpc框架让远程调用就向本地调用一样简单方便，提升了开发的速度。在实际工作中用的最多就是webservice，feign。rpc框架原理简单来说，就是rcp客户端先通过动态代理，创建调用的接口实例；在调用代理的接口时，则序列化请求信息，再通过网络发送给服务提供方；服务提供收到请求后，反序列化，调用实际的服务接口，并将结果序列化后通过网络返回客户端；客户端再进行反序列，返回给代理接口调用者。当然这部分是简单实现，真正实际生产环境，还需要增加很多非功能需求，如服务注册、服务发现、服务负载均衡、流量控制等，具体的就是dubbo的开源实现了。

- 微服务

  单体应用有这简单，方便维护的好处，微服务是解决业务复杂的一种方法，不是银弹，也带来很多问题，如：服务配置，服务集群与路由，流控等。当然，目前都成熟的方案来解决这些问题，目前火热的Spring Cloud技术栈，在微服务领域有整套的解决方案，注册中心有Consul、Eureka、业务网关有Zuul、Spring Gateway，接口发布与调用有Feign、Ribbon；流量控制有Hytrix、Sentinel；调用链有pinpoint、zipkin、skywalking；日志监控有ELK、promethus+Grafana等待。