访问远程服务

- **远程服务调用RPC**
  - **进程间通信IPC**
    - **通信方式：**管道或具名管道，负责在进程间传递字符流或字节流。
    - **通信载体**
      - 信号
      - 信号量
      - 消息队列
      - 共享内存
      - 本地套接字接口
  - **通信成本**
    - **通过网络进行分布式运算的八宗罪**
      - “网络是可靠的”
      - “延迟是不存在的”
      - “带宽是无限的”
      - “网络是安全的”
      - “拓扑结构是一成不变的”
      - “总会有一个管理员”
      - “不必考虑传输成本”
      - “网络都是同质化的”
    - **RPC定义：**指位于互不相同的内存地址空间中的两个程序，在**语言层面**上，以同步的方式来使用带宽有限的信道来传输程序控制信息。
    - **RPC与IPC区别：**RPC是一种高层次的或者说语言层次的特征，而IPC是低层次的或者说系统层次的特征。
  - **RPC协议三个基本问题**
    - **如何表示数据：**包括传递给方法的参数和方法返回值，其实就是**序列化协议**。如Protocol Buffer、JSON、XML等序列化。
    - **如何传递数据：**指如何通过网络在两个服务的EndPoint之间交换数据，称为Wire Protocol。如SOAP、HTTP协议等。
    - **如何表示方法：**指如何用跨语言的标准表示特定方法。如使用UUID、Web服务描述语言WSDL、接口描述语言IDL等。
  - **统一的RPC**
    - **跨系统、跨语言的CORBA：**设计繁琐，大量无效冗余代码。指定的规范脱离实际，导致后续各个语言厂商有自己解读，最终各不兼容。
    - **Web Service：**基于XML，缺点是性能较差，且包含一大堆其他协议，学习困难。
  - **分裂的RPC**
    - **朝着面向对象发展：**不满足于RPC将面向过程编码方式带到分布式，希望能进行跨进程的面向对象编程。如RMI、.NET Remoting。
    - **朝着性能发展：**性能取决于序列化效率和信息密度。序列化输出结果的容量越小，速度越快，效率越高。信息密度取决于有效负载占总传输数据的比例大小，使用的传输协议层次越高，密度越低。如gRPC和Thrift都有专用序列化器，gRPC基于HTTP/2，支持多路复用和Header压缩，Thrift基于传输层TCP协议。
    - **朝着简化发展：**协议简单轻便，接口和格式更为通用，尤其适合Web浏览器这种不会有额外协议支持的应用场合。代表有JSON-RPC。
    - **向更高层次与插件化发展：**不再追求独立解决RPC三个问题，将一部分功能设计为扩展点，由用户选择。聚焦于提供核心的更高层次功能，如负载均衡、服务注册等。代表为阿里的Dubbo，支持多种传输协议选择，支持多种序列化器选择。
- **REST设计风格**
  - **RPC和REST区别**
    - **思想上：**面向过程和面向资源的编程思想不同
    - **概念上：**REST并不是一种远程服务调用协议，只是一种风格，不带有一定规范性和强制性。
    - **范围上：**RPC的发展方向包括分布式对象、提升调用效率、简化调用复杂性。其他，分布式对象的应用与REST毫无关系；而REST应用最多的浏览器可选传输协议和序列化器不多，提升调用效率困难；基于简化调用复杂性，几乎只有JSON-RPC可与REST竞争。
  - **REST相关概念**
    - **资源：**信息数据本身
    - **表征：**资源的不同表现形式
    - **状态：**基于特定场景的上下文信息
    - **转移：**服务端将表征经过某种操作实现状态转移。即”表征状态转移“。
    - **统一接口：**HTTP中GET、PUT、POST等，表征状态转移的具体方式
  - **REST六大原则**
    - **客户端与服务端分离：**将用户界面关注逻辑与存储关注的逻辑分离，提高用户界面跨平台的可移植性。

- **无状态：**服务端不用负责维护状态，客户端来处理或维护必要的上下文信息、会话信息。但绝大多数系统都无法满足，因为大型系统上下文信息庞大。
- **可缓存：**无状态提升了系统可见性、可靠性和可伸缩性，但降低网络性，使得有状态的设计在无状态时需要多次请求。REST希望客户端与中间的通信传递者如代理可以将部分服务端的应答缓存起来。
- **分层系统：**客户端无需知道是否直接连接到最终服务器或中间服务器，中间服务器可以使用负载均衡和共享缓存提高系统可扩展性，便于缓存、伸缩和安全策略部署。代表应用是内容分发网络CDN。
- **统一接口：**REST希望面向资源编程，侧重于系统该有哪些资源，使用HTTP的统一接口来实现资源相关的操作
- **按需代码：**可选原则，指任何按照客户端请求将可执行代码从服务端发送至客户端的技术，如Java Applet。

- **REST好处**
  - **降低服务接口学习成本**
  - **资源天然具有集合与层次结构**
  - **REST绑定与HTTP协议，既是好处又是缺点**
- **REST成熟度RMM**
  - **第0级：**完全与REST不相关
  - **第1级：**引入资源概念，通过资源ID作为主要线索与服务交互
  - **第2级：**引入统一接口，使用HTTP协议的Status Code、Header等信息
  - **第3级：**超文本驱动，请求应该自己描述清楚后续可能发生的状态转移
- **不足与争议**
  - **面向资源编程对CURD之外操作设计麻烦**，面向过程思想符合计算机世界主流交互方式，面向对象思想符合现实世界主流交互方式，面向资源思想符合网络世界主流交互方式。
  - **REST与HTTP完全绑定，不适合应用于要求高性能传输场景**
  - **REST协议不具有直接支持分布式服务对多个数据同时提交的统一协调能力**，但实现最终一致性是可以的。
  - **REST没有传输可靠性支持**，若未收到响应，无法确定服务端操作进行到哪一步，只能重发一次请求，这要求服务具有幂等性。
  - **REST缺乏对资源进行部分和批量处理的能力**
  - GraphQL可以解决上述问题，但是脱离了HTTP，推广成为问题