China · Beijing

# 微服务的集成，远远不止一个API

王延炯 博士

主任架构师
普元信息 软件产品部

ArchSummit
全 球 架 构 师 峰 会 2016

[ 北京站 ]

主办方 Geekbang. 极客邦科技 InfoQ

# 关于我

普元 PRIMETON

- 最初从安全领域切入企业IT
- 长期参与SOA / 服务治理 / OpenAPI 等分布式中间件的交付

- 经常思考与反思：企业敏捷与精益运营、软件价值的本质

- 曾任：架构师，西门子（中国）研究院 / 信息安全部 (CT ITS)
- 曾任：资深架构师，平安健康互联网股份有限公司 / 技术平台部
- 现任：主任架构师，普元信息技术股份有限公司 / 软件产品部

# 大纲



1. 从现象出发，分析服务集成的背景与问题
2. 从本质出发，设计服务编排的落地路径
3. 从实现出发，剖析服务的运行态
4. 向未来看齐，机器学习离我们有多远？

# 1. 从现象出发，分析服务集成的背景与问题



- 手工代码集成服务
- REST
- Docker

# 一个现实中的例子



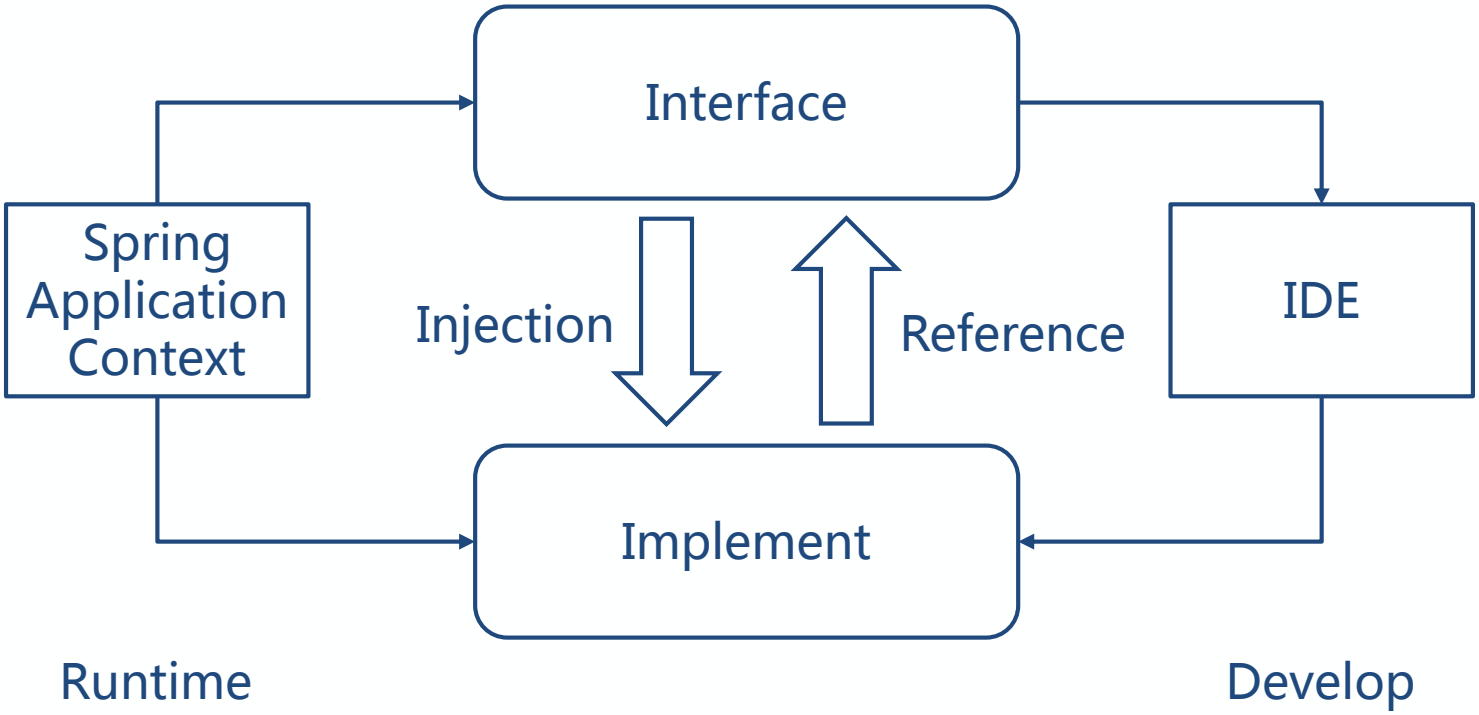
一个大约在9个月前的项目...

# 用开发大应用的思维，开发分布式服务，再通过API集成

普元 PRIMETON

架构演进与升级，普通程序员的思维习惯

- Method as Interface
- Service不能依赖Service
- Service依赖DAO



# 手工编码的服务集成，隐患重重 —— 编排太遥远

普元 PRIMETON

```
/**
 * Register user.
 *
 * <p>Request body schema:
 * {
 *     inviteCode: xxx,
 *     userName: xxx,
 *     email: xxx,
 *     password: xxx
 * }
 * </p>
 * @param userVO user info object
 * @return Register Result
 * @throws PortalCapabilityException
 */
@Override
public Boolean register(UserVO userVO) throws PortalCapabilityException {
  if (userVO == null) {
    throw new IllegalArgumentException("UserVO is null!");
  }
    if (StringUtils.isBlank(userVO.getUserName()) || userVO.getUserName().length() < 5) {
        throw new PortalCapabilityException(PortalErrorCode.inviteCode_user_name_verify_failed, "userName length must more than or equal to 5,but it is" + (userVO.getUserName() == null ? "0" : userVO.getUserName().length()));
    }
     if (!ValidateUtil.validateUserName(userVO.getUserName())) {
         throw new PortalCapabilityException(PortalErrorCode.inviteCode_user_name_verify_failed, "userName must be '_',  digital or English,but it is "+ userVO.getUserName());
     }

  if (StringUtils.isBlank(userVO.getUserPassword()) || userVO.getUserPassword().length() < 8) {
    throw new PortalCapabilityException(PortalErrorCode.inviteCode_user_password_verify_failed, "password length:" + (userVO.getUserPassword() == null ? "0" : userVO.getUserPassword().length()));
  }
  if (StringUtils.isBlank(userVO.getUserNickName())) {
    if (!StringUtils.isBlank(userVO.getUserEmail())) {
      String[] emails = userVO.getUserEmail().split("@");
      if (emails.length > 0) {
        userVO.setUserNickName(emails[0]);
      }
    }
  }

  try {
    InviteCodeVO inviteCodeVo = null;
    if (StringUtils.isNotEmpty(userVO.getInviteCode())) {
      inviteCodeVo = inviteCodeSpi.getInviteCode(userVO.getInviteCode());
      if (!StringUtils.equals(inviteCodeVo.getToUserEmail(), userVO.getUserEmail())) {
        throw new PortalCapabilityException(PortalErrorCode.inviteCode_ser_mail_verify_failed, "userEmail must be invited.");
      }
    }
    if (userSpi.exists(userVO.getUserName())) {
      throw new PortalCapabilityException(IamErrorCode.user_already_existed, "user already existed, cannot create: "+ userVO.getUserName());
    }
    //注册
    userVO = userSpi.register(userVO);
    log.info("step1: IAM create user success!");

    //增加VCS账户
    if (userVO != null) {
      vcsUserSpi.createUser(userVO);
      log.info("step2: VCS create user success!");
    }

    if (StringUtils.isEmpty(userVO.getInviteCode())) { //普通用户
      //设置角色
      userSpi.assignRoles(userVO.getUserName(), new String[]{PlatformRoleVO.ORDINARY_USER_ROLE_CODE});
      log.info("step3: assign user success!");

      //创建BPR用户
      bprSpi.create(userVO);
      log.info("step4: BPR create user success!");

      //增加租户关系
      TenantUserRelationVO[] tenantUserRelationVOs = tenantUserRelationSpi.queryTenantUserRelationsByEmail(userVO.getUserEmail());
      for (TenantUserRelationVO tenantUserRelationVO : tenantUserRelationVOs) {
        tenantUserRelationVO.setUserStatus(TenantUserRelationVO.USER_STATUS_REGISTERED);
        tenantUserRelationSpi.saveTenantUserRelation(tenantUserRelationVO);

        //BPR赋权限
        bprSpi.appendAuth(userVO.getUserName(), tenantUserRelationVO.getTenantCode() + "~r");
      }
      log.info("step5: create relationship between user and tenant success!");
      log.info("step6: BPR authorization success!");

    } else { //租户管理员
      //设置角色
      userSpi.assignRoles(userVO.getUserName(), new String[]{PlatformRoleVO.TENANT_ADMIN_ROLE_CODE, PlatformRoleVO.PRODUCT_MANAGER_ROLE_CODE});
      log.info("step3: assign tenantManager success!");

      //增加租户关系
      String tenantCode = inviteCodeVo.getTenantCode();
      TenantUserRelationVO tenantUserRelationVO = new TenantUserRelationVO();
      tenantUserRelationVO.setTenantCode(tenantCode);
      tenantUserRelationVO.setUserName(userVO.getUserName());
      tenantUserRelationVO.setUserEmail(userVO.getUserEmail());
      tenantUserRelationVO.setUserStatus(TenantUserRelationVO.USER_STATUS_REGISTERED);
      tenantUserRelationSpi.saveTenantUserRelation(tenantUserRelationVO);
      log.info("step4: create relationship between tenantManager and tenant success!");

      //更新租户信息
      Tenant tenant = tenantSpi.getTenantByCode(tenantCode);
      tenant.setUserId(userVO.getId());
      tenant.setUserName(userVO.getUserName());
      tenantSpi.modifyTenantInfo(tenant);
      log.info("step5: update information of tenant success!");

      //创建BPR仓库并赋权限
      bprSpi.create(userVO); //创建BPR用户
      bprSpi.createMavenHosted(tenantCode); //创建BPR仓库
      bprSpi.createMavenGroup(tenantCode);  //创建BPR组
      bprSpi.createBatch(tenantCode);   //创建BPR角色
      bprSpi.appendAuth(userVO.getUserName(), tenantCode + "~rw");
      log.info("step6: BPR create and authorization success!");
    }

    return true;
  } catch (PortalCapabilityException e) {
    throw e;
  } catch (PortalResourceException e) {
    cancelRegister(userVO);
    throw new PortalCapabilityException(e.getErrorCode(), e.getMessage(), e);
  }
}
```

- 120行代码
  - 25行注释
  - 12行空行
  - 83行功能
    - 12次参数校验
    - 18次RPC
      - 4查询
      - 14次写操作
    - 6大业务步骤
      - 注册用户
      - 设置角色
      - 注册VCS
      - 注册Nexus用户
        - 新增
        - 赋权
      - 维护租户关系
      - 维护Nexus权限
      - …

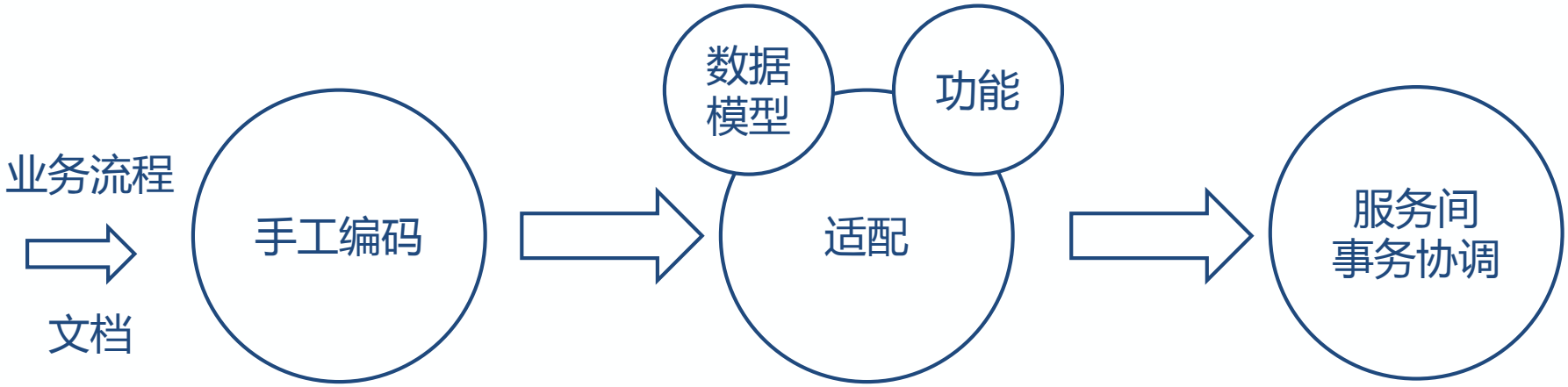
绝大部分RPC没有事务控制！

# 服务集成、编排的现状 —— 缺乏工程性的手工编码



交付应用的程序员，其价值就是翻译业务逻辑？

既要比业务部门的人员更精通业务规则，还要驾驭分布式计算机系统，要求太高




代码质量，不能仅仅取决于老师傅的手艺

# 另一个现实中的例子



初见REST

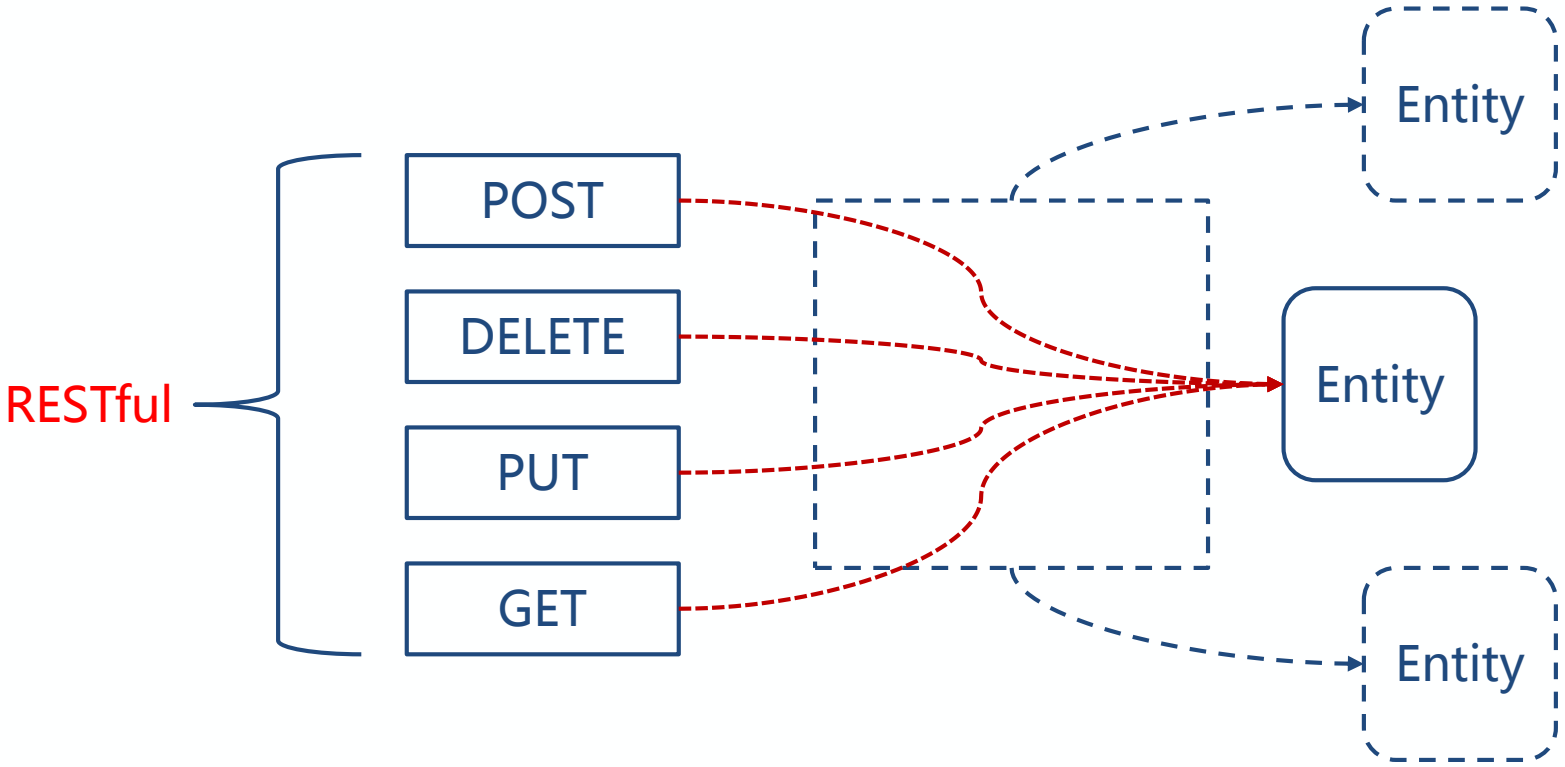
…

初用REST

…

# 服务的操作行为，应包含过程与结果(实体)两个方面
# ——REST的重点在单个实体






- 绝大多数的业务场景，被操作的都不止一个实体，且存在(分布式)事务
- REST的关注重点在单个实体

# 几乎所有的REST科普例子，都以单个实体操作为中心

POST http://<domain>/articles/

DELETE http://<domain>/articles/123

GET http://<domain>/articles/123

PUT http://<domain>/articles/123

问题是
在POST文章时

Article

1 Category 如果Category索引更新失败？

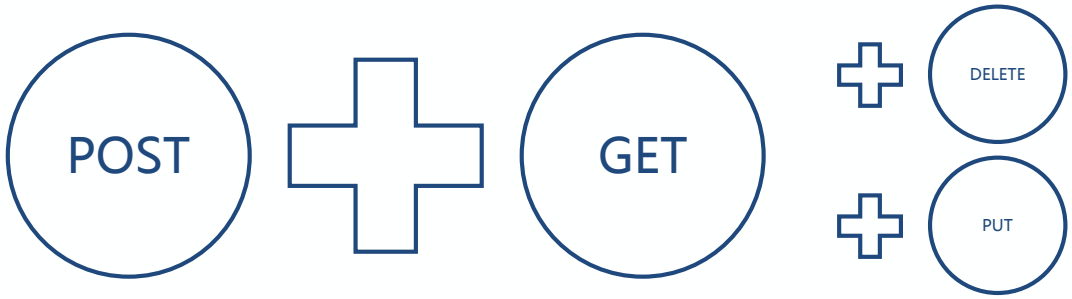
1 Tags 如果Level不够，社区不允许新建Tag？

POST ? PUT http://<domain>/categories/software

POST ? PUT http://<domain>/tags/architect

POST ? PUT http://<domain>/tags/microservice

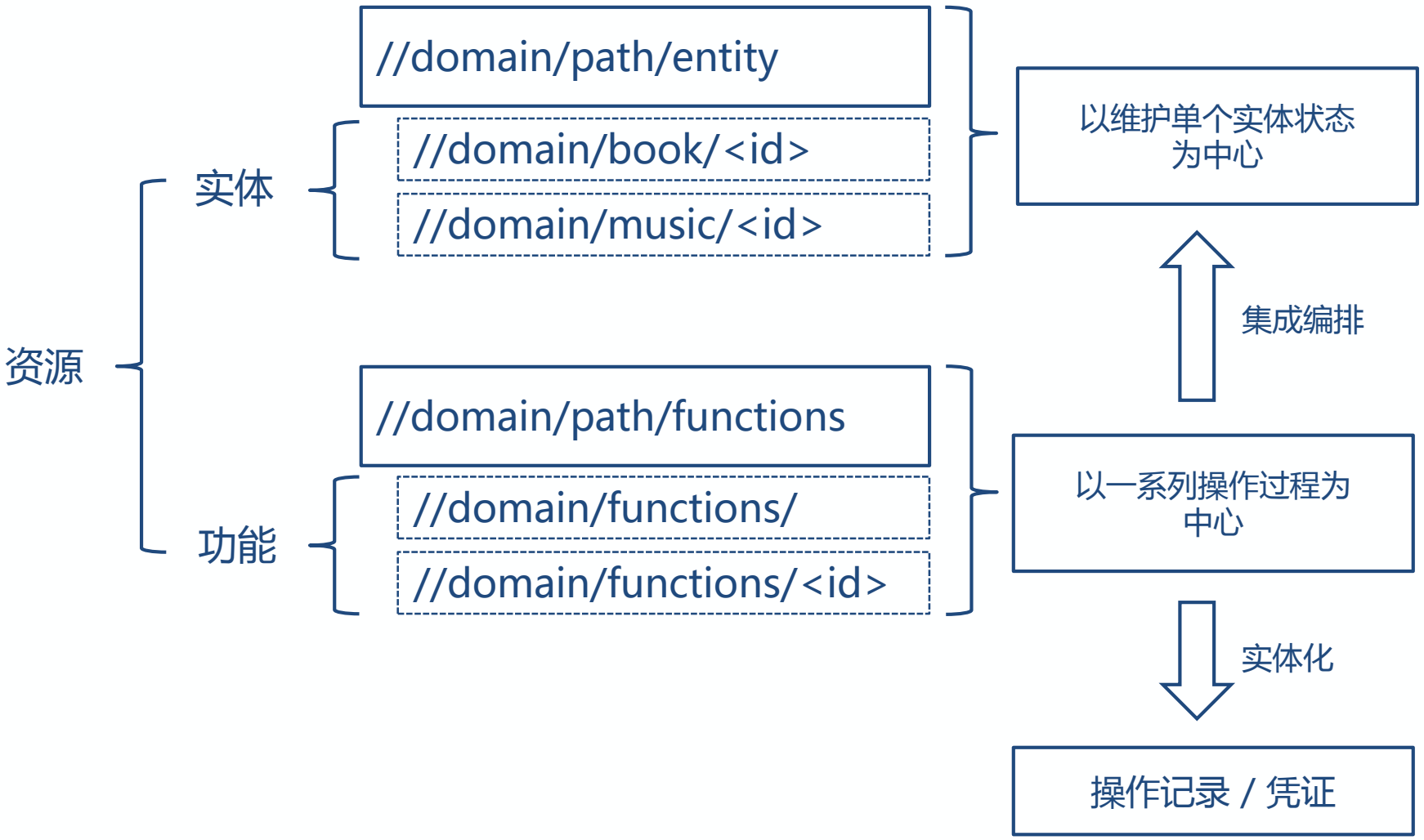
# 几乎周围的REST实践例子，都以POST+GET为主




## 原因很多

- HTTP Method 不熟悉 → POST不是什么都能干嘛？
- HTTP Header 不熟悉 → 浏览器里有设置吗，调试怎么调？
- HTTP Status Code 不熟悉 → 到底是技术含义，还是业务含义？
- Request 内容复杂 → 浏览器URL有长度限制，2KB，4KB，8KB？
- 想明白了，没有人力资源对其进行改造
- 想明白了，团队没有现成的框架，水平不一的程序员开发出来的结果 □

# REST & URL：资源本应该有两类 —— 实体 & 功能



# Docker

# Docker？——对于服务的集成太薄



软件的使用价值 = 代码+数据 ⇒ 薄弱的根本原因：不懂数据

正面意义

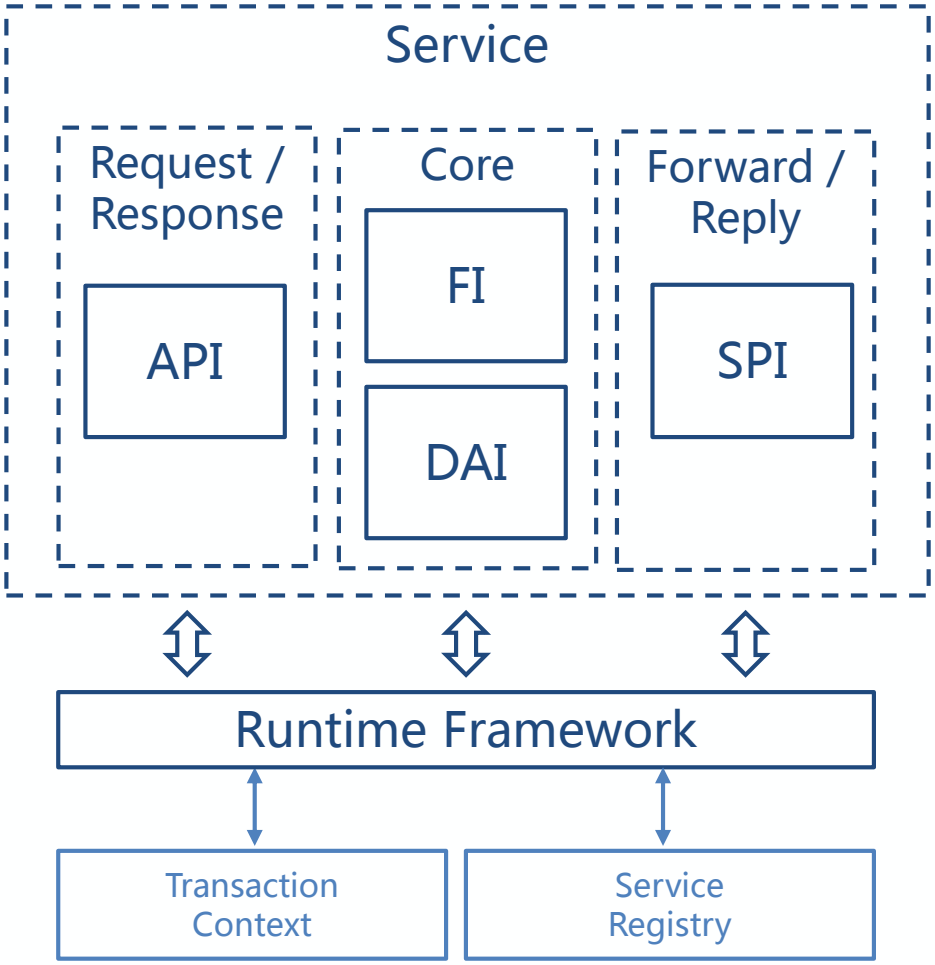
- 标准化运行环境基线 → OS + Service/App RTF
- Docker Compose → 标准化应用/服务的部署集群
- Docker Container → OS Level 虚拟化

# 2. 从本质出发，设计服务编排的落地路径

普元 PRIMETON

- 服务在代码上的物理形态
- 服务的三种基本工作模式
- HTTP/1.1 在 CC / TCC 模式服务中的应用
- 元数据驱动数据适配的两种方式

# 高内聚 / 松耦合，用代码定义服务 —— ASDF模型

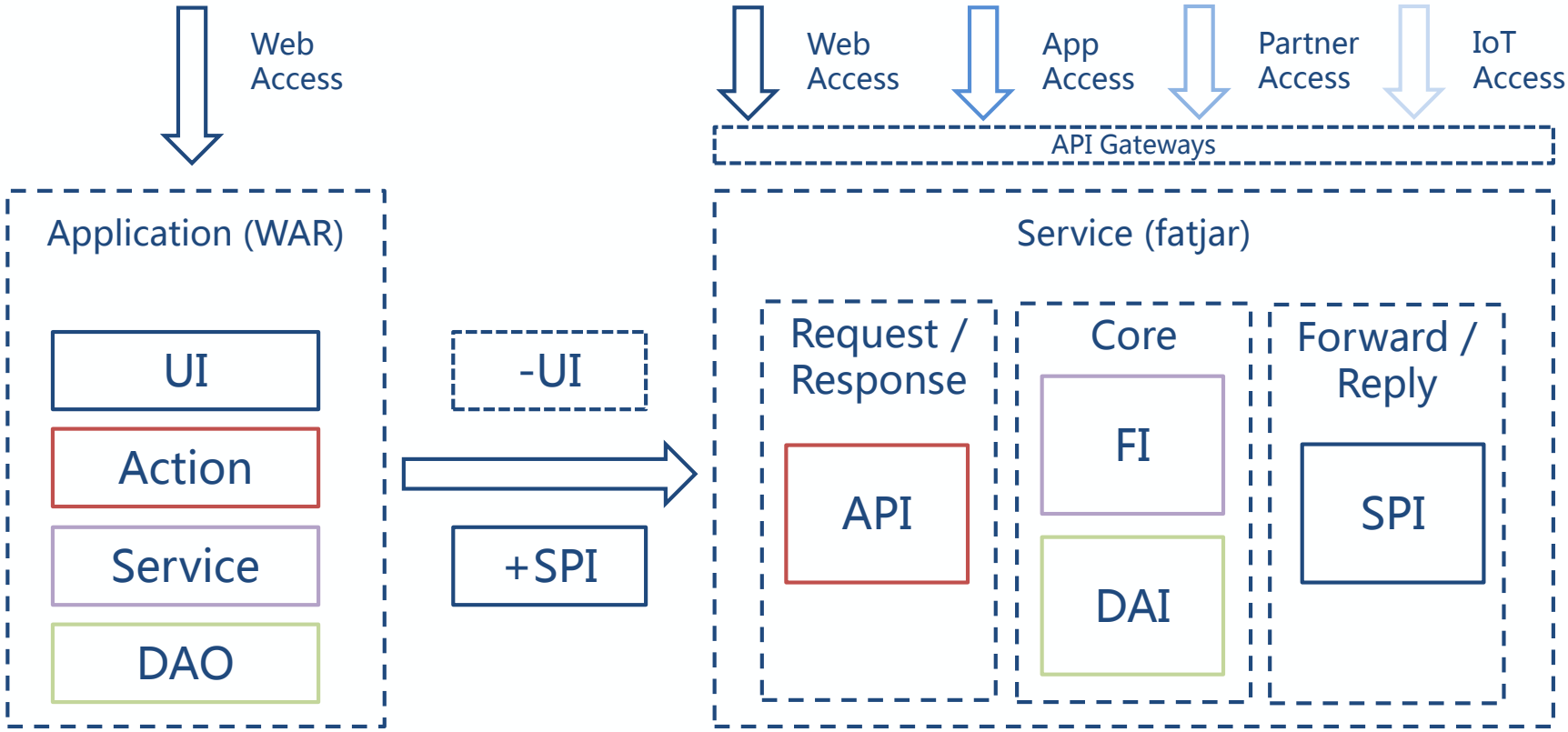



- API 请求 / 响应
- SPI 转发 / 应答
- DAI 数据访问
- FI 基础独立功能

SPI 与 DAI

强弱互成反比

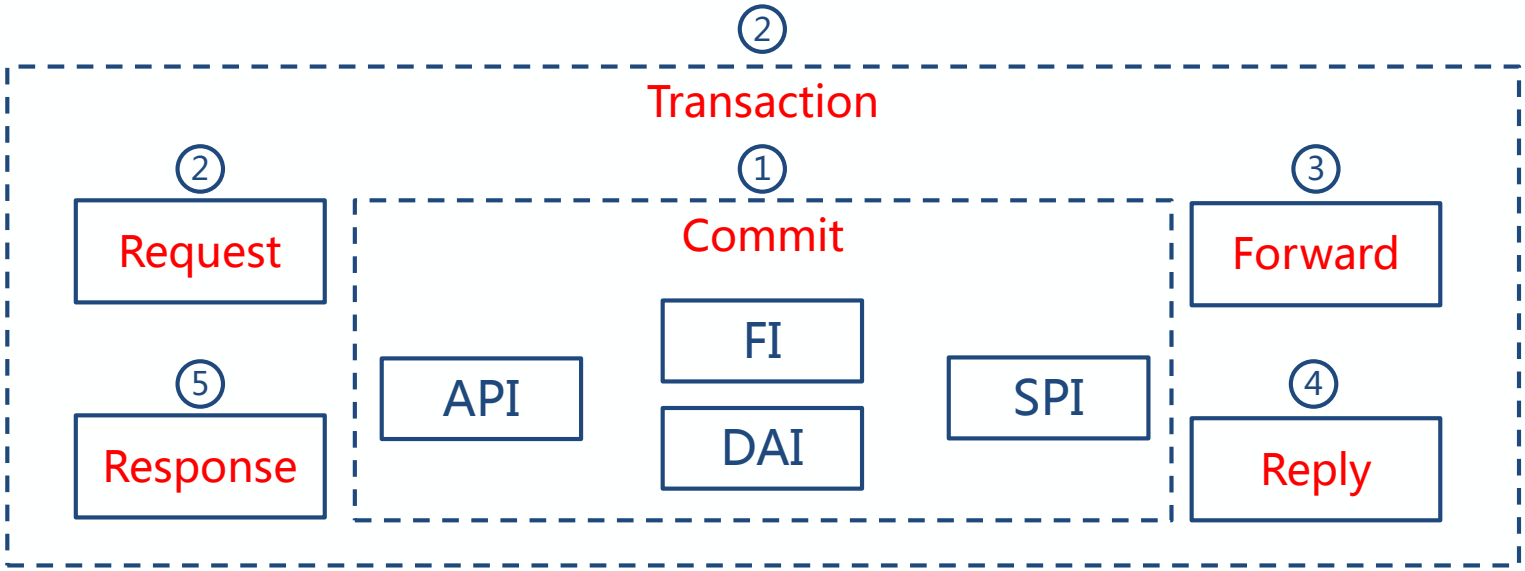
# 一加一减，是微服务与单体应用的距离—— +SPI -UI




代价

- 线程切换 → RPC，可靠性下降
- 网络 QoS / SLA 很少在系统设计中被提及
- 数据模型离散化，数据实体碎片化
- 多团队开发，从全栈交付到分栈交付
- ....

# 服务，在分布式上的基本要求

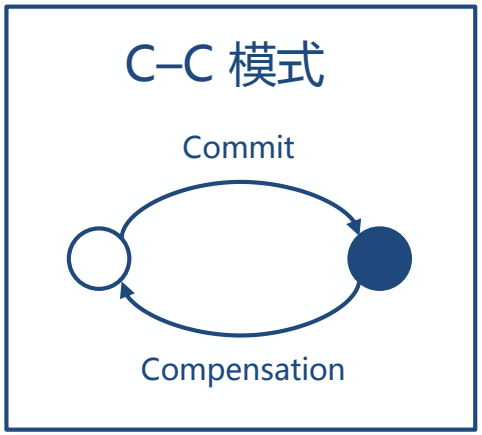


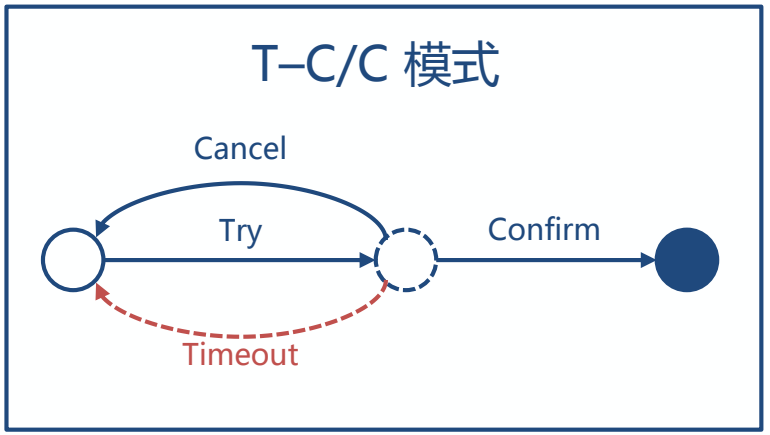

1. 生成commit
2. 解析请求报文
3. 转发请求（多个）
4. 解析转发结果（多个）
5. 响应请求

6组ID是关键

# 服务的三种基本工作模式

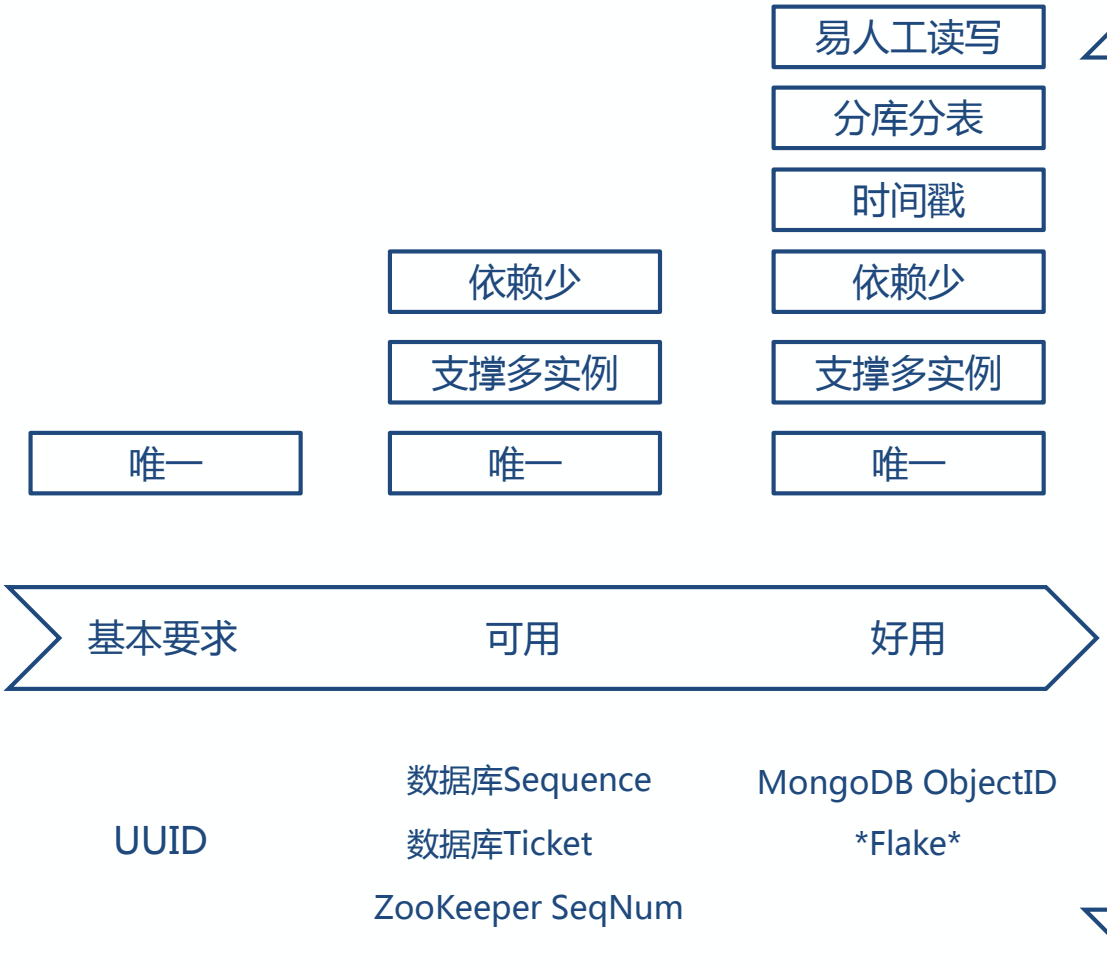
- Request Processing
  1. Commit - Compensation
  2. Try - Confirm / Cancel




- Event Processing
  3. Event Sourcing + Event Handler

# 唯一ID的生成，是实现幂等的必要条件




套路

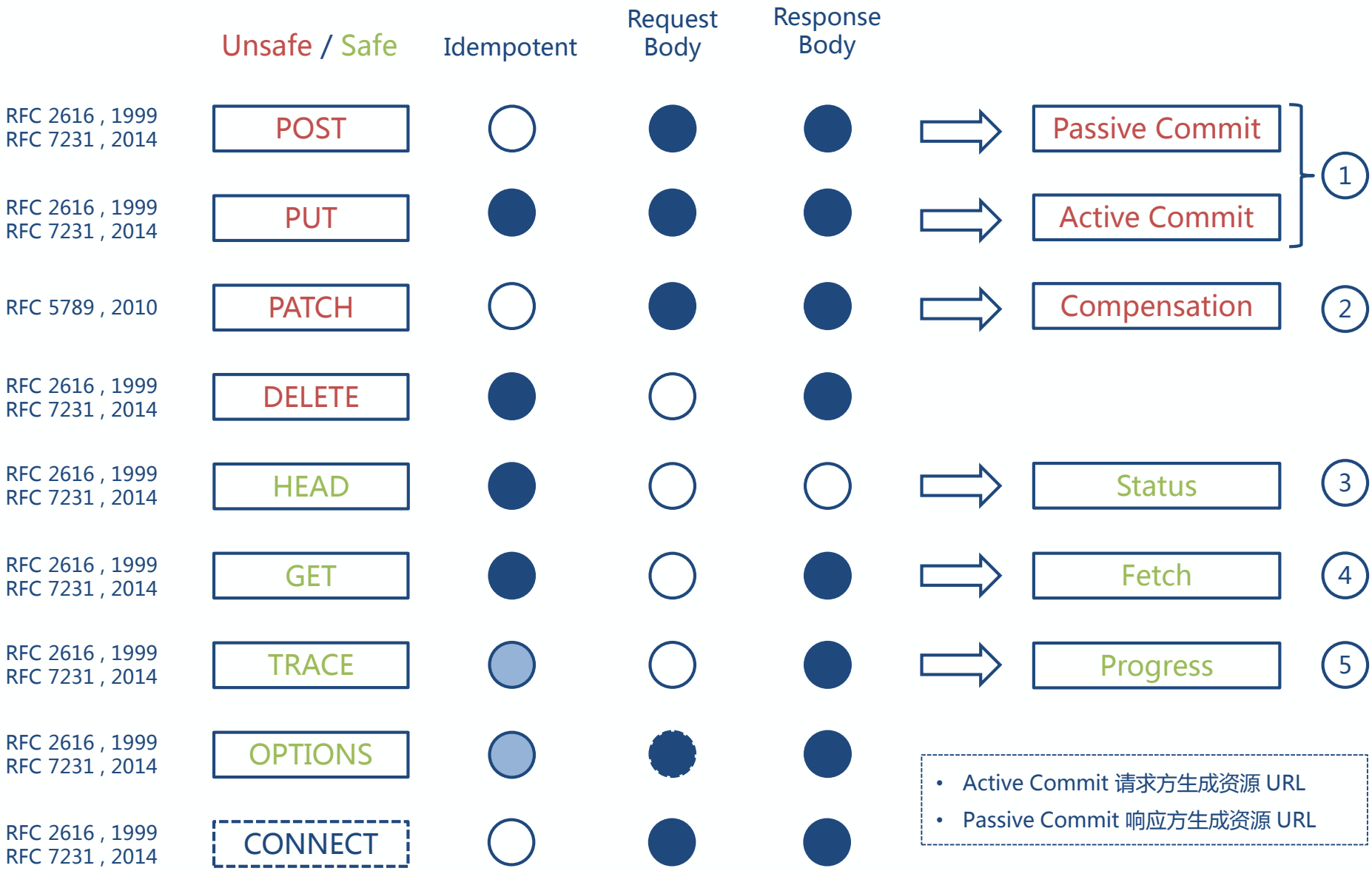
- 命名空间
  - 时间
  - 生成者*
- 递增序列
- 随机初始值
- 序列化/编码

*与DevOps结合

long, 使用方便，但64-bit容量太小
→ 128-bit 自己做序列化/反序列化

# 一个URL，六种用法(Before)
# ——用HTTP Method来表示一个CC服务

普元 PRIMETON



# CC模式流水号在HTTP请求与响应中的位置(Before)

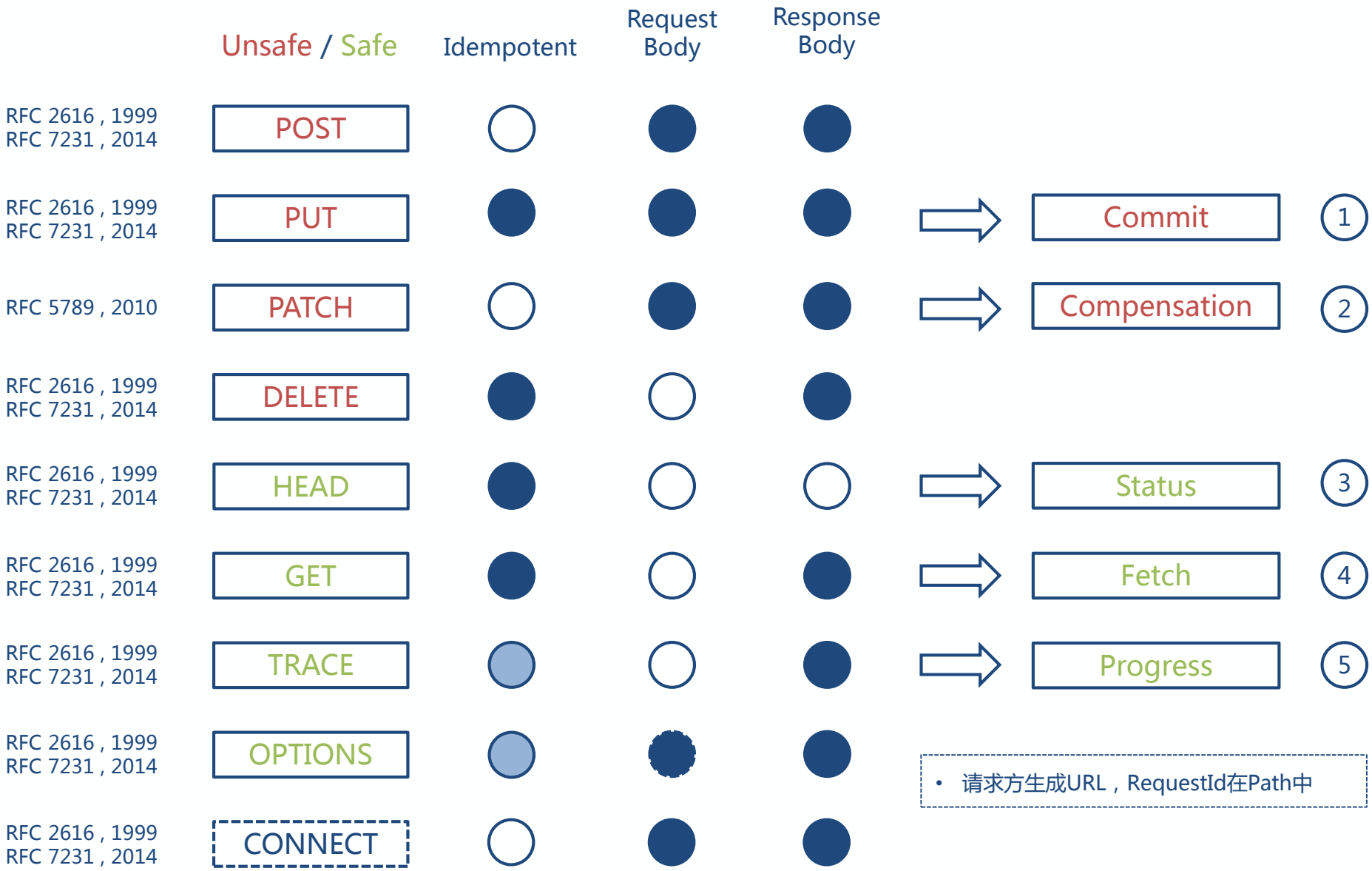
普元 PRIMETON

- Transaction URL: http(s)://<domain>/<paths>/{RequestId}

| | Commit (Passive) | Commit (Active) | Compensation | Status | Fetch | Progress |
|---|---|---|---|---|---|---|
| **HTTP Method** | POST | PUT | PATCH | HEAD | GET | TRACE |
| **RequestId (Request)** | - | Path | Path | Path | Path | Path |
| **RequestId (Response)** | HTTP Header X-CC-RequestId | HTTP Header X-CC-RequestId | HTTP Header X-CC-RequestId | - | - | - |
| **ResponseId (Request)** | - | - | - | - | - | - |
| **ResponseId (Response)** | HTTP Header X-CC-ResponseId | HTTP Header X-CC-ResponseId | - | - | - | - |
| **Transaction (Request)** | HTTP Header X-CC-TransactionId | HTTP Header X-CC-TransactionId | HTTP Header X-CC-TransactionId | - | - | - |
| **Transaction (Response)** | - | - | - | - | - | - |
| **C-RequestId (Request)** | - | - | HTTP Header X-CC-C-RequestId | - | - | - |
| **C-ResponseId (Request)** | - | - | HTTP Header X-CC-C-ResponseId | - | - | - |

# 一个URL，五种用法(After)
# ——用HTTP Method来表示一个CC服务

普元 PRIMETON



# CC模式流水号在HTTP请求与响应中的位置(After)

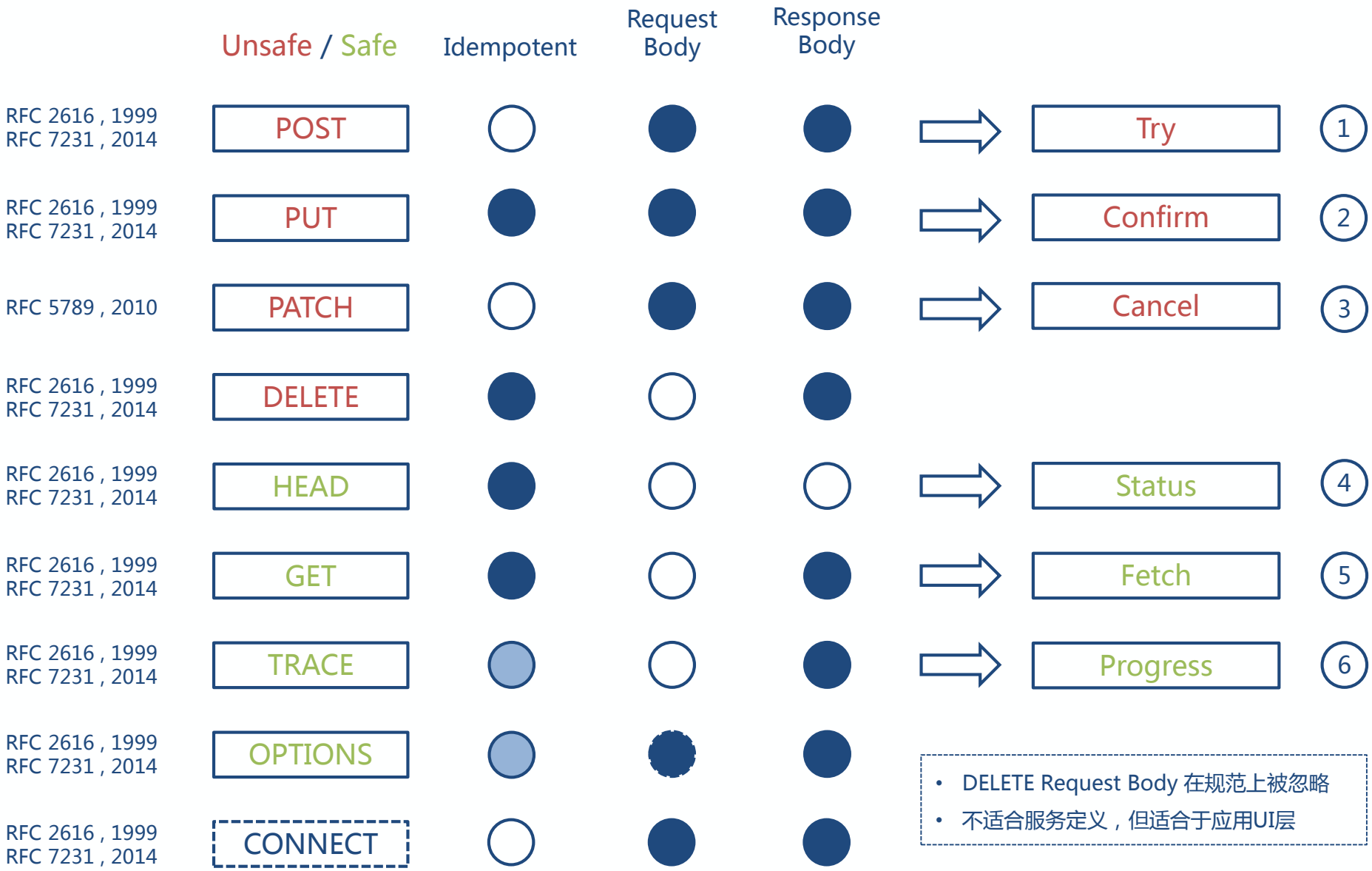
普元 PRIMETON

- Transaction URL: http(s)://<domain>/<paths>/{RequestId}

| | Commit | Compensation | Status | Fetch | Progress |
|---|---|---|---|---|---|
| **HTTP Method** | PUT | PATCH | HEAD | GET | TRACE |
| **RequestId (Request)** | Path | Path | Path | Path | Path |
| **RequestId (Response)** | HTTP Header X-CC-RequestId | HTTP Header X-CC-RequestId | - | - | - |
| **ResponseId (Request)** | - | - | - | - | - |
| **ResponseId (Response)** | HTTP Header X-CC-ResponseId | - | - | - | - |
| **Transaction (Request)** | HTTP Header X-CC-TransactionId | HTTP Header X-CC-TransactionId | - | - | - |
| **Transaction (Response)** | - | - | - | - | - |
| **C-RequestId (Request)** | - | HTTP Header X-CC-C-RequestId | - | - | - |
| **C-ResponseId (Request)** | - | HTTP Header X-CC-C-ResponseId | - | - | - |

# 一个URL，六种用法
# ——用HTTP Method来表示一个TCC服务

普元 PRIMETON



# TCC模式流水号在HTTP请求与响应中的位置

普元 PRIMETON

- Transaction URL: http(s)://<domain>/<paths>/{RequestId}

| | Try | Confirm | Cancel | Status | Fetch | Progress |
|---|---|---|---|---|---|---|
| **HTTP Method** | POST | PUT | PATCH | HEAD | GET | TRACE |
| **RequestId (Request)** | Path | Path | Path | Path | Path | Path |
| **RequestId (Response)** | HTTP Header X-TCC-RequestId | HTTP Header X-TCC-RequestId | HTTP Header X-TCC-RequestId | - | - | - |
| **ResponseId (Request)** | - | - | - | - | - | - |
| **ResponseId (Response)** | HTTP Header X-TCC-ResponseId | HTTP Header X-TCC-ResponseId | - | - | - | - |
| **Transaction (Request)** | HTTP Header X-TCC-TransactionId | HTTP Header X-TCC-TransactionId | HTTP Header X-TCC-TransactionId | - | - | - |
| **Transaction (Response)** | - | - | - | - | - | - |
| **C-RequestId (Request)** | - | - | HTTP Header X-TCC-C-RequestId | - | - | - |
| **C-ResponseId (Request)** | - | - | HTTP Header X-TCC-C-ResponseId | - | - | - |

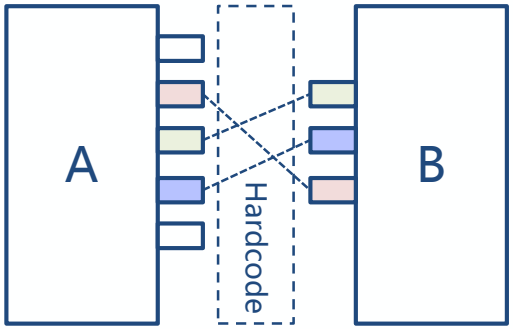
# 直接映射与间接映射，数据适配的两种方法

普元 PRIMETON

A / B是两个内部服务，对于同一个客体，有不同视角

手工

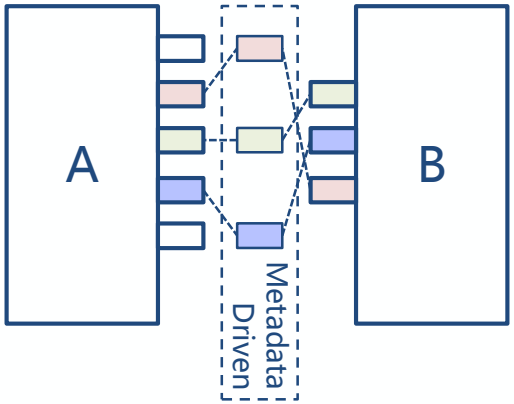
直接适配




- 方法：手工 / 代码生成工具
- 效率：运行期高 / 开发期低

元数据

间接适配




- 方法：适配引擎 / 元数据注解
- 效率：运行期低 / 开发期高

DevOps

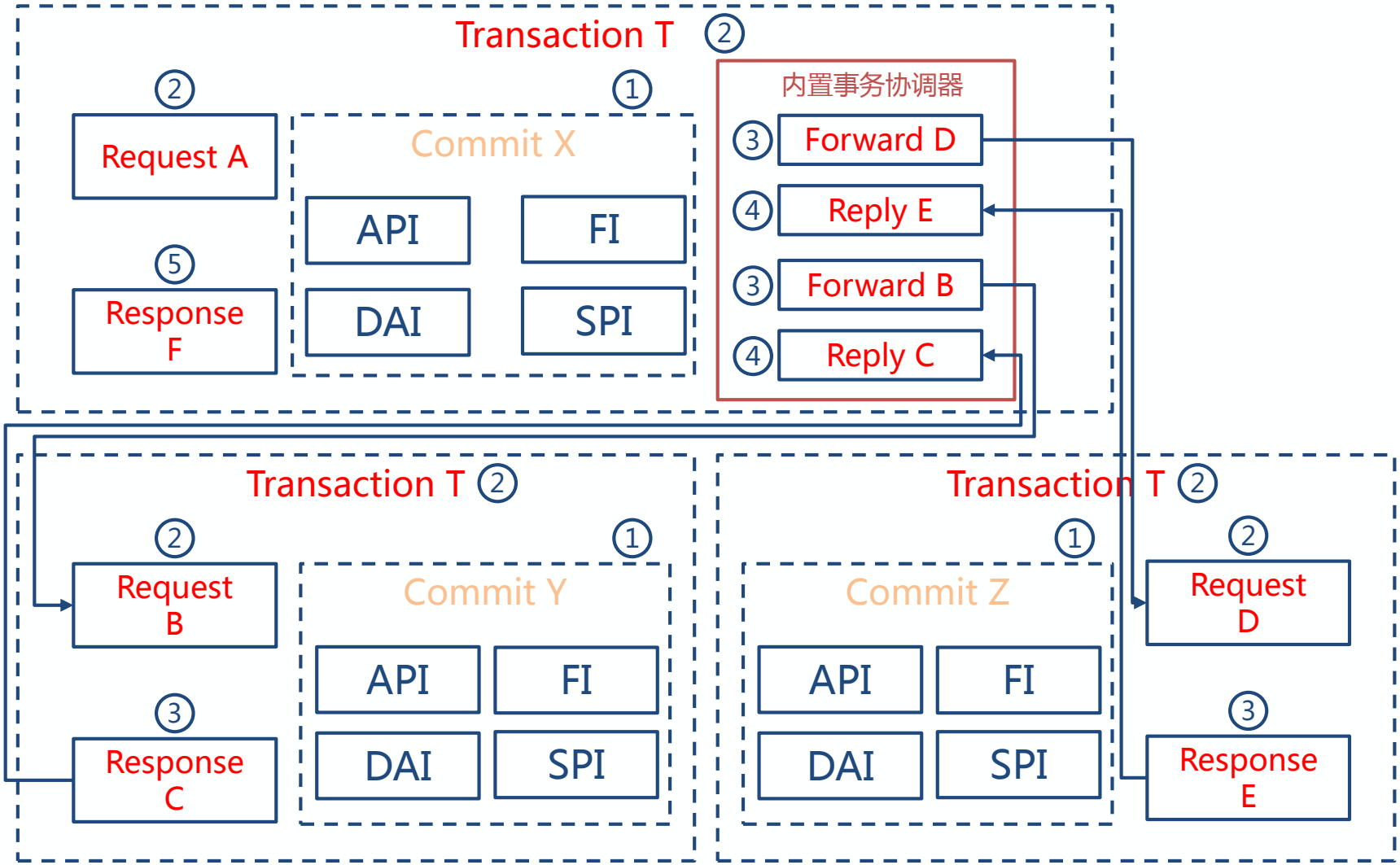
数据标准 ⇨ 适配二方库

# 3. 从实现出发，剖析服务的运行态



- 流水号的传递
- 根据上下文，计算服务的执行策略
- 服务协调器与服务的状态迁移

# Commit - Compensation 级联方式集成的流水号传递



# 研究生毕业转单——现实生活中的服务编排

北京邮电大学毕业研究生离校通知单

学院 _____ 学号 _____ 姓名 _____

前往贵处办理离校手续，请予以办理。

学生事务管理处

| | | |
|---|---|---|
| 研究生导师<br>签字 | 研究生院<br>(教1—430)<br>盖章 | 财务处<br>（后勤楼1层）<br>盖章 |
| 学院党委<br>（办理党组织关系） | 保卫处<br>（转户口关系）<br>（体育馆西厅北侧）<br>盖章 | 图书馆<br>（凭论文呈缴证明借阅证和一卡通办理）<br>盖章 |
| 校团委<br>（办理团组织关系） | | |
| 档案馆<br>(无需盖章，请登录档案馆主页，登记档案邮寄地址) | 学生处公寓中心<br>（本楼交钥匙，领退房清单）<br>（学八楼一层大厅）<br>盖章 | 学生事务管理处<br>（注销学生证）<br>（行政办公楼西侧院内）<br>盖章 |
| 一卡通<br>（办理注销）<br>（信息网络中心109） | 各学院<br>收通知单<br>发派遣证<br>毕业证 | |

注意事项：

1、导师签字→研究生院盖章→财务处盖章这三项应按顺序进行。图书馆应在一卡通之前办理。领取毕业证为最后一项，其他项无顺序要求。

2、档案馆主页进入：www.bupt.edu.cn(北邮主页→公共服务→档案馆→学生档案)

3、一卡通如代办注销，应凭委托人身份证或学生证办理。

注意事项1：

- 导师签字 → 研究生院盖章 → 财务处盖章
- 图书馆应在一卡通之前办理
- 领取毕业证为最后一项

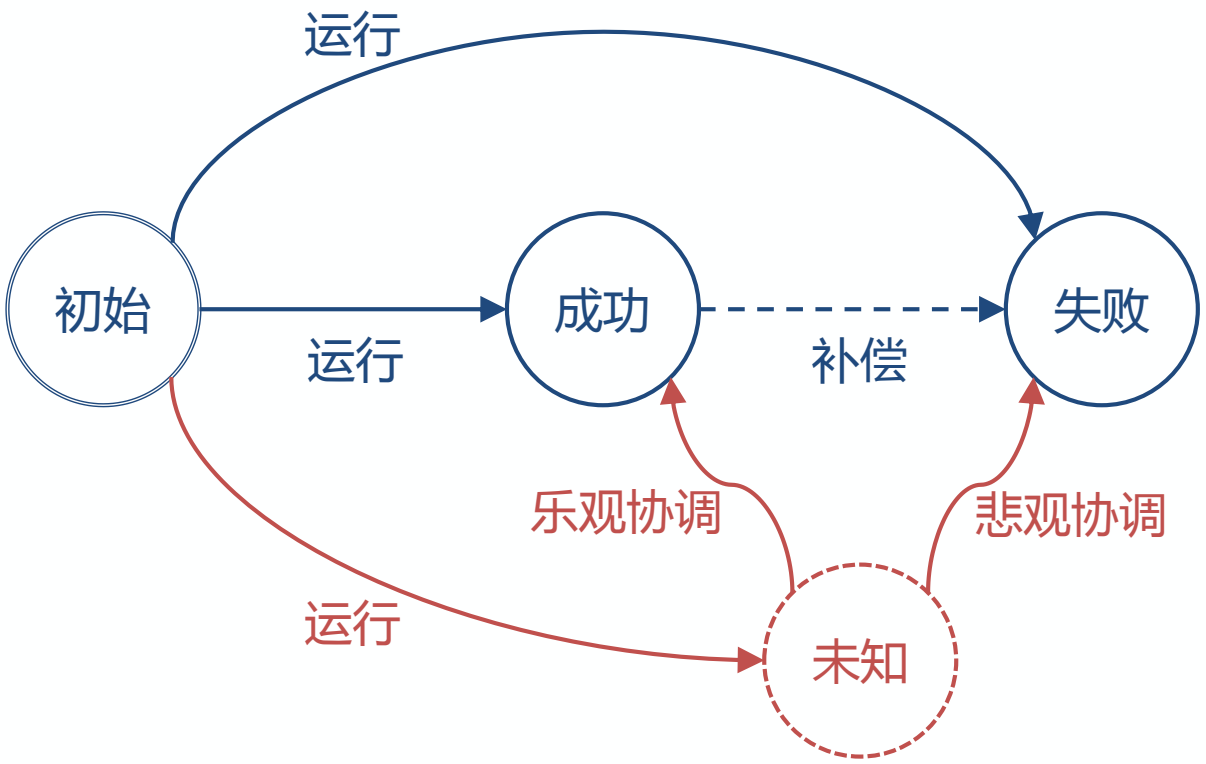
- 每一项都提供了「服务路由」
- 七个步骤可以并行，只是没有分身术

# CC服务运行期的自身状态迁移，与协调器能感知的状态

普元 PRIMETON

## 服务状态的两个视角

- 服务自身稳态
  - 初始
  - 成功
  - 失败
    - 表示失败的响应码
    - 可识别的异常
    - Connect Timeout
- 被协调器感知的状态
  - 未知
    - 不可识别的响应码
    - 不可识别的异常
    - Read Timeout

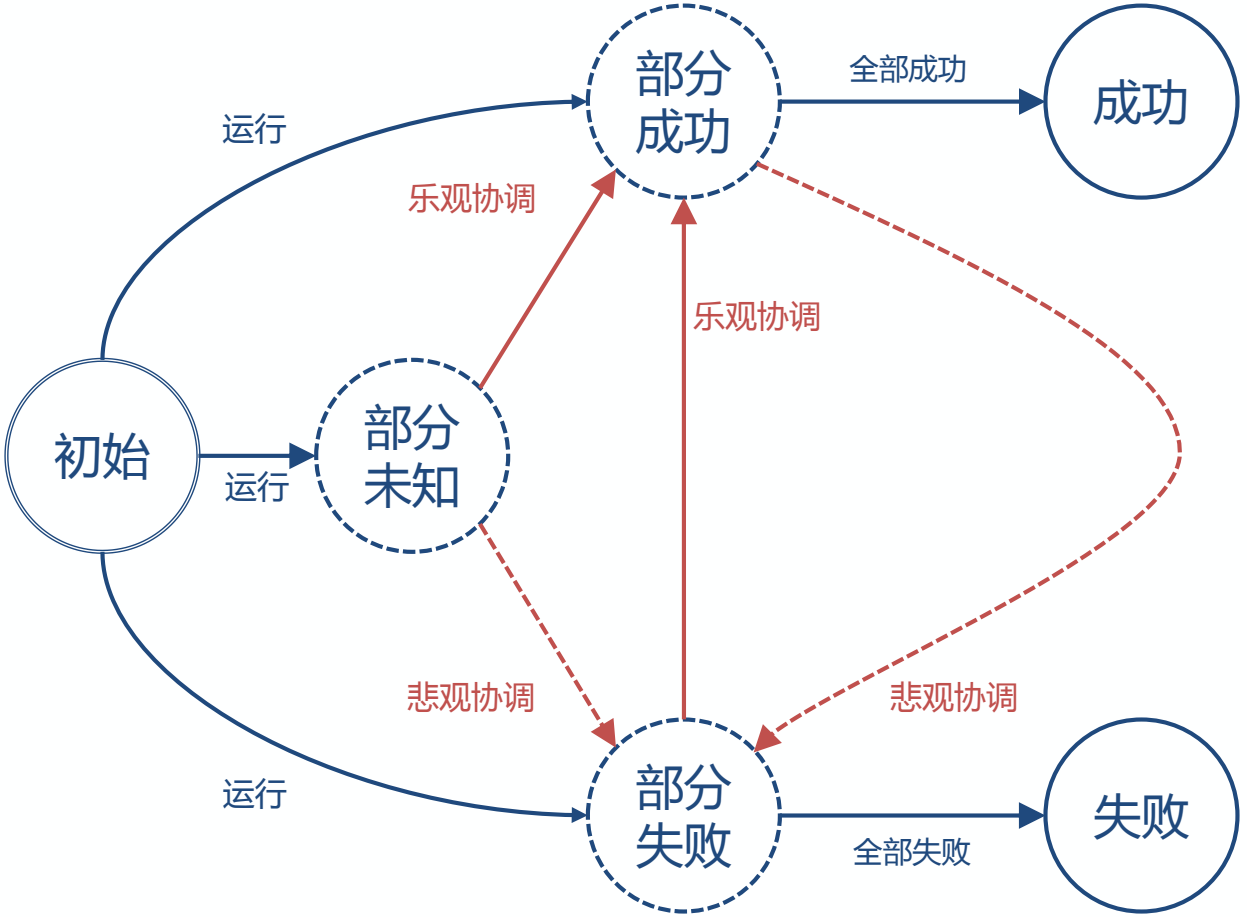


# CC服务协调器的运行期状态迁移

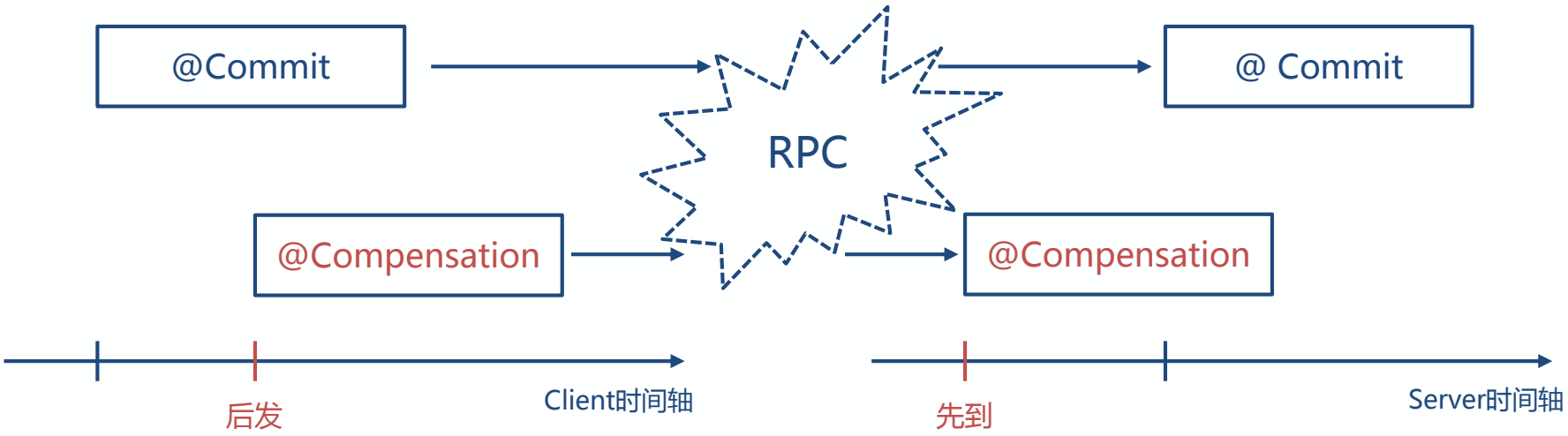
普元 PRIMETON

## 服务协调器

- 服务执行条件
  - 入参完备 / 前序正常
  - 结果成功 / 后续异常
- 服务执行顺序
  - 并行
  - 顺序
- 服务执行结果
  - 触发重试
  - 触发补偿
- 协调倾向
  - 乐观
    - 重试
    - 查询结果
  - 悲观
    - 补偿



# CC服务的运行时——一个小细节




1. 补偿请求中必须包含原请求的RequestId以及对应请求内容
2. 补偿请求处理过程中，如果发现原请求没有到达，需要先进行补记

# 4. 向未来看齐，机器学习离我们有多远？



- 服务运行期优化
- 从面向请求的服务，到面向事件的服务

# 机器学习在服务集成中的想象空间

普元 PRIMETON

数据库处理SQL

1. 语法解析
2. 优化
3. 生成执行计划
4. 执行

由合到分


运行期协调优化 → 合并部署 + 动态调度

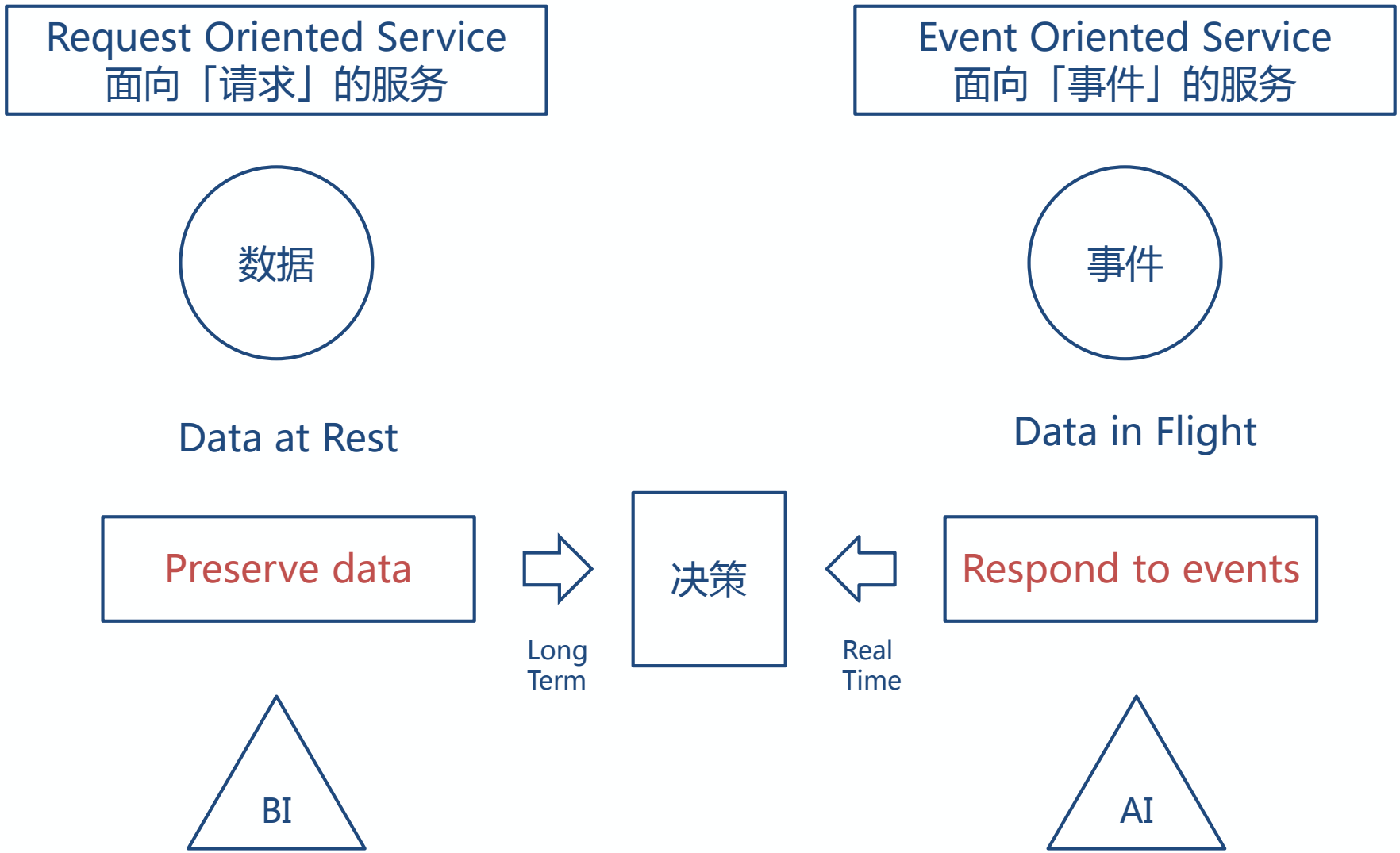
由分到合


关键点

1. 动态调度：执行计划的选择，是一个决策
2. 合并部署：考验持续集成的能力

# 服务驱动→事件驱动：从以数据为中心到以事件为中心





# 总结与建议



## 总结

1. 规范服务的三种基本模式CC / TCC / ES
2. 可以通过数据标准，由业务元数据驱动数据自动适配
3. 可以根据上下文依赖，实现运行期的并行计算，甚至智能调度

## 建议

1. REST之路，需要深入实践HTTP/1.1，持续关注HTTP/2
2. 小团队，不要轻易从微服务起步，但要做好准备。
3. 大团队，先制定好微服务的标准，提供相应的支撑平台，再迁移到微服务。

# THANKS

ArchSummit
全 球 架 构 师 峰 会 2016

PRIMETON · 普元

[ 北京站 ]

主办方 Geekbang. 极客邦科技 InfoQ