

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

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

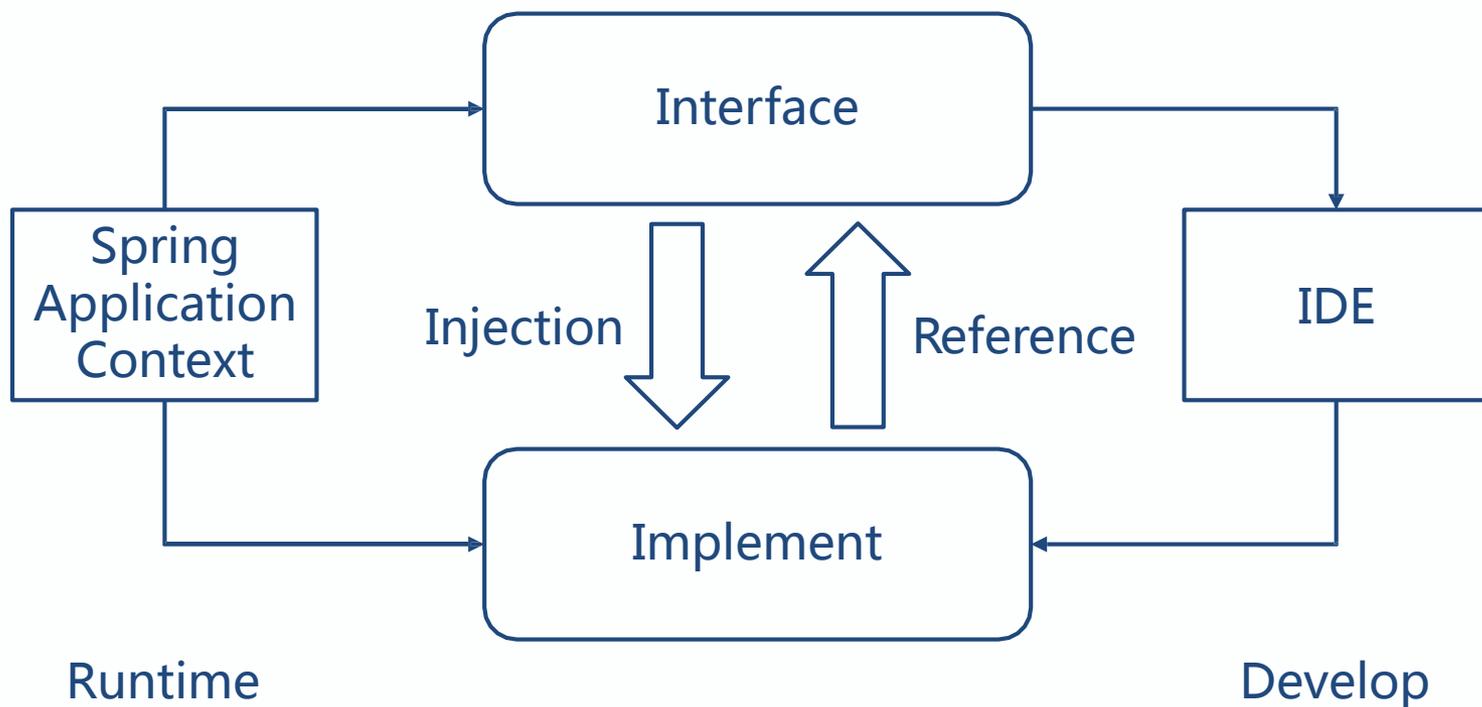
- 手工代码集成服务
- REST
- Docker

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

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

```
/**
 * Register user.
 * @param Request body schema:
 * {
 *   inviteCode: xxx,
 *   userName: xxx,
 *   email: xxx,
 *   password: xxx
 * }
 * @param userVO user info object
 * @param 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.INVITE_CODE_USER_NAME_VERIFY_FAILED, "userName length must more than or equal to 5, but it is " + userVO.getUserName() + " null ? " + userVO.getUserName().length());
    }
    if (!ValidatorUtil.validateUserName(userVO.getUserName())) {
        throw new PortalCapabilityException(PortalErrorCode.INVITE_CODE_USER_NAME_VERIFY_FAILED, "userName must be '_', digital or English, but it is " + userVO.getUserName());
    }
    if (StringUtils.isBlank(userVO.getPassword()) || userVO.getPassword().length() < 8) {
        throw new PortalCapabilityException(PortalErrorCode.INVITE_CODE_USER_PASSWORD_VERIFY_FAILED, "password length" + (userVO.getPassword() == null ? "0" : userVO.getPassword().length()));
    }
    if (StringUtils.isBlank(userVO.getUserNickname())) {
        String[] emails = userVO.getEmail().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.getUserEmail(), userVO.getUserName())) {
                throw new PortalCapabilityException(PortalErrorCode.INVITE_CODE_SER_MAIL_VERIFY_FAILED, "userEmail must be invited.");
            }
        }
        if (userSpi.exists(userVO.getUserName())) {
            throw new PortalCapabilityException(PortalErrorCode.USER_ALREADY_EXISTED, "user already existed, cannot create: " + userVO.getUserName());
        }
        //注册
        userVO = userSpi.register(userVO);
        log.info("step1: BR create user success");
        //创建VCS用户
        if (userVO != null) {
            vcsUserSpi.createUser(userVO);
            log.info("step2: VCS create user success");
        }
        if (StringUtils.isEmpty(userVO.getInviteCode())) { //创建用户
            //设置角色
            userSpi.assignRoles(userVO.getUserName(), new String[]{PlatformLevelV0.ORGANIZER_USER_ROLE_CODE});
            log.info("step3: assign user success");
            //创建租户
            tenantSpi.create(userVO);
            log.info("step4: BR create user success");
            //创建租户关系
            TenantUserRelationVO[] tenantUserRelationVOs = tenantUserRelationSpi.queryTenantUserRelationsByEmail(userVO.getUserName());
            for (TenantUserRelationVO tenantUserRelationVO : tenantUserRelationVOs) {
                tenantUserRelationVO.setUserStatus(TenantUserRelationVO.USER_STATUS_REGISTERED);
                tenantUserRelationSpi.saveTenantUserRelation(tenantUserRelationVO);
            }
            //授权权限
            tenantSpi.appendAuth(userVO.getUserName(), tenantUserRelationVO.getTenantCode() + "-*");
            log.info("step5: create relationship between user and tenant success");
            log.info("step6: BR authorization success");
        } else { //租户管理
            //设置角色
            userSpi.assignRoles(userVO.getUserName(), new String[]{PlatformLevelV0.TENANT_ADMIN_ROLE_CODE, PlatformLevelV0.PRODUCT_MANAGER_ROLE_CODE});
            log.info("step7: assign tenantManager success");
            //创建租户关系
            String tenantCode = inviteCodeVO.getTenantCode();
            TenantUserRelationVO tenantUserRelationVO = new TenantUserRelationVO();
            tenantUserRelationVO.setTenantCode(tenantCode);
            tenantUserRelationVO.setUserStatus(TenantUserRelationVO.USER_STATUS_REGISTERED);
            tenantUserRelationVO.setUserEmail(userVO.getUserName());
            tenantUserRelationVO.setUserNickname(userVO.getUserName());
            tenantUserRelationSpi.saveTenantUserRelation(tenantUserRelationVO);
            log.info("step8: create relationship between tenantManager and tenant success");
            //更新租户信息
            Tenant tenant = tenantSpi.getTenantByCode(tenantCode);
            tenant.setUserInfo(userVO.getUserInfo());
            tenant.setUserEmail(userVO.getUserName());
            tenant.setUserNickname(userVO.getUserName());
            tenantSpi.modifyTenantInfo(tenant);
            log.info("step9: update information of tenant success");
            //创建租户用户
            tenantSpi.createUser(); //创建租户用户
            tenantSpi.createAdminUser(tenantCode); //创建租户管理员
            tenantSpi.createProductManager(tenantCode); //创建租户产品经理
            tenantSpi.createProductManager(tenantCode); //创建租户产品经理
            tenantSpi.createProductManager(tenantCode); //创建租户产品经理
            tenantSpi.appendAuth(userVO.getUserName(), tenantCode + "-*");
            log.info("step10: BR create and authorization success");
        }
    }
}
return true;
}
```

• 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

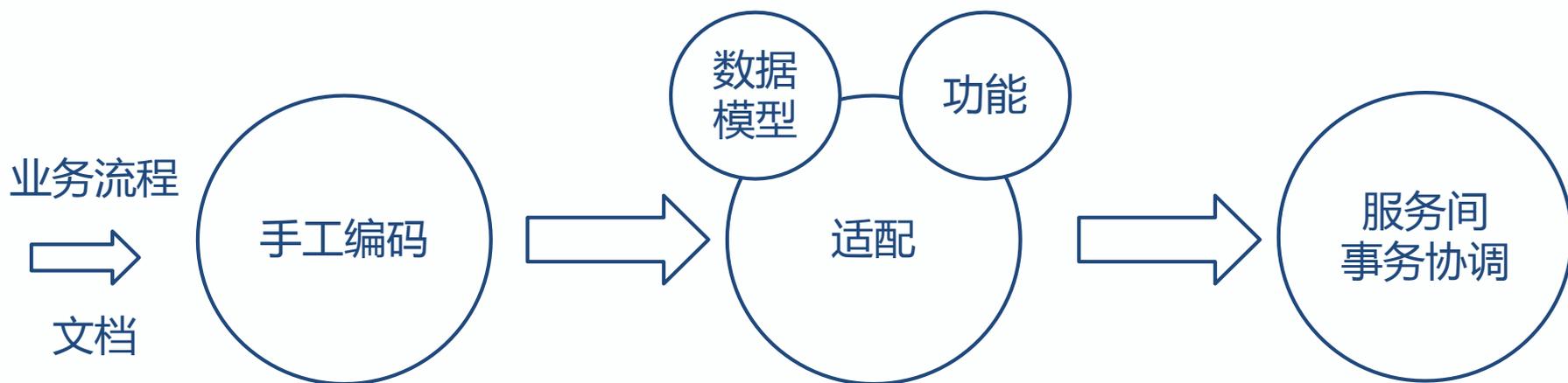
• ...

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

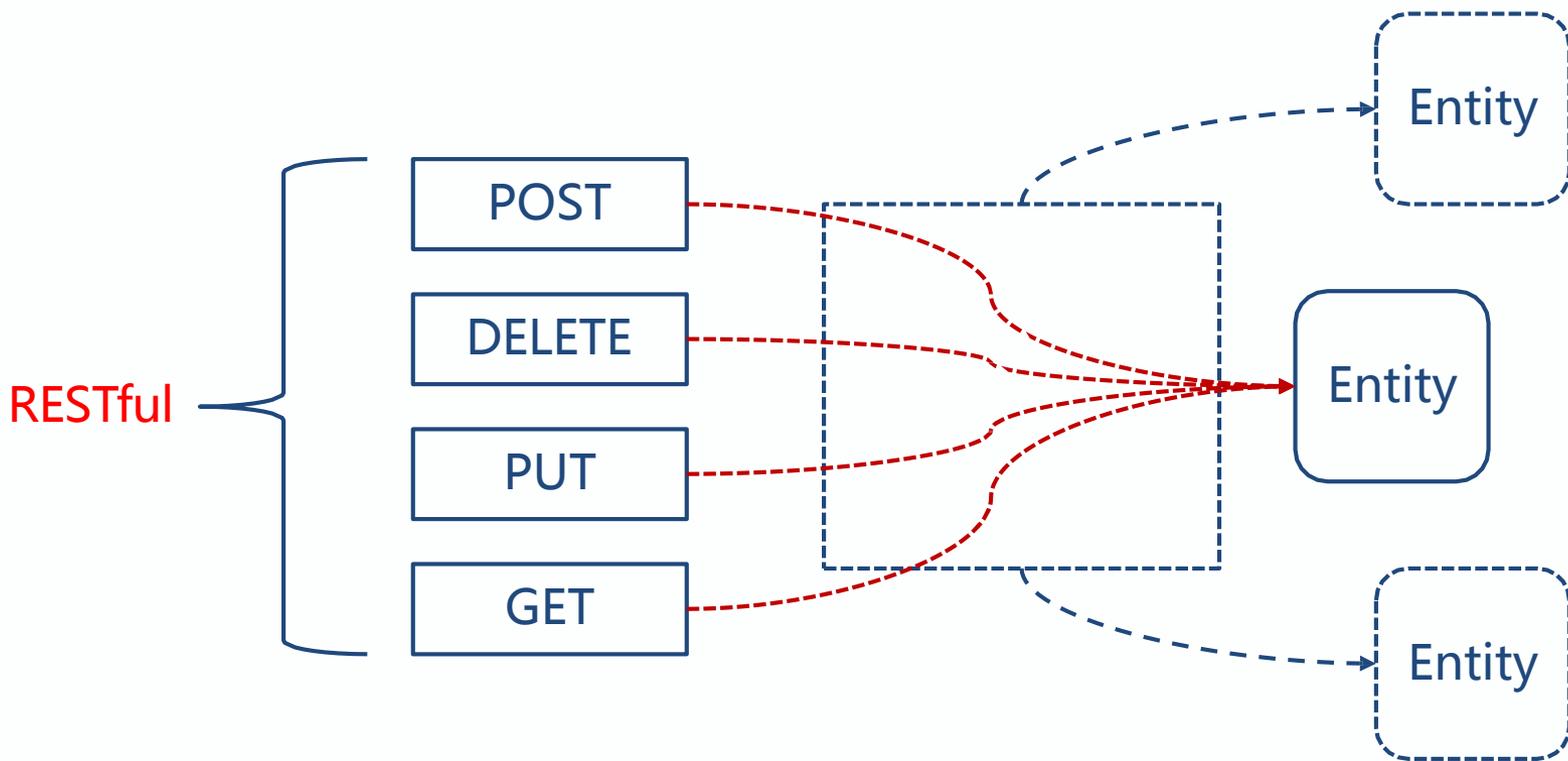
初见REST

...

初用REST

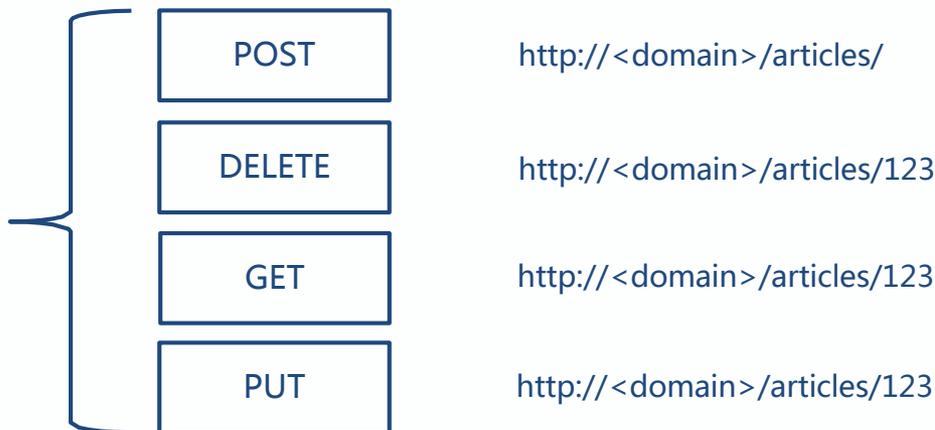
...

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

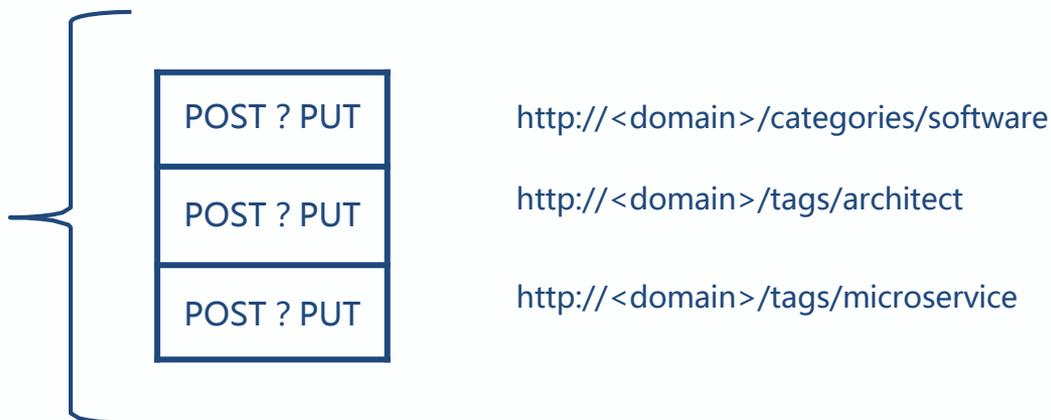
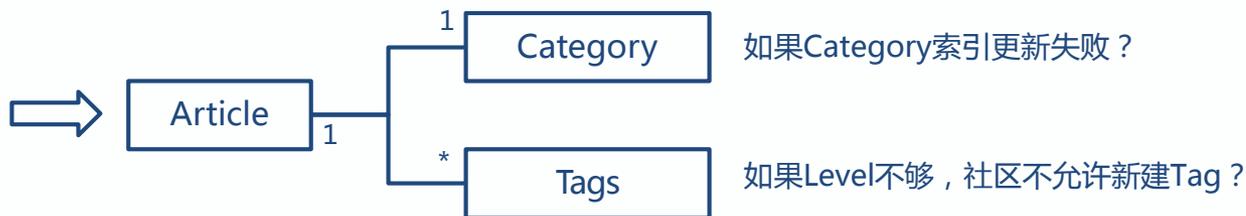


- 绝大多数的业务场景，被操作的都不止一个实体，且存在(分布式)事务
- REST的关注重点在单个实体

几乎所有的REST科普例子，都以单个实体操作为中心



问题是在POST文章时



几乎周围的REST实践例子，都以POST+GET为主

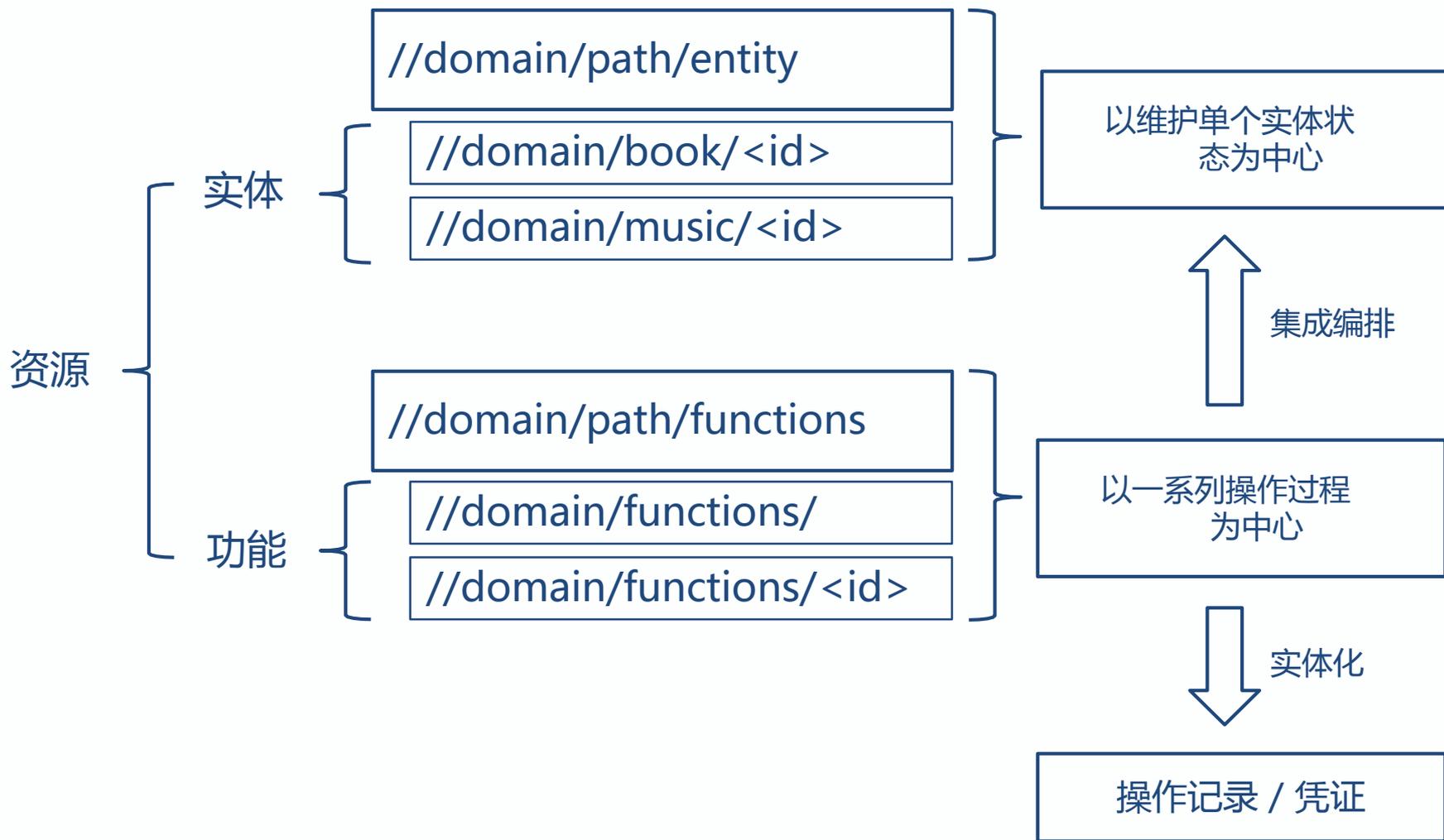


原因很多

- 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

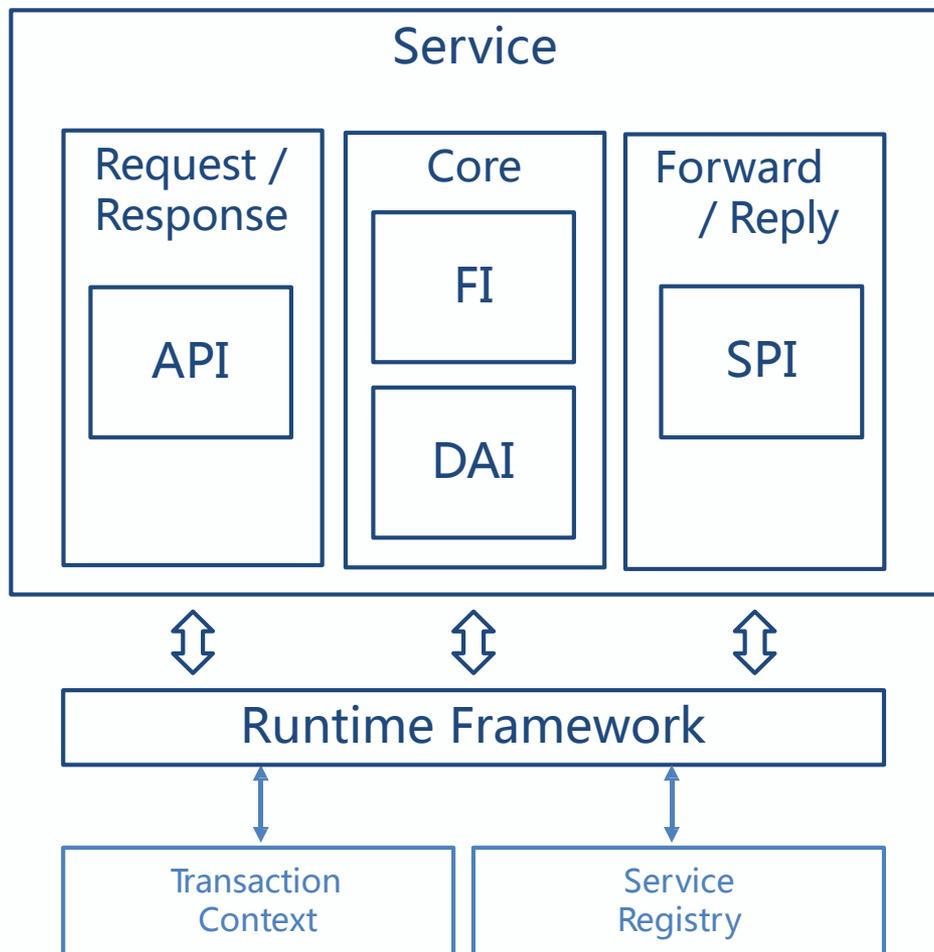
Docker Compose → 标准化应用/服务的部署集群

Docker Container → OS Level 虚拟化

2. 从本质出发，设计服务编排的落地路径

- 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

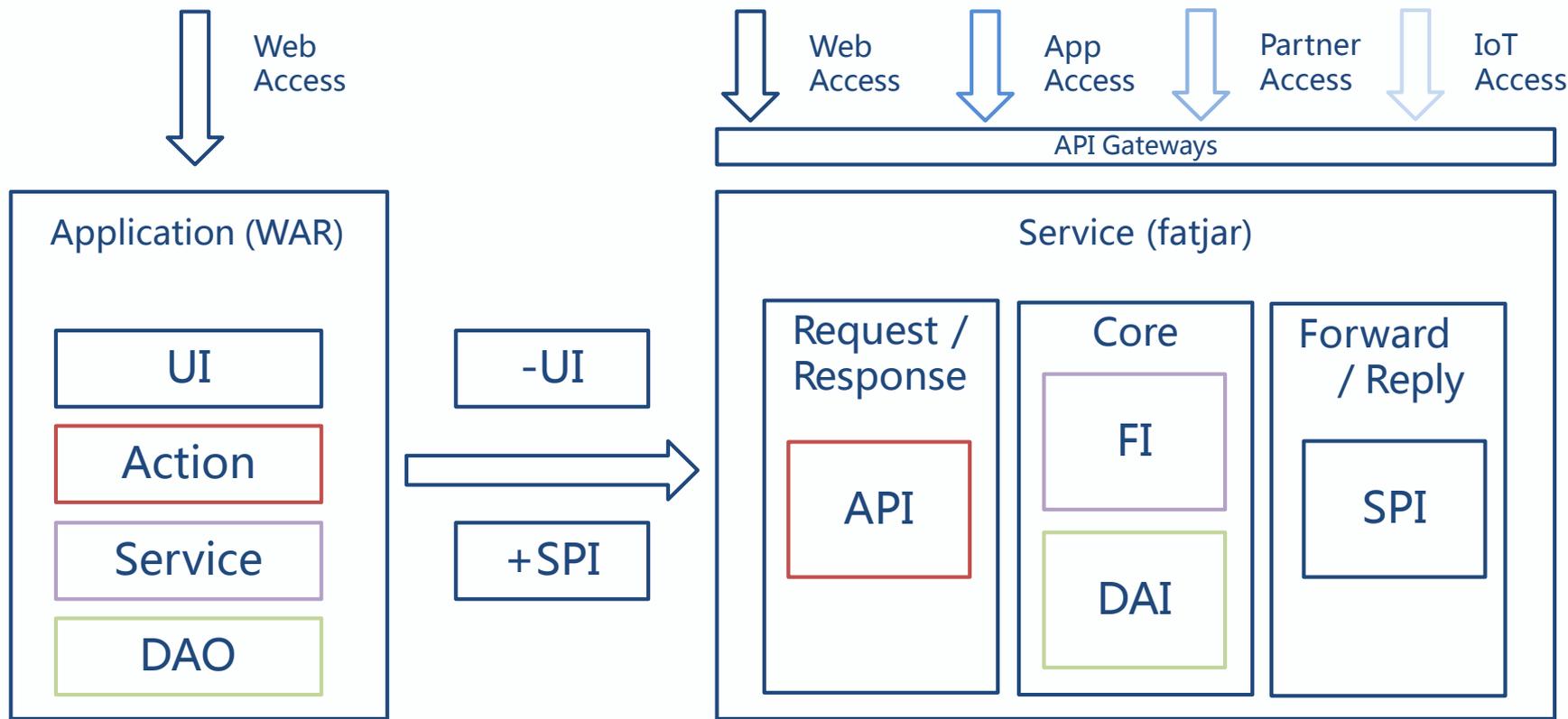


- API 请求 / 响应
- SPI 转发 / 应答
- DAI 数据访问
- FI 基础独立功能



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

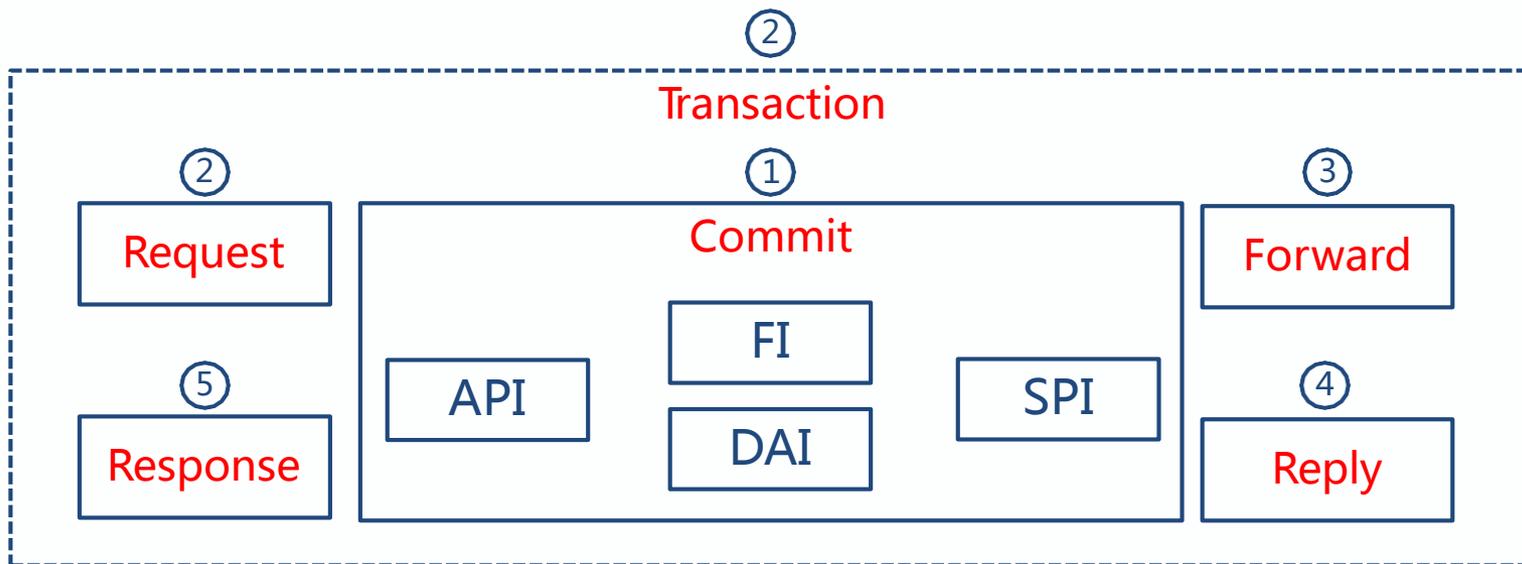


代价



- 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

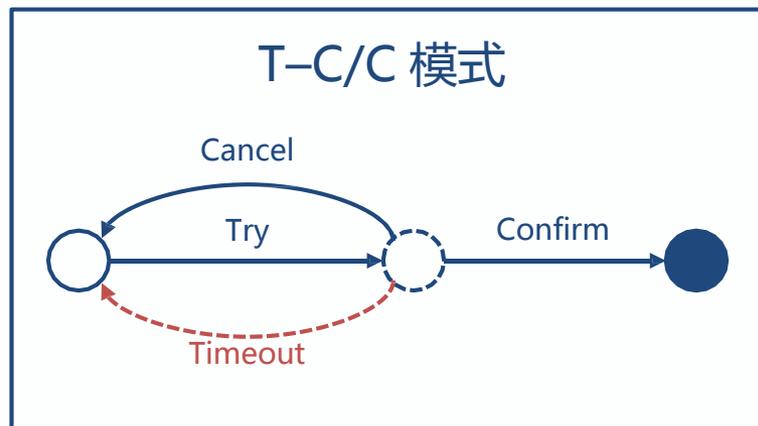
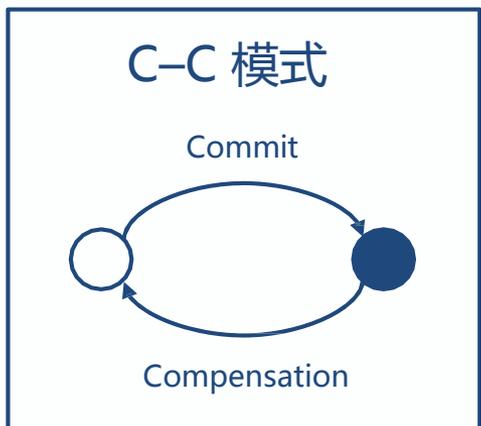


1. 生成commit
2. 解析请求报文
3. 转发请求（多个）
4. 解析转发结果（多个）
5. 响应请求

6组ID是关键

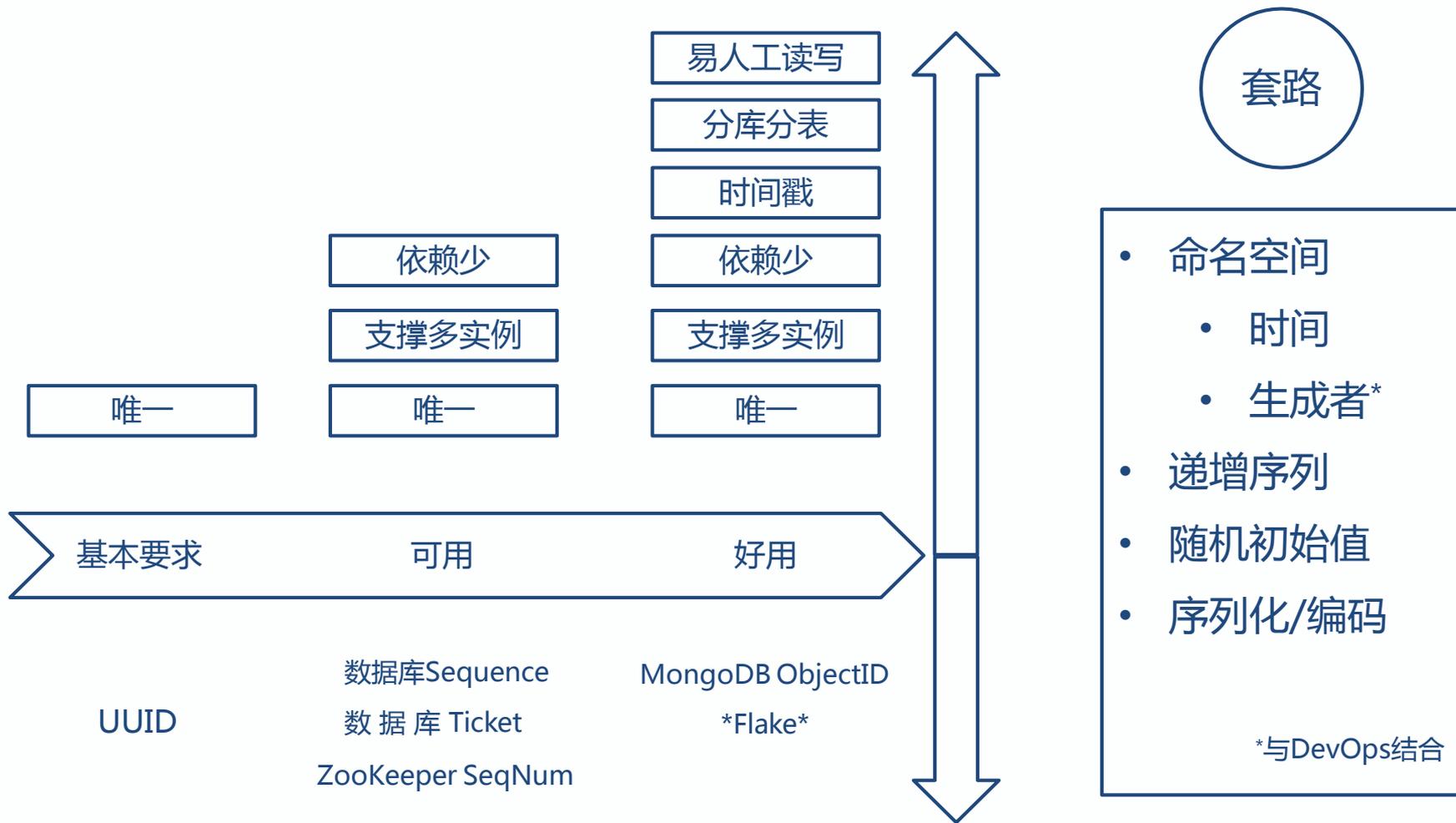
服务的三种基本工作模式

- 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



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

	Unsafe / Safe	Idempotent	Request Body	Response Body		
RFC 2616, 1999 RFC 7231, 2014	POST	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	→	Passive Commit
RFC 2616, 1999 RFC 7231, 2014	PUT	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	→	Active Commit
RFC 5789, 2010	PATCH	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	→	Compensation
RFC 2616, 1999 RFC 7231, 2014	DELETE	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>		
RFC 2616, 1999 RFC 7231, 2014	HEAD	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	→	Status
RFC 2616, 1999 RFC 7231, 2014	GET	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	→	Fetch
RFC 2616, 1999 RFC 7231, 2014	TRACE	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	→	Progress
RFC 2616, 1999 RFC 7231, 2014	OPTIONS	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>		
RFC 2616, 1999	CONNECT	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>		

①

②

③

④

⑤

- Active Commit 请求方生成资源 URL
- Passive Commit 响应方生成资源 URL

CC模式流水号在HTTP请求与响应中的位置(Before)

- 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

	Unsafe / Safe	Idempotent	Request Body	Response Body		
RFC 2616, 1999 RFC 7231, 2014	POST	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>		
RFC 2616, 1999 RFC 7231, 2014	PUT	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	⇒	Commit (1)
RFC 5789, 2010	PATCH	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	⇒	Compensation (2)
RFC 2616, 1999 RFC 7231, 2014	DELETE	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>		
RFC 2616, 1999 RFC 7231, 2014	HEAD	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	⇒	Status (3)
RFC 2616, 1999 RFC 7231, 2014	GET	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	⇒	Fetch (4)
RFC 2616, 1999 RFC 7231, 2014	TRACE	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	⇒	Progress (5)
RFC 2616, 1999 RFC 7231, 2014	OPTIONS	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>		
RFC 2616, 1999	CONNECT	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>		

• 请求方生成URL, RequestId在Path中



CC模式流水号在HTTP请求与响应中的位置(After)

- 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

	Unsafe / Safe	Idempotent	Request Body	Response Body			
RFC 2616, 1999 RFC 7231, 2014	POST	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	⇒	Try	①
RFC 2616, 1999 RFC 7231, 2014	PUT	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	⇒	Confirm	②
RFC 5789, 2010	PATCH	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	⇒	Cancel	③
RFC 2616, 1999 RFC 7231, 2014	DELETE	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>			
RFC 2616, 1999 RFC 7231, 2014	HEAD	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	⇒	Status	④
RFC 2616, 1999 RFC 7231, 2014	GET	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	⇒	Fetch	⑤
RFC 2616, 1999 RFC 7231, 2014	TRACE	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	⇒	Progress	⑥
RFC 2616, 1999 RFC 7231, 2014	OPTIONS	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>			
RFC 2616, 1999	CONNECT	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>			

- DELETE Request Body 在规范上被忽略
- 不适合服务定义，但适合于应用UI层

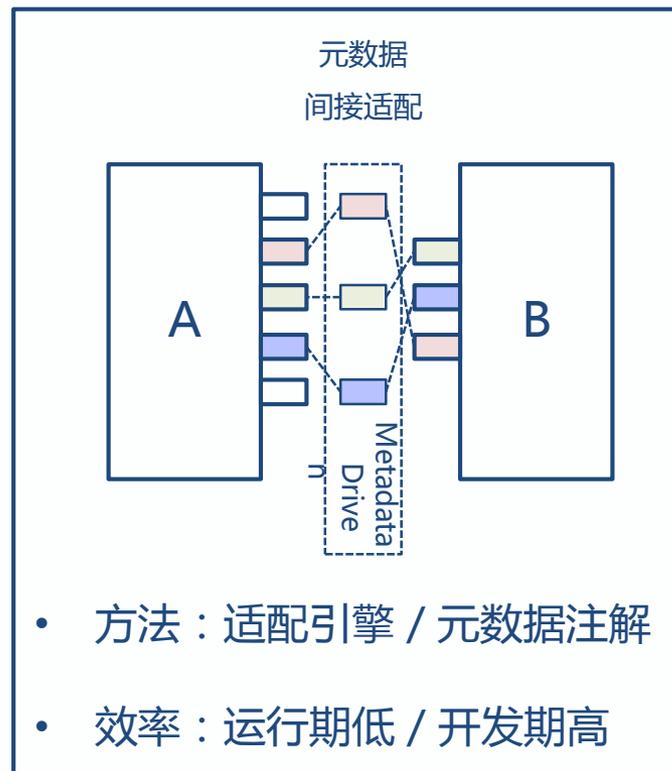
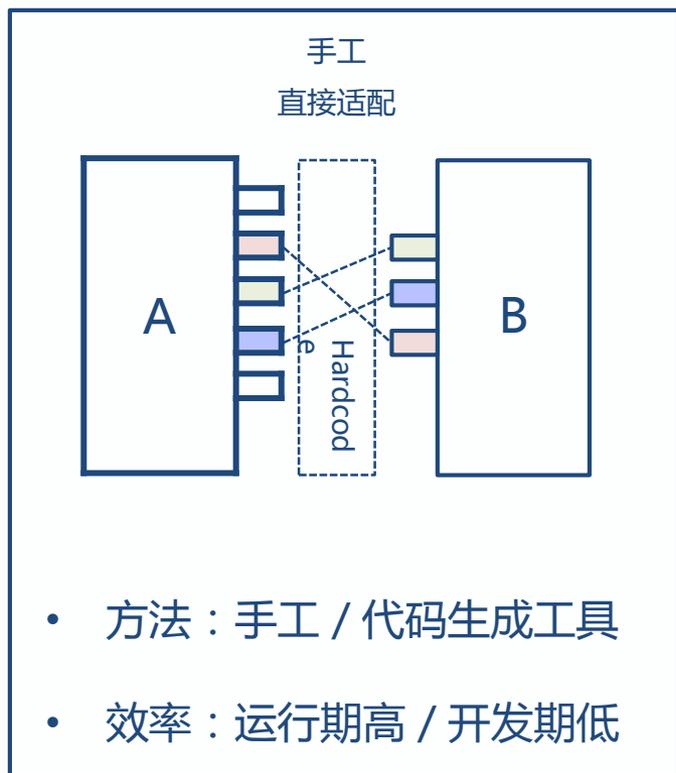
TCC模式流水号在HTTP请求与响应中的位置

- 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

A / B是两个内部服务，对于同一个客体，有不同视角



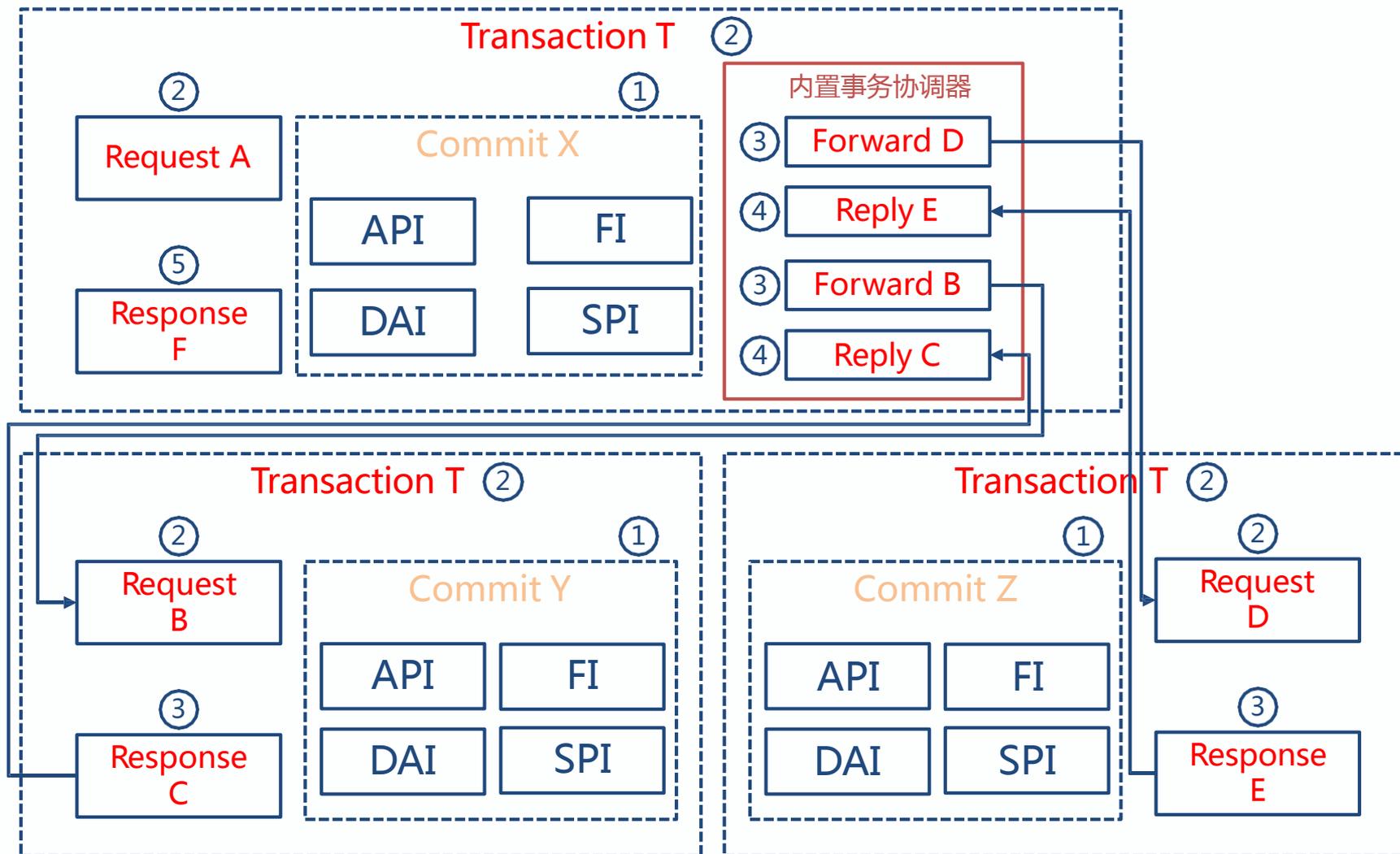
DevOps

数据标准 \rightleftarrows 适配三方库

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。



研究生导师 签字	研究生院 (教1-430) 盖章	财务处 (后勤楼1层) 盖章
学院党委 (办理党组织关系)	保卫处 (转户口关系) (体育馆西厅北侧) 盖章	图书馆 (凭论文呈缴证明 借阅证和一卡通办理) 盖章
校团委 (办理团组织关系)		
档案馆 (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 (本楼交钥匙,领退房清 单) (学八楼一层大厅) 盖章	学生事务管理处 (注销学生证) (行政办公楼西侧院 内) 盖章
一卡通 (办理注销) (信息网络中心 109)	各学院 收通知单 发派遣证 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：

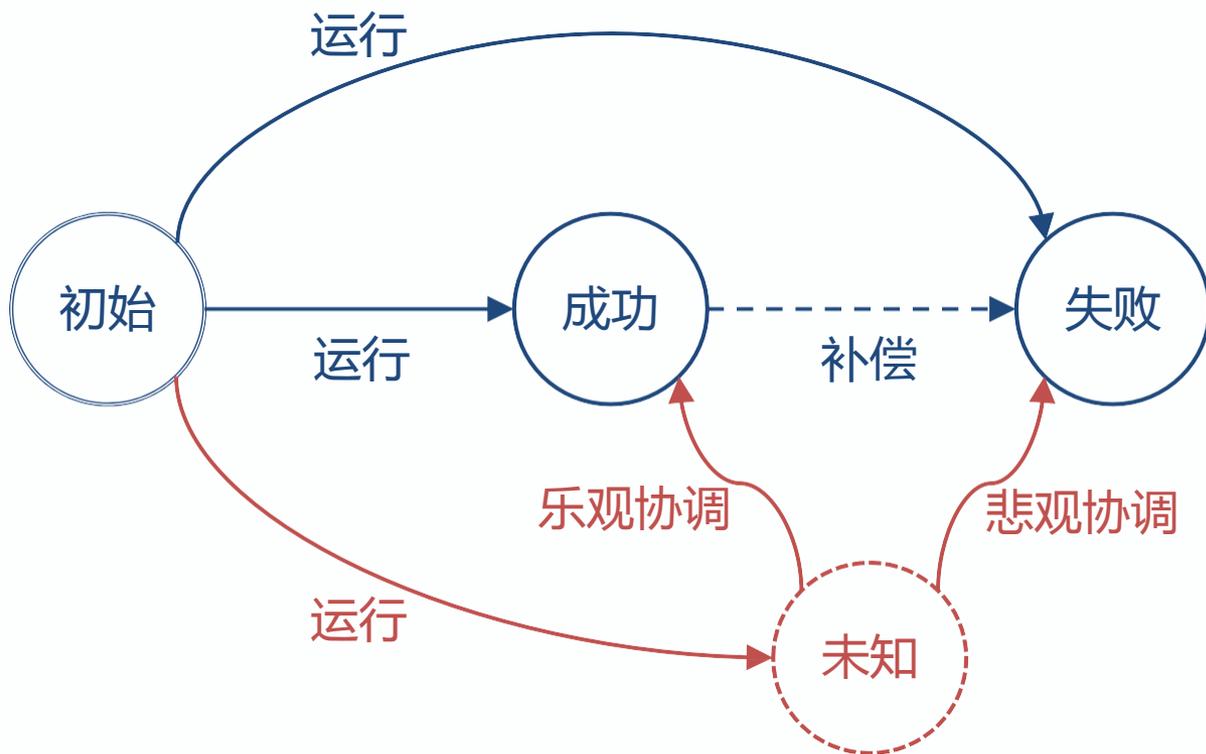
- 导师签字 → 研究生院盖章 → 财务处盖章
- 图书馆应在一卡通之前办理
- 领取毕业证为最后一项

- 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

服务状态的两个视角

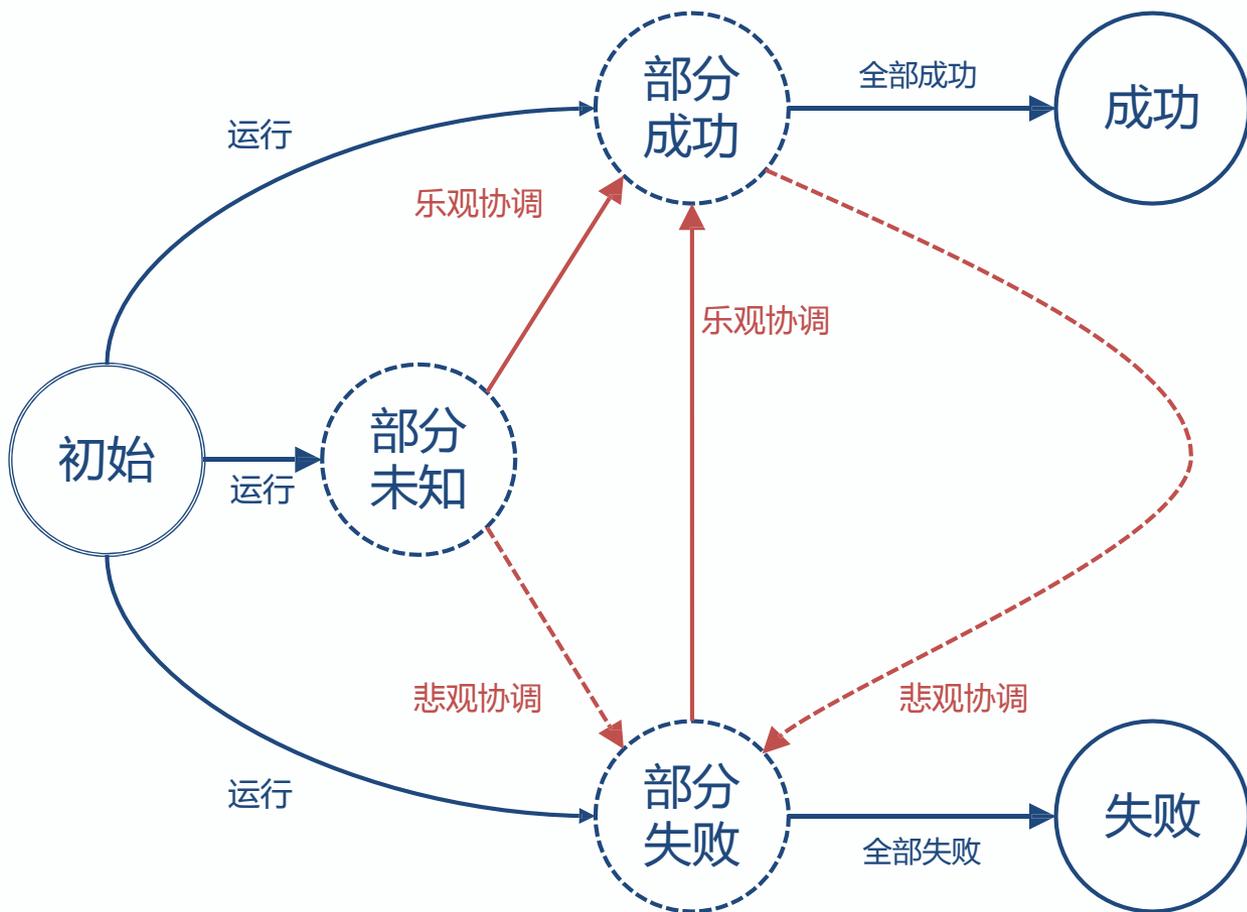
- 服务自身稳态
 - 初始
 - 成功
 - 失败
 - 表示失败的响应码
 - 可识别的异常
 - Connect Timeout
- 被协调器感知的状态
 - 未知
 - 不可识别的响应码
 - 不可识别的异常
 - Read Timeout



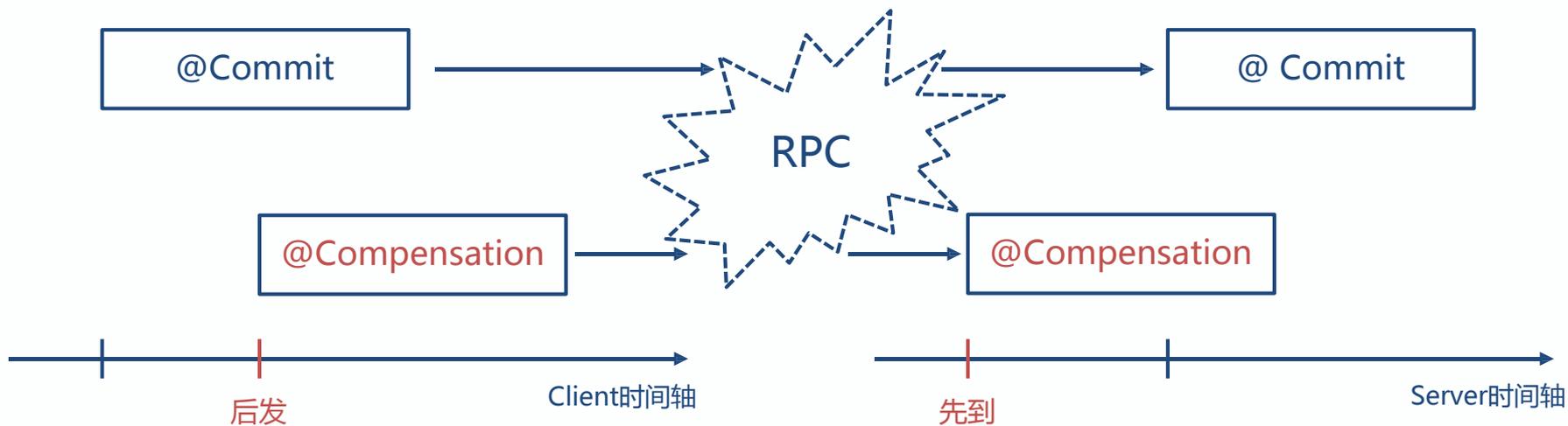
CC服务协调器的运行期状态迁移

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

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

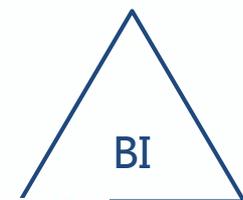
服务驱动→事件驱动：从以数据为中心到以事件为中心

Request Oriented Service
面向「请求」的服务



Data at Rest

Preserve data

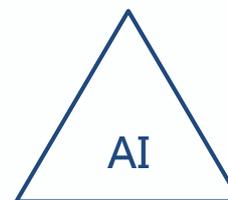


Event Oriented Service
面向「事件」的服务



Data in Flight

Respond to events



Long
Term

决策



Real
Time

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