看完这篇，你的API服务设计能力将再次进化!

京东零售技术 2024-07-11 247 阅读19分钟 专栏：京东零售技术

智能总结 复制 重新生成

这篇文章主要探讨了 API 服务设计的关键因素，包括服务路径和模块、请求方式、出入参、业务处理建议、异常处理、强弱依赖、监控日志、降级点、过时服务处理、保障安全措施等，还介绍了敏感字段加密、系统准入、接口防篡改、出入参加解密等安全保障措施，强调服务设计要全面考虑以应对各种需求和挑战。

关联问题: 如何设计服务路径? 服务异常如何兜底? 怎样处理敏感字段?

基于该文章内容继续向AI提问

引言

在当今快速演变的技术场景中，服务设计不仅仅是遵循通用的设计规范和最佳实践的问题，它更深层次地触及到如何在满足这些标准的同时，确保服务能够灵活适应未来的变化、满足用户的期望。本篇文章旨在探讨在遵循通用设计规范之外，服务设计过程中需要考虑的关键因素。希望经过我们一系列的分享，能和大家一起讨论如何将API设计的易对接，易理解和易扩展。

系统介绍

> 京东企业业务VOP(智采)：以API形式为客户提供京东供应链能力；VOP提供上百个标准API服务，为上千家客户搭建自采商城提供底层能力。

服务设计tips

1、服务路径和模块

• 服务路径从域名后开始，应有统一的一级路径，如domain/api/，这样做的好处是:将所有API请求集中到一个明确的路径下可以简化安全控制的实施。比如可以在NP或API网关层面上，针对特定路径应用特定的安全策略，如**认证、授权、日志、限流和监控**等，从而增强API的安全性;

•再向后则是**业务分组**，比如订单/api/order，商品/api/product等，可以让调用方清晰明了此接口是属于哪个业务模块。

•还要考虑各服务的**粒度**，比如:商品价格是做成单独的服务，还是耦合在商品详情接口直接吐出呢?订单物流信息应该包含在订单详情服务里吗等等。考虑这部分，需要以当前业务场景作为基础，拿VOP来说，因为有客户级别的商品变价消息存在，所以调用方在拉取到商品变价消息之后，可以单独查询价格服务更新自己本地存储的价格，而不用全量查询商品信息，所以解耦更优。在保证**业务场景完整度**的情况下，尽可能的为调用方提供**可灵活组装**的服务。

2、服务请求方式

对于大部分服务来说，应支持ajax请求，并且指定固定的数据格式(JSON/XML等)，既方便后续扩展，又能支持业内绝大部分的客户业务处理。但对于部分特殊场景的服务，如收银台，登录/登出等接口，则需要非ajax请求，由服务端进行后续的转发跳转处理。

3、服务出入参

•API服务出入参的定义，尤其是对外开放服务的出入参，需要先考虑**扩展性**，防止后续扩展受限以至于不得已增加新的接口导致重复建设、接口混乱不清、同一个接口有好些大同小异的版本等问题;其次就是对于各字段的定义和说明，通用的规范不用赘述，想要强调的是:

•为了后续的扩展性以及对调用方屏蔽服务内部变化，某些数字字段要**优先设计成Long甚至于直接定义成字符串**，防止服务内部变化(比如某些字段Integer不够存储需要转换成Long时)，给调用方增加额外的改造工作量。

•在某些情况下，字段可能需要区分“未设置”和“零值”这两种情况。此时，需要考虑将字段设计为包装类型，这样，null可以表示“未设置”，而具体的零值（如0、0.0等）则有其特定含义。

•字段取舍:出参字段**非必要不暴露**，在满足接口功能要求的前提下，尽可能的减少字段暴露;对于一些敏感字段，比如客户地址，手机号等信息，更要仔细斟酌是否需要吐出该字段，如必需，则还需要考虑**字段脱敏和加解密**。

•字段定义及说明:详细且合规的字段定义和说明可以降低调用方对于服务字段的理解成本，确保数据的一致性和准确性，常见的说明有以下几种:

•字段长度的限制

•数字类型字段的范围

•各业务域相同含义的字段名保持一致

•字段格式说明：对于日期、时间等特殊格式的字段，明确其格式

•枚举类型的说明:对于枚举类型的字段，所有可能的枚举值，并对每个值的含义进行说明等等

•已发布服务需要新增出/入参:已经发布出去的服务，如果要新增出入参，需要考虑对调用方的影响，如序列化/反序列化，字段处理等等;若调用方在进行接口出参反序列化时使用的是**严格模式**，那服务端随意的新增返回字段将会使调用方在服务出参数据**反序列化时失败**，进而影响到业务进行。我们可以在**服务入参增加扩展字段，当调用方传入指定数据时，服务再在返回体中增加新的出参**。

京东零售技术 LV.5
技术运营 @京东
作者榜No.17 优秀作者
147 文章 108k 阅读 1.1k 粉丝
关注 私信

·出参格式统一:对外服务要求有固定的格式，清晰的标明处理结果，状态码以及业务数据等字段，能让调用方清晰的进行业务处理已经异常处理，如:

```json
{
    "success": true,
    "resultMessage": "success",
    "resultCode": "0000",
    "result": {}
}
```

4、服务内业务处理的一些建议

·对于服务逻辑处理来说，尽量支持批量数据的处理。对于调用方来说，可以将自身重心放在数据处理逻辑上，而不是如何高并发的调用服务来处理这部分数据，可以有效减轻服务方压力。如订单/商品查询，发票开具等服务

·做水平越权处理:防止调用方查询到不属于自身的数据，具体校验严格程度视业务而定。比如VOP对于大部分的查询服务，都允许属于同一合同下的不同pin互相查询数据(订单，发票等)

·对于重点业务逻辑需要多线程处理时，尽量使用单独的线程池，防止线程池内线程被其他业务抢占影响重要业务

5、异常处理

·对于开放服务，尤其是写操作的服务，定义明确的异常码是尤为重要的，调用方系统需要准确的错误码和错误信息来自动化地处理接口调用的结果:**重试、告警或直接忽略、业务中断或继续进行**。 相信我们都遇到过调用一些没有任务错误码文档的接口，调用方完全不知道这个接口会抛出或者返回一些什么样的错误码以及错误描述，没办法决策这些异常是否可以展示给客户!(对于一些技术化的错误提示:比如'调用XX接口异常'，'XX数据为空'等等)

·第3点时我们提到了服务的出参格式需要统一，所以在出现业务异常时，我们要包装合适的异常码以及异常信息返给调用方，另外还需要注意出现未知异常(空指针，调用依赖接口超时等)时，对出参的包装，防止出现直接抛异常给调用方的情况，可以在最外层使用拦截器做未知异常的兜底出参处理。

6、服务的强弱依赖

在构建一个复杂的对外服务时通常会依赖多个外部接口。对于这些依赖，我们可以将它们分为两类：强依赖和弱依赖。强依赖的接口是服务运行不可或缺的部分，如果这类接口出现异常，我们通常采用的策略是短路处理，即立即中断当前操作，以避免进一步的错误传播或数据不一致。

另一方面，对于弱依赖的接口，它们对主流程的影响相对较小，可以容忍一定程度的异常或失败。在这种情况下，如果弱依赖接口发生异常，我们可以选择忽略这些异常，并继续执行服务的核心流程。同时，为了确保服务的稳定性和问题的可追踪性，我们会实施业务告警机制，以便及时发现并处理这些异常情况。这种做法旨在保证服务的整体可用性，同时确保问题不会在无人知晓的情况下被忽略。

7、服务的监控和日志记录

·**日志记录**:对于一个对外服务(尤其是直接面向外部客户的服务)来说，除了系统中常用的技术监控以外，服务出入参日志的记录是至关重要的一环，尤其是涉及**资金、订单、支付**等各种类型的写操作服务，一方面我们可以监控重要服务中，各调用方都使用了哪些字段，从而为在后续**新增、下线或修改某些字段时提供风险评估数据**；另一方面还可以从日志记录出入参中**分析业务数据，为后续业务决策、业务告警等提供数据支撑**。另外，当出现调用方和服务方数据不一致而产生严重后果的情况时，这些日志的记录也是判断问题点的重要信息。

·**调用数据收集**:对于开放平台的流量细节做统计和分析，是做流量治理、保障服务安全的重要前提，通过对各调用方的调用分析可以得知调用方的调用频率、调用规律、请求是否合理等信息，对于一些非法调用(刷单、爬虫等)进行有效识别，可以统计出调用方的平均调用量进而为服务限流、安全防护等提供数据支持。

8、合理设置降级点

在构建复杂的服务时，特别是在分布式系统和微服务架构的背景下，我们一般需要依赖很多第三方服务或数据。在这种情况下，服务降级策略就成为确保整体系统稳定性的关键。**服务降级**主要是指在某些服务模块遇到性能瓶颈或者故障时，主动地减少或关闭这些组件的某些功能，以此来保障整个系统的核心运作。

为了有效实施服务降级，需要在开发和设计过程中识别到潜在的风险点，并据此划分出服务中的**关键与非关键模块**。在非关键模块的代码层面嵌入降级逻辑，让我们可以在出现问题时，手动或自动执行降级操作，从而确保关键服务的持续可用性。这种设计不仅有助于提升系统的鲁棒性，也可以有效地减少系统故障时对用户的影响。

9、处理过时服务

每一个系统都会出现很多初期简单设计、只满足简单业务场景的服务，随着业务的发展，初期设计的很多服务将不再能满足新的业务场景，在原服务无法扩展的情况下，我们一般会增加新的服务(需要能覆盖旧服务的能力)，久而久之，就会产生很多几乎无客户使用的过时服务，有可能会产生影响代码问题排查、触发业务场景漏洞等问题，对于这样的服务，我们需要有两种处理办法:

·若逻辑兼容，可以将旧服务路由到新服务

·下线处理，但不建议直接删除代码，可以统一拦截需要下线的路径，给调用方返回统一的错误码和提示信息，一旦监控到拦截告警或者调用方反馈，可以先停止拦截并和调用方沟通后续处理方案。

10、不要相信任何调用方和依赖的第三方

在服务设计和编程过程中，对于所有依赖的第三方的服务都要保持不信任的前提，对其返回的数据、服务的超时时间、异常的返回等均需要细致处理，要考虑每一个可能出现的问题点，防止某一个依赖的服务出现问题进而影响全局。

同样的，也不能信任调用方。无论事先如何保证或者规约，都无法保证调用方的调用行为符合预期，所以我们在服务上线前就需要考虑和构建多种防御机制。这包括但不限于：实施有效的限流措施、进行严格的输入验证、以及设置精细的权限控制等。通过这样的设计，我们不仅能够提升服务的稳定性和安全性，还能确保在面对不可预测的外部因素时，我们的系统能够保持弹性和韧性。

保障服务和信息安全的措施

(1)敏感字段

在B2B场景中，企业对于用户敏感数据的要求较高，比如手机号，用户名，住址等信息，这就要求服务提供方在接口出参时对于这些数据进行加密处理，加密这些信息可以防止在数据传输过程中被未经授权的第三方截取和读取，从而保护客户和企业的隐私和利益。

加密可以在不同的层面上实施：

·传输层加密：使用SSL/TLS等协议对客户端和服务器之间的整个通信进行加密。这意味着所有传输的数据，包括敏感字段，都会被加密，从而保障数据在传输过程中的安全。

·应用层加密：在数据发送之前，直接在应用层对特定的敏感字段进行加密。这通常涉及到使用加密算法（如AES、RSA等）对敏感数据进行编码。即便在传输层已经有加密，应用层加密也提供了另一层安全保障。

·数据库层加密：对存储在数据库中的敏感字段进行加密，这种方式使用不多，一般用来保存密码等信息。

在使用敏感字段加密时，需要注意以下几点：

·选择合适的加密标准和算法，确保是业内常用的、经过验证的。

·管理好加密密钥，密钥的生成、存储、交换和销毁都应当安全可靠。

·考虑性能影响，加密和解密操作会增加计算复杂度，需要确保服务性能不会因此受到明显影响。

(2)系统准入

部分对系统安全要求较高的客户，会要求服务方对非法的调用方进行拦截，防止非法第三方在获取到身份信息之后使用客户身份进行调用进而产生资损或更多的数据泄露。面对这种情况，我们可以使用设置IP黑白名单的方式来进行拦截，限制某个身份标识只能是指定的ip来访问或指定的ip不能访问。

> IP白名单:只有在白名单中的IP地址被允许访问系统或接口。不在列表中的所有IP地址都会被拒绝。 优点：提供了高级别的安全性，在客户维度下，只有预先批准的IP地址可以访问，其余IP均不能以该客户身份进行访问。可以有效防止未授权的访问尝试，在某些特定场景下，甚至可以取消授权校验，IP即代表客户身份。 缺点：管理起来可能比较繁琐，尤其是当合法用户的IP地址经常变化时。对于使用动态IP地址的用户不太友好。 适用场景：适用于访问者数量有限且IP地址固定或变化不大的环境。

> IP黑名单:在黑名单中的IP地址不被允许访问系统或接口。不在列表中的IP地址则可以访问。 优点：简单易于管理，只需要列出已知的恶意IP地址。对于大多数合法用户来说，不会影响他们的访问。 缺点：安全性较低，因为新的未知攻击者仍然可以访问系统。恶意用户可以通过更换IP地址来绕过黑名单控制。适用场景较少，目前VOP平台尚未有客户选择此种IP校验类型。 适用场景：适用于开放性较高的环境，或者作为其他安全措施的补充。

(3)接口出入参防篡改

确保通过接口发送的数据在传输过程中未被修改。这对于保障调用双方数据的完整性和安全性至关重要，常用的防篡改方法有以下几种:

a. 使用HTTPS

确保所有数据传输通过HTTPS进行，这样数据在传输过程中会被加密，从而降低被截取和篡改的风险。

b. 数字签名

在发送数据前，调用方使用数字签名对数据进行签名，接收方可以使用相应的公钥验证签名，以确保数据自签名以来未被修改。

数字签名的步骤如下：

发送方使用私钥对数据（可以是数据的哈希值或直接将参数按规则拼接成字符串）进行签名。

发送方将数据和签名一起发送给接收方。

接收方收到数据后，使用发送方的公钥验证签名。

如果签名验证成功，说明数据未被篡改；如果失败，说明数据在传输过程中可能遭到篡改。

c. 消息认证码（MAC）

与数字签名类似，消息认证码（MAC）是一种确保消息完整性的技术，它使用一个密钥和消息内容生成一个MAC值。接收方使用**相同的密钥**生成MAC值，并与发送方提供的MAC值进行比较。

d.使用API密钥和时间戳

结合API密钥和时间戳可以提供另一层安全性。时间戳可以防止**重放攻击**，API密钥则确保只有授权的用户可以发送请求。

(4)出入参加解密

出入参加解密是指在客户端发送请求到服务器（入参解密）和服务器返回响应到客户端（出参加密）的过程中，对数据进行加密和解密的操作。这样做可以保护数据在传输过程中的安全性，防止敏感信息被窃取或篡改。

出入参要求全是加密后的数据，这种需求一般来自银行或者国央企等十分看重数据安全的客户。作为接口数据安全的可选项，并不适用于全部客户，而且要求数据密文传输的客户，一般都会指定数据的加密方式，有可能是业内通用的加密方式，也有可能是客户内部自定义的一个jar包。

关于后者，我们搭建了一个可以加载不同客户的加密SDK的ECI平台，通过这个平台，将每个客户提供的加密SDK隔离加载，确保不会因为某个客户SDK出现问题而影响全部客户;在平台上传客户的SDK并加载完成之后，会对外提供一个服务接口，该接口支持客户维度的加密和解密两种操作。我们只需要按客户维度配置该客户对应的加解密方法，在拦截器层面进行统一处理加解密操作即可，对实际业务代码完全无侵入。

伪代码示例:

scala 代码解读 复制代码

```scala
/**
 * @description 指定客户接口出入参加解密拦截器
 */
@Service
public class EncryptInterceptor extends HandlerInterceptorAdapter {
        //请求进入，业务处理前解密
    @Override
    public boolean preHandle(HttpServletRequest request， HttpServletResponse response， Object handl
            if(!(request instanceof  DecryptionRequestWrapper)){
                    return true;
            }
            //获取客户授权信息，获取到当前客户身份
            try {
                //判断该客户是否开启接口加解密
                     //组织参数，获取原始入参，进行解密操作
                     //解密完成后，将解密得到的参数全部加入到request的参数内
                     //继续向下进行
                    return true;
            } catch (Throwable throwable){
                //异常处理
            }
    }
        //返回前加密
    @Override
    public void postHandle(HttpServletRequest request，HttpServletResponse response， Object handler，
                    ModelAndView modelAndView) throws Exception {

            //获取客户身份标识
            try{
                //判断该客户是否开启接口加解密
                //调用原始的 postHandle 方法，让响应数据被写入包装类的输出流
                //调用加密方法，获取加密后数据
                //将加密后的数据写回响应
            }catch (Throwable throwable){
                //异常处理
            }
    }
}
```

结语

构建一个健壮的服务和系统不仅是技术领域专业人士的终极追求，也是确保我们能够在面对未来不断演变的业务需求和技术挑战时保持竞争力的关键。在遵守广泛认可的设计规范之外，我们必须进一步探索和优化，确保我们的服务不仅可以全面覆盖各种业务场景，还要在安全性、可扩展性以及可伸缩性和降级能力方面表现出色。

这种追求不仅要求我们在设计之初就深入考虑潜在的风险和挑战，还要求我们在整个开发过程中对服务进行持续评估和优化。这样的系统将成为支持企业业务持续增长和创新的坚实基础，帮助我们在不断变化的技术和业务环境中保持领先!

读完此篇，加入光荣的进化吧!

作者：企业业务 张博文

来源：京东零售技术 转载请注明来源

标签： 后端

评论 1

登录 / 注册 即可发布评论!

最热 | 最新

Coding小菜鸡 Software Developer

666

7月前 点赞 评论 ···

为你推荐

看完这篇,你的服务设计能力将再次进化!

京东云开发者 | 7月前 | 270 | 8 | 评论 设计模式

学习系统设计：了解API 设计

一只拉古 | 1月前 | 81 | 点赞 | 评论 后端 面试 架构

API 体系构建

飞书技术 | 2年前 | 1.5k | 4 | 1 API

API 体系构建

字节跳动技术团队 | 2年前 | 5.5k | 32 | 3 API

浅谈6种流行的API架构风格

追逐时光者 | 1月前 | 527 | 6 | 1 后端 架构

常见API架构介绍

yangnk | 1年前 | 474 | 4 | 评论 Java

API 接口设计的未来趋势：探索技术方向与实践方案

Swift社区 | 2月前 | 93 | 2 | 评论 Harmo... ArkUI ArkTS

API 设计：从基础到最佳实践

Apifox | 8月前 | 1.7k | 4 | 2 API 程序员 强化学习

微服务架构的外部 API 集成模式

MobotStone | 2年前 | 525 | 8 | 评论 微服务 API

深入理解 JSON 和 Form-data

Hong1 | 11月前 | 579 | 点赞 | 评论 JSON 前端 数据结构

为什么需要监控API

eoLinker | 3年前 | 622 | 点赞 | 评论

关于API：好的设计和坏的设计【eolink翻译】

eoLinker | 2年前 | 921 | 1 | 评论 大数据

构建开放式服务：API接口与用户认证实践指南 | 青训营

纯虚函数 | 1年前 | 482 | 点赞 | 评论 青训营笔记

REST API设计原则：构建可扩展、易维护的 API

MobotStone | 1年前 | 947 | 2 | 评论 后端 架构

构建 API 接口和用户认证的实践指南 | 豆包MarsCode AI刷题

lann | 2月前 | 45 | 1 | 评论 青训营笔记