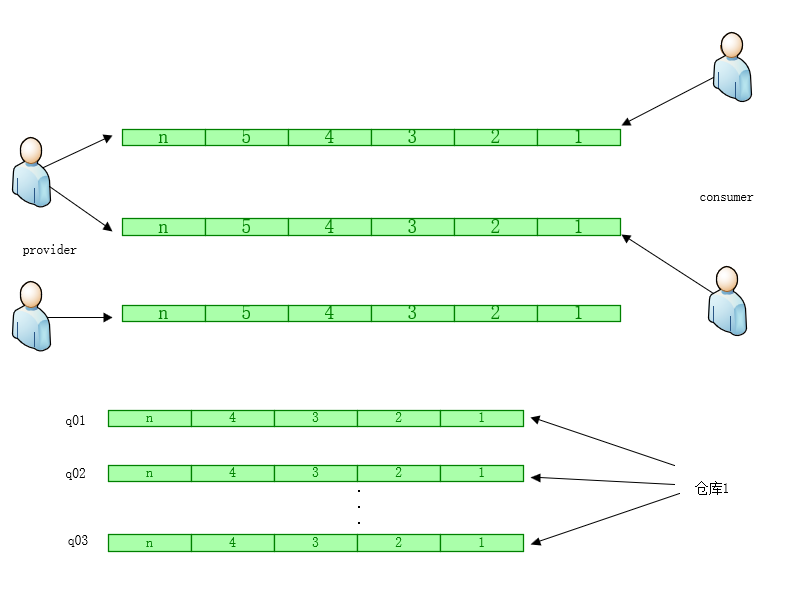
1. 数据的存储



1. 以队列（数组）方式存储，里头存放Msg（产品），队列一头进，一头出。

2. 左边是若干生产者(Producer)，往队列里头发消息，

右边是若干消费者(Consumer)，从队列里头消费消息。

多对多关系。

3. 每一个（或多个）队列表示一个仓库，里面存放Msg（产品）id。

4. 一个仓库由若干个队列组成，这些队列也叫分区(Partition)，不同仓库可能分配有相同队列。仓库可以按需扩容、按需增加。细节由MQ系统屏蔽。

1. 对应的技术栈
2. 后端技术

编程语言: Java

应用框架: SpringBoot ：快速开发、轻量级、高效、安全，适合微服务架构。

1. 数据库技术

数据库: MySQL/PostgreSQL ：MySQL适用于大部分业务场景，而PostgreSQL适用于复杂查询场景。

1. 前端技术

UI框架: React或Angular - React的虚拟DOM和单向数据流有利于大型应用的性能优化，而Angular提供了更多的内置功能，适合全面的企业级应用开发。

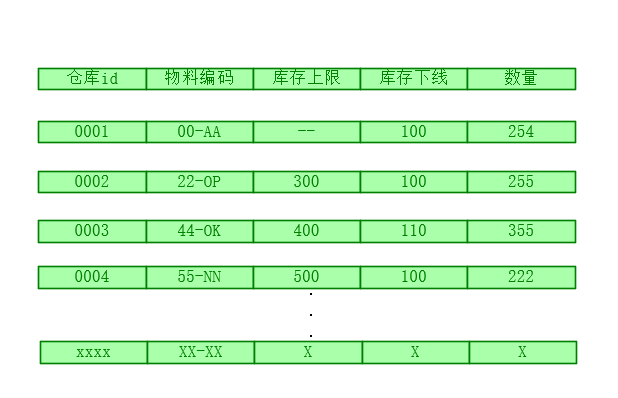
1. 其他技术

容器化: Docker - Docker可以简化环境配置，提升开发部署的一致性和效率。

持续集成/持续部署: Jenkins - Jenkins在自动化构建、测试和部署方面非常灵活，可以大幅提升开发效率。

消息队列: Apache Kafka - Kafka提供了高吞吐量的消息处理能力，适合大规模的数据传输和处理场景。

1. 数据存储模块



1. 仓库 ID 代表产品的存放位置。
2. 物料编码用于标识产品的类型。
3. 库存上限和下限对应产品的安全存量范围，若显示为 “--” 则表示没有限制。
4. 数量表示当前仓库内所有该物料的总量。
5. 订单管理模块



1. 订单创建: 用户可以通过填写客户信息、货物详情、配送方式等创建新的物流订单。系统将进行数据验证并生成唯一订单号。
2. 订单修改: 提供修改界面，允许用户修改订单的详细信息，如更改送达时间或修改客户信息。
3. 订单查询: 用户可以通过输入订单号或客户信息等查询条件快速检索订单。
4. 状态更新: 订单状态将根据物流进展进行更新，用户可以手动或自动更新订单状态，如“已接收”、“在途中”、“已送达”等。
5. 接口设计：

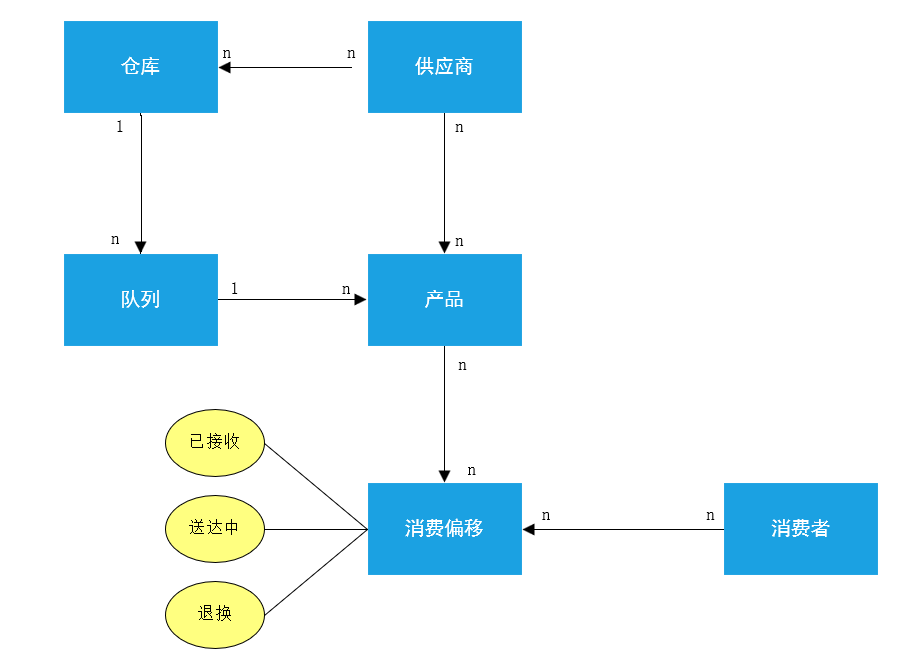
POST /orders: 创建新订单

GET /orders/{orderId}: 根据订单号查询订单

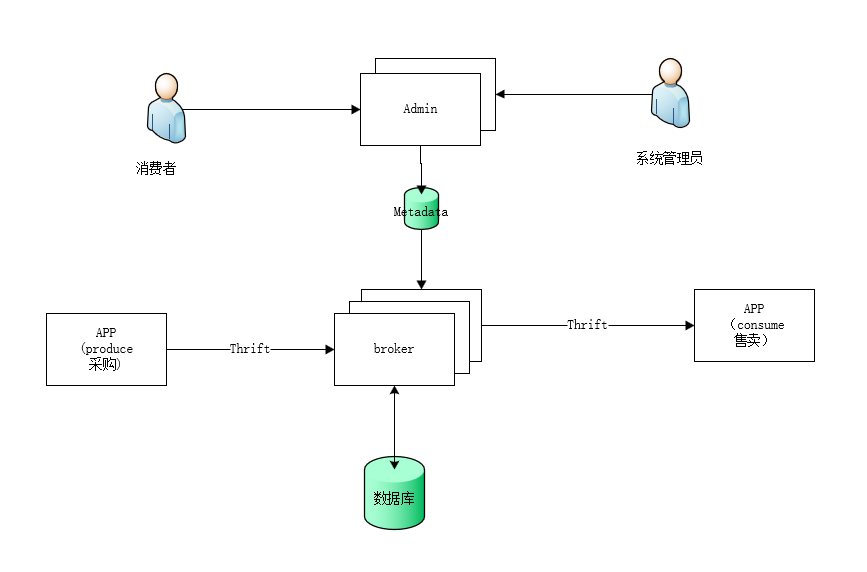
PUT /orders/{orderId}: 更新订单信息

PATCH /orders/{orderId}/status: 更新订单状态

1. 元数据模型



1. 整体架构

1. Admin模块中管理数据库节点，记录采购，销售，队列，消费偏移等元数据信息。系统管理员具有更改，查看这些的权限，而普通消费者仅有查看库存、相关订单的权限。

2. Broker模块定期从Admin数据库同步元数据，接受生产者消息，按路由规则将消息存入对应的数据库表(队列)中；同时接受消费者请求，根据元数据从对应数据库表读取消息并发回消费者端。Broker模块也接受消费者定期提交消费偏移。

3. Producer接受应用发送消息请求，将消息发送到Broker。

4. Consumer从Broker拉取消息，供上层应用进一步消费。

5. 客户端和Broker之间走Thrift over HTTP协议，中间通过域名走Nginx代理转发