阿里巴巴中文站架构实践

何崚@阿里巴巴

Email&google+: helin8@gmail.com

MSN: helintc@hotmail.com

2011-10

我

- 2001-2005 中科院下属研究所
- 2006-至今 阿里巴巴中文站
  - 阿里巴巴中文站架构师
  - 阿里巴巴B2B网站优化领域负责人
- 感兴趣领域: Java, SOA, Performance tunning, Erlang
- Email, google+: helin8@gmail.com
- MSN: helintc@hotmail.com

阿里巴巴中文站简介

- 成立与1999年。域名： china.alibaba.com，1688.com

- 阿里巴巴B2B旗下访问量最大，注册会员最多的网站
  - PV占整个B2B的78%，注册会员数也是B2B最多
  - 应用数量，集群规模在B2B最大

- 业务：
  - 国内最大的网上批发市场
  - 供应商的网店 - 旺铺
  - 会员工作平台 – work平台
  - 诚信商人社区（行业资讯，sns，生意经，博客，论坛）
  - 诚信保障服务
  - ……

阿里巴巴中文站架构发展历程

时间	关键字
1999 第一代网站架构	Perl ， CGI ， Oracle
2000 进入JAVA时代	Java ， Servlet
2001-2004 EJB时代	EJB（SLSB，CMP，MDB）， Pattern（ServiceLocator，Delegate，Façade，DAO，DTO）
2005-2007 Without EJB 重构	去EJB重构: Spring + iBatis+ Webx，Antx, 底层架构：iSearch， MQ+ESB， 数据挖掘， CMS
2008-2009 海量数据	Memcached集群, Mysql +数据切分 = Cobar, 分布式存储，Hadoop，KV , CDN
2010 安全，镜像	安全，镜像，应用服务器升级，秒杀, No Sql，SSD

第五代网站架构

- 第四代网站架构解决了
  - 性能和海量数据问题
    - 大规模的Memcached集群，高性能应用服务器升级，KV,CDN，一定程度解决了网站的性能问题
    - 数据切分和分布式存储解决了网站海量数据的问题。
  - 安全问题
    - 镜像站解决了网站的灾备问题
    - 网站框架的安全特性升级透明的过滤了常见的网站安全漏洞
- 但到了2010年底，我们却不得开始实施第五代网站架构改造

2011 第五代网站架构	？？？？？？？

第五代网站架构的使命

- 敏捷
  - 业务快速增长，每天都要上线大量的小需求。
  - 应用系统日益膨胀，耦合恶化，架构越来越复杂，会带来更高的开发成本。如何保持业务开发敏捷性?
- 开放
  - **Facebook和 AppStore**带来的启示，如何提升网站的开放性，吸引第三方开发者加入到网站的共建中来?
- 体验
  - 网站的并发压力快速增长，用户却对体验提出了更高的要求

2011
第五代网站架构

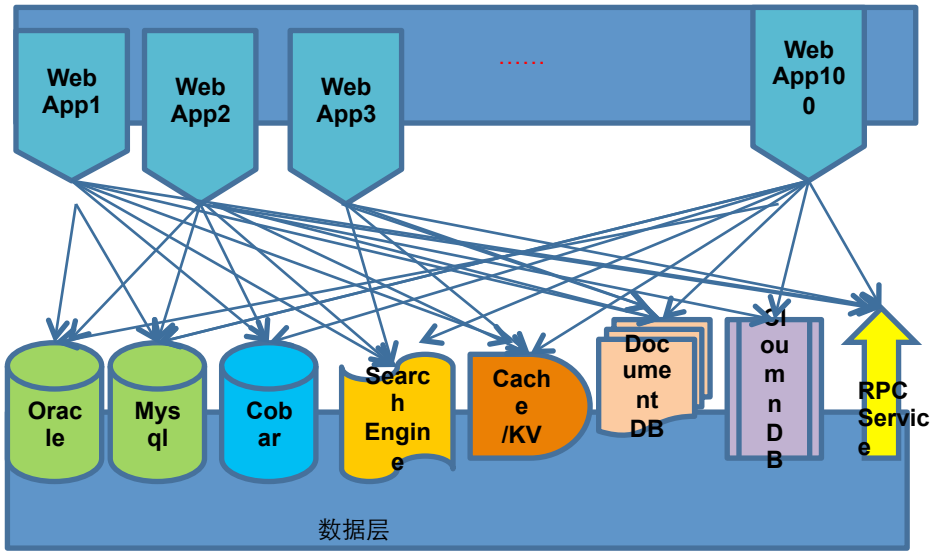
敏捷，开放，体验

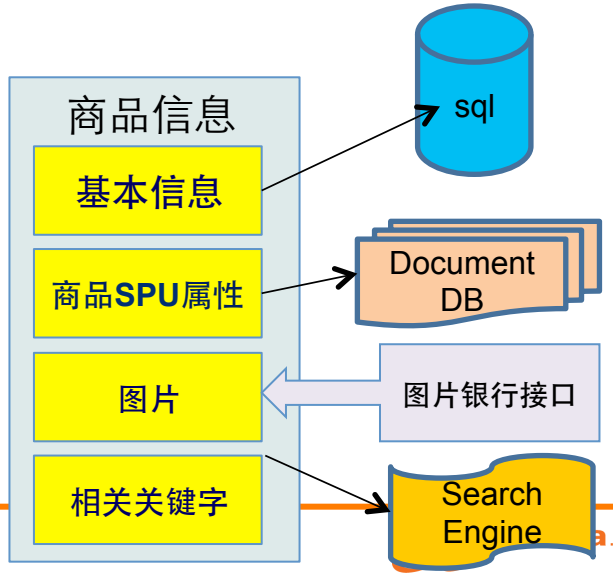
介绍内容

- 数据层
  - 挑战
  - 解决方案：统一数据服务平台
- 业务层
  - 挑战
  - 解决方案：服务化中心 + 模型关系路由框架
- 展现层
  - 挑战
  - 解决方案：**CMS+ Service tag+** 页面组件化框架 **+**店铺装修平台**+**组件服务平台

数据架构日益复杂

- 数据架构现状
- 数据架构非常复杂
  - 在不同的场景采用了多种类型的数据源
    - 关系数据库，
    - 搜索引擎，提供商业搜索服务
    - **Cache，KV** ，高性能场景
    - 外部数据接口：如淘宝/支付宝接口
    - 文档数据库，**Schema free**的结构化数据检索/管理场景
    - 列数据库，后台大规模计算场景。
- 业务模型的各个字段分布在不同数据源





数据架构日益复杂

网站应用层

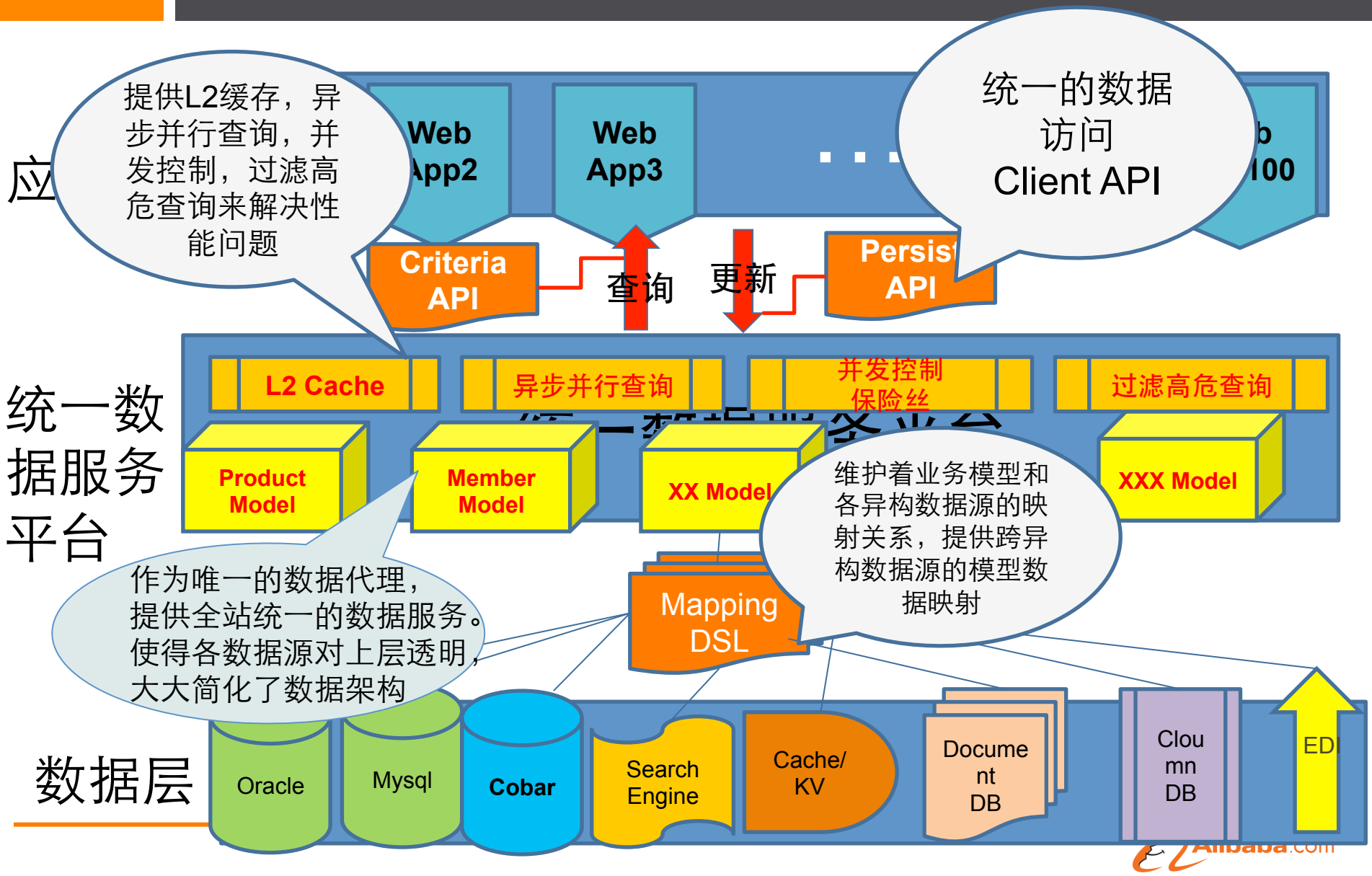
- 问题：
  - 数据架构复杂，应用需要直接依赖多种类型的数据源
  - 开发人员需要熟悉各种数据源，以及访问方式
    - 对开发人员能力提出很高的要求
  - 网站应用组装业务模型的数据需要查询多种数据源
    - 带来开发的不敏捷，大量的资源消耗在无意义的模型组装上
  - 网站应用直接依赖底层数据源，模型发生变更将导致所有相关应用大面积重构
    - 例如商品模型的图片属性由数据库迁至图片银行
  - 数据源改造也会导致相关应用的大面积重构
    - 数据水平切分
  - 跨数据源定位查找问题，实施缓存和性能优化都很困难

解决方案：统一数据服务层UDSL

网站应用层

- 在网站应用集群和底层数据源之间，构建一层代理，统一数据层
- 统一数据层的特性
  - 模型数据映射
    - 实现 业务模型 各属性 与 底层不同类型数据源的 模型数据映射
    - 目前支持关系数据库，**iSearch**，**redis**，**mongodb**
  - 统一的查询和更新**API**
    - 提供了基于业务模型的统一的查询和更新的**API**，简化网站应用跨不同数据源的开发模式。
  - 性能优化策略
    - 字段延迟加载,按需返回设置
    - 基于热点缓存平台的二级缓存。
    - 异步并行的查询数据:异步并行加载模型中来自不同数据源的字段
    - 并发保护： 拒绝访问频率过高的主机**IP**或**IP**段
    - 过滤高危的查询：例如会导致数据库崩溃的全表扫描

数据层 解决方案： UDSL (统一数据服务平台)



数据层

UDSL：模型字段的数据路&Mapping DSL

- 问题
  - 传统O/Rmapping框架，不能实现非关系数据库和跨不同类型数据库的对象数据映射
    - 早期 O/R Mapping框架，简化了对关系数据库的查询方式，采用面向对象的方式查询管理数据，但对非关系数据库类型，不能实现对象数据映射，更不能跨不同数据源映射
  - UDSL提供除关系数据库以外的数据源的对象关系映射
    - 目前支持 关系数据库，iSearch, Mongodb，Redis
  - UDSL支持同一业务模型的各字段映射到不同的数据
- 设计1：模型数据映射 DSL，定义模型字段和底层数据库的映射关系
  - 我们设计了一套DSL，
    - 描述模型对应哪些类型的数据源，各个数据源的访问方式是什么
    - 模型有哪些字段，每个字段对应哪个数据源的哪个字段或哪个数据接口方法
    - 模型字段是否延迟加载
    - 模型字段的值如果需要对原始数据进行逻辑处理，用何种方式逻辑处理
  - UDSL阅读DSL定义的数据路由，访问不同数据源，组装模型
    - 通过这套DSL我们定义模型各字段和各数据源的数据路由,UDSL在查询数据时通过阅读DSL路由规则来聚合各数据源数据，组装模型。

UDSL：统一查询更新API

- UDSL采用统一的查询/更新API，统一了不同数据源的查询方式
  - 查询API (Criteria API)： 提供类似JPA
    - 基于模型表达式的查询方式:
      - 商品模型的行业属性是否等于“服装”
    - 支持按模型属性的结果排序
    - 支持限定结果返回条数和返回哪些字段
    - 和JPA非常不同的是：
      - 支持按需加载：可以指定只返回哪些字段 ，或者过滤哪些字段。

方法	说明
find	查找
orderBy	排序
limit	约束

表达式	说明	示例
eq	等于	object1.field1.eq（“xxx”）
gt	大于	object1.field2.gt（100）
desc	倒序	object1.field3.desc

  - 持久化API (Persist API)：提供简单的接口
    - 只有四个方法：**insert**，**update**，**delete**，**insertOrUpdate**，参数就是数据对象，非常简单
      - **Dml.insert(product)， Dml.update(product)， Dml.delete(product)**
      - **Dml.insertOrUpdate(product)**
- **UDSL自动的分析查询/更新参数，并根据模型字段映射配置，转换成底层各数据源的native 语句进行数据操作**

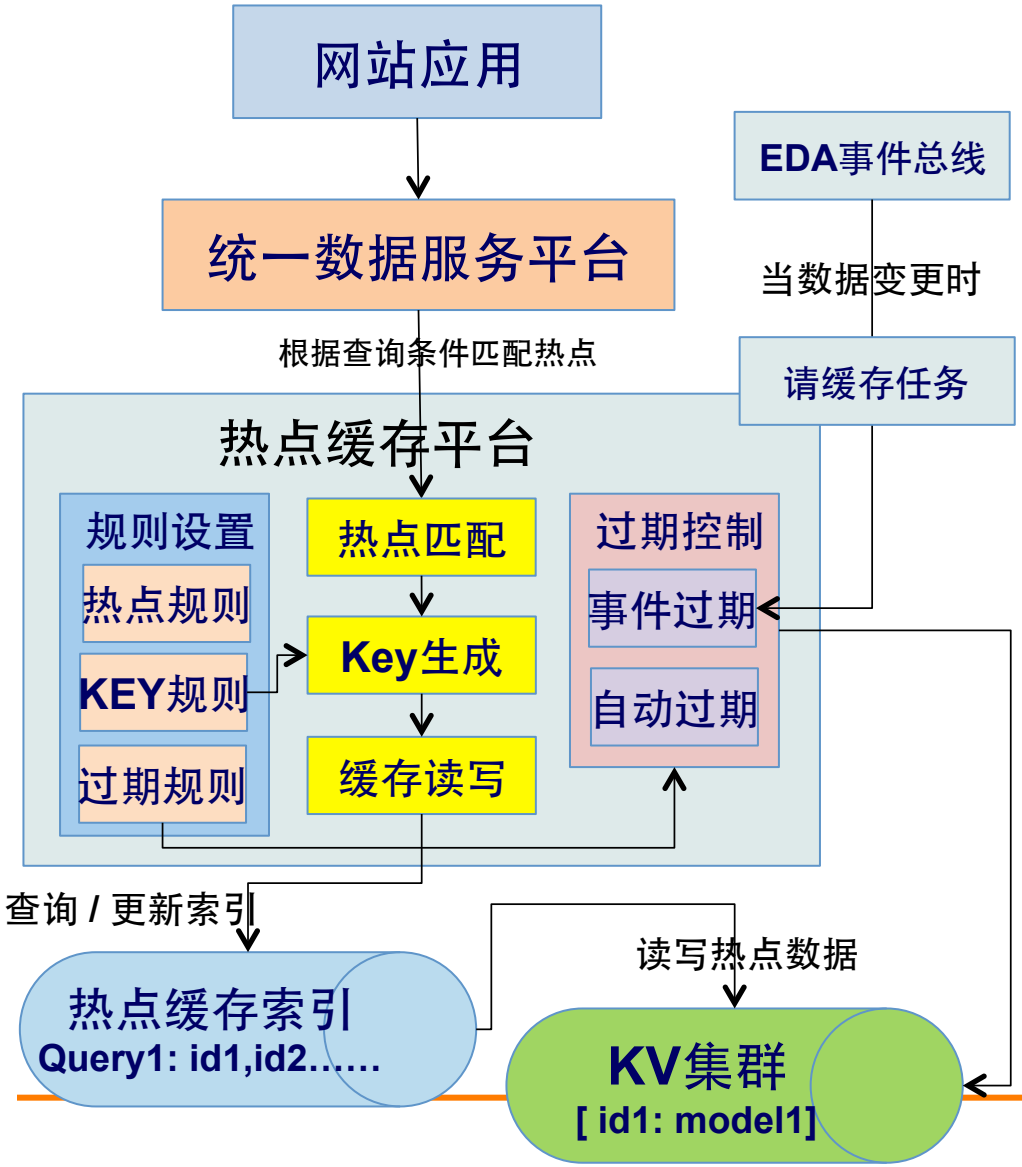
UDSL:性能优化

- 问题：UDSL完成跨数据源的对象查询非常消耗资源
  - 业务模型的不同字段可能会来自不同的数据源，组装对象成本很高
- UDSL的性能优化策略
  - 字段延迟加载
  - 字段按需返回设置
  - 架设L2 Cache.
  - 并步并行的加载：异步并行加载模型中来自不同数据源的字段
  - 并发保护：拒绝访问频率过高的主机IP或IP段
  - 过滤高危的查询：例如会导致数据库崩溃的全表扫描

UDSL:热点缓存 背景

- 问题1：性能问题
  - UDSL上线后，数据架构大幅简化，开发敏捷，但性能问题亦很严重
    - 业务模型的不同字段可能会来自不同的数据源，组装对象成本很高
    - 延迟加载和按需返回设置不当，加载冗余数据，会导致性能问题
    - 呼唤缓存
- 问题2：网站数据非常庞大，缓存过多数据消费比不高，只能缓存热点数据
- 分析：网站数据访问存在明显的热点分布.
  - 例如，行业热点，占总数10%的热门行业的商品的浏览量占全站商品浏览总量的90%以上
- 结论：缓存热点数据
  - 以较少的存储代价，大大缓解应用系统及数据库的压力。
- 解决方案：开发了热点缓存平台，提供给UDSL作为缓存系统.

UDSL:热点缓存 设计



- 热点缓存平台
  - 作为UDSL的二级缓存
  - 提供DSL，支持定义数据热点规则
  - 热点缓存平台有一级缓存索引，查询UDSL时，首先根据查询条件匹配DSL，判断热点
  - 如果匹配热点，根据查询条件生成Cache key。
  - 根据Cache key访问Cache Index，如果命中，返回一组查询结果数据的ID列表。
  - 根据查询结果数据的ID列表去KV系统中取得查询结果数据。

UDSL:热点缓存 规则设置

- 设计了一套DSL，设置热点规则，Cache key规则，过期规则
- 热点规则设置
  - 根据UDSL查询条件表达式来设定热点规则
    - 例如 缓存服装行业的产品， DSL：

```
<hotspot name="rule1" expression ="Product.category.eq('服装')"/>
```

  - 通过UDSL Criteria API 调用传入的查询条件表达式来匹配热点
    - 热点匹配:Query.find(Product.category.eq('服装')).orderby(Product.salenum.desc).limit(100).list();
- Cache key规则
  - 默认根据查询条件生成key
    - 例如： Query.find(Product.category.eq('服装')).orderby(Product.salenum.desc).limit(100).list();
      - Key: Query.find(Product.category.eq('服装')).orderby(Product.salenum.desc).limit(100)
- 缓存过期规则设置
  - 绝对过期时间 & 相对过期时间：生存周期

热点规则

过期规则

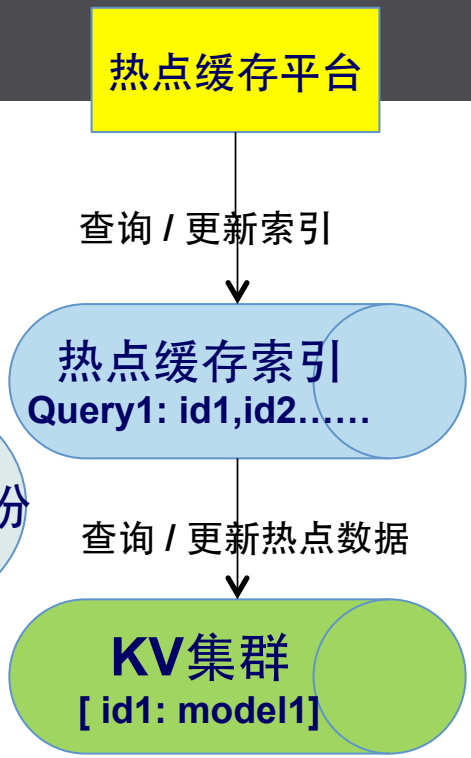
KEY规则

UDSL:热点缓存 索引策略

- 缓存索引index
  - 缓存结果集合所有对象的主键列表
  - 使用索引的原因：节约存储空间
    - 假设不使用Index，直接缓存至KV

Cache Key	Cache Value
Query Key1	Model1，Model2，Model3
Query Key2	Model2，Model3，Model4

数据保存了多份



- 使用Index ， 建接缓存至KV

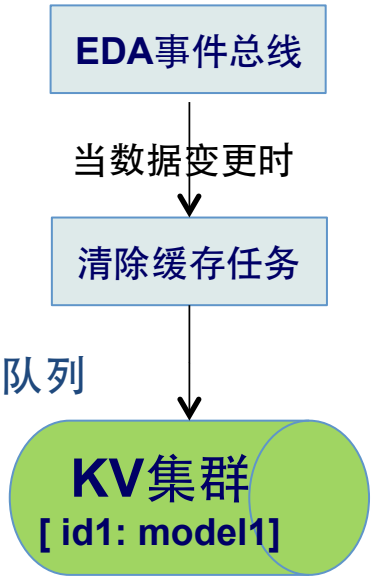
Index Key	Index Value
Query Key1	Model_id1，Model_id2，Model_id3
Query Key2	Model_id2，Model_id3，Model_id4

Cache Key	Cac
Model_id1	Mod
Model_id2	Model2
Model_id3	Model3

数据无冗余

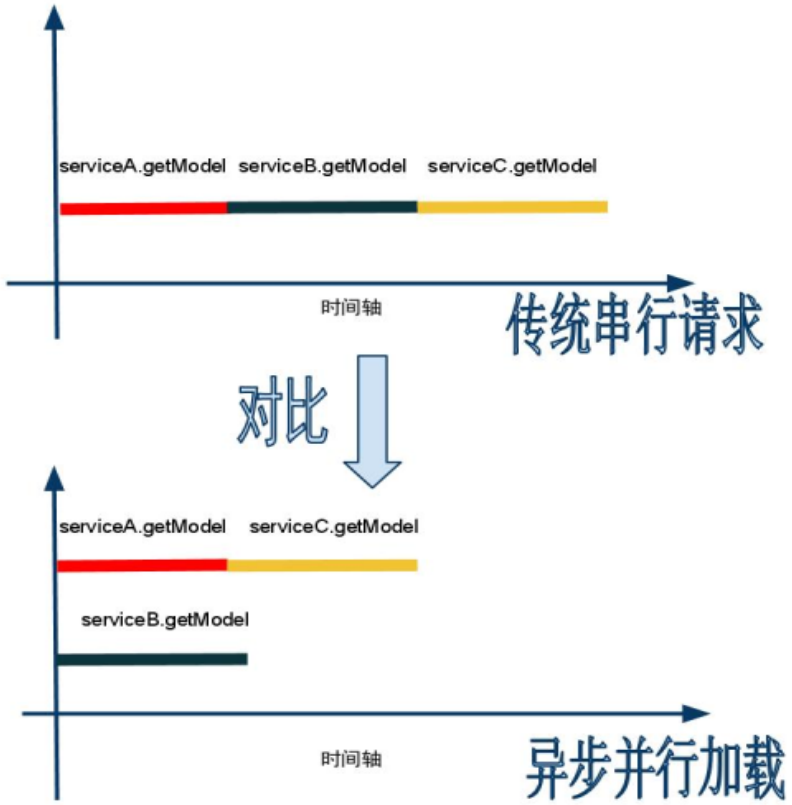
UDSL:热点缓存 缓存失效机制

- 自动过期
  - 绝对过期时间 ： 绝对的时间点
  - 相对过期时间：生存周期
- 基于EDA的事件过期
  - 中文站的商业数据发生变化时，会产生消息发送到MQ集群的指定队列
  - 热点缓存失效任务监听数据变更事件，踢掉KV中的数据
  - 数据变更时，只更新缓存，不更新索引
- 索引失效
  - 数据变更时，更新索引需要遍历所有索引，非常耗时，所以不主动更新索引
  - 客户端API的容错机制保证，当索引内结果集的对象ID值，在KV中找不到时，索引被动失效。



数据层

UDSL：查询异步化，并行化




```
ModelA modelA = serviceA.getModel(); //1. 异步发起请求
ModelB modelB = serviceB.getModel(); //2. 异步发起请求
// 3. 此时serviceA和serviceB都在各自并行的加载model
if(modelA.isOk()){//4. 此时依赖了modelA的结果，阻塞等待modeA的异步请求的返回
    ModelC modelC = servicec.getModel(); //5. 异步发起请求
}
// 6.  此时serviceB和serviceC都在各自并行的加载model
......
modelB.xxxx() //7. 数据处理，modelB已经异步加载完成，此时不会阻塞等结果了
modelC.xxxx() //8. 数据处理，modelB已经异步加载完成，此时不会阻塞等结果了
```

问题：页面的数据请求过程耗时

- 一个页面经常要查询多个数据源
- 传统的编程模式，是一种串行的请求方式
  - 先查询第一个数据，查完后再查下一个数据
- 整个请求的耗时，等于所有数据请求的总和

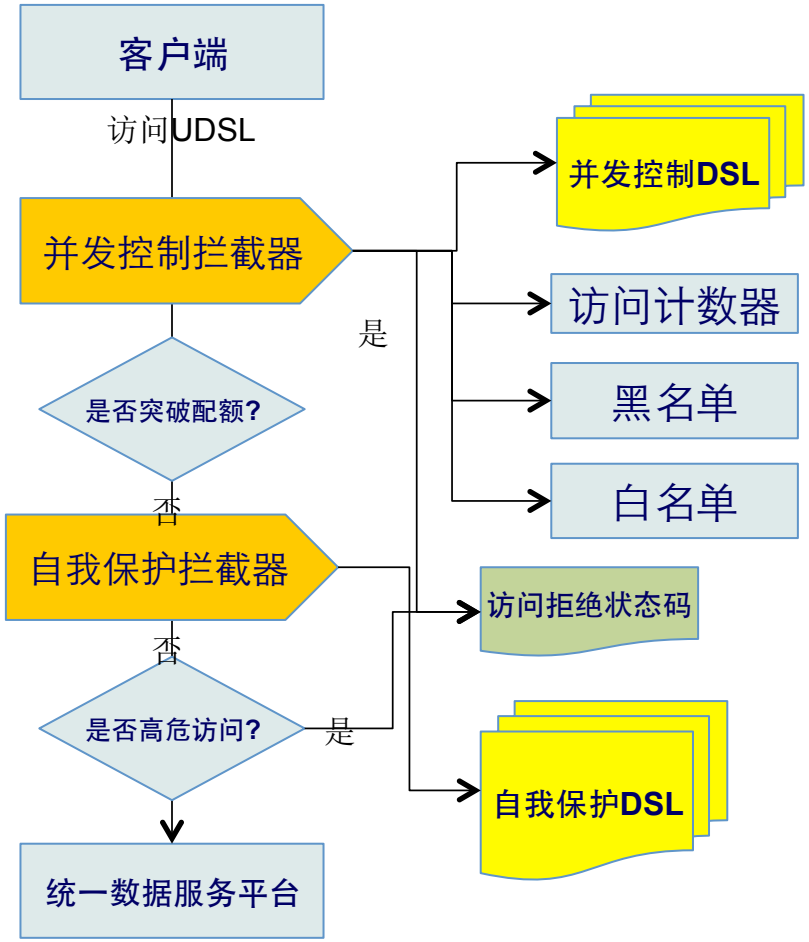
解决思路：异步并行查询

- 借鉴**Ajax**的思想运用到数据服务层
- 异步的并行加载同一业务场景中对不同数据源的查询

解决方案：异步并行加载框 架**Asyncload**

- 整个业务处理流程的总耗时，只取决于最慢的一次查询
- 相对于**Ajax**，不会对页面**seo**有任何的影响
- 减少了**Ajax**异步发起的**http**请求
- 两块代码的资源不存在重复请求，允许进行资源共享

UDSL：并发控制，自我保护



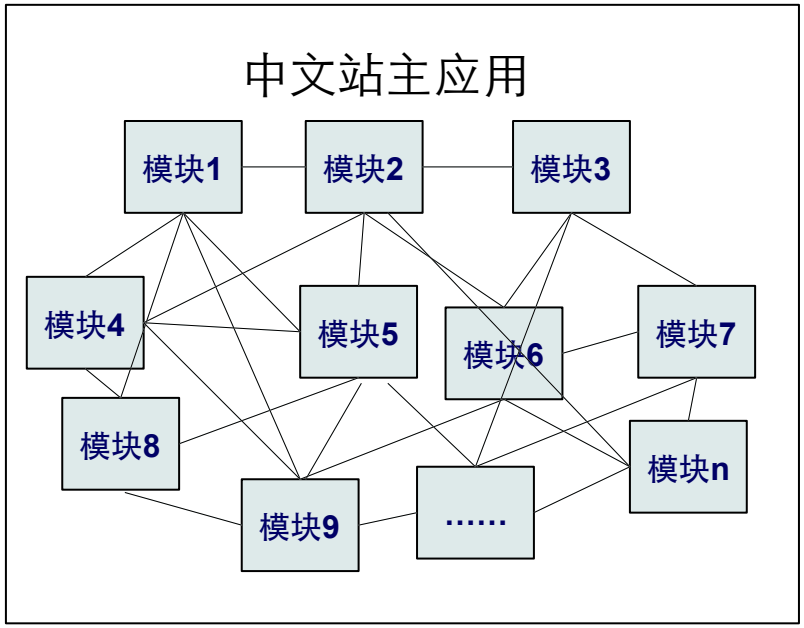
- 并发控制机制：过滤危险IP段
  - 并发控制DSL
    - 可以配置访问并发阀值
      - 例如某IP段单位时间内访问某个接口频次就自动列入黑名单
    - 白名单
- 自我保护机制：过滤危险数据请求
  - 自我保护DSL
    - 设定基于Cretria API 查询参数的过滤规则，例如可以限度请求商品返回条数不能超过100
    - 设定超时机制

数据层架构改造回顾

问题	解决方案
数据源类型很多，架构及其复杂，导致网站应用开发的不敏捷	使用UDSL作为异构数据源的统一代理，应用采用统一的API访问异构数据，使得数据源对应用透明，大大降低了数据架构的复杂度，提升了开发和维护的效率
业务模型变更或数据源改造，会导致相关应用大面积重构	模型或数据源变更后，只需修改UDSL中模型/数据源Mapping DSL中的字段路由配置，对应用改造较小，无需大面积网站应用重构
数据源改造导致相关应用大面积重构	模型或数据源变更后，只需修改UDSL中模型/数据源Mapping DSL中的字段路由配置，对应用透明，无需大面积网站应用重构
跨数据源下性能优化，并发控制等均很困难	通过**热点缓存平台**定义热点规则缓存热点数据，以较小的存储代价换来较大的性能提升 通过**异步并行加载框架**对同一场景的不同数据查询实施异步并行加载，大大降低了查询等待时间 通过**UDSL**数据访问的**并发控制，自我保护策**略，有效的保证了UDSL的可用性。

问题：耦合严重，架构腐化

- 早期：
  - 中文站按照域名划分为应用
  - 大部分业务由中文站主应用提供服务
- 问题：
  - 主应用日益膨胀
  - 耦合失控
    - 单一应用内建立依赖成本很低
    - 依赖失控，无法管理
    - 甚至出现环形依赖
  - 应用/故障不隔离，可用性下降



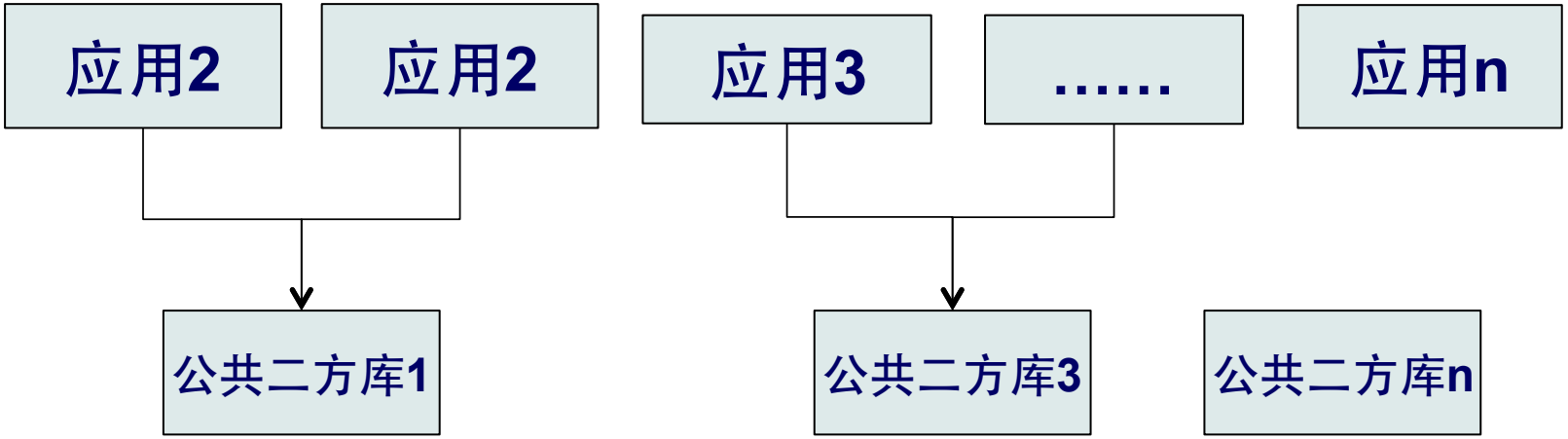
解耦2：应用拆分

- 将中文站主应用按业务线拆分，物理隔离解耦

- 结果：
  - 拆分成业务相对独立的系统，降低了复杂度，开发效率提升
  - 各应用物理隔离，无法建立依赖
  - 应用/故障隔离
  - 可做应用读写分离

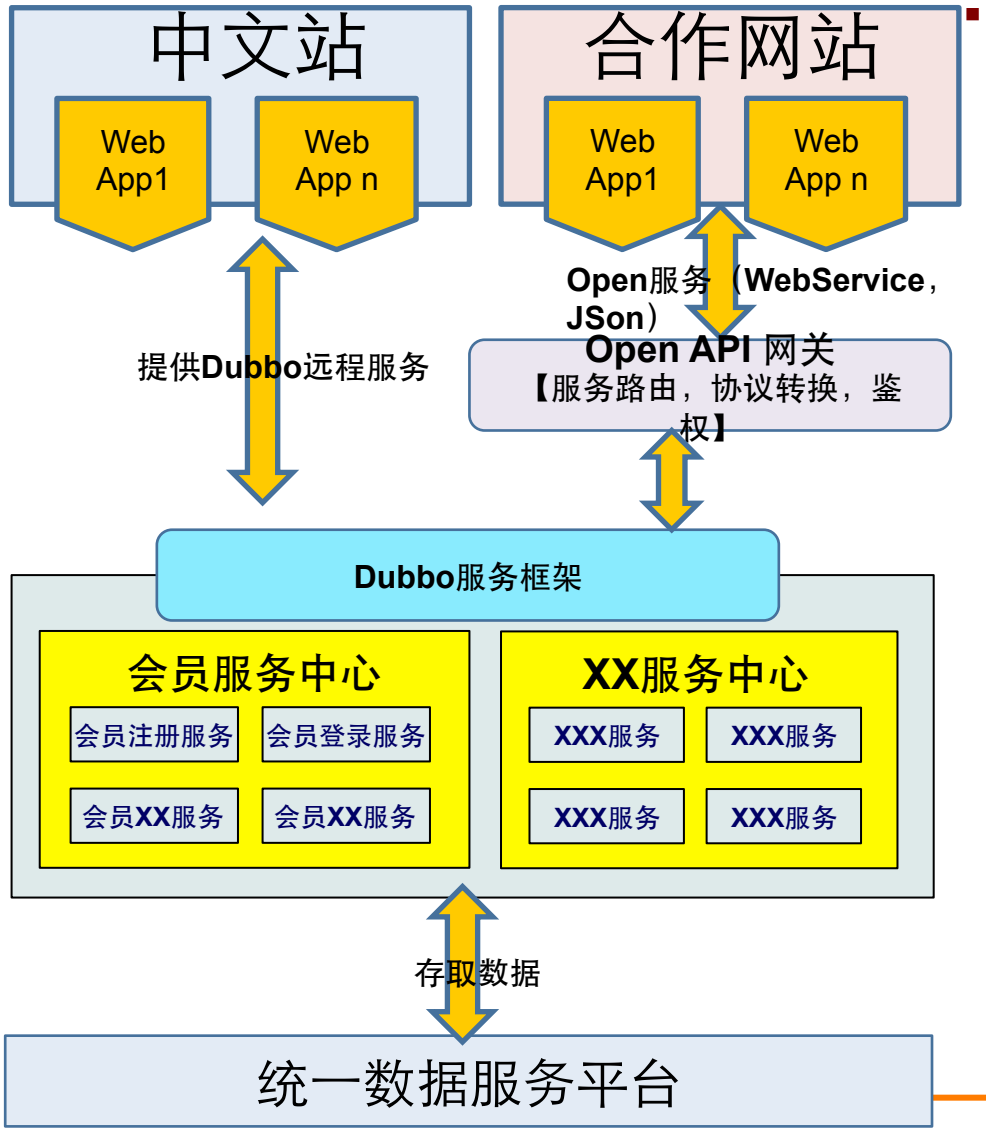
解耦3：建立公共二方库

- 问题：各应用中存在公共代码
  - 第一个极端，导致代码Copy，公共代码在应用间Copy
  - 第二个极端，重复发明轮子，每个应用各搞一套实现
- 解决方案：创建二方库
  - 将公共代码封装成二方库，供应用依赖
  - 最大限度的代码重用



业务层

最终解耦方案：服务化中心



- 问题
  - 二方库发布升级困难
    - 二方库升级会导致所有依赖应用升级
    - 只修改实现不改接口也要大面积升级
  - 二方库不适合作为Open API 提供方式
- 解决思路：采用远程RPC服务取代二方库
  - 服务实现改变不会导致大面积发布
  - 方便服务治理，能够有效的管理依赖
- 解决方案：服务中心
  - 阿里巴巴高性能服务框架 Dubbo
  - 所有公共模块改造成Dubbo服务
  - 服务按产品线划分部署，每个产品线都有自己的服务集群，即服务中心。
  - 服务化中心提供的服务通过Open API提供给第三方合作网站授权使用
  - 网站应用调用服务快速构建

业务层

服务化带来的开发模式的转变

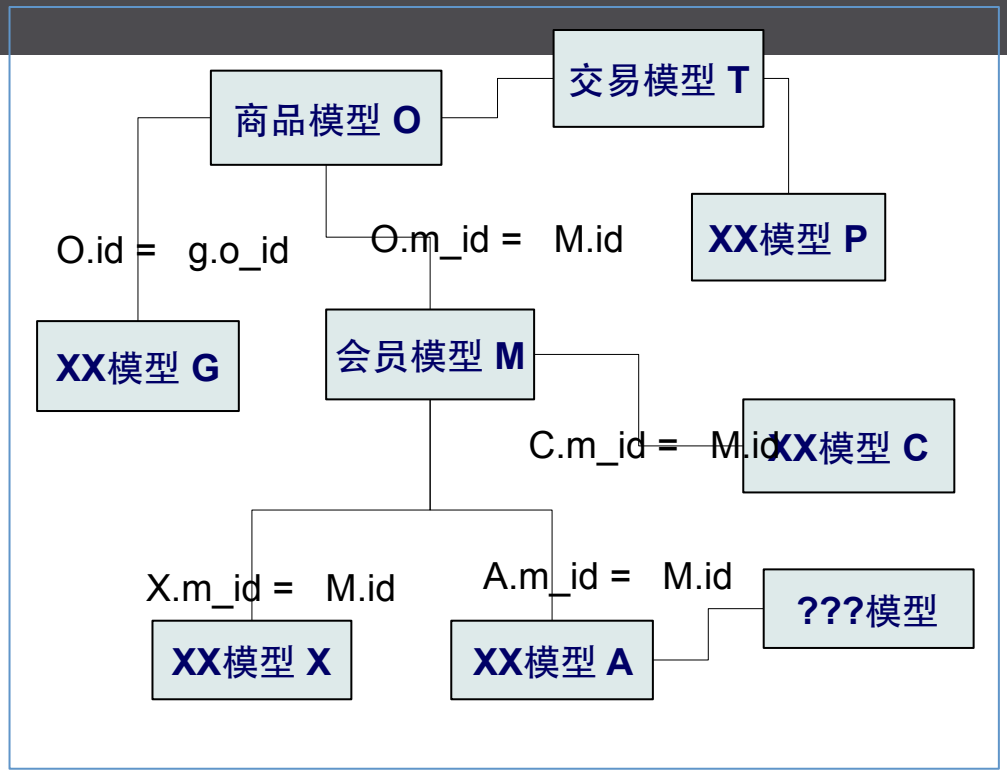
- 服务化前
  - 跨业务线的产品开发，需要依赖别的团队资源，沟通成本很高
  - 例如交易线要开发新产品，新产品中涉及其他业务线内容，需要申请别的团队的开发资源进入项目组，很容易产生资源瓶颈
- 服务化后
  - 减少了跨业务线合作沟通成本
    - 无需依赖其他业务线开发资源，阅读相关业务线服务化API文档，调用服务化中心服务开发。
  - 为网站开放化奠定基础
    - 业务线服务化中心提供的核心服务，成为网站Open API的基础。

业务层 新的问题：页面取数逻辑复杂

- 问题：网站页面取数逻辑复杂
  - 需要聚合展示来自各业务线的数据
  - 需要处理各数据间复杂的关联关系
  - 需要调用多次服务化接口，取得关联数据
  - 相关工作 占 全站所有开发人日 的很大比例

模型关系图

- 业务模型关系图
  - 把主要的业务模型排列出来
  - 每个关联模型之间用实线连起来
  - 每条实线上标明模型之间的关联条件



- 全站主要业务模型 + 模型间关联条件 → 无向图
- 由已知图中任一模型数据可以推导出 其他模型数据
- 网站大部分页面的应用场景 都是 这一关系图的遍历过程
  - 例如商品详情页面：根据商品ID 获取 商品详情，交易条件，卖家，评价等 数据并展示页面

模型关系图框架 ModelMap & Mapping DSL

- 模型关系图框架 **ModelMap**：
  - 通过单一查询条件获取应用场景所有的关联数据
    - 通过商品ID获取商品详情页面的所有关联数据
  - 模型图定义**DSL**：定义模型的查询方法和 模型之间的关联关系
    - **Step1**：定义模型的查询方法

```
<!-- 定义全站业务模型 -->
▼<models>
  <!-- 定义商品模型 -->
  <model id="product" alias="商品" class="com.china.alibaba.model.ProductModel" keyFieldName="id"
  keyFieldType="int" factoryInterface="com.china.alibaba.service.ProductService"
  factoryMethod="getProductById"/>
  <!-- 定义会员模型 -->
  <model id="member" alias="会员" class="com.china.alibaba.model.MemberModel" keyFieldName="id"
  keyFieldType="String" factoryInterface="com.china.alibaba.service.MemberService"
  factoryMethod="getMemberById"/>
  ……
</models>
```

    - **Step2**：定义业务模型之间的关联关系：用基于模型的关系表达式

```
<!-- 定义模型关系 -->
▼<relationships>
  <relationship from="product" to="member" expression="product.memberId == member.id"/>
  ……
</relationships>
```

模型关系图框架 ModelMap & Query API

- 模型关系图框架 **ModelMap**
  - 模型图遍历**API:** 通过一次查询关联获取各服务中心的所有相关模型
    - 通过 初始的查询条件 遍历 获取模型图中指定模型数据
      - 根据商品**ID**获取商品详情所需的所有数据
    - 查询条件表达式是基于模型属性的逻辑表达式
      - 例如： 商品**ID=XXXX**
    - 可指定返回哪些模型 **[**商品，会员**]**
    - 返回结果是包含模型对象结果集的**Map**
    - **Map modelMap = ModelQuery.query('product.Id=1111','product,member');**

业务层架构改造回顾

问题	解决方案
网站应用耦合失控，导致，架构腐化，敏捷下降的问题	抽取核心服务，建设各业务线的服务化中心来解耦，并实现大粒度的代码重用。 同时服务化中心提供的核心服务也成为网站Open API的基础。
网站主要页面查询关联数据对象逻辑复杂带来的巨大开发成本	设计模型关系图框架来定义管理业务模型的查询方式和模型之间的关联关系，并提供API，通过单一的初始条件获取到所有页面逻辑所需要的关联数据对象，实现页面的快速开发。

展示层

建站需求受开发瓶颈制约

- 背景：中文站由大市场向行业化运营方向转变
  - 需要建设多个垂直行业化网站，如服装批发市场，小商品批发市场
  - 需要不定期推出运营专题推广网站，如1688开业秒批网站
- 问题：快速增长的建站需求受到有限的开发资源的制约
- 解决思路：网站运营自己DIY网站页面,解放开发资源
  - 有利条件：中文站有CMS系统，运营人员普遍有一定的HTML编写能力
  - 唯一问题：运营不知道如何取得网站页面的数据
  - 结论：建立一种运营可接受的获取页面数据的方式，就可以实现运营DIY站点
- 解决方案：Service tag 【模板服务化标签】
  - 扩展了CMS系统的模板标签，使得用户能以模板标签的方式访问查询服务
  - 网站运营中CMS的Html 编辑框中用模板标签调查询服务取数据，所见即所得的编辑页面 eg. $Product = $ProductService.getProductById(id)

展示层

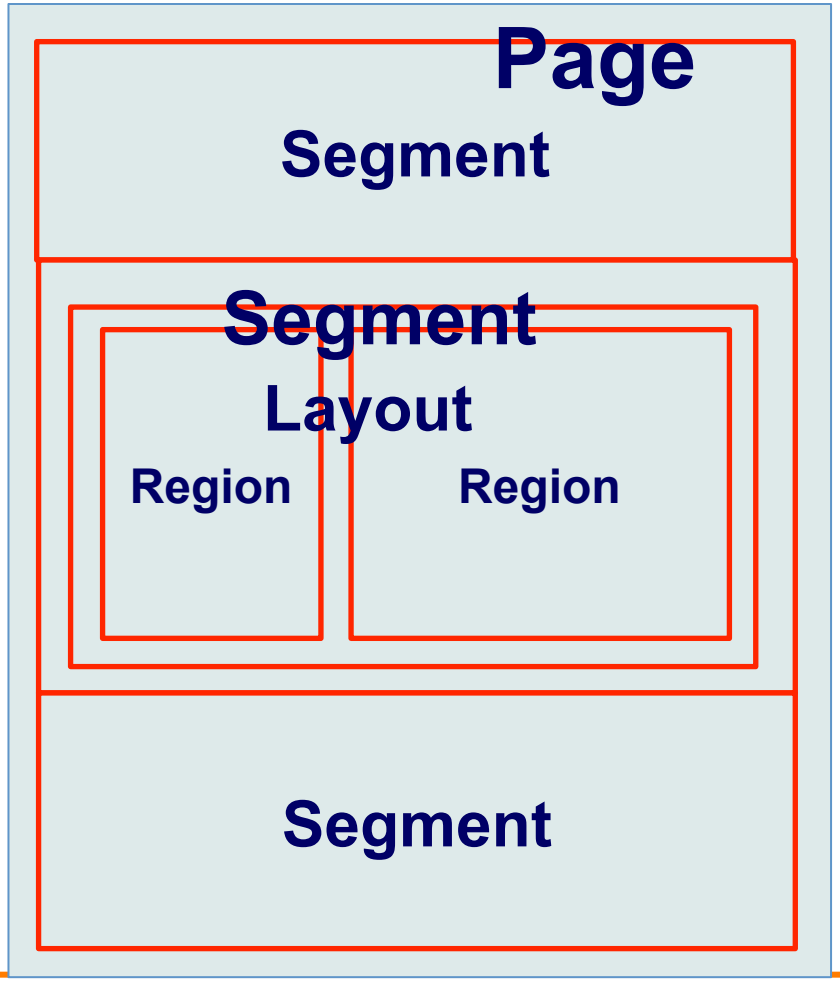
网站开放性需求：卖家希望个性化DIY网店

- 背景：卖家迫切希望能个性化DIY网店，更好的展示商品
  - 自定义网店的展示内容，包括定义页面的板块
  - 自定义网店的展示风格，包括页面布局，皮肤
- 解决思路：页面的组件化开发
  - 将网店主要板块设计成组件，供用户选择，例如：商品详情，店铺搜索，卖家联系方式
  - 提供网店的布局，皮肤，供用户选择
  - 提供可视化编辑器，用户拖拽组件到页面指定区域，**WYSIWYG DIY**页面
- 解决方案：页面组件框架 + 旺铺装修平台
  - 设计了页面组件框架，将网店主要功能板块改造成组件。
  - 推出了店铺装修后台，提供 版块库，布局库，皮肤库，供用户个性化装修店铺

展示层

页面组件化：组件框架的概念模型

- 组件（Component）：构成网站展示的基本单元

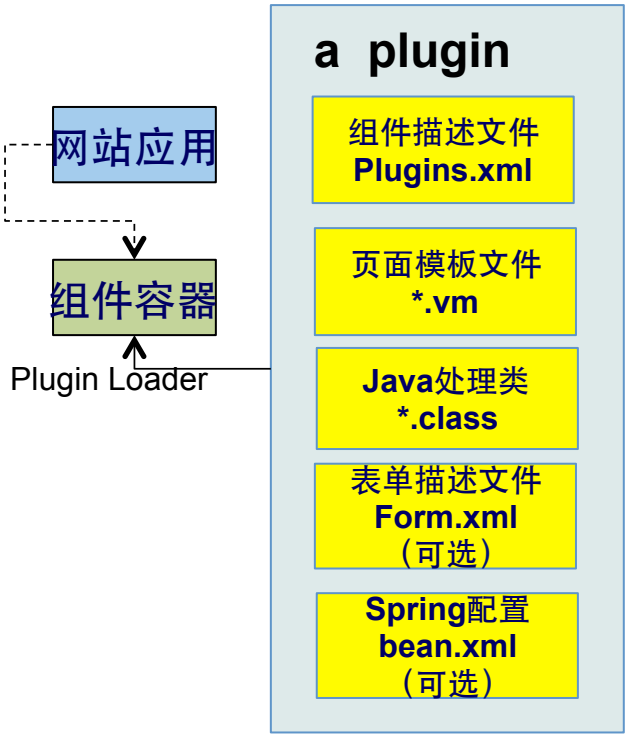


概念	描述
页面(Page)	用户的信息组织表现形式
段(Segment)	页面从上往下切分成不同段，默认是头、中、尾三段
布局(Layout)	对段里面内容的排版
区域(Region)	布局当中一个小格局
版块(Widget)	基于HTML的Web页面上一个块代码段，也就是网站的功能模块
主题(Theme)	一套页面集合与一种皮肤风格的一种组合
皮肤(Skin)	Web页面表现样式风格，实现层面就是css

组件的部署方式：插件

- **Plugin（插件）：** 一组组件的集合，也是网站的功能单元，物理上就是一个jar包
  - 是组件的物理部署方式
  - Page，Layout，Widget，Skin都以这种方式部署

- **Plugin（插件）的内容**
  - 组件描述文件 **Plugins.xml**：
    - 定义组件的扩展点和结构：
      - 组件参数，模板，处理Java类，服务接口
  - 页面模板文件 ***.vm**
  - 组件所需的java类 ***.class** （ action class)



- 网站的组件化改造后，整个网站由一个个插件组成了一个松耦合的高度可扩展的系统

页面组件化：页面结构设计

- 组件框架的页面结构设计
  - 页面由上而下划分为几个段（Segment）
    - 默认是上中下三段
  - 每个段有自己的布局(Layout)
  - 每个布局(Layout)划分为几个区域（Region）
    - 默认是 二通栏，左边是side，右边是main
  - 每个区域可以放置多个版块（ Widget ）
  - 页面采用Json描述Page->Segment->Layout->Region->Widget树状的页面结构
  - 每个页面有一个默认的json文件定义页面默认的结构。
  - 用户自定义页面结构后，会生成自己的json结构到数据库
- 页面的 渲染
  - 根据JSON描述的页面结构，得到层层嵌套的Layout和Widget组件，递归渲染，生成页面

展示层

页面组件化：主题(Theme)，皮肤

- **主题(Theme)：** 一套页面（Page）集合+一种皮肤（Skin）风格
  - 定义了一个站点下有哪些页面（Page），站点用哪个皮肤(Skin)
  - 一个网站可以多个主题，提供给不同角色的用户
  - 可以给不同用户分配不同主题
- 用户第一次进入旺铺装修平台，会根据特定规则分配给一套站点主题

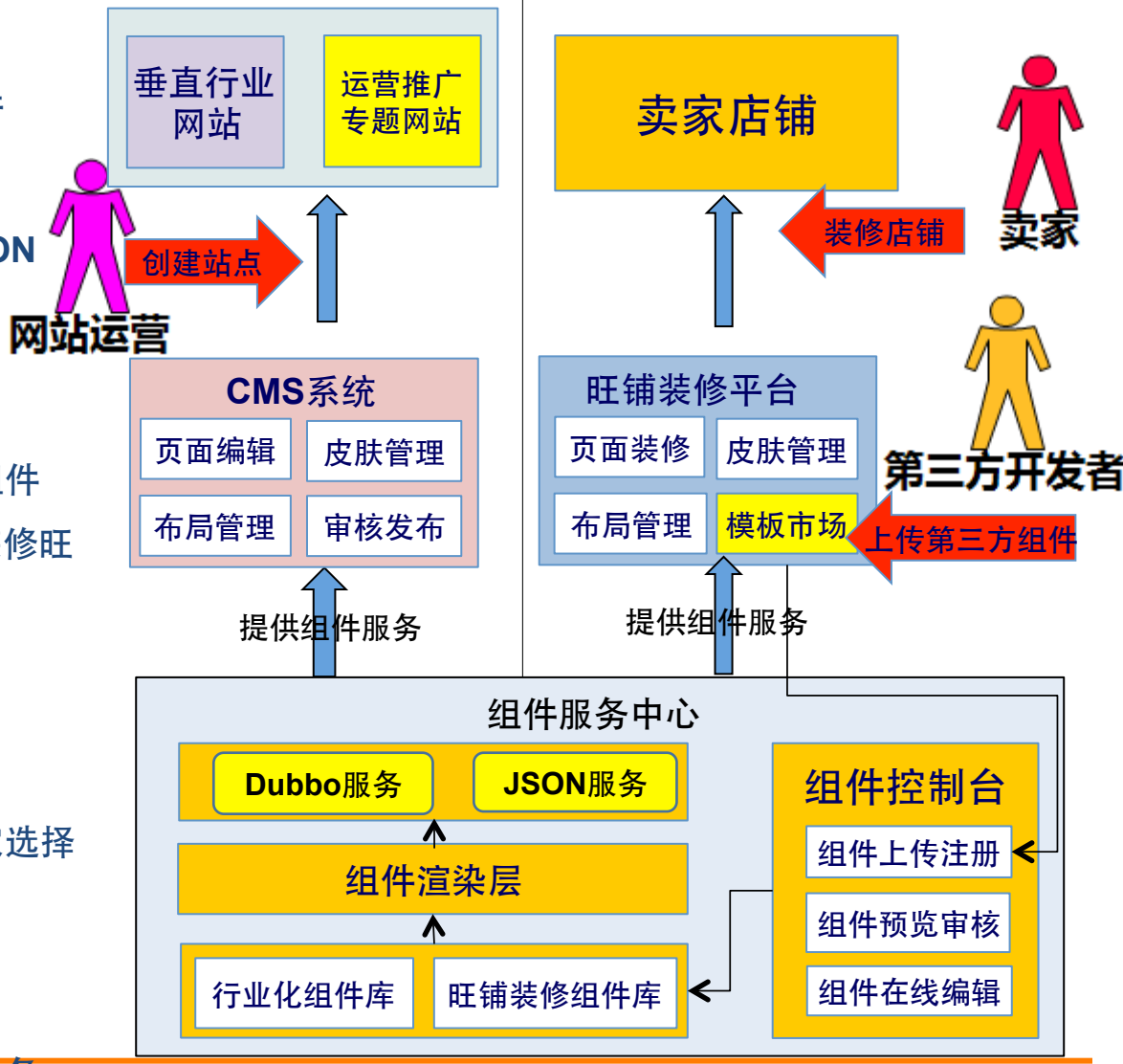
  主题定义的所有页面配置json都存一份到User_Page表用户页面自定义表。

皮肤（skin）在组件框架中一种特殊的组件，对应着一套站点的css，供用户选择保存在用户自定义站点表

展示层

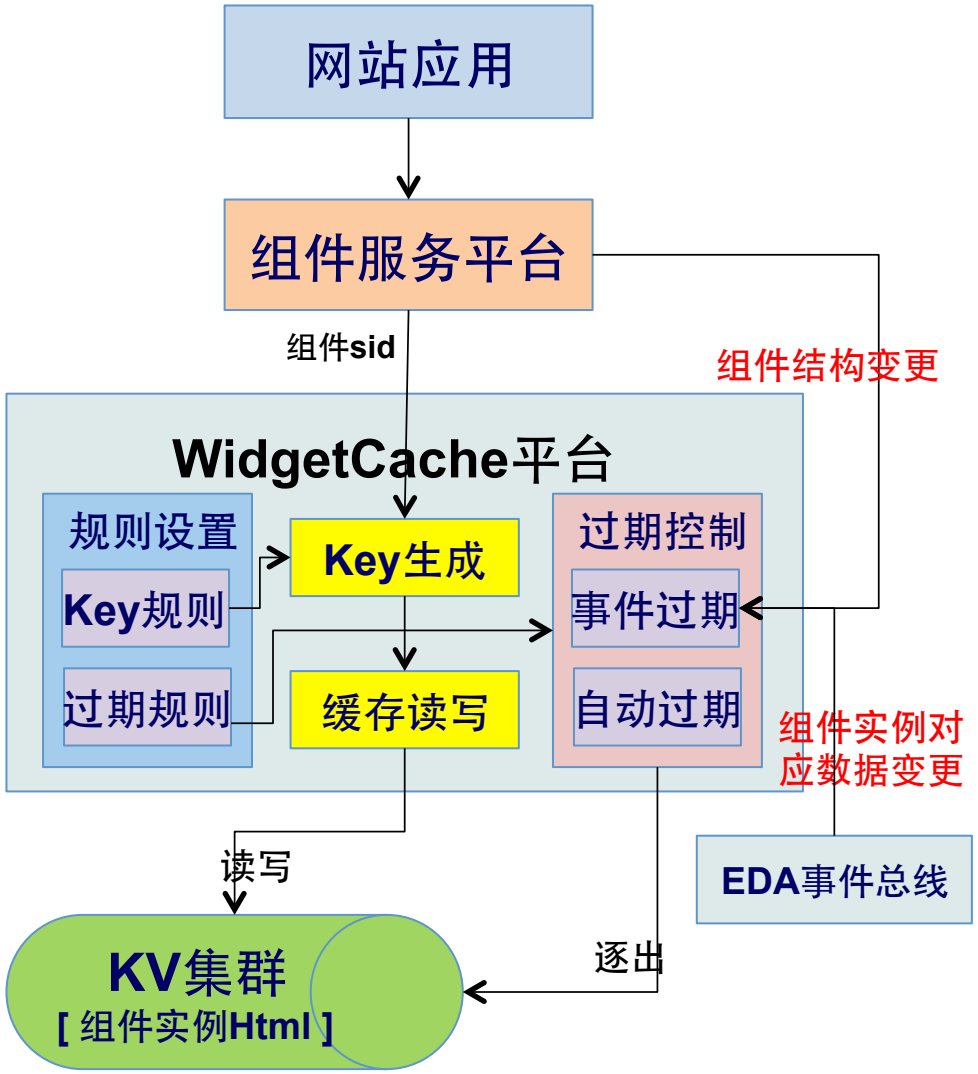
基于页面组件的开发模式

- 组件服务中心
  - 组件控制台：上传注册编辑页面组件
  - 组件库：Namespace,行业化，旺铺
  - 组件实例的渲染结果以Dubbo和JSON
- 卖家使用装修平台装修旺铺
  - 卖家使用组件服务中心提供的旺铺组件的拖拽生成页面，并选择主题皮肤装修旺铺
- 第三方开发者
  - 开发并上传组件到模板市场，供卖家选择
- 运营 使用 CMS系统建站
  - 运营使用行业化组件库提供的组件服务所见即所得构建行业化站点 和推广专题站



展示层

高性能：组件缓存系统 WidgetCache



- 页面缓存的问题：
  - 缓存整个页面，而很多页面部分板块实时性要求很高，不适合全页缓存
  - 组件缓存可选择性的缓存页面部分区域
- 组件缓存系统
  - 缓存组件实例的结果html至KV
  - 加快页面响应时间，提升用户体验
  - 组件Cache Key生成
    - 根据组件Sid生成
  - 组件过期策略
    - 自动过期
    - 事件过期
      - 组件结构定义发生变更时
      - 组件实例对应数据变更时

展现层

前端优化：前端资源的版本控制，资源合并

- 背景：
  - 网站运营力度加大，
    - 秒杀，定时媒体投放广告 日益常态化，给系统造成巨大的压力
  - 用户对网站的页面响应速度等用户体验提出了更高的要求
- 前端优化
  - 请求过多影响加载速度
    - 出于CSS，JS最大限度重用的需要，页面引入的CSS,JS文件都切割的很小，导致页面引入的CSS/JS较多，产生较多的请求，影响加载速度
  - CSS/JS文件版本控制困难，无法实现动态回滚
    - 出问题需要重新发布，过程较慢
  - CSS/JS出于版本升级后即时生效的需要，没有走浏览器缓存
    - 出于版本升级后即时生效的需要，CSS,JS文件URL后带上时间戳参数，避免走浏览器缓存，每次都请求style服务器，导致页面加载慢，服务器压力较大
- 解决方案： 独角兽系统,管理CSS/JS的版本，自动合并CSS/JS。

前端优化：独角兽的设计

- UED上传CSS/JS到独角兽平台
- 独角兽管理网站CSS,JS文件的版本，保存各个版本的CSS/JS，以便回滚
  - CSS,JS文件和最新版本号 缓存中内存中。每次文件发布时更新索引
- 应用服务器保存一份CSS/JS文件版本号的索引，
  - 定时从独角兽服务器取最新的版本号
- 页面模板引入静态资源时，使用独角兽提供的宏声明需要合并的静态资源
  - #url('a.css,b.css,c.css')，请求文件名会转换成 a1.3-b1.2-c.1 .0.css
  - 独角兽服务器分析请求URL，将合并好的结果缓存，并返回给客户端，
  - 下一次同样请求走缓存。

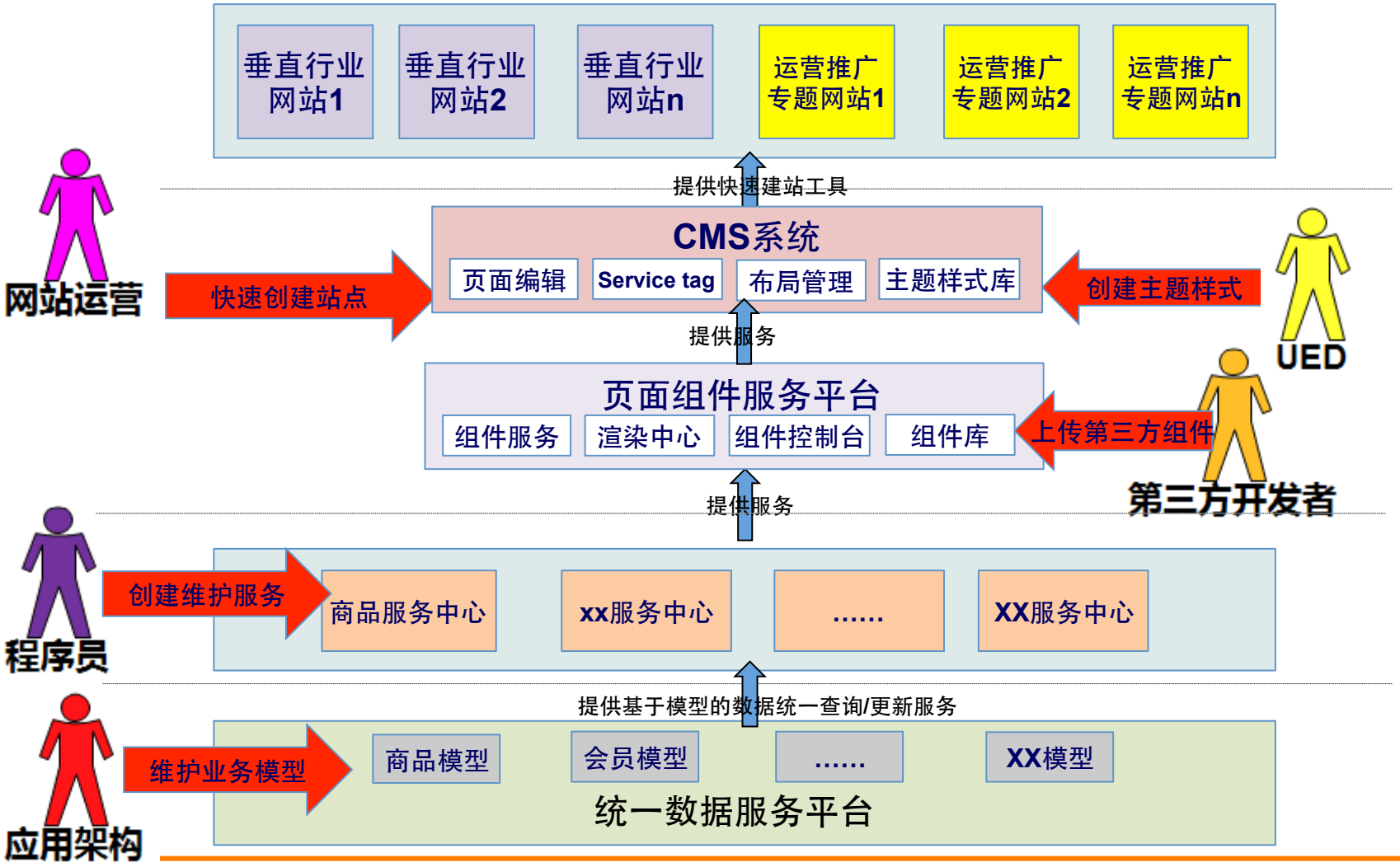
展现层架构改造回顾

问题	解决办法
页面版块不能重用 （eg，会员名片）	组件化： 1. **页面组件化**，将页面版块定义为可以接入到页面的组件（widget），板块提供定制参数，实现调整板块展示内容 2. **建立公共组件库**， 将可重用的组件保存在全站的公共组件库中，供全站各业务线应用页面调用
用户无法定义旺铺的版块，布局，皮肤	旺铺装修平台： 用户可以装修自己的旺铺的内容和展示风格 1.**定义页面**（Page）可支持哪些组件（Widget） 2. 定义页面的**布局**(layout)和**主题**（theme） 3.定义组件（Widget）可支持版块操作（view，edit）
网站运营无法快速搭建行业化网站和运用推广专题网站	ServiceTag，CMS，组件服务平台： 1.通过service tag标签获取页面数据，通过CMS系统编辑发布站点。 2.通过组件库提供的组件定制行业化站点 1）使用公共组件库板块提供定制参数，实现调整板块展示内容 2)开发人员通过组件服务平台上传，审核组件，管理组件库 3）使用公共组件库提供的版块拖拽生成页面，站点。
增强用户体验	前端优化：使用独角兽合并css，js请求，前端延迟加载框架等 加快页面响应速度，提升用户体验

总结

最终开发模式

所有人都在各自的领域里干着擅长的工作



总结

网站架构改造回顾

架构挑战	架构改造过程
敏捷	1. **数据层**：通过**UDSL**来实现隔离网站应用和各种底层数据源，使得复杂的底层数据架构对应用透明，大大的降低了了数据层开发和维护的成本 2. **业务层**：通过抽取核心业务服务，建设业务线**服务化中心**，实现应用的解耦，基于服务的开发大大加快了应用的开发效率。 3. **展现层**：通过开发 **Service tag**，**页面组件库**和基于组件平台的**CMS**等建站工具，使得网站运营人员可自行DIY行业化站点和运营推广站点，解决了开发资源瓶颈，加快了站点上线的周期.
开放	**1.Open API**: **服务化中心**提供的核心服务API 为网站Open API的基础，通过**Open平台**暴露给第三方开发者开发网站增值应用。 **2. 开放的店铺** 通过**旺铺装修平台**，卖家可以定义自己店铺的组件，布局，主题，装修店铺。 **3. 组件市场** 制定并开放统一的阿里巴巴中文站**页面组件**标准，组件接入规范，组件开发工具，第三方开发者可以开发页面组件，并上传至旺铺组件市场，给卖家装修自己的店铺
体验	1. 通过**热点缓存**，**异步并行加载框架**，大大加快了数据请求的时间。 2. 通过**页面组件缓存**，大幅加快了页面响应时间，增强了用户体验

Thanks

更进一步的问题?

Email: helin8@gmail.com

MSN: helintc@hotmail.com