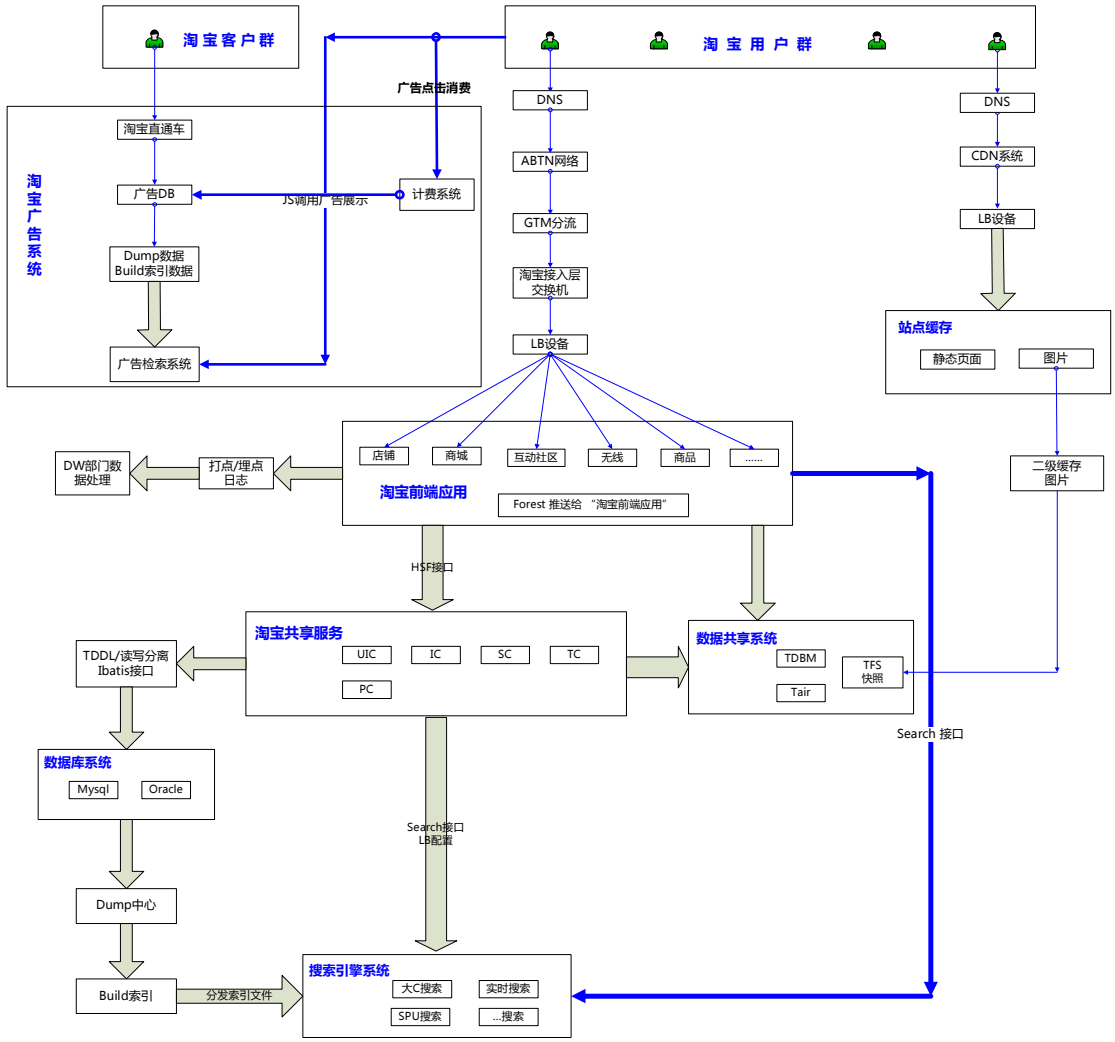
淘宝技术架构分享

中文网站技术部-B2C 商城 -- 郎中锋【花名:八神】

Email：zhongfeng.langzf@alibaba-inc.com

一，淘宝技术架构

1. 淘宝整体网络架构：



我介绍下图中提到的各个系统缩写是神马意思：

1.UIC: 用户中心(User Interface Center),提供所有用户信息相关的读写服务，如基本信息，扩展信息，社区信息，买卖家信用等级等等。淘宝现在有两类卖家 B 和 C，这是通过在用户身上打不同的标签实现的，我们这次的无名良品卖家也是通过在用户身上打特殊的标签来区别于淘宝已有的 B 和 C 类卖家。淘宝的 TOP 平台已经开放了大部分的 UIC 接口。

2.IC：商品中心(Item Center),提供所有商品信息的读写服务，比如新发商品，修改商品，删除商品，前后台读取商品相关信息等等，IC 是淘宝比较核心的服务模块，有专门的产品线负责这块内容，IC 相关接口在 TOP 中占的比重也比较大。

3.SC：店铺中心(Shop Center),类似中文站的旺铺，不过淘宝的 SC 不提供页面级应用，提供的都是些远程的服务化的接口,提供店铺相关信息的读写操作。 如：开通店铺，店铺首页，及 detail 页面店铺相关信息获取，如店内类目，主营，店铺名称，店铺级别：如普通，旺铺，拓展版，旗舰版等等。装修相关的业务是 SC 中占比重较大的一块，现在慢慢的独立为一个新的服务化中心 DC(design center),很多的前台应用已经通过直接使用 DC 提供的服务化接口直接去装修相关的信息。

4.TC：交易中心(Trade Center),提供从创建交易到确认收货的正 向交易流程服务，也提供从申请退款到退款完成的反向交易流程服务.

5.PC：促销中心(Promotion Center),提供促销产品的订购，续费，查询，使用相关的服务化接口，如：订购和使用旺铺，满就送，限时秒杀，相册，店铺统计工具等等。

6.Forest：淘宝类目体系：提供淘宝前后台类目的读写操作，以及前后台类目的关联操作。

7.Tair：淘宝的分布式缓存方案，和中文站的 Memcached 很像。其实也是对 memcached 的二次封装加入了淘宝的一些个性化需求。

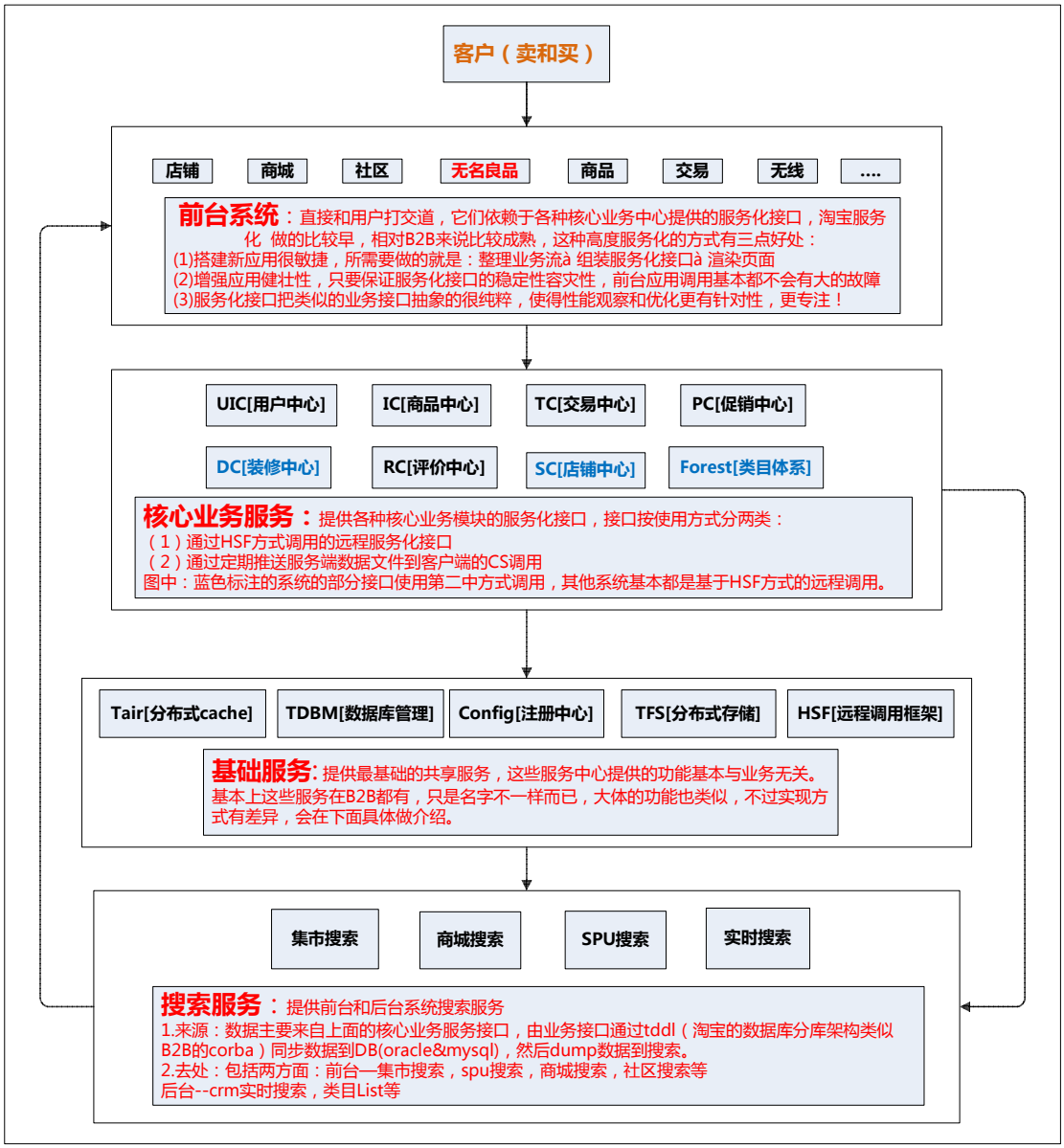
8.TFS：淘宝分布式文件存储方案(TB File System)，专门用户处理静态资源存储的方案，淘宝所有的静态资源，如图片，HTML 页面，文本文件，页面大段的文本内容如：产品描述，都是通过 TFS 存储的。具体设计和优缺点会在下面详细介绍。

9.TDBM：淘宝 DB 管理中心(TB DB Manager)，淘宝数据库管理中心，提供统一的数据读写操作，具体设计在下面详细介绍。

10.RC：评价中心(Rate center),提供评价相关信息的读写服务，如评价详情，DSR 评分等信息的写度服务。

11.HSF :淘宝的远程服务调用框架和平台的 Dubbo 功能类似 ,不过部署方式上有较大差异 ,所有的服务接口都通过对应的注册中心(config center) 获取。

2.淘宝服务化架构：



用户：淘宝用户分两类：买家和卖家，不过很多的卖家也会在淘宝买东西，所以他们既是卖家也是买家，因此最好是按照用户行为分：卖家行为和买家行为。买卖家行为都会涉及的系统包括：店铺，商城，交易，商品，社区等等。涉及到的功能有较大差异：

(1)卖家行为:主要定位在个人后台，且主要定位在后台的“我是卖家”TAB 页下的功能，包括：店铺管理，商品管理，订单管理，服务订购和使用，广告系统，社区发帖，淘宝客等等，前台浏览相对使用较少。

(2)买家行为:集中在前台：店铺浏览，宝贝的浏览，社区浏览等比重较大，买家后台功能主要定位在后台的“我是买家”Tab 页下，包括拍商品，付款，确认，退款，评价，社区互动等。

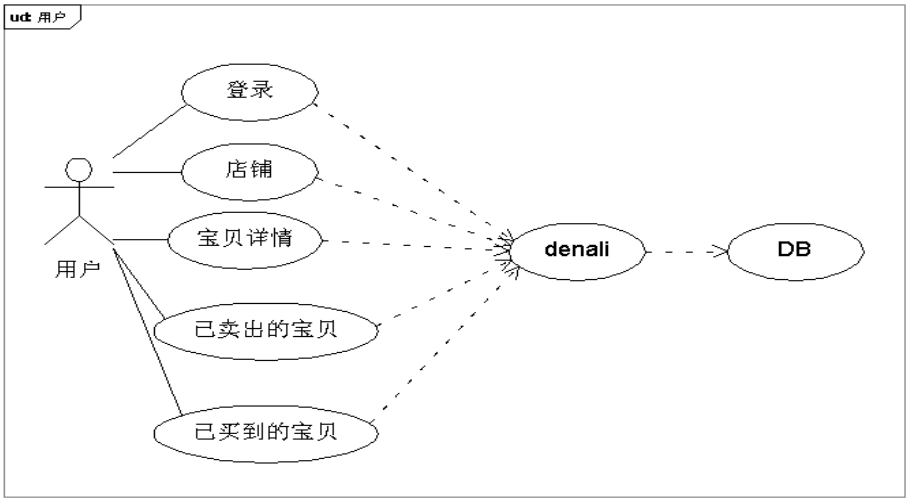
产品：淘宝对产品定义和 B2B 有差别，淘宝的业务拆分较细，服务化做的较成熟，所以前台应用对应的业务非常纯粹，如 Detail 系统可能就一个 detail 页面，无数据库连接，所有数据来自底层的各种服务化中心，功能专一逻辑清晰纯粹，不过也正因为这样，淘宝的一个产品可能是分布在多个应用系统中，单个应用很难称为一个产品。

淘宝更喜欢用产品线的概念，往往是一个团队负责一条产品线，跨几个功能纯粹的应用，如：商品线 交易线 店铺线 江湖线 商城线等。

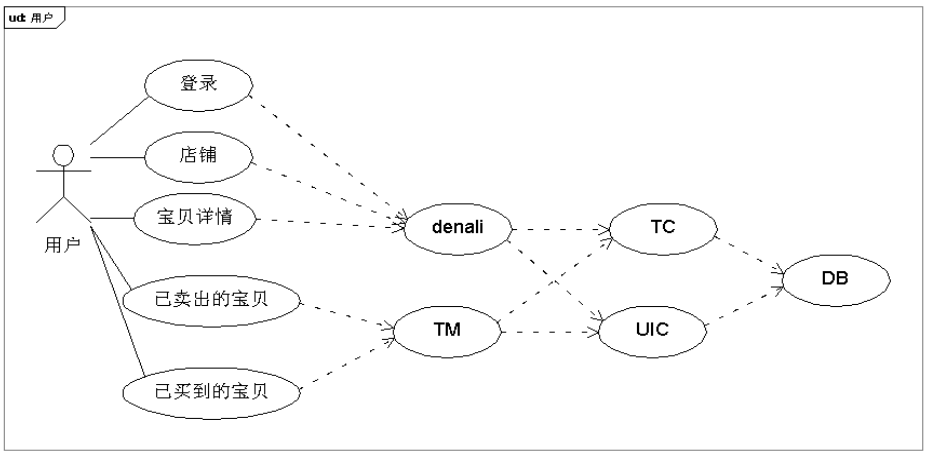
二.淘宝服务化进程

罗马非一日建成，上面看到的相对成熟的服务化中心也是在淘宝业在务发展过程中不断的遇到问题，解决问题的过程中沉淀下来的，下面给大家简单图示下淘宝的服务化进程：

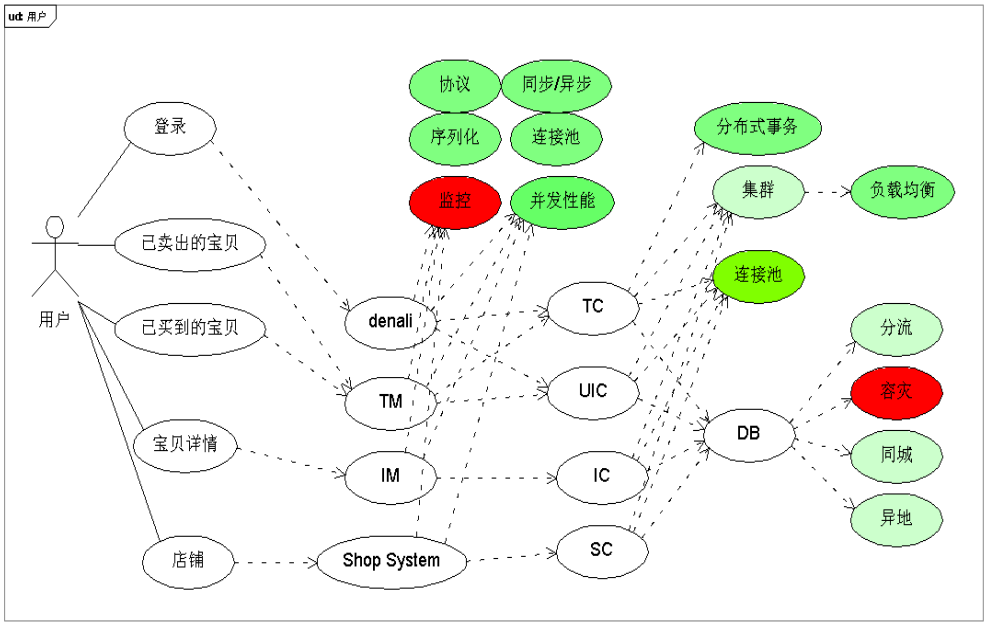
Ⅰ.最初的淘宝应用架构状况：业务高度耦合，不同的功能模块基本都糅合在一个系统中，错综复杂，很有 exodus2 当初的味道啊！



Ⅱ.随着业务发展，抽提部分核心业务之后的状态：对前台业务做了切分，抽提了部分核心业务成为独立的服务化提供中心。不过抽提的业务较少，这个阶段缺少对不同层的针对性优化，比如监控，容灾，事务等等



Ⅲ.大规模使用 HSF，服务化相对成熟之后的状况：前端应用高度分化，拆分成一个个功能单一的应用系统，后端服务化接口也划分的较细同时针对不同业务和具体的技术实现做个性化优化，增强了各个节点的健壮性，高效性。



通过上面的几张图可以明显的看出服务化给整个系统架构带来的好处：

(1) 应用间的耦合降低，在服务化接口成熟的情况下，扩展新的业务需求变得方便，主要就是一个组合服务化接口的过程。

(2) 开发人员可以专心关注业务，而不用考虑分布式领域中的各种细节技术，例如远程通讯、性能损耗、调用的透明化、同步/异步调用方式的实现等等问题。

(3) 各个系统负责的业务更纯粹，方便做针对新的性能调优和功能改进：如容灾性，并发，监控等方面的优化。这种针对性有助于增强系统的高性能，高健壮。

三，淘宝服务化利器--HSF

前台应用和核心业务层由于和业务结合比较紧密，篇幅限制这里就不展开讲了，后续会有针对各个细节业务和技术的分享，下面主要说说服务化中最重要的基础技术架构：HSF（淘宝远程服务调用框架），并附实例做说说它们的使用方式。

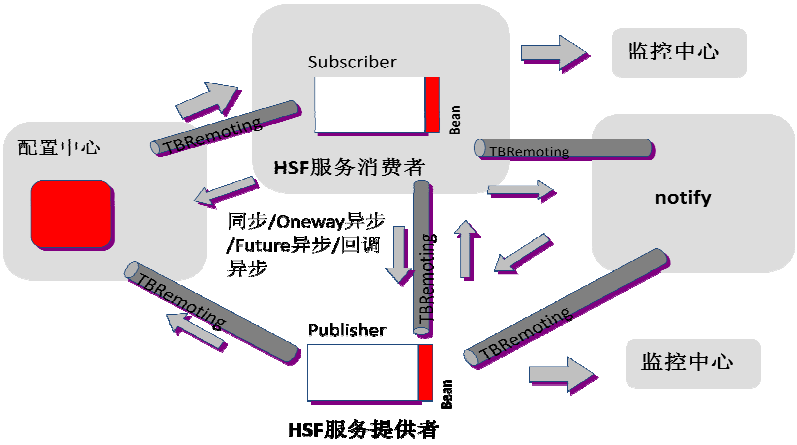
(1) HSF 是神马？

[High-Speed Service Framework] 淘宝的高速远程调用框架，类似中文站的 Dubbo，基于 Mina 框架开发，它是淘宝应用从**集中式**走入**大型分布式**的基础支撑。使开发人员无需过多的关注应用是集中式的，还是分布式的，可以更加专注于应用的业务需求的实现，这些纯技术的需求都由 HSF 来解决。

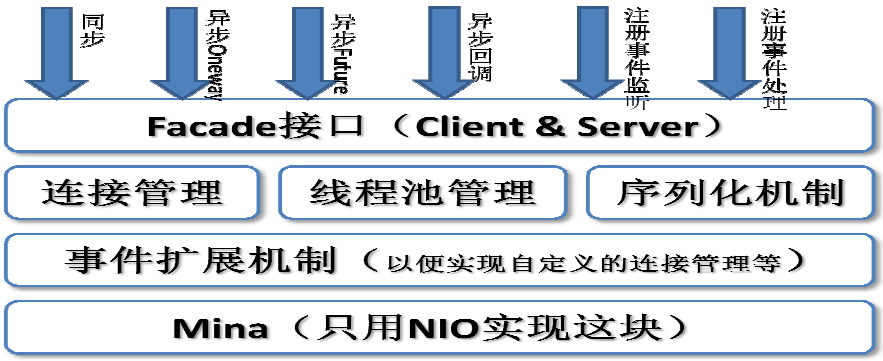
（2）HSF 的系统架构

I. HSF 交互场景图

客户端（消费端）从配置中心获取服务端地址列表—>和服务端建立连接开始远程调用—>服务端更新通过 notify（类似 B2B 的 naplio）系统通知客户端。服务端和客户端都有对应的监控中心，实时监控服务状态。客户端，配置中心，服务端，notify，之间的通信都是通过 TB Remotion API 去搞定的。



II. TB Remoting 架构图



底层基于分布式框架 Mina，主要的代码都是通过 NIO 实现的。关于 Mina 的具体实现大家可以 google，篇幅限制不细讲.

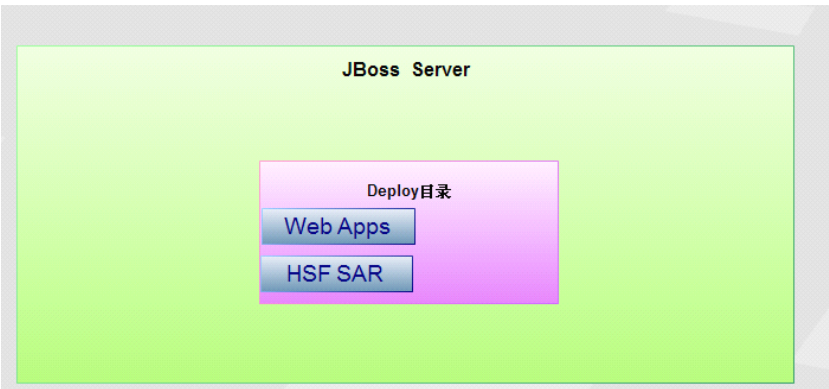
B2B 的 Dubbo 也是基于这个 NIO 框架的。Mina 上层封装的事件扩展，连接管理，线程池管理，序列化，包括最上面的 Façade 接口都是淘宝根据自身的需求做了一些封装，如果对这块感兴趣可以看看 HSF 的源码，需要的可以联系我！

（3）HSF 的工作原理：

淘宝的 HSF 在使用方式上面和 Dubbo 有比较大的区别，HSF 使用的时候需要单独的下载一个 hsf.sar 文件放置到 jboss 的 deploy 下面这样做的好处是：hsf 的升级不需要应用做改动，方便统一升级统一管理；弊端也很明显：增加了环境的复杂度，需要往 jboss 下扔 sar 文件，对 jboss 版本也有要求，这也是当初 Dubbo 没有完全参考 hsf 设计的主要原因。HSF 工作原理如下图：

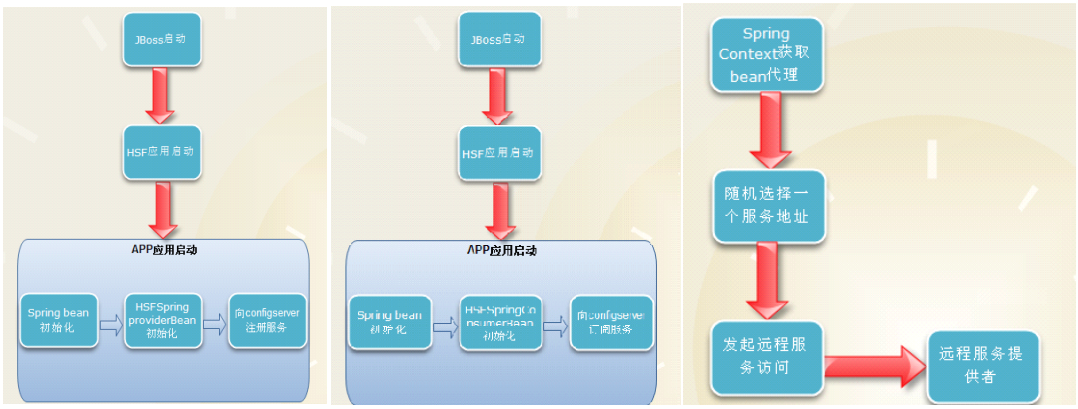
1.1. HSF 部署模型

下图 HSF 部署模型：使用之前，需要拷贝淘宝提供的 HSF SAR 文件到 Jboss 的 Deploy 目录。



*SAR 文件 download URL：*http://hsf.taobao.net/hsfversion/

1.2.HSF 发布和订阅服务和调用



1. JBoss 服务器启动后，启动HSF SAR 应用（即taobao.hsf.sar，本地开发机放在%DEPLOY_DIR%中，线上机器放置在/home/admin/${appName}/target下）

2. 应用自身启动，Spring 容器初始化。这时：

HSFSpringProviderBean: 会进行初始化，将向config server 注册当前这个bean 服务。这一注册过程，简单理解其实就是告诉config server：IP 为xxx.xxx.xxx.xxx 的机器提供了xxx 服务。这样，config server 就可以根据服务调用者的请求中的服务名来转发推送服务地址了。

HSFSpringConsumerBean: 会进行初始化，将向config server 订阅当前这个bean 服务，过程可以简单理解为：告诉config server :IP 为xxx.xxx.xxx.xxx 的需要提供xxx 服务，Config server 就会根据这一服务名返回给应用相应的服务地址。

3. 在HSF 服务调用者订阅到服务地址后，就可以使用该服务地址执行服务调用了。一个HSF 服务通常并不是由一台机器提供的，所以，订阅到的服务地址通常是一个地址列表，里面包含了所有提供了该服务的地址。HSF 会随机选择一个服务地址进行服务调用。如上图所示。

（4）HSF 使用实例代码：

这也是和我们最相关的，代码的方式和Dubbo大同小异，主要是些spring服务的申明和注入。

Ⅰ. 服务端：发布服务代码：

```
<bean   class="com.taobao.hsf.app.spring.util.HSFSpringProviderBean"   init-method= “init” >
        <property name="serviceInterface “>
            <value>com.taobao.hsf.test.HelloWorld</value>
        </property>
        <property name="target">
            <ref bean="HelloWorld"/>
        </property>
</bean>
```

Ⅱ. 客户端：订阅服务代码：

```
<bean id="OnewayConsumerBean" class="com.taobao.hsf.app.spring.util.HSFSpringConsumerBean" init-method="init">
        <property name="interfaceName “>
                <value> com.taobao.hsf.test.HelloWorld</value>
        </property>
        <property name="version">
                <value>1.0.0</value>
        </property>
        <property name="target">
                <value>socket://192.168.207.91:4446</value>
        </property>
</bean>
```

红色的属性是单元测试特有的，应用启动状态下不需要配置这块内容，因为这个 target 是从 config（注册中心）获取的。

上面的代码就不做具体说明了，不复杂，用过 hession 和 dubbo 的应该都可以看懂。

一个多月的淘宝感受：

1. 淘宝的服务化做的较早，现在已比较成熟，系统功能单纯，比如我们这次迁移了淘宝的 Detail 页面，整个应用就一个页面，其余的都是对服务化接口的调用，无任何数据库连接，只要服务化接口都调通了，开发起来很快很给力！
2. 文档较全，貌似淘宝的开发都比较擅长写文档，估计是受了写 UC 能力的影响（淘宝的 UC 全部由开发编写），文档带来很多好处，特别是服务化比较成熟之后，通过 API 文档可以减少很多沟通成本,同时也是给新人熟悉产品和技术提供了很大帮助。
3. 流程具有针对性（淘宝的 SCM 很有趣，他们的原则是让那些平时规范和流程遵守较好的人拥有更大的权力，区别对待！），一般产品和技术较熟悉的，需求流程基本都由他们自己把控。这种弹性的流程控制确实为我们项目节省了不少时间。后续会有专门的淘宝流程分享，这里就不细说了。
4. 也有感觉不太给力的：

   I.开发环境还停留在 B2B 几年前的水平，本地开发还需要手动从 jboss 启动，模板的配置方式也不太好，如果要所见即所得，还需要借助于 Eclipse 的同步插件，杯具！

   II.无法绑定到正式环境的某一台服务器做测试，测试单台机器必须通过 beta 发布方式，线上验证和排查很不方便。

   III.线上环境无法 ssh 和 scp 到测试和本地环境，download 预发和正式环境的文件异常复杂。

资料参考：

淘宝核心业务团队博客：http://rdc.taobao.com/blog/cs/

HSF 百科：http://baike.corp.taobao.com/index.php/HSF

B2C 商城团队黄花会 wiki：：http://share.b2b.alibaba-inc.com/index.php/B2C%E5%95%86%E5%9F%8E%E5%88%86%E4%BA%AB