**基础技术体系规划**

**2011年**

目录

[1. 目的和要求 5](#_Toc309395493)

[1.1 名词解释 5](#_Toc309395494)

[1.1.1 敏捷 5](#_Toc309395495)

[1.1.2 轻量级 6](#_Toc309395496)

[1.1.3 SOC 6](#_Toc309395497)

[1.1.4 SOA 7](#_Toc309395498)

[1.1.5 CAP 7](#_Toc309395499)

[1.1.6 有状态和无状态 7](#_Toc309395500)

[1.1.7 四层负载均衡 7](#_Toc309395501)

[1.1.8 七层负载均衡 7](#_Toc309395502)

[1.2 开发敏捷性要求 8](#_Toc309395503)

[1.3 运维高效性要求 8](#_Toc309395504)

[1.4 核心思路 8](#_Toc309395505)

[1.5 承接下文 9](#_Toc309395506)

[2. 基础技术体系的构建思路 9](#_Toc309395507)

[2.1 关于三层体系结构 9](#_Toc309395508)

[2.2 业务场景分解 9](#_Toc309395509)

[2.2.1 三层体系结构的业务应用 9](#_Toc309395510)

[2.2.2 三层体系结构中的业务解读 10](#_Toc309395511)

[2.2.3 抽象的解读三层体系结构 11](#_Toc309395512)

[2.3 “状态”对于部署视图的意义 12](#_Toc309395513)

[2.4 承接下文 13](#_Toc309395514)

[3. 基础技术体系蓝图 13](#_Toc309395515)

[3.1 基础技术体系的意义 13](#_Toc309395516)

[3.1.1 提供基于公共抽象的业务能力 13](#_Toc309395517)

[3.2 分层 15](#_Toc309395518)

[3.3 分层图示 16](#_Toc309395519)

[3.4 说明 16](#_Toc309395520)

[3.4.1 基础服务 16](#_Toc309395521)

[3.4.2 基础产品 16](#_Toc309395522)

[3.4.3 基础框架 17](#_Toc309395523)

[3.4.4 公共管理服务 17](#_Toc309395524)

[4. 基础服务规划 18](#_Toc309395525)

[4.1 蓝图 18](#_Toc309395526)

[4.1.1 图示 18](#_Toc309395527)

[4.1.2 子系统说明 19](#_Toc309395528)

[4.2 跨机房部署 19](#_Toc309395529)

[4.2.1 可静态化的跨机房能力 19](#_Toc309395530)

[4.2.2 动态化的跨机房能力 21](#_Toc309395531)

[4.3 分组及分域 25](#_Toc309395532)

[4.3.1 分组及分域的要求 25](#_Toc309395533)

[4.3.2 访问安全性 26](#_Toc309395534)

[4.3.3 数据互斥性 26](#_Toc309395535)

[4.3.4 数据编码约束 28](#_Toc309395536)

[4.4 接口能力 30](#_Toc309395537)

[4.5 平滑扩展能力 31](#_Toc309395538)

[4.5.1 分布式文件、键值系统等横向扩展 31](#_Toc309395539)

[4.5.2 Mysql集群横向扩展 31](#_Toc309395540)

[5. 基础产品规划 34](#_Toc309395541)

[5.1 蓝图 34](#_Toc309395542)

[5.2 负载均衡能力 34](#_Toc309395543)

[6. 公共管理服务规划 35](#_Toc309395544)

[6.1 蓝图 35](#_Toc309395545)

[6.2 子系统边界 35](#_Toc309395546)

[6.2.1 Boss系统 35](#_Toc309395547)

[6.2.2 配置管理系统 35](#_Toc309395548)

[6.2.3 OA系统 36](#_Toc309395549)

[6.2.4 版本管理系统 36](#_Toc309395550)

[6.2.5 监控系统 36](#_Toc309395551)

[6.3 子系统交互示例 36](#_Toc309395552)

[6.3.1 开发阶段 36](#_Toc309395553)

[6.3.2 测试阶段 37](#_Toc309395554)

[6.3.3 上线阶段 38](#_Toc309395555)

[6.3.4 系统监控 39](#_Toc309395556)

[6.3.5 与Boss系统交互能力 40](#_Toc309395557)

[7. 基础框架规划 40](#_Toc309395558)

[7.1 iCommon 40](#_Toc309395559)

[7.1.1 Common.conf 41](#_Toc309395560)

[7.1.2 Common.plugin 41](#_Toc309395561)

[7.1.3 Common.dm 41](#_Toc309395562)

[7.1.4 Common.remoting 41](#_Toc309395563)

[7.1.5 Common.workflow 41](#_Toc309395564)

[7.2 Noah 41](#_Toc309395565)

[8. 部分子系统概述 41](#_Toc309395566)

[8.1 基础服务部分 41](#_Toc309395567)

[8.1.1 分布式文件系统 41](#_Toc309395568)

[8.1.2 MySql集群 43](#_Toc309395569)

[8.1.3 Map/reduce 及 分布式计算 43](#_Toc309395570)

[8.1.4 NOSQL数据库 44](#_Toc309395571)

[8.1.5 通用队列服务 44](#_Toc309395572)

[8.1.6 通用缓存服务 44](#_Toc309395573)

[8.2 基础产品部分 44](#_Toc309395574)

[8.3 公共管理服务部分 45](#_Toc309395575)

[8.3.1 Boss系统 45](#_Toc309395576)

[8.3.2 配置管理系统 49](#_Toc309395577)

[8.3.3 OA系统 50](#_Toc309395578)

[8.3.4 监控系统 51](#_Toc309395579)

[9. 实施计划 52](#_Toc309395580)

[10. 遗留问题 52](#_Toc309395581)

**本文档修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **修订时间** | **修订人** | **修订内容** |
| V1.0 | 2011-10 | 金明岩、李毅 | 初稿 |
| V1.1 | 2011-11-18 | 金明岩 | 根据技术部管理人员意见修改和增加部分章节 |
|  |  |  |  |

# 目的和要求

本文在对凤凰新媒体现有的IT技术体系进行梳理的基础上，明确一个中远期的基础技术体系规划，进而保障长期技术演进符合整体业务发展的需要。

在明确规划的前提下，规避开发行为中不符合架构趋势的方向性错误，减少重复性的开发工作；明确软件系统的分类和最终部署关系；明确软件系统的边界和相关接口方式，进而建立一个规范统一的技术体系。

## 名词解释

此章节不必阅读，在下文中遇到相关名词时可以链接到这里来阅读。

### 敏捷

全称为**敏捷软件开发**或**敏捷开发**，是从1990年代开始逐渐引起广泛关注的一些新型软件开发方法，是一种应对快速变化的需求的软件开发能力。

它们的具体名称、理念、过程、术语都不尽相同，但相对于“非敏捷”，更强调程序员团队与业务专家之间的紧密协作、面对面的沟通（认为比书面的文档更有效）、频繁交付新的软件版本、紧凑而自我组织型的团队、能够很好地适应需求变化的代码编写和团队组织方法，也更注重软件开发活动中人的作用。

**原则（宣言）：**

对我们而言，最重要的是通过尽早和不断交付有价值的软件满足客户需要。

我们欢迎需求的变化，即使在开发后期。敏捷过程能够驾驭变化，保持客户的竞争优势。

经常交付可以工作的软件，从几星期到几个月，时间尺度越短越好。

业务人员和开发者应该在整个项目过程中始终朝夕在一起工作。

围绕斗志高昂的人进行软件开发，给开发者提供适宜的环境，满足他们的需要，并相信他们能够完成任务。

在开发小组中最有效率也最有效果的信息传达方式是面对面的交谈。

可以“工作”的软件是进度的主要度量标准。

敏捷过程提倡可持续开发。出资人、开发人员和用户应该总是维持不变的节奏。

对卓越技术与良好设计的不断追求将有助于提高敏捷性。

简单——尽可能减少工作量的艺术至关重要。

最好的架构、需求和设计都源自自我组织的团队。

每隔一定时间，团队都要总结如何更有效率，然后相应地调整自己的行为。

### 轻量级

全称：“**轻量级开发**”，通常与一套开发方法、框架和设计原理一起使用。

轻量级方法： 包括敏捷过程，例如极限编程（XP）和 Scrum。它们强调开发中测试第一，积极调动客户和重构。

轻量级框架： 鼓励人们使用简单原始的 Java 对象（POJO）编程，而不是类似 EJB 的重量级面向组件模型。

轻量级设计模式： 使您可以在对象和集成服务之间进行松散耦合，而无需艰苦地编写业务逻辑或领域模型。

**原则：**

争取简单性。这种观念应该渗入到您所有的工作中。您的过程应该刚好生成足够完成工作的工件。开发人员应该尽量使用最简单的方法解决问题。您的工具应该使您构建一个清晰、简洁的解决方案。

修补漏洞。许多开发方法可能不鼓励在过程中进行重构或变更，因为这些行为不直接用于产生客户代码。轻量级开发要求可以自由地修补太复杂或充斥 bug 的代码。您需要为它做出预算。

自动化单元测试。您应该优先编写测试用例。您可能还没有在测试第一的开发中成功过，但测试会间接给您带来重构代码的便利。广泛的单元测试改善您的客户体验，并提高代码的设计水平，这是因为它强迫您解耦联系过于紧密的代码。

使用短开发周期并积极调动客户参与其中。许多顶级的软件工作室通过剔除不必要的工件来简化开发周期。如果您已经顺利得到客户的参与，那么很多的功能规范会变得越来越没必要。客户会很满意这种交互，并感激您的短周期开发，因为这稳步提高了客户的业务价值。

技术：

1 依赖注入

依赖注入: 将对象和它所依赖的东西交给第三方。然后第三方创建所有对象并将它们绑在一起。

2 面向对象编程

3 透明持久性

持久性也是建立在较简单的编程模型之上。透明持久性框架通过配置而不是编写代码，来使您为应用程序添加持久性。

### SOC

关注点分离，是软件设计的普遍原则。

将一个复杂的问题，变成若干个简单问题的组合能力，使得程序开发过程只是面对简单的问题，从而使得问题的思考和描述落入更多人的能力范围内。

关注分离是一个非常泛化的原则，不只是针对程序设计方面，对人员分工同样有效。

### SOA

SOA，Service Oriented Architecture，面向服务的架构。从业务角度理解SOA，它是针对企业的一些旧的软件体系重新利用，进行整合，构建一套松散耦合的软件系统，同时也能方便的结合新的软件共同服务于企业的一个体系。使系统能够随着业务的变化更加灵活适用。

### CAP

来自Eric A. Brewer的观点

Consistency（一致性）：即数据一致性。

Availability（可用性）：好的响应性能。

Partition tolerance（分区容错性）：可简单的理解为是节点的可扩展性。

数据存储要求的基本属性。

定理：任何分布式系统只可同时满足二点，没法三者兼顾。

忠告：架构师不要将精力浪费在如何设计能满足三者的完美分布式系统，而是应该进行取舍。

### 有状态和无状态

有状态和无状态(Stateful and Stateless)

有状态就是有数据存储功能。有状态对象(Stateful Bean)，就是有实例变量的对象，可以保存数据，是非线程安全的。在不同方法调用间不保留任何状态。

无状态就是一次操作，不能保存数据。无状态对象(Stateless Bean)，就是没有实例变量的对象.不能保存数据，是不变类，是线程安全的。

### 四层负载均衡

OSI模型的第四层是传输层。传输层负责端对端通信，即在网络源和目标系统之间协调通信。在IP协议栈中这是TCP（传输控制协议）和UDP（用户数据报协议）所在的协议层。

第四层交换的一个简单定义是：它是一种功能，它决定传输不仅仅依据MAC地址（第二层网桥）或源/目标IP地址（第三层路由），而且依据TCP/UDP（第四层）应用端口号，指向物理服务器。

在第四层交换中为每个供搜寻使用的服务器组设立虚拟IP地址（VIP），每组服务器支持某种应用。

当某用户申请应用时，一个带有目标服务器组的VIP连接请求（例如一个TCP SYN包）发给交换服务。交换服务在组中选取最好的服务器，将终端地址中的VIP用实际服务器的IP取代，并将连接请求传给服务器。

### 七层负载均衡

七层及OSI的最高层，应用层。四层是基于IP+端口的负载均衡；七层是基于URL等应用层信息的负载均衡；七层通过虚拟的URL或主机名接收请求，然后再分配到真实的服务器。

## 开发敏捷性要求

互联网业务和用户规模在不断的发展，其发展速度有时已经超过了产品经理和架构师的预期，为了保障业务系统的功能、性能以及可靠性能够满足不断变化和增加的用户需求，在软件设计和开发活动中，越来越多“[敏捷](#_敏捷)”、“[轻量级](#_轻量级)”等高效快速提交结果的开发方法被引入产品开发的过程。这些思路在需要高效协同的开发组织、完善且不断改进的开发流程的同时，也对基础技术体系，尤其是基础架构提出了更高的要求。

如何通过架构的合理性以及基础产品体系的完善度提供一个便于高效开发的场景，成为了越来越多互联网公司系统设计人员的主要工作任务。

提供高效的技术架构，可以使得单一业务系统的结构更“单纯”，即只需要考虑某些方面（Aspect）的问题，减少设计过程中设计人员所面对场景的思维复杂度；另外使得设计过程可以充分忠实于[SOC](#_SOC)（关注点分离）的软件设计原则；同时技术架构可以将一些公共的业务抽象为线上已有的服务，采用[SOA](#_SOA)的方式构建线上的服务平台，使得业务系统的开发可以直接使用这些能力。

## 运维高效性要求

随着业务和用户规模的快速扩展，面对大量的服务器的运维场景，将超出纯粹人工维护可以接受的工作量要求以及复杂度要求。

面对这些可能的困难，运维活动要求统一的系统架构和技术架构应该具有以下的属性：

1 服务器有效的分类。

2 每个分类的服务器所需要的运行环境要尽量的单纯。

3 有效的负载均衡无单点故障的调度体系。

4 几乎全部的故障都可以自动完成故障切换，运维人员只需要在事后修复硬件故障。

5 自动的安装部署体系，无需频繁的登录服务器，而是通过配置管理软件来完成系统的安装、维护和升级。

6 有效的监控体系，让故障自动呈现，无需定期关注和检查各种服务。

以上的这些要求，无论是靠行政指令，管理流程，还是软件服务去解决，都需要在技术架构的角度给予有效的规划和支持，明确各种类型的行为所负责的边界。

## 核心思路

1. 业务逻辑、基础服务能力、业务数据存储分离。业务系统重点关注于实现业务逻辑，实现的过程中可以依赖基础服务系统提供的能力从而有效的降低业务系统的复杂度；业务数据存储的[CAP](#_CAP)(Consistency、Aviailability、Partition tolerance)要求由专门的基础存储服务实现。

2. 数据存储能力支持无缝的分布式部署能力，为业务系统的分布式部署提供可能的支持方式。

3. 系统、服务及其依赖的设备可以比较容易的线性扩容或收缩，一般情况下性能提升靠增加业务系统、底层服务的服务器数量和部署规模即可实现。

4. 提高设备的通用，各种不同的业务系统使用的设备尽量统一。具体到架构中，应该做到对应一个系统组件部署的场景，其依赖的运行环境应该有统一的分类；其硬件要求的方面应该单纯，即单纯需要高计算性能、单纯需要大存储能力等。

5. 所有的“部署”都可以比较容易的被管理和监控。

## 承接下文

后续的章节，将以本章节的目的和要求为导向，逐渐描述和细化基础技术体系。

# 基础技术体系的构建思路

## 关于三层体系结构

三层架构(3-tier application) ：通常意义上的三层架构就是将整个业务应用划分为：表现层（UI）、业务逻辑层（BLL）、数据访问层（DAL）。区分层次的目的是为了完成“高内聚，低耦合”的设计思想。

　　1. 表现层（UI）：通俗讲就是展现给用户的界面，即用户在使用一个系统时的所见所得。

　　2. 业务逻辑层（BLL）：针对具体问题的操作，也可以狭义的理解为是对数据层的操作和对数据业务逻辑的处理。

3. 数据访问层（DAL）：直接操作数据存储。

## 业务场景分解

### 三层体系结构的业务应用

面对互联网公司需要的应用系统，几乎所有的系统都可以“套用”三层体系结构给予一个“抽象”的描述。

即所有的业务场景均可抽象如下图：



***2.2 附图1 - 三层体系结构***

### 三层体系结构中的业务解读

其中每个层次可以被描述（元数据化）为：

***2.2 附图2 - 三层体系结构解读***

可以看到：

表现层中的UI主要通过某种接口能力（Html,Xml or 其他调用方式）呈现给体验者，而为了完成某种动态呈现能力时，则需要一些code给予支持。

业务逻辑层中着重对表现层提供足够的接口能力（可能是本地的或者远程的），这些接口能力通常要通过“配置”和Code来共同完成。

数据访问层着重为上层提供足够的数据存储和相关访问能力的接口，这些数据可能包括文件、单索引数据（Key/Value）,关系数据、有序或无序的结构化数据等。

### 抽象的解读三层体系结构

我们借用来自EJB的“[有状态和无状态](#_有状态和无状态)”来对三层体系结构做一个解读。

Code:在数据和业务分离的前提下，所有的代码都可以是无状态的。

* 不要泛化理解数据和业务分离，这里的数据并不排斥使用缓存数据、临时数据等，但需要业务代码本身将“权威”的数据与缓存数据和临时数据分开考虑。
* 代码如何在存储数据的同时保持无状态性，取决于业务系统的自我选择。即业务系统需要在不进行持久化的同时，保持自己的“无状态”性不被破坏。

Data:数据本身是状态的表现，是业务能力“状态性”的核心。

表现方式（html,xml,js）：所有的表现方式都可以抽象为“表现数据”（如.html文件），并采用发布的方式发布为具体的部署。

**所有的表现层都可以用一个无状态对象来描述，即表现层可以通过发布（将表现层数据发布出来）和代码的组合来实现灵活的扩展部署能力。**

**所有的业务逻辑层都可以组件化、无状态化，即业务逻辑本身的实现均由代码来构成，数据和配置数据均交由数据层提供。**

**数据层是状态的维护者，是使得每个“用户体验”具有不同能力的状态轴心。**

## “状态”对于部署视图的意义

所有无状态的组件，均可以天然利用[四层负载均衡](#_四层负载均衡) /[七层负载均衡](#_七层负载均衡)能力，构建一个节点间无关的可伸缩部署模型。

***2.3 附图1 - 无状态部署***

有状态部署能力则需要维护状态的分配和一致性，需要独立的负载均衡能力来完成任务调度和数据一致性维护，其伸缩模型受限于负载均衡能力的需要。

***2.3 附图2 - 有状态部署***

## 承接下文

下文将依赖本章节对有状态和无状态的解读，映射我们的业务需要，开始定义我们的服务层次。

# 基础技术体系蓝图

本章节将针对技术体系做一个明确的分类，进而得到软件产品和项目的分类，明确其所负责的边界，明确不同分类之间的接口和调用方式。

## 基础技术体系的意义

### 提供基于公共抽象的业务能力

为了有效的降低开发一个业务需要的系统的复杂度，从“[敏捷](#_敏捷)”的角度理解，我们往往需要：

1. **降低编码难度。**
2. **降低编码的数量。**

**以上两个原则，被软件工程学科用“抽象”来解释，即将表现为不同业务逻辑的功能，理解为一样或者近似的能力，通过特有的不同方面的描述（配置，子类）来实现不同性，以便做到相同的特性用相同的代码或组件来实现。**

我们将抽象的应用场景更广泛化，把抽象应用于系统角度。

我们将一个业务系统“强制解释”（这里可以理解为泛化）为一个三层体系结构，即都划分为：

1 表现层。

2 业务逻辑层。

3 数据访问层。



***复制 2.2 附图1 - 三层体系结构***

其中我们将数据访问层，分离为代码和存储两个逻辑 或 部署部分，将代码部分抽象进入业务逻辑层的数据接口部分，将存储部分抽象定义为狭义的数据访问层，并将这部分数据访问层抽象为基础服务，即提供基础数据存储能力的最底层服务，通过一个基础服务的接口能力，与业务逻辑层的数据接口部分“粘合”。

同时，我们将业务逻辑层中的一些公共的业务能力，定义在一些公共的业务组件中，并将这些组件部署为一个线上服务，提供这些服务的统一负载均衡能力，使其从业务系统中独立出来，成为一个独立的部署，这些部署完成某一种特有的业务能力，我们将其称为基础产品。



***3.1.1 附图1 层次抽象***

## 分层

现在我们抛弃掉三层体系结构的约束，面向我们的开发行为，我们将我们开发活动对应的产品，面向部署所在位置和依赖关系，分为如下几种：

1 基础服务。

2 基础产品。

3 公共管理服务

4 业务系统。

5 基础框架。

## 分层图示

***3.3 附图1 - 技术体系蓝图***

## 说明

在上图中，我们明确了以下几种类别的软件服务及其部署能力：

### 基础服务

基础服务是指提供基础（最底层）存储能力的服务平台，其本身是一个部署态应用服务，但一般不提供某种业务能力，而是为完成某种应用提供某一个层次和某一方面问题的解决能力（目前狭义的定义为存储能力）。基础服务及组件是提高开发效率的最有效手段，它可以让开发一个业务产品时不需要，或者很少的需要考虑某一个层次或方面的问题，使得业务系统能更关注本身的业务实现。

目前在互联网应用中，数据和内容的存储是迫切需要的基础服务，在下面的章节中将针对基础服务展开，明确基础服务所包含的内容。

针对数据的基础服务的存在大大的降低了部署角度的业务系统开发难度，使得业务系统的设计模式应用更加灵活。

### 基础产品

基础产品是指一些公共的业务产品，以业务服务的形式部署在线上，供需要此能力的其他业务系统调用。

基础产品本身将被部署为一个服务（Service），该服务提供某种业务能力，业务系统可以使用此服务来完成业务系统的相关功能。

基础产品着重降低业务实现的编码量，将公共的业务直接以服务形式开发给业务系统使用，业务系统可以直接使用基础产品完成某一个用例中步骤或过程。

### 基础框架

基础框架是一些基础开发工具，本身是一个编码级的框架，在业务系统的设计和编码阶段引入业务系统中，与业务系统的程序一并部署并完成业务系统的业务能力，主要的目的是降低编码的复杂度和结构复杂性。

基础框架针对的并非一个部署场景，只是会影响开发活动和一个软件产品本身的结构。

基础框架有可能在所有的部署层次（基础服务、基础产品、业务系统）中使用。

### 公共管理服务

公共管理服务是指一些面向内部管理和维护的服务能力，这些服务提供了统计分析、监控、管理等相关能力，一般被内部的业务和运维人员使用，用来了解系统运行的情况和各种数据，用以决策和分析。

公共管理服务将提供一个全网管理的平台能力，所有的软件服务都需要向公共管理服务负责，进而保障软件服务本身可部署、可统计、可监控、可管理。

公共管理服务是一种“特殊的”业务系统，其逻辑上与业务系统在同样的位置，可以在开发的过程中使用基础服务、基础产品的业务能力，但本身在用例表现中实际处于一个管理位置，需要与各个层次的“部署”交互，完成各种管理行为。

# 基础服务规划

## 蓝图

### 图示

***4.1.1 附图1 基础服务蓝图***

### 子系统说明

基础服务着力于提供各种存储方面的能力。其中：

#### 分布式文件系统

主要用于存储海量文件，业务系统可以在分布式文件系统存储用户数据文件，也可以存储自身需要分布式能力的配置文件。

#### MySql集群

主要用于所有需要关系数据库存储数据的场景，将提供一个统一的数据库集群，业务系统将不再需要安装独立的数据库，而是用数据库集群提供的数据存储能力。

#### 分布式计算服务

提供公共的Map/Reduce能力，业务系统可以使用此能力完成需要的分布式计算而不需要独立安装和部署类似的服务。

#### Key/Value数据库

提供统一的Key/Value存储及相关域管理能力，完成业务系统需要的相应存储能力。

#### 缓存服务

提供一个有结构数据的缓存能力，以便业务系统可以管理一个分布式缓存，使得部署于不同服务器上的业务系统可以使用一个公共的缓存服务来完成相互之间的数据交换，提供一个类数据总线能力。

#### 队列服务

提供一个公共队列能力，使用put get的接口方式完成不同组件或不同服务之间的数据通讯和传递能力。

## 跨机房部署

因为基础服务中存储了业务系统的数据，故其需要考虑业务系统部署的变化，进而提供适应这种变化的部署能力。

### 可静态化的跨机房能力

对于静态服务可以考虑直接使用CDN系统或者类CDN系统的部署模式,利用其七层负载均衡能力实现用户体验的最佳化。

如下图所示：



***4.2.1 附图1 跨机房部署静态方式访问示例***

步骤如下：

1 用户访问首先到达本地的DNS.

2 如果采用DNS负载均衡,则到达CDN系统中的动态DNS服务，动态DNS根据用户的源DNS确认其位置，并将解析后的距离用户最近的Node对应的Cache – VIP地址.

如果采用IPSERVER完成负载均衡，则用户直接会访问ipserver，然后被重定向到距离用户最近的Cache-VIP地址。

3 用户访问对应的Cache服务。

4 Cache本地存在对应的数据时，则直接为用户提供服务，否则反向代理到父节点。

5父节点存在数据则直接为用户提供服务，否则反向代理到源的动态服务。

6 源动态服务从基础服务中取得数据，然后返回给父节点。

适用于静态内容的场景常见于较大数据量的用户访问行为，如访问一个图片、视频等。这些动态服务本身由于最终返回的链接可以转化为静态的，所以可以采用CDN完成用户请求的负载均衡，动态服务本身的压力可控，所以可以部署在一个中心的互通性较好的机房中，然后由中心机房的一套基础服务为其提供数据存储服务。

### 动态化的跨机房能力

业务系统另外一种常见的部署场景由于要考虑其对应服务用户的网络体验，将业务系统的动态服务部署于距离用户较近的位置。由于这种服务提供的大多是一些动态能力，比如社交服务、用户校验、数据校验等，这些能力无法静态化运行在CDN中，所以就需要将这些动态服务部署在距离用户较近的机房中。

如下图：



***4.2.2 附图1 跨机房部署动态方式业务系统访问示例***

在上述场景中，如果动态服务采用基础服务提供的数据存储能力，在数据量和访问量不大的情况下，也可以将基础服务部署在中心的一个互通性较好的机房中，提供统一的存储能力。

如下图：



***3.2.2 附图2 跨机房部署动态方式中心服务能力***

在数据量或者访问量较大时，动态服务本身的体验要求需要基础服务距离自己的位置更近，这时就要求使用“本地化”的基础数据存储为用户提供可靠的服务质量。这就需要基础服务考虑跨机房部署问题。

 4***.2.2 附图3 跨机房部署动态方式***

我们在基础服务跨机房时，引入针对所有基础服务类型的统一的负载均衡服务。

针对数据负载均衡和复制的问题，我们将其分为两个场景。

以分布式文件系统为例。分布式文件系统本身提供一个强一致性的负载均衡及数据复制体验：



***4.2.2 附图4 基础服务内部数据复制（IFS）示例***

负载均衡服务在现有的分布式文件系统本地强一致性的前提下，提供一个跨部署的远程负载均衡和数据复制能力，但数据复制提供的一致性为“最终一致性”。



***4.2.2 附图5 跨机房最终一致性数据复制示例***

如上图，步骤为：

1 client调用负载均衡服务，询问其对应的本地IFS系统。

2 负载均衡服务计算其位置，返回其对应的服务区域调用地址。这个地址在client端可以长期缓存，因为client端为一个业务系统的动态服务，其部署的位置是固定的。

3 如图3 4 5步骤所示，client完成向本地系统的数据写操作（读操作无需数据同步）。

4 如图6步骤所示，完成以后client需要报告负载均衡服务一个数据同步的消息。

5 负载均衡服务发起数据同步，要求远程需要同步的其他子系统向对应最优路径的系统请求数据，完成数据同步操作，如图8 9 10 11步。

6 当数据同步失败时，负载均衡服务只能尝试重试，如果最终重试失败，需要将已经同步完成的数据设置为失效的。以便业务系统访问时可以感知。

当在一个新的位置部署新的基础服务子系统时，需要完成一个数据同步操作，其大致的流程如下图：



***4.2.2 附图6跨机房新增基础服务部署数据复制示例***

## 分组及分域

### 分组及分域的要求

在基础服务提供的服务能力规划中，所有的业务系统会共用了一套基础服务。这样基础服务本身就需要考虑如下几个问题：

1 如何做到基础服务数据访问的安全性，即只允许经过授权的机器访问基础服务。

2 在基础服务内部数据的访问互斥性，即某一个业务只能访问业务自己的数据，而不能访问其他业务的数据。

3 业务系统内部以及业务之间不同的数据定义者定义的数据最大限度的避免可能的各种“碰撞”，如避免Key/Value中的不同业务系统生成一样的key。

### 访问安全性

由公共管理服务 - 配置管理系统提供统一的管理数据，该数据描述针对某一个基础服务系统（最终细化到部署态的服务器上），都有哪些ip地址可以访问基础服务能力对应的通信端口。



***4.3.2 附图1 访问安全性配置表***

配置创建或变更后，配置系统会下发此列表到基础服务对应的服务器上，基础服务服务器上的操作系统防火墙将按照此列表配置自身访问列表。这样就可以保障只有经过授权的服务器才可以访问对应的基础服务。

### 数据互斥性

对于一个业务系统来说，其数据会被基础服务分散存储到基础服务归属的各个服务器中（MySql集群除外）。这样对于基础服务的一个存储空间来看，其存储的数据将属于不同的各种的业务系统，最终通过数学期望来保障所有的基础服务对应的存储单元的存储空间消耗基本一致。

 ***4.3.3 附图1 数据分布示例***

在上文描述的业务逻辑的前提下，基础服务本身需要根据3.3.2中的“访问安全配置表”来控制业务系统的访问行为即：

只允许对应业务系统ip地址来源的请求访问对应的业务系统数据。

如何约束和识别不同的业务系统将在3.3.3中说明。在可识别业务系统的前提下，完成下面的时序：



***4.3.3 附图2 数据互斥时序***

### 数据编码约束

为了最大限度的避免不同业务系统或者业务系统内部的不同组件在在使用基础服务时出现数据冲突，我们对基础服务涉及到的数据做出一些编码的约束。

对于所有可以隐含为Key值的访问数据（如分布式文件系统中的文件名称、Key/Value存储中的Key值、队列服务中的队列标识）均采用以下编码：



***4.3.4 附图1 数据编码规则***

其中：

Vesion为1个字节，存放此编码约束对应的版本，从1开始。每次变更+1.

首个len字段长度为1个字节，放入sysId的长度，在版本中给予长度固定的约束，如在版本1中sysId长度为2.

sysId放入系统标志，每个业务系统均有一个固定的系统标志。在配置管理系统中申请获得。

第二个len字段长度也为1个字节，放入sysId对应的业务组件的id，由业务系统自己定义，但长度为版本本身的约束。

AppId字段放入组件标志，该标志长度固定，由业务系统自行确定。

Key字段放入真正的业务key值。

数据编码具体的使用时序如下：

 4***.3.4 附图2 数据编码使用时序***

在上图的“读/写数据”调用中，业务系统传递数据编码给基础服务系统，基础服务通过数据编码中的信息校验安全性。

## 接口能力

基础服务要最终被上层的基础产品、业务产品以及公共管理服务调用，故基础服务需要开发相应的接口能力。

基础服务可以提供如下几种接口能力：



***4.4 附图1 基础服务接口类别***

Stub接口：Stub接口使用在分布式文件存储、Key/Value存储等服务系统；支持RMI等远程调用协议，需要在调用端引用基础服务相应的Stub库进行远程访问或调用。

Memcached接口：可以使用在缓存系统和Key/Value存储中，调用者可以通过Memcached调用直接访问基础服务。

Http接口：基础服务提供Http接口，它具有易于访问、方便调试等优点，但访问效率较低。

二进制（ProtoBuff，Thrift）接口：跨语言级的二进制访问协议接口，使用在分布式文件存储、键值存储等服务系统中。会在基础框架中采用类似Gearman等调用机制进行包装。

Webservice、RESTful接口：通用的SasS调用方式，由于序列化和反序列化代价较大，访问效率不如上述的接口方式。

Mysql协议接口：用于Mysql数据库访问以及Mysql代理访问的接口方式。

## 平滑扩展能力

### 分布式文件、键值系统等横向扩展

跨机房部署参考3.2，在同一个区域里系统的扩展方式为增加服务器，注册到Master时将采用自有的负载均衡策略和容错机制把其他服务器的“数据”同步到新加入的服务器。

具体的内部负载均衡策略和分组机制不做约束。

### Mysql集群横向扩展

#### Mysql的分区策略（Partition）

* 简介

数据库划分把一个大的数据库分割为更小的数据库集合，每个分区遍及多个服务节点，用户能够在分区上实现本地事务处理。

划分方式：

数据库表、索引、事务日志和数据元素。

水平划分（Shard）：

把不同的行放入不同的数据库表中。

垂直划分：

少量的列和剩余的列分开存储在不同的数据库表中。

* 划分的统一标准

范围划分：

划分主键值在一定范围之内，例如邮编在70000和79999之间作为一个划分标准。

队列划分：

划分主键值在一个数值集合内，例如把Iceland, Norway, Sweden, Finland or Denmark 作为北冰洋国家的一个划分。

哈希划分：

划分主键值为一个哈希函数的值。

复合划分：

上述3种方式的组合。

#### Mysql的碎片化策略（Sharding）

* 简介

水平扩展(Scale Out，亦或横向扩展、向外扩展)的解决方案，其主要目的是为突破单节点数据库服务器的 I/O 能力限制，解决数据库扩展性问题。

数据Sharding的策略与分区表的方式有很多类似的地方，有基于表、ID 范围、数据产生的时间或是SOA 下理念下的基于服务等众多方式可选择。而与传统的表分区方式不同的是，Sharding策略和业务结合的更为紧密，成功的Sharding必须对自己的业务足够熟悉，进行众多可行性分析的基础上进行，"业务逻辑驱动"。

* Sharding策略

冷热数据分离：根据业务特点，根据时间范围对主要的业务数据做Sharding，把不到 10% 的"热"数据有效隔离开来，同时对这部分数据用以更好的硬件，提供更好的用户体验。而另外 90% 的数据因用户很少访问，所以尽管访问速度稍慢一点，对用户来说，影响也很小。

关联数据抽象：采用一定策略抽象数据，抽象出来的数据对象之间的关联数据很小，关联数据尽可能保持到同一分区。

#### Mysql代理

MySQL-Proxy是处在你的MySQL数据库客户和服务端之间的程序，它还支持嵌入性脚本语言Lua。这个代理可以用来分析、监控和变换（transform）通信数据，它支持非常广泛的使用场景：

* + 负载平衡和故障转移处理
  + 读写分离
  + 查询分析和日志
  + SQL宏（SQL macros）
  + 查询重写（query rewriting）
  + 执行shell命令



**4.5.2 附图1 MysqlProxy读写分离**

# 基础产品规划

## 蓝图

 ***5.1 附图1 基础产品蓝图***

基础产品主要处理的方面是一些公共的业务能力，这些业务能力将被其他系统引用并完成业务系统的业务能力，也可能独立为一个业务能力，但是独立的业务能力基本不应该面对最终的互联网用户，而是由内部系统或者内部人员来使用。

基础产品会随着业务系统的开发活动需要而扩展，所以在本文不做过多描述。

## 负载均衡能力

所有的基础产品必须考虑负载均衡模型，达到如下要求：

1 任何组件均无无单点故障。

2 所有的用户访问需要的组件均可平滑扩展。

# 公共管理服务规划

## 蓝图

***6.1 附图1公共管理服务蓝图***

## 子系统边界

### Boss系统

Boss系统即业务运营支持系统，它融合了业务支撑系统(BSS)与运营支撑系统(OSS)，是一个综合的业务运营和管理平台。我们目前的Boss 系统中集成了一些基础运营系统，并包含了一部分权限管理能力，未来需要着力于为全网各个子系统提供统一的权限管理的能力。

### 配置管理系统

配置管理系统管理整个产品开发、测试、安装、运维以及后续的升级流程。在配置管理系统中各个流程环节可追踪；把机器自动化作业与人机交互有效的结合起来。作为一个系统运维平台，主要功能是发布、自动编译、上线及后续的升级；它还对线上所有机器进行状态维护和远程控制。

### OA系统

内部员工登录OA系统，在工作流引擎的支持下，处理各种OA工作流事务，能够直观的从待办事项和已办事项看到自己的工作内容。OA系统还能看到自己工作中消息提醒和系统公告。

### 版本管理系统

版本管理系统将结合svn以及boss系统具有的认证能力，通过有效的流程整合软件开发、测试以及上线的各个流程，保障各个流程的交付物可追踪可控制。

### 监控系统

监控系统是我们对线上服务主机和上线系统的运行状况实时监控分析的运维管理系统。它定义监控指标；监控线上服务器的运行状态；并对线上机器的异常行为进行监控报警和逻辑控制。

## 子系统交互示例

### 开发阶段

***6.3.1 附图1 开发范例***

在具有通过审核的开发计划的前提下：

1. 开发人员首先通过版本管理系统创建新的开发项目，然后在此项目中完成符合开发计划的开发活动。

2．完成开发活动后，发布对应的版本计划对应的dev版本，dev版本的命名规则参见[**《IFENG-2011-TS-PR-01(凤凰新媒体-2011-技术规范-流程管理规范-No.01\_版本发布管理规范)》**](规范/IFENG-2011-TS-PR-01(凤凰新媒体-2011-技术规范-流程管理规范-No.01_版本发布管理规范).docx)**。**

3. 发布DEV版本后，开发人员需要将此版本安装到开发活动内部的开发环境，并进行单元测试和集成测试，其中单元测试需要达到规定的覆盖率目标。

4. 测试通过后，将DEV版本对应的安装文件提交到配置管理系统，配置管理系统将此发布物提交到测试的SVN上，并通过工作流引擎发起一个测试流程，测试人员将收到通知，并可以在oa系统的待办工作列表中查看到对应的测试任务。

5 进入测试阶段。

### 测试阶段

 ***6.3.2 附图1 测试范例***

1. 测试人员查询到自己的待办事项，发现一个新的待测试版本。
2. 测试人员通过配置管理系统将软件安装到测试环境。配置管理系统通知部署在测试服务器上的agent检出测试SVN对应的软件版本，并完成安装和启动的过程。
3. 测试人员在测试环境上进行测试，期间如果发生缺陷，将提交缺陷至缺陷跟踪系统（bugfree）并通过工作流引擎将缺陷放入开发人员的待办事项中。开发人员通过检查自己的待办事项处理属于自己的缺陷，处理完毕以后重新发布dev版本，回归上述流程。
4. 测试通过以后，测试人员完成测试工作，并确认测试通过的同时通过工作流引擎将此消息传递给开发人员，进入其待办事项。
5. 开发人员发布alpha 版本，并创建一个软件上线流程。
6. 软件上线事件进入运维人员待办事项。

### 上线阶段

***6.3.3 附图1 上线范例***

1. 运维从oa系统获取待上线的项目名称。
2. 在配置管理系统发送上线请求，待相关领导从oa系统中审批完成后提交上线配置到配置管理系统中。
3. 提交系统上线申请至配置管理系统，配置管理系统向线上服务器发送上线指令，线上服务器从版本管理系统中对应的运维svn中获取资源软件安装文件，并安装。
4. 安装完毕以后系统上线完毕，通知运维人员，并通过oa通知相关人员。

### 系统监控

 ***6.3.4 附图1 监控系统范例***

1. 运维人员通过监控系统定义监控指标。并将监控需要的采集探针上传到配置管理系统中。
2. 通过配置管理系统的安装能力，监控系统将探针部署到线上环境中。
3. 监控探针定期或者根据事件反馈监控数据。监控系统分析及汇总数据。
4. 运维人员可以通过监控系统查看各种数据运行的状况和趋势。
5. 监控系统也可以根据配置的规则将各种异常或者其他需反馈的信息通过oa系统的待办事项或告警服务的能力通知给相关人员。

### 与Boss系统交互能力

***6.3.5 附图1 各系统与BOSS交互***

## 工作流引擎

参见《[工作流概念及概要设计.docx](规范/工作流概念及概要设计.docx)》

# 基础框架规划

我们将着手定义统一的开发框架，目前开发框架主要有：

1 Java 的IOC框架：iCommon.

2 PHP 的 MVC框架：Noah

## iCommon

着力于为Java开发主要应对的场景：基础服务场景、基础产品场景、公共管理系统场景以及可能的业务系统场景提供统一的Ioc（依赖注入）能力，以及依赖Ioc（依赖注入）的组件。

目前包括如下几个能力：

### Common.conf

通用依赖注入能力，类似Spring提供的注入能力，但更轻量级，更贴近我们的使用需要，配置方式更简单灵活。同时未来将提供针对xsd和“个性化配置方言”的支持，提高注入能力的可读性。

### Common.plugin

插件结构体系，将一个复杂的业务逻辑分解为若干个步骤，并通过框架提供的组合能力，将有限多的原子性不愁，聚合为能力分子，从而最大限度的节约代码。

### Common.dm

标准的DAO实现，底层为hibernate,但是暴露出的接口均为对象接口，隔离了关系数据和代码之间的关系，并以此为基础提供各种可能的缓存、异步及通信能力。

### Common.remoting

远程调用能力，可以在无需额外编码的情况下将系统的逻辑视图和部署视图分离，使得开发人员在开发阶段只关注逻辑视图，而在部署阶段注入部署能力。

### Common.workflow

轻量级工作流引擎，依赖remoting能力被调用，内部用plugin描述一个“活动”，关联外部“角色”响应挂起及唤醒。

## Noah

着力于为PHP代码提供模块化的MVC体验，并模块化实现各种基础能力，开发人员可以以此为基础构建自己的业务系统。目前已经有很多应用，后续将结合基础产品和基础服务的规划，在noah中提供天然的支持能力，并考虑将noah与iCommon联通实现“异构”系统之间的通信和调用。

# 部分子系统概述

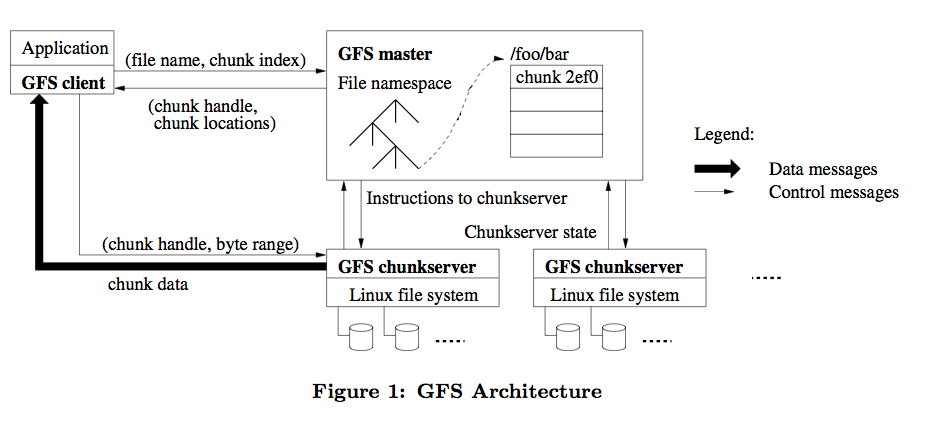
## 基础服务部分

### 分布式文件系统

#### 可参考的GFS模型

分布式文件系统提供一个分布式的、无单点故障的、高性能的文件存储服务。

业内可以参考的文件系统有很多，其结构大多类似，下图是GFS的结构图：

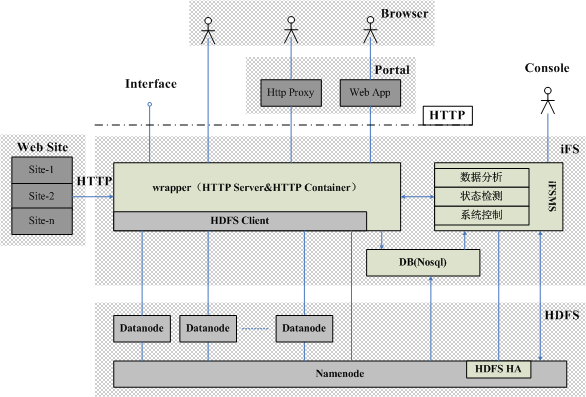


***8.1.1 附图1 GFS结构示意***

#### IFS现状

在本文定义以前，技术部-基础研发组曾经开发过IFS系统。IFS本身以 HDFS为核心。并封装了客户端部分，提供内部路由体验，提供http和memorycache接口。

IFS现有结构如下：



***8.1.1 附图2 IFS结构示意***

其中Datanode = GFS-chunkserver

Namenode = GFS-master

HDFSclient = GFS-client

IFS的主要能力为封装了一个 wrapper ，进而屏蔽客户端和句柄缓存对真正业务系统的影响。

另外，考虑到HDFS的接口适应度，IFS增加了一些接口能力。

#### 改进方向

结合业内经验和IFS系统现状，我们将一个部署态的分布式文件系统大致分为如下几个组件：

1 客户端。

2 负载均衡服务。

3 master。

4 chunkserver.

增加的负载均衡服务将解决跨机房部署问题。

大致的时序图如下：

 ***8.1.1 附图3 分布式文件系统数据复制时序***

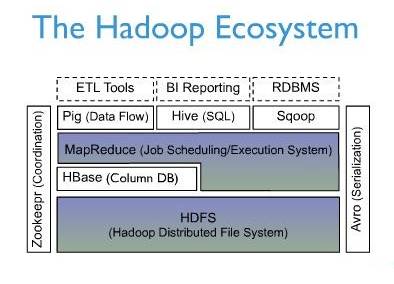
在现有的ifs基础上，新增一个用于跨机房的负载均衡服务，负载均衡服务为一个单独的服务。

### MySql集群

待补充

### Map/reduce 及 分布式计算

这两个基础服务，目前主要的应用场景在统计和搜索系统中，可以采用HBase + zookeeper来完成。



***8.1.3 附图1 HBASE结构示意***

类似Google Bigtable利用GFS作为其文件存储系统，HBase利用Hadoop HDFS作为其文件存储系统；Google运行MapReduce来处理Bigtable中的海量数据，HBase同样利用HadoopMapReduce来处理HBase中的海量数据；Google Bigtable利用 Chubby作为协同服务，HBase利用Zookeeper作为对应。

### NOSQL数据库

我们需要的是一个NOSQL数据库集群，提供一个统一的NOSQL能力，集群内部完成不同业务系统的域划分及管理即可。

目前我们的原有产品为inetdb.

短期内主要的任务是明确Nosql存储的设计目标，针对Inetdb的实现，做改进或者重构。

### 通用队列服务

通用队列服务目前我们的自有产品为icqueue,提高一个标准队列体验，但距离一个部署级的队列集群还有一定的差异。

### 通用缓存服务

目前我们没有通用的缓存服务产品，下一步的重点是选型，比较倾向于Redis，具体的还要看最后的比较结果，缓存服务自主开发的部分主要在分布式和分域。

## 基础产品部分

待补充

## 公共管理服务部分

### Boss系统



***8.3.1 附图1 Boss系统结构***

统计系统：对于站内文章访问趋势的统计和专题数据的统计。基于现有的统计系统不断改进完善。

用户管理：用户注册审核，重置密码；关键词访问，生成用户统计报表。部分为现有的SSO服务能力。

邮件系统：邮件任务管理；邮件用户、邮件组管理；邮件发送统计。部分为现有的EDM系统能力。

分发系统：管理从主站向各个省分站进行数据、视频的复制分发，部分为现有的INMS系统能力，关于INMS与监控系统的定位和分工，有待开发改进阶段再进行讨论和设计。

检索系统：现有的站内搜索系统能力。

海纳系统：现有的海量系统能力。

系统管理：包括编辑频道管理、用户管理、权限管理、权限组管理和资源管理。

#### 关于权限管理

我们将使用统一的权限管理，管理全网各个管理系统和业务系统需要的操作权限。

统一的权限管理将作为BOSS系统的一个组件存在，并由BOSS系统管理和维护。

统一权限管理中的数据模型如下：



***8.3.1 附图2 权限管理数据模型***

其中：

用户应对应一个自然人，一般为公司内部的一个人员。用户具有的最基本属性为：

1 登录能力：用户名、密码。

2 通知能力：手机号码、邮件地址。

系统为一个抽象或者实际存在的软件系统，对应我们的一个线上服务的部署态。

系统存在如下继承关系：



***8.3.1 附图3 System数据模型***

即系统内部存在一个树状结构的继承关系，使得系统可以一层一层的描述为一些具体的业务能力的聚合。

角色为一个能力的集合，其主要的含义为该角色将具有某些能力，能力为系统或者系统的子节点的抽象定义。也就是说一个角色将具有一些系统、子系统、组件或者能力的操作权限。



***8.3.1 附图4 角色数据结构***

Department 部门将直接映射公司的组织结构，为一个典型的树状组织结构。员工会具体属于某一个部门。部门数据将直接对应管理职能，一般某一个部门内的人员应该具有同样能力属性。



***8.3.1 附图5 Department数据模型***

Group组织本身不存在从属关系，为一个散列的数据，每一个组织由一些人员组成，一般情况下一个组织将向某一个业务行为负责，比如视频业务组中可能包含视频部门的编辑人员、技术部门的技术支持人员、技术部门的产品研发人员、技术部门的运维人员。

一个User只能属于一个部门，但可以属于多个组织。



***8.3.1 附图6 User从属关系***

任何一个人、组织或者部门都可以具有某一个角色。同一个组织、部门或者人可以对应多个角色。



***8.3.1 附图7 角色映射关系***

当一个部门具有某一个角色时，其子部门和部门及子部门归属的人员也同时具有该角色对应的能力。

当一个组织具有某一个角色时，属于这个组织的人员也同时具有这个角色的能力。

依赖于上面的模型，在Boss系统中构造一个统一的权限管理模块，达到如下的几个能力。

1. 部门组织结构管理。
2. 业务组织管理。
3. 人员管理。
4. 角色管理。
5. 系统功能管理。

在上述能力中，将完成人员的入职离职，人员的权限分配；系统上下线的权限分配，系统线上的统一登录和权限管理等能力和接口。

### 配置管理系统



***8.3.2 附图1 配置管理系统结构***

配置管理系统整个软件架构分为业务层、事务层和传输层。业务层实现了配置管理系统通过Web方式对外提供的服务功能；业务层还实现了控制台服务，它接收并处理运维人员从虚拟控制台传递的命令行请求；事务层主要实现了上线任务的事务控制和会话状态管理；传输层实现了基本消息转发、与各个子系统通讯、并向运维人员提供Linux标准的虚拟控制台(PTY)接口。

* 业务层

项目管理：创建、发布新系统；系统的统一业务交互管理。

发布上线：部署系统安装包到线上环境并安装执行。

控制台服务：接收并处理运维人员从虚拟控制台传递的命令行请求。

* 事务层

任务管理：指令执行单元，通过异常捕获和一致性处理等方式正确执行上层的业务请求。

工作流模块：上线过程的流程管理和状态流转。

命令解释器：控制台命令的翻译, 并传递指令到上层的控制台服务。

接口调用单元：各个接口单元的事务处理和会话控制，处理调用过程的超时和异常错误。

* 传输层

消息转发：接收线上环境或版本服务器的消息数据、结构化处理，传递到上层；上层的任务请求和数据通过消息转发模块发送到线上环境或版本服务器。

虚拟控制台接口：向运维人员提供的Linux标准虚拟控制台(PTY)接口，运维人员能够使用比较熟悉的命令行方式查询线上系统的运行状况或上线活动。

### OA系统



***8.3.3 附图1 OA系统结构***

OA系统整个软件架构分为业务层、事务层和传输层。业务层实现了OA系统过Web方式对外提供的服务功能；事务层主要实现了工作流程的事务控制和会话管理；传输层实现了与各个子系统通讯。

* 业务层

已办、代办：获取起草、发起或流转本用户的工作事务列表和明细。

事务提醒：通过消息、短信或展示页面等方式醒目的告诉用户代办事宜或消息。

通知公告：管理员给已知人群发送的系统消息和公告。

鉴权认证：用户登录通过第三方Boss系统进行身份认证和操作权限判定。

审批事务：对起草或流入的文档、信息进行审批。

* 事务层

工作流模块：系统流程管理和状态流转。

接口调用单元：各个接口单元的事务处理和会话控制，处理调用过程的超时和异常错误。

* 传输层

略。

### 监控系统



***8.3.4 附图1 监控系统结构***

监控系统整个软件架构分为业务层、事务层和传输层。业务层实现了监控系统对外提供的监控服务功能集合；业务层还实现了控制台服务，它接收并处理运维人员从虚拟控制台传递的命令行请求。事务层主要实现了事务控制和会话状态管理，并实现了监控任务的语义分析、控制执行。传输层实现了基本消息转发、与各个子系统通讯、并向运维人员提供Linux标准的虚拟控制台(PTY)接口。

* 业务层

监控项定义：定义需要采集的监控项，监控项不仅包括上线主机的指标（例如CPU占用、内存占用、IO占用等等），也包括各个上线系统的运行情况指标项。

监控语义：定义了监控动作、触发规则、报警规则等等。

监控数据收集：接收各个上线主机的监控数据或主动向各个主机发送监控数据请求。

监控报表：创建、生成各个监控主机的历史数据报表和趋势图。

控制台服务：接收并处理运维人员从虚拟控制台传递的命令行请求。

* 事务层

任务管理：包含监控决策部件和监控动作执行单元，通过异常捕获和一致性处理等方式正确执行上层的业务请求。

命令解释器：控制台命令的翻译, 并传递指令到上层的控制台服务。

接口调用单元：各个接口单元的事务处理和会话控制，处理调用过程的超时和异常错误。

系统报警：接收任务管理模块的报警请求，调用底层接口发送报警短信、邮件等等。

日志处理：监控数据存储到分布式文件系统，日志分析。

* 传输层

消息转发：接收线上环境监控消息、数据，传递到上层；上层的监控请求和数据通过消息转发接口发送到线上环境。

虚拟控制台接口：向运维人员提供的Linux标准虚拟控制台(PTY)接口，运维人员能够使用比较熟悉的命令行方式查询线上系统的运行状况和监控数据。

# 实施计划

待补充

# 遗留问题

1. 基础产品的具体产品列表需要待基础服务和管理系统完善以后逐渐定义。
2. 部分章节需等待初稿通过审核以后补充完善。
3. 基础框架部分由于和部署无关，未作详细描述，后续待补充。