logo8.png

详细设计说明书(SaaS)

**《SaaS》**

文档一旦发布，请务必按文档执行并坚持遵守。如果您有改进的建议，请将您的建议发邮件或当面告知所列作者。

修订历史记录

| **版本** | **日期** | **作者** | **修正原因** |
| --- | --- | --- | --- |
| 1.0 | 2013-05-23 | 蔡源 | 初始化文档 |
| 1.1 | 2013-08-30 | 蔡源 | 增加【定制化、个性化】章节 |
| 1.2 | 2013-09-29 | 蔡源 | 增加自动登录的设计 |
| 1.3 | 2013-10-15 | 蔡源 | 增加参数字典设计  增加客户管理设计 |
| 1.4 | 2013-10-18 | 蔡源 | 增加应用场景及解决方案章节，用于描述特定业务流程或者功能流程的实现 |
| 1.5 | 2013-12-04 | 蔡源 | 增加【团队协作】相关设计，主要包括项目管理和任务管理 |
| 1.6 | 2013-01-15 | 蔡源 | 增加用户与租户空间一对多的设计，用户可以在不同空间中切换 |
|  |  |  |  |
|  |  |  |  |

目录

修订历史记录 2

目录 3

1.1 编写目的 5

1.2 背景 5

1.3 参考资料 5

1.4 术语定义 5

2. 设计概述 7

2.1 任务和目标 7

2.2 运行环境 7

2.3 条件与限制 7

3. 系统架构详解 7

3.1 升级列表 7

3.2 配置、注解、约定 7

3.3 接口 or 实现 7

3.4 本系统约定 7

3.5 开源技术 8

3.6 自主技术 11

3.7 模式应用 11

3.7.1 单例模式 11

3.7.2 工厂模式 11

4. 系统技术模块 11

4.1 数据存储 11

4.1.1 Cache 12

4.1.2 CLI连接池 13

技术特性 13

数据采集 13

服务与组件 14

处理器与线程 16

服务与插件 17

控制与自动化 17

预警与自动修复 17

开放接口 17

拓扑发现与拓扑图 18

5. 系统主要数据模型 19

5.1 Node 19

5.2 ProxyServer 20

6. 系统功能模块 20

6.1 Agent 20

6.1.1 私有MIB实现 20

6.2 中心采集控制系统 22

6.2.1 Framework 22

6.2.2 发现模块 23

6.2.3 采集模块 24

6.2.4 事件模块 36

6.2.5 性能管理 41

6.2.6 告警模块 45

6.2.7 通知模块 48

6.2.8 Trap模块 52

6.2.9 数据清理模块 54

6.2.10 数据同步模块 57

6.3 Web 59

6.4 Message Queue 60

7. 接口设计 60

7.1 外部接口 60

7.2 内部接口 60

8. 机制指南 60

8.1 系统使用证书校验机制 60

8.2 用户安全校验机制 60

8.3 自动登录机制 60

8.4 数据传递机制 61

8.5 异常处理机制 61

8.6 数据清理、更新机制 61

8.7 运行配置信息处理机制 61

8.8 客户端连接服务器切换机制 61

8.9 服务器切换机制 61

9. 开发、测试、生产 62

9.1 数据库 62

9.2 服务器 62

10. 持续集成指南 62

10.1 持续构建服务器 62

10.2 持续集成Web平台 62

# 引言

## 编写目的

详细设计的主要任务是对概要设计方案做完善和细化。说明书编制的目的是说明一个软件系统各个层次中的每个程序（每个模块或子程序）和数据库系统的设计考虑，为程序员编码提供依据。

本文档在概要设计的基础上，进一步完整详尽的描述了系统实现的技术细节，及根据业务需求制定的系统所需要实现的业务功能，功能模块的详细定义。

## 背景

## 参考资料

## 术语定义

| **缩写** | **英** | **中** |
| --- | --- | --- |
|  |  |  |
|  |  |  |

# 设计概述

## 任务和目标

本文档详细描述Framework层面组件和模块的详细设计及实现。

## SaaS用户特点

* 没有大笔的软件授权费用，通常是按月收取使用费。
* 除了个人电脑和互联网连接之外，客户不需要进行任何其它IT投资。
* 通过互联网，多个公司多重租赁（Multi-tenancy）一套服务器软硬件设备,月租费包含服务和升级两项费用。

## SaaS模式优点

* 面向中小企业用户，支持海量用户群体，以负载均衡和身份认证等技术保证系统平台的稳定和安全;
* 实施成本低、效率高，在线租用模式下，中小企业用户可以先试后买，实际应用体验让中小企业用户享有充分的准备和缓冲时间以及更多的选择机会;
* SaaS平台的运营商拥有专业的技术人员、先进的设备和完善的解决[方案](http://solution.chinabyte.com/new/)，能够及时处理突发问题、完成系统升级，提供专业服务，免去了中小企业的维护运营耗费。

## 多租户的潜在问题

* 隔离：因为承租者共享软件和硬件的同一实例，所以一个承租者可能会影响其他承租者的软件的可用性和性能。例如，如果共享的软件没有充足的安全保障措施，则可能会发生因一个承租者关闭共享软件而导致对共享该实例的所有承租者拒绝服务。
* 安全性：如果共享软件没有充分的安全保障措施，则一个承租者的用户可能会访问属于另一个承租者的数据。
* 自定义能力：因为软件是在承租者之间共享的，所以各个承租者不可能自定义该软件。例如，在没有充足的扩展点的情况下，一个承租者不可能为业务流程提供自己的实现。
* 应用程序升级将为承租者带来问题：同时升级共享的软件可能并不是所有承租者都希望的。
* 恢复：在承租者之间共享数据库使得很难为每个承租者单独备份和恢复数据。

## 多租户带来的挑战

可以根据面临这些挑战的主要组织和角色对多承租应用程序的技术挑战分类：解决方案开发人员和服务提供者。

**解决方案开发人员面临的技术挑战包括：**

* 访问控制：如何在承租者之间共享应用程序资源（例如，虚拟门户、数据库表、工作流、Web 服务和 Java™ 2 Platform Enterprise Edition (J2EE) 构件），以便只有属于承租者的用户可以访问属于该承租者的实例？例如，如何确保其他银行（如北卡罗来纳州第一银行）的用户不能访问加拿大第二银行的资源（如虚拟门户）？
* 自定义能力：
  + 数据库：在一个承租者不影响其他承租者的情况下，如何自定义共享的数据库模式？例如，在不影响北卡罗来纳州第一银行的模式定义的情况下，加拿大第二银行如何将新的数据字段引入客户概要的共享数据库表？
  + 用户界面：如何仅通过配置来自定义网站外观（即，在不更改代码的情况下）？例如，如何确保北卡罗来纳州第一银行和加拿大第二银行的银行管理员可以配置不同的设计，并在其客户概要 Portlet 中显示其他字段？
  + 业务逻辑：在不进行代码更改的情况下，如何允许为每个承租者自定义业务逻辑？例如，北卡罗来纳州第一银行如何使用不同于加拿大第二银行的最小信用记录自动拒绝贷款申请？
  + 工作流：您如何让承租者银行在共享的工作流中自定义人工任务和其他有条件任务的分配？例如，北卡罗来纳州第一银行如何确保将共享工作流中的贷款审批任务仅分配给北卡罗来纳州第一银行的员工？
* 承租者预置：如何自动执行新承租者的预置？例如，如何使用非常少的手动步骤（即如何自动化创建新的 LDAP 子树或数据库、创建新的虚拟门户、部署 portlet 的新实例和注册新的 IBM DB2® XML 模式之类的步骤）引入新银行（例如 Fairfield Trust 第三银行）？
* 基于使用情况的测定：如何记录服务的使用情况，以便仅根据服务使用情况向每个承租者收取服务使用费？例如，银行服务提供者管理员如何测定承租者北卡罗来纳州第一银行和加拿大第二银行的服务使用情况，以及他们的客户调用贷款申请服务的次数？

**服务提供者面临的技术挑战包括：**

* 数据库共享、自定义、备份和特定于承租者的数据的恢复：服务提供者如何基于性能、管理和可扩展性标准在不同的数据库划分模式之间进行选择？例如，服务提供者如何满足从多个承租者之间共享的表仅备份自己数据？
* 快速支持现有 Web 服务的多承租：如何在很少或不进行代码更改的情况下使单个承租者的 Web 服务支持多承租？例如，在不对 Web 服务接口和实现进行代码更改的情况下，如何使单个承租者的信用审核服务支持多承租？
* 在大型企业中管理大量的第三方服务提供者和部门服务客户之间的连接性：在 Web 交付的应用程序中，大型企业中的业务部门 (LOB) 表现了承租者的许多特征。同一企业中不同的业务部门可以使用来自不同的第三方或内部服务提供者的服务。在企业中，大量的此类服务提供者可能导致中心 IT 部门的管理问题。例如，银行服务提供者企业中的不同业务部门（如信用额度和抵押贷款部门）可能使用不同的信用审核服务提供者。中心 IT 部门如何监视、授权和测定企业中不同业务部门的多个信用审核服务的使用。
* 可伸缩性改进了硬件使用情况和特定于承租者的服务质量 (QoS)：服务提供者如何改进在不同承租者之间共享的硬件使用和提供可绅缩性？服务提供者如何为不同的承租者提供不同的服务质量？例如，您如何满足有别于加拿大第二银行的服务质量需求，以便使用专用硬件承载其服务并收取较高的服务使用费用？

## SaaS模式的不足

### 场景一

CallCenter与SaaS的不兼容。很多CRM或者客户系统最重要的一个功能就是语音交互，也就是说，当客户打电话过来的时候，能够自动根据客户的电话号码进行转接，转接到对口的业务员那边，以减少客户的等待时间，提高客户的满意度。其技术实现是基于传统固话网络的。与部署在客户本地的应用相比，部署在云中的应用无法实现该功能。即使采用IPPHONE，由于网络带宽等因素的影响，仍然无法同局域网的IO达到同一平面。

解决方案：Local Agent+C/S模式来实现这部分高级功能。

### 场景二

大文件传输，海量基础数据的查询、导出等操作，通过互联网传递这些数据同局域网相比有不小的差距。统计分析等功能对共享模式下的计算和网络吞吐也有较大影响。

### 场景三

如果企业在实施SaaS前，已经实施了相关业务应用，如已有成熟的CRM系统,财务等，如何实现部署在SaaS中的业务与部署在企业内部的业务的数据交互，功能交互，将是决定SaaS成功的一个关键。因为对于企业来说已实施并使用多年的成熟应用也是企业资产的一部分，无论是废弃还是转移到SaaS都存在的很多潜在风险。

如果企业的信息化管理是一清二白的，那么采取SaaS模式的信息化应用可能比较合理，因为不用考虑跟现有的信息化项目集成。而对于那些已经具有一定信息化管理基础的企业来说，如果后续新的系统考虑采用SaaS的模式，那么就需要先断掉与现有信息化系统集成的想法。这基本上没有实现的可能性。

# SaaS设计

## 多租户模式

### 数据隔离

* 将每个承租者的数据隔离到不同的数据库。
* 共享数据库，Multi-Schema，将每个承租者的数据隔离到独立的表和模式。
* 共享数据库，Share-Schema，在所有承租者之间共享一组相同的表和模式。

### 实现多租户的三种模式

* 无共享，完全独立：每个租户独立使用一套应用程序和一个数据库，应用与数据库均不包含租户信息，通过访问入口路由到指定租户的路径上。
  + 优点：
    - 无需修改原有应用程序跟数据库。
    - 租户间不会相互影响，可对个别租户做自定义。
  + 缺点：
    - 部署跟运维相对繁琐。
    - 物理设施资源开销最大。
    - 无法对多租户数据进行查询归并，存在数据孤岛
* 共享应用，多数据源：使用同一套应用程序，数据库访问时根据租户信息路由到指定数据库或Schema上。
  + 优点：兼顾了开发和性能。
  + 缺点：
    - 无法对多租户数据进行查询归并，存在数据孤岛
* 共享应用，单一数据源：使用同一套应用程序，使用同一个数据库，数据模型中定义了租户信息，通过过滤条件过滤租户数据。
  + 优点：性能最优，部署简便
  + 缺点：
    - 对系统架构和开发工程师要求较高，否则可能存在数据安全性问题
    - 运维复杂，当数据发生异常需要恢复时，无法简单依赖数据库的恢复机制，并将影响到多个租户的数据

### 数据过滤

在共享同一数据源的模式下，需要对每个数据查询增加租户信息的过滤条件；在单app环境下，一个用户只对应一个租户，通过登录用户信息即可获得租户信息，比较简单。但是在平台模式下，一个用户可以租用多个app，用户与租户是一对多的情况。

解决方案：用户在登录一个app时，app通过appKey去平台获取该用户的信息，并在本地session中保存用户登录信息，平台可以根据appKey与用户ID获得唯一的tenant，即app本地session中只需保存用户对象与tenant对象一对一的关系。只有用户在登录平台系统时才有一对多tenant的情况。

### 总结

实际使用中可能综合运用3种模式，即如果客户较为重要，愿意为安全性、性能等额外付费，可部署为独立模式。常规情况下则使用共享数据库模式，但根据性能或部署需要，可能根据用户数切分为多个domain，每个domain中的用户共享一个数据库，这样如果某个domain失效，不会影响其他用户的使用。但基本原则是所有数据表均按SaaS模式设计，以便实现不同模式下的切换。

## 定制化、个性化

定制化指的是同一SaaS服务可以为不同用户在相同基础功能的基础上提供一定程度的功能定制或强化，在不改动或尽量小改动服务的基础上实现不同用户的差异化功能性需求。如：数据模型的定制化，业务流程的定制化。

个性化指的是为客户提供的，满足用户企业或个人个性需要的非功能性需求，如国际化、主题、收藏夹、菜单结构调整、Logo或程序名调整、Dashboard等。

## MetadataDB

元数据数据库，定义了多租户相关信息，用于租户信息管理，作为基础的公共服务独立于业务系统数据库。

## 系统用户角色

### 租户拥有者

租用app的用户，作为app的拥有者，其拥有app的所有功能模块使用权限；同时作为拥有者，可以对app进行续费、升级、停用等操作。

此外作为app的第一个默认用户，也是默认的租户管理员（租户开通时默认创建），具有租户“系统管理”模块的功能权限，可以在租用范围内创建角色，邀请其他用户加入，分配权限。

### 租户管理员

租户拥有者出于管理角度考虑（如租户拥有者是老板，但是管理员是IT管理员），可以将租户中的任意用户提升为系统管理员，由其作为租户管理员协助或负责租户内相应的管理工作，如用户管理，角色管理，功能权限分配，邀请用户加入等。

租户管理员在权限上与租户拥有者一致，但租户拥有者作为最高级别，可随时将租户管理员降级成普通用户；而反之则不行。

### 租户成员

租户开通后，默认只有拥有者一个成员，此时拥有者可通过邀请方式请求其他用户加入到该租户中共同使用租用的app。如：老板租用了CRM系统，邀请公司内部员工加入到该系统中，员工即可使用CRM系统的功能，并在租户范围内共享数据。

用户在加入一个租户后，需要租户管理员为其开通相应功能模块的使用权限（通过设置角色），否则只能共享【个人事务】中公开部分的数据。

## 客户用户角色

### 系统管理员

管理系统用户、角色与权限，保证系统正常运行。

### 高管

审查客户贡献数据、客户构成数据、客户服务构成数据和客户流失数据。

### 客户经理

维护负责的客户信息。

接受客户服务请求，在系统中创建客户服务。 处理分派给自己的客户服务。 对处理的服务进行反馈。 创建销售机会。

对特定销售机会制定客户开发计划。 执行客户开发计划。

对负责的流失客户采取“暂缓流失”或“确定流失”的措施

### 销售主管

对客户服务进行分配。 创建销售机会。 对销售机会进行指派。

对特定销售机会制定客户开发计划。

分析客户贡献、客户构成、客户服务构成和客户流失数据，定期提交客户管理报告。

### 销售人员

接受销售任务，负责与客户接触，实施销售任务，跟踪客户消费。

### 合作伙伴

部分数据交互，并提供合作伙伴关系的数据，可由合作伙伴自行访问（SelfService）。

### 客户

提供客户关心的数据，可由客户自行访问查询（SelfService）

## 身份验证与授权

身份验证和授权是现实应用程序的安全性概念中主要的两个：

* 身份验证允许一个应用程序在连接时验证一个人（或一个应用程序、智能卡等）是否与它声明的一样。
* 授权定义一个用户在一个系统上的权利与权限。用户身份验证通过之后，授权会决定该用户在系统上有权做什么。因此，授权应该发生在身份验证之后。

身份验证和授权在 SaaS 应用程序中很复杂。在一个安全性 SaaS 解决方案中，底层的身份验证和授权基础设施有两种设计方法：集中式或联邦式。

### 授权模式

黑盒模式：即简化的权限模型，不开放授权功能给用户，角色和权限由系统内置，用户在加入App时自动绑定角色，对于一个App来说通常有：创建人，管理员和普通成员3个角色。

白盒模式：即允许用户授权，App创建人可在自行创建用户组和角色，并对每个功能模块进行细分授权。该模式可实现更精细的权限控制，类似传统的企业应用。

## 可扩展性

### 基础设施可扩展性

* 计算资源快速供给
* 应用快速部署
* 资源按需分配
* 自动化管理

### 应用架构可扩展性

* 应用服务器水平扩展
* 数据库水平扩展
  + MySQL Sharding（）
* 异步消息队列
* 缓存机制
* 负载均衡
* 流程可定制
* 功能可配置

## 个性化

### 界面个性化

用户可在一定程度上对界面做定制化，如使用个性化主题，个性化布局，可自行调整菜单结构等。

#### 系统菜单可配置性

菜单对不同的租户来说，可能有不完全一样的名字。例如客户管理，在医院使用时，就得改成病人管理，客户服务人员就得改成医生，客户服务记录就是就诊记录等。另外菜单的层次结构和分布，不同的租户可能也会有不同的要求。在设计上需要考虑以下几个问题：

* 一个租户一套菜单；
* 一个菜单可以关联一个子功能；
* 组织成树型结构，构成上下级菜单结构；
* 同级菜单之间还存在显示顺序的问题

#### 页面元素可配置性

各功能界面上的内容也是供用户和系统交互的界面元素。不同的租户可能有各种不同的需求。由于租户可以自定义扩展数据，这些数据是需要在页面上展示的，因此无论对页面元素的个数、位置、顺序，还是元素的含义，租户都会有一些个性化的需求。同时对于在设计时设定的界面元素，一般情况下是不允许删除的，但有时候还是允许租户将一些无关紧要的字段隐藏。

### 数据个性化

在实际应用中，不同租户之间需求的差异导致系统需要针对不同租户保存许多扩展性数据。在传统应用中，可以通过定制实例，增加客户的扩展数据，来满足其个性化要求。在多租户SaaS应用中，所有租户都使用同一个数据架构，常见的解决办法就是实现扩展数据的可配置。

名称值对的方式将扩展数据的保存和原数据表分离，另外用一个统一的扩展数据表来保存。扩展数据表将数据表的横向扩展列转换为纵向的数据集，将每一条原始数据记录的一个扩展字段，都保存成一条扩展数据行。将数据表中的数据记录与配置元数据表中的配置记录关联，构成扩展数据记录。可以提供无限数量的自定义扩展字段。 但是其增加数据操作的复杂性，查询时也要多次访问数据库才能得到完整的业务数据。这些都会影响数据访问性能。

此外可结合使用NoSQL，通过SchemaFree模式提供高扩展性和个性化。

#### 参数字典

不同的用户在对参数字典的使用上也会存在差异，如客户等级，有的喜欢用1、2、3表示，有的喜欢用A、B、C表示，这就需要参数字典也需要能够支持多租户，并可定制。

### 功能个性化

对于SaaS应用，面对为数众多的租户，大部分租户可能只会使用到应用中的部分功能。因此系统需要支持租户有选择的使用自己需要的功能，满足功能可配置要求。

#### 原子功能划分

要实现功能可配置，首先需要将整个系统的功能进行分解。整个应用需要分解成最基本、相对独立、互不重叠的原子功能。所有原子功能叠加起来，就是整个应用所提供的全部功能。

进行原子功能划分，首先就是功能分解，即将整个系统的功能分解成最基本的相对独立的原子功能，应遵循以下几个原则：

* 每个功能都是有价值的； 
* 每个功能都是不可再细分； 
* 功能间互不重叠； 
* 功能之间不循环依赖； 
* 整个系统功能是完整的。

将功能分解完毕后，由于不是所有的原子功能都是可以单独使用的。有些功能是需要依赖其他功能才能使用，功能之间是存在一定的依赖关系。因此功能分解完毕后，还需要对功能进行定义，描述相关依赖关系。

#### 功能包设计

当系统功能被划分为许多原子功能后，直接配置原子功能给每个租户是比较复杂的。需要根据用户类型和使用的场景，对原子功能进行打包，然后为每个用户配置其合适的功能包。功能包的设计要遵循高内聚、低耦合的原则，尽量将相关的和相互依赖的原子功能设计在一个功能包中。同时应该减少功能包和功能包之间的依赖，使得各个功能包尽可能独立的进行操作使用。

通过功能包的设计，虽然可以将系统功能组合成几个相对比较独立的部分，但是这些功能包仍然不可以完全独立使用，也就不能够单独销售。为了让用户购买了系统以后可以充分使用其同能，需要按照不同的商业意图构造合适用户的销售包。例如，按照客户使用功能的多少，可以把系统划分为最小版、标准版、完整版。

#### 功能使用校验

在经过对系统进行原子功能划分和功能包的设计后，系统的不同租户可以按照不同版本使用了，系统需对原子功能进行校验，确定租户在系统中可以使用和操作哪些原子功能。

# 数据模型

## 用户信息（UserInfo）

用户信息表中只保存比较固定的数据，便于快速查询和缓存，其他经常要变的数据放到附属表中

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| loginId | 登录ID | String | 登录名 |
| password | 密码 | String | 密码 |
| userType | 用户类型(1001) | int | 1：个人用户  2：企业用户 |
| userName | 用户名称 | String | 用户姓名 |
| nickName | 昵称 | String |  |
| gender | 性别(0002) | int | 0:未知  1：男  2：女 |
| email | 电子邮件 | String |  |
| mobile | 手机号 | String |  |
| status | 状态（1002） | int | 0：未激活  1：正常  2：注销（可恢复）  3：删除（仅超级管理员恢复）  4：锁定 |
| locale | 语言 | String | 支持用户自定义 |
| timezone | 时区 | int | 支持用户自定义 |
| theme | 主题 | String | 支持用户自定义 |
| headImgUrl | 头像 | String | 头像图片路径 |
| createDatetime | 创建时间 | Date |  |
| updateDatetime | 更新时间 | Date |  |

## 用户扩展信息（UserExt）

与用户表一对一，相同的生命周期

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 等于用户ID |
| level | 级别 | int |  |
| source | 来源 | String |  |
| birthday | 生日 | Date |  |
| zipCode | 邮编 | String |  |
| address | 地址 | String |  |
| district | 区 | String |  |
| city | 城市 | String |  |
| state | 省份 | String |  |
| country | 国籍 | String |  |
| securityQuestion | 安全问题 | String |  |
| answer | 安全答案 | String |  |
|  |  |  |  |

## 用户状态信息（UserState）

与用户表一对一，相同的生命周期

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 即用户ID |
| currentTenantId | 当前租户ID | int | 当一个用户绑定了多个租户时，用户需要选择当前使用哪个租户 |
| totalLoginTimes | 总登录次数 | int |  |
| loginRetryTimes | 登录重试次数 | int | 默认值0  连续登录失败超过指定次数，则帐号禁用指定时间 |
| lastLoginDatetime | 上次登录时间 | Date |  |
| lastLoginIp | 上次登录IP | String |  |

## 用户自动登录信息（UserAutoLogin）

记录用户自动登录的信息

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | UUID | Integer | UUID |
| autoLoginIp | 自动登录IP | String | 可检测是否为授权IP |
| loginId | 登录ID | String |  |
| encryptedPassword | 加密的密码 | String | 可检测密码是否一致 |
| createDatetime | 创建时间 | Timestamp | 时间戳 |
| expiredTime | 过期时间 | long | 为0或负数代表永不过期。使用cookie时用cookie的过期机制即可 |

## 用户组（UserGroup）

通过组的方式将相关用户关联起来，一个组的用户相互知晓。组可以是系统创建的，也可以由用户自行创建。

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| parentId | 父组织ID | int |  |
| code | 编号 | String |  |
| name | 名称 | String |  |
| type | 类型 | int |  |
| description | 描述 | String |  |
| status | 状态 | int |  |
| createUserId | 创建用户 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateUserId | 更新用户 | int |  |
| updateDatetime | 更新时间 | Date |  |

## 用户组成员（UserGroupMember）

用户组成员列表。

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| groupId | 用户组ID | int |  |
| userId | 成员用户ID | int |  |
| type | 类型 | int | 1：创建人  2：普通成员  3：管理员 |
| status | 状态 | int | 1：正常  2：锁定 |
| createDatetime | 创建时间 | Date |  |

## 租户信息（Tenant）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 租赁ID |
| code | domain编号 | String | 英文，由用户指定，用来实现类似email后缀的区分，通常可以是企业的英文缩写或者域名 |
| appCode | 应用编号 | String |  |
| appAlias | 应用别名 | String |  |
| bagCode | 功能包编号 | String |  |
| isolatedMode | 租赁模式 | int | 数据隔离模式  1：完全共享型  2：独立Schema  3：独立数据库 |
| status | 状态 | int | -3：到期  -2：禁用  -1：删除  0：尚未激活  1：启用 |
| createUserId | 创建者 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateDatetime | 更新时间 | Date |  |
| maxMembers | 最大成员数 | int | 最大加入用户数 |
| trialDays | 试用期限 | int | 试用天数 |
| expiredDate | 租赁到期日 | Date |  |
| price | 购买价格 | int |  |
| logoUrl | logo链接地址 | String |  |
|  |  |  |  |
|  |  |  |  |

## 租户成员信息（TenantMember）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 租赁ID |
| tenantId | 租户ID | int |  |
| appId | 应用ID | int |  |
| userId | 用户ID | int |  |
| status | 状态 | int | -1：禁用  1：启用 |
| isAdmin | 是否管理员 |  |  |
| createUserId | 创建人 |  |  |
| createDatetime | 创建时间 |  |  |
| updateUserId | 更新人 |  |  |
| updateDatetime | 更新时间 |  |  |
|  |  |  |  |
|  |  |  |  |

## 应用商店（AppStore）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer |  |
| uuid | 唯一ID | String |  |
| code | 编号 | String | 由系统自动生成，方便管理，如：CRM-001 |
| key | 授权KEY | int | 系统根据应用特征自动生成的key，应用在调用平台API时必传参数 |
| password | 授权密钥 | String | 应用在调用平台API时必传参数 |
| name | 名称 | String |  |
| type | 类型 | int |  |
| authType | 授权方式 | int |  |
|  |  |  |  |
|  |  |  |  |

## 参数字典类型（DictParamType）

参数字典分类表，在参数字典可配置时才有用。

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer |  |
| dictType | 参数类型 | String | 统一编码 |
| dictTypeName | 名称 | String |  |
| alterMode | 更改方式 | int | 1：不可修改  2：可增加  3：可修改  4：可删除 |
| status | 状态 | int | 1：正常  2：屏蔽  9：系统 |
| remark | 备注 | String |  |
|  |  |  |  |

## 参数字典（DictParam）

参数字典

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer |  |
| dictType | 参数类型 | String | 统一编码 |
| dictValue | 参数值 | String |  |
| dictContent | 显示内容 | String | 国际化时通过type和value映射，该属性仅作为默认值 |
| position | 排序 | int |  |
| status | 状态 | int | 1：正常  2：屏蔽  9：系统 |
| remark | 备注 | String |  |

## 租户参数字典（TenantDictParam）

由租户自行定义的参数字典，仅在租户范围内有效

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer |  |
| tenantId | 租户ID | int | 租户隔离 |
| dictType | 参数类型 | String | 统一编码 |
| dictValue | 参数值 | String |  |
| dictContent | 显示内容 | String |  |
| position | 排序 | int |  |
| status | 状态 | int | -1：删除  1：正常 |
| remark | 备注 | String |  |

## 菜单（Menu）

通过菜单统一管理用户可使用的功能模块的接入，所有功能模块的跳转不使用直接URL，而是基于菜单的URL，以此实现权限校验、访问统计等功能。同时基于菜单可由用户实现一定程度的个性化，如菜单排序，快捷菜单，分类菜单。

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer |  |
| appId | 所属应用ID | int | 关联具体应用，为0则为默认系统菜单 |
| parentId | 父菜单 | int |  |
| code | 菜单编号 | String |  |
| name | 名称 | String | 默认名称，国际化后通过code获得国际化名称 |
| url | 实际URL | String |  |
| isLeaf | 是否叶子节点 | Boolean |  |
| status | 状态 | int |  |
| position | 位置 | int |  |
| iconUrl | 图标URL | String |  |
| imageUrl | 图像URL | String |  |
|  |  |  |  |
|  |  |  |  |

## 角色（Role）

系统预置全局角色：

1：personal，用户开通后即拥有，包含个人事务中的所有基本功能。

2：tenantLevel1，一级租户，用户开通租户后获得app的对应权限。

3：tenantLevel2，二级租户，用户开通租户后获得app的对应权限。

4：tenantLevel3，三级租户，用户开通租户后获得app的对应权限。

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int | 该值为0时表示全局角色 |
| name | 名称 | String |  |
| status | 状态 | int |  |
| remark | 备注 | String |  |
|  |  |  |  |

## 用户角色（UserRole）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| userId | 用户ID | int |  |
| roleId | 角色ID | int |  |
|  |  |  |  |
|  |  |  |  |

## 用户组角色（UserGroupRole）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| groupId | 用户组ID | int |  |
| roleId | 角色ID | int |  |
|  |  |  |  |
|  |  |  |  |

## 角色功能权限（RoleFuncPermission）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户id | int |  |
| roleId | 角色id | int |  |
| permissionCode | 权限编号 | String |  |
|  |  |  |  |

## 激活码（ActivationCode）

由系统生成全局唯一的激活码，用于在用户注册、找回密码等模块中进行数据验证，并匹配相应的用户。同时在激活码中保存相关业务参数，可在激活时通过这些参数获取相关对象信息。

目前主要的激活码有三种：

* 一种是通过UUID生成的长度为36位（去掉横线为32位）的字符串，生成方式较简单，但输入麻烦，主要用于邮件激活，找回密码中；
* 一种是通过一定算法生成的较简短的字符串，通常是6到8位，可由用户输入；但由于存在重复性，在不同批次间需要校验重复性；
* 第三种是生成4-6位的随机数，用户输入体验好，但由于重复性加大，需要用户同时输入邮箱地址，登录名等其他辅助验证的信息。

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | String | UUID |
| userId | 用户ID | Integer |  |
| moduleCode | 模块 | String | register  forgetPassword  join |
| params | 动态参数 | String | JSON格式存放，KEY-VALUE模式，可直接转成Map |
| securityKey | 安全密钥 | String | 备用，用于增强安全性 |
| expiredDatetime | 过期时间 | Date |  |

## 快速查询视图(QuickView)（暂未实现）

通过保存填写的查询条件并映射到一个视图上，用户无需每次反复填写查询条件而只用选择一个视图，实现查询结果的快速过滤。

视图有两种：一种是系统预置的，所有人都能看到；一种是用户自定义的，只有创建的人才能看到

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| name | 名称 | String | 登录名 |
| filterString | 过滤条件字符串 | String |  |

# 系统辅助数据模型

主要包含一些非核心的功能

## 附件(Attachment)

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户id | int |  |
| moduleCode | 模块编号 | String | 模块使用时通过该值过滤 |
| recordId | 数据记录ID | int | 关联具体数据 |
| displayName | 显示名称 | String |  |
| fileName | 文件名 | String |  |
| filePath | 文件路径 | String | 相对HOME.PATH的路径 |
| createDatetime | 创建时间 | Date |  |
| createBy | 创建人 | int |  |
| updateDatetime | 更新时间 | Date |  |
| updateBy | 更新人 | int |  |

## 标签(Tag)

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户id | int |  |
| type | 类别 | String | Task |
| name | 名称 | String |  |
| totalCount | 关联对象数 | int |  |
| createDatetime | 创建时间 | Date |  |

## 用户评论(UserComment)

统一存放各模块的用户评论，可附加到具体的模块中。

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户id | int |  |
| dataId | 数据ID | int |  |
| moduleCode | 模块编号 | String | 对应模块的英文名：  customer  custContacts  custContactLog  Task  Topic |
| content | 内容 | String |  |
| createDatetime | 创建时间 | Date |  |
| createUserId | 创建用户ID | int |  |

# 消息中心数据模型

## 消息中心(UserMessage)

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户id | int |  |
| userId | 所属用户 | int | 当为0时表示系统消息 |
| moduleCode | 模块编号 | String | 对应模块的英文名： |
| params | 参数 | String | JSON格式，只存属性，呈现时再根据模板动态组织 |
| createUserId | 创建用户ID | int | 当为0时表示系统消息 |
| createDatetime | 创建时间 | Date |  |

## 短信发送日志（SmsOut）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | Int |  |
| sender | 发件人 | String |  |
| receiver | 收件人 | String |  |
| content | 内容 | String |  |
| sendFlag | 发送标志 | boolean |  |
| createDatetime | 创建时间 | Date |  |
| sendDatetime | 发送时间 | Date |  |
|  |  |  |  |

## 短信接收日志（SmsIn）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | Int |  |
| sender | 发件人 | String |  |
| receiver | 收件人 | String |  |
| content | 内容 | String |  |
| createDatetime | 创建时间 | Date |  |
|  |  |  |  |

# 个人事务数据模型

## 站内信（SHORT\_MESSAGE）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| title | 标题 | String |  |
| type | 类型(2001) | int | 1：系统  2：用户 |
| content | 内容 | String |  |
| createUserId | 创建用户 | int |  |
| createDatetime | 创建时间 | Date |  |
| receiveUserId | 接收人 | int |  |
| receiveFlag | 接收标志 | Boolean |  |
| receiveDatetime | 接收时间 | Date |  |
|  |  |  |  |

## 记事本（NOTE）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| title | 标题 | String |  |
| content | 内容 | String |  |
| createUserId | 创建人 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateDatetime | 更新时间 | Date |  |
|  |  |  |  |

## 代办事项（TODO）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| content | 内容 | String |  |
| status | 状态(2101) | int | -2：推迟  -1：取消  1：未开始  2：正在进行  3：完成 |
| isNeedRemind | 是否需要提醒 | Boolean |  |
| remindDatetime | 提醒时间 | Date |  |
| todoDatetime | 代办时间 | Date | 通过该字段过滤和排序 |
| createUserId | 创建人 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateDatetime | 更新时间 | Date |  |
| iconUrl | 图标 | String |  |

## 联系人（CONTACTS）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| subject | 主题 | String |  |
| content | 内容 | String |  |
| assignee | 受派人 | int |  |
| dueDate | 截止日期 | Date |  |
| tags | 标签集 | String |  |
| createDatetime | 创建时间 | Date |  |
| createUserId | 创建人 | int |  |
| updateDatetime | 更新时间 | Date |  |
| updateUserId | 更新人 | int |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 团队协作数据模型

## 项目（Project）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| name | 标题 | String |  |
| descr | 内容 | String |  |
| status | 状态 | int |  |
| tags | 标签集 | varchar(255) | 用逗号分隔的标签ID，用于根据任务查询标签，长度有限，限制最多5个标签 |
| createUserId | 创建用户 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateUserId | 更新人 | int |  |
| updateDatetime | 更新时间 | Date |  |
|  |  |  |  |

## 项目标签（ProjectTag）

项目与标签关系表

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| projectId | 项目ID | int |  |
| tagId | 标签ID | int |  |
|  |  |  |  |

## 项目组（ProjectUserGroup）

项目与用户组关系表，只有授权的用户组才能看到项目相关的数据。

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| projectId | 项目ID | int |  |
| userGroupId | 用户组ID | int |  |
|  |  |  |  |

## 讨论（Topic）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| projectId | 项目ID | int |  |
| title | 标题 | String |  |
| content | 内容 | String |  |
| createUserId | 创建用户 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateUserId | 更新人 | int |  |
| updateDatetime | 更新时间 | Date |  |
|  |  |  |  |

## 任务（Task）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| projectId | 项目ID | int | 对于日常工作没有项目的，可创建一个日常工作的项目使结构统一 |
| title | 标题 | String |  |
| content | 内容 | String |  |
| assignee | 受任人 | int |  |
| priority | 优先级 |  | 0：Suggestion  1：Minor  2：Major  3：Critical  4：Blocked |
| status | 状态 | int | -3：重开  -2：中止  -1：取消  0：创建  1：进行中  2：暂停  3：完成 |
| tags | 标签集 | varchar(255) | 用逗号分隔的标签ID，用于根据任务查询标签，长度有限，限制最多5个标签 |
| createUserId | 创建用户 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateUserId | 更新人 | int |  |
| updateDatetime | 更新时间 | Date |  |
|  |  |  |  |
|  |  |  |  |

## 任务标签（TaskTag）

任务与标签关系表

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| taskId | 任务ID | int |  |
| tagId | 标签ID | int |  |
|  |  |  |  |

## 知识空间（KNOWLEDGE\_SPACE）

知识库空间，用于知识的分类管理

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| code | 编号 | String |  |
| name | 名称 | String |  |
| descr | 详细描述 | String |  |
| createUserId | 创建用户 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateUserId | 更新人 | int |  |
| updateDatetime | 更新时间 | Date |  |
|  |  |  |  |

## 知识页（KNOWLEDGE\_PAGE）

记录具体内容的页面

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| spaceId | 知识空间ID | int | 关联KNOWLEDGE\_SPACE |
| parentId | 父页面ID | int |  |
| subject | 主题 | String |  |
| brief | 摘要 | String |  |
| keywords | 关键字 | String |  |
| content | 内容 | String |  |
| createUserId | 创建用户 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateUserId | 更新人 | int |  |
| updateDatetime | 更新时间 | Date |  |
|  |  |  |  |

# CRM数据模型

## 客户信息（CUSTOMER）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| code | 编号 | String | 自动生成 |
| parentId | 父节点ID | int | 构造集团客户的树型架构 |
| isLeaf | 是否叶子节点 | Boolean |  |
| name | 名称 | String |  |
| email | 电子邮件 | String |  |
| phone | 联系电话 | String |  |
| fax | 传真号 | String |  |
| address | 联系地址 | String |  |
| zipCode | 邮编 | String |  |
| website | 网址 | String |  |
| remark | 备注 | String |  |
| chargeType | 负责人类型 | int | 个人或者组织，用于区分chargeUserId |
| chargeUserId | 负责人ID | int |  |
| contactsId | 默认联系人ID | int | 关联客户联系人 |
| createUserId | 创建人 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateDatetime | 更新时间 | Date |  |
| logoUrl | 图标的URL | String |  |
| isDeleted | 是否已删除 | Boolean |  |
| rank | 客户等级(3001) | int | 1：1星  2：2星  3：3星  4：4星  5：5星 |
| status | 客户状态(3002) | int | 1：潜在  2：有意向  3：失败  4：已流失  5：已成交  6：维护  7：重点维护 |
|  |  |  |  |

## 客户特征信息（CUSTOMER\_FEATURE）

记录客户的特征信息，用于分类统计和市场细分决策。

注：目前暂时由系统预置，后期可增加字典表实现动态维护特征列表

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int |  |
| customerId | 客户ID | int |  |
| code | 特征编号 | String | 全局唯一，建立索引 |
| name | 名称 | String |  |
| dataType | 数据类型 | int |  |
| dataValue | 数据值 | String |  |
| dataValueInt | 整型值 | int |  |
| dataValueDate | 日期值 | Date |  |
| dataValueDecimal | 浮点值 | BigDecimal |  |
| createDatetime | 创建日期 | Date |  |
| updateDatetime | 更新日期 | Date |  |

**特征列表：**

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 描述 | 数据类型 | 备注 |
| source | 客户来源 | int | 1：电话来访  2：老客户  3：客户介绍  4：独立开发  5：媒体宣传  6：代理商  7：合作伙伴  8：公开招标  9：直邮  10：网站  11：展会  12：其他 |
| industry | 所属行业 | int |  |
| companyType | 公司性质 | int | 1：国有企业  2：外资企业  3：民营企业  4：集体企业  5：股份制企业  6：合资企业  7：独资企业  8：其他 |
| foundDate | 成立日期 | Date |  |
| turnover | 营业额 | int | 1：人民币 10 万元/年以下  2：人民币 10 万元/年 - 30 万元  3：人民币 30 万元/年 - 50 万元/年  4：人民币 50 万元/年 - 100 万元/年  5：人民币 100 万元/年以下  6：人民币 200 万元/年 - 300 万元/年  7：人民币 100 万元/年 - 200 万元/年  8：人民币 300 万元/年 - 500 万元/年  9：人民币 500 万元/年 - 700 万元/年  10：人民币 700 万元/年 - 1000 万元/年  11：人民币 1000 万元/年 - 2000 万元/年  12：人民币 2000 万元/年 - 3000 万元/年  13：人民币 3000 万元/年 - 5000 万元/年  14：人民币 5000 万元/年 - 1 亿元/年  15：人民币 1 亿元/年以上 |
| employees | 员工数 | int | 1：5 - 10 人  2：11 - 50 人  3：51 - 100 人  4：101 - 200 人  5：201 - 300 人  6：301 - 500 人  7：501 - 1000 人  8：1000 人以上 |
| credit | 信用等级 | int | 1.款到发货；  2.月结30天，  3发货后7日内；  4发货后30日内，  5，发货后60日内.  6 其他 |
| region | 区域 | String | 省市名称 |
| segment | 客户类型 | int | 1经销商  2机加工,  3模具,  4船舶，  5半导体，  6电子，  7航空  8 其他 |

## 客户联系人（CUST\_CONTACTS）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int | 关联租户 |
| customerId | 客户ID | int | 关联客户信息 |
| name | 名称 | String |  |
| gender | 性别 | int | 0 - unknown  1 - male  2 - female |
| mobile | 手机号 | String |  |
| email | 电子邮件 | String |  |
| phone | 联系电话 | String |  |
| fax | 传真号 | String |  |
| address | 联系地址 | String |  |
| zipCode | 邮编 | String |  |
| remark | 备注 | String |  |
| createUserId | 创建人 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateUserId | 更新人 | int |  |
| updateDatetime | 更新时间 | Date |  |
| remark | 备注 | String |  |

## 客户往来（CUST\_ACTIVITY）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 含义 | 数据类型 | 备注 |
| id | 序号，主键 | Integer | 由数据库自动生成 |
| tenantId | 租户ID | int | 关联租户 |
| customerId | 客户ID | int | 关联客户信息 |
| contactId | 联系人 | int | 关联联系人 |
| subject | 主题 | String |  |
| content | 内容 | String |  |
| actType | 往来类型 | int |  |
| chargeUserId | 负责人 | int |  |
| actDate | 往来日期 | Date |  |
| createUserId | 创建人 | int |  |
| createDatetime | 创建时间 | Date |  |
| updateUserId | 更新人 | int |  |
| updateDatetime | 更新时间 | Date |  |

# Framework设计

## 分布式

分布式Web集群应无状态，使得请求可以发送给任一服务器做处理，同时后台定时任务应剥离，避免重复处理。

## 分布式Session(SNA)

在多用户高并发环境下，单台Web服务器的处理能力有限，当需要增加Web服务器实现水平集群扩展实现高可用性的话，用户状态的共享处理就成了难点，早先的做法是通过Web服务器的Session复制，实现不同服务器间的会话共享，然而高昂的复制成本、低下的复制效率、有限的扩展能力使得该方案无法有效解决海量用户高并发环境。

**关键点：**

* 集中式：通过Memcached或者Redis实现集群环境下Session缓存和共享；同时在Session失效或者服务器集群减少时能有效的清除过期缓存。
* 高可用：通过主备实现Cache高可用；
* 无状态：通过cookie记录用户登录状态，由客户端浏览器维持状态，而不是Web服务器；不再使用Web服务器提供的Session管理，Web服务器无状态以实现水平扩展；
* 安全性：通过附加时间戳和对cookie加密，实现数据传递的安全性；通过数字签名实现对篡改数据的校验；
* 负载均衡：各个服务器的负载相当，请求被合理的调度；

### Sticky Session、Non-sticky Session和Replicated Sessions

* Sticky Sessions：粘性会话。即同一个会话中的请求必须被转发到同一个节点上，除非该节点宕机才转发到故障转移节点。一个节点宕机，所存储的 Sessions 完全丢失。通俗的话就是，将用户“粘”在某一个服务器节点上。
* Non-Sticky Sessions：非粘性会话。每一次请求都可能转发到不同节点。
* Replicated Sessions：把一个节点上的 Sessions 复制到集群的其他节点上，防止数据丢失，允许失效无缝转移。如node 0复制到node 5，node 1复制到node 6，以此类推。多数应用服务器（如 Tomcat ）都支持会话复制机制。

### 基于 ZooKeeper 集群的分布式 Session 方案

要解决基于 memcache 方案的数据丢失问题，可以引入持久化存储介质 ZooKeeper（下面简称 ZK）。

依托于 ZK 的一致性复制（在多个副本间保证数据的强一致性）和容错能力，结合上面的 MSM 思想，

由 ZK 负责 session 数据的存储，而我们自己实现的 session manager 将负责 session 生命周期的管理。



### 基于Cookie的分布式SessionId

cookieValue：userId+clientIp+timestamp+expiry

如果返回的Cookie值不为null，则有3种可能性：其一，已经实例化过一个Session对象并且可以正常使用；其二，虽然已经实例化过了，但是可能此Session已经超时失效；其三，分布式环境中的其他子系统已经实例化过了，但是本系统中还未实例化过此Session对象。

### 问题

#### 浏览器关闭再打开，由于是通过cookie判断的，仍然会是登陆状态。

#### SessionID的生成

SessionID由Web服务器生成，对单个服务器来说是唯一的，但是集群环境无法保障，需要有效的算法重算sessionId。

#### 缓存中session的失效机制

Session是有生命周期的，当Session的空闲时间(maxIdle属性值)超出限制时，Session就失效了，这种设计主要是考虑到了Web容器的可靠性。当一个系统有上万人使用时，就会产生上万个Session对象，由于HTTP的无状态特性，服务器无法确切的知道用户是否真的离开了系统。因此如果没有失效机制，所有被Session占据的内存资源将永远无法被释放，直到系统崩溃为止。在分布式环境下，Session被简单的创建，并且通过某种机制被复制到了其他系统中。你无法保证每个子系统的时钟都是一致的，可能相差几秒，甚至相差几分钟。当某个Web容器的Session失效时，可能其他的子系统中的Session并未失效，这时会产生一个有趣的现象，一个用户在各个子系统之间跳转时，有时会提示Session超时，而有时又能正常操作。因此我们需要一种机制，当某个系统的Session失效时，其他所有系统的与之相关联的Session也要同步失效。

#### 负载均衡

Sticky Session机制，同一session的请求应被分配到同一服务器。

## 用户、部门、组织、角色与权限

* 用户：每个在系统中注册并可登陆的个体称为一个用户，通常对应到一个人。
* 组织：包含一个或多个相关或相似用户的集合称为一个组织，如一个公司，一个兴趣小组，一个技术圈子。
* 部门：是一种特殊的组织，其存在于企业内部，部门与部门之间通常构成一个父子关系，具有层次结构。

## DynamicQueryObject

通过数据访问层屏蔽数据库结构的细节，实现多表数据的动态组合。通过创建数据访问对象，实现表间关联、分库分表、动态路由、对象映射等高级功能。如需要同时查询用户基本信息和用户扩展信息时，查询语句为：SELECT ID, USER\_NAME, EMAIL, LAST\_LOGIN\_IP, TOTAL\_LOGIN\_TIMES, AREA, TOTAL\_FRIENDS FROM…其中ID, USER\_NAME和EMAIL在基础表中，LAST\_LOGIN\_IP, TOTAL\_LOGIN\_TIMES在状态表中，AREA在地区表中，TOTAL\_FRIENDS在统计表中，其实际的SQL可能是：SELECT USER\_NAME, EMAIL, LAST\_LOGIN\_IP, TOTAL\_LOGIN\_TIMES, AREA, TOTAL\_FRIENDS FROM T1 LEFT JOIN T2 ON… LEFT JOIN T3 ON…

由于屏蔽了数据库结构，因而我们可以很容易的在后面实现数据库优化。同时注意该实现的局限性，主要还是用于查询。

## FuncInceptor

request中隐含参数:

\_\_moduleCode

\_\_funcCode

## 电子邮件服务

提供统一的电子邮件发送接口，支持简单文本内容，HTML内容，带图片内容，带附件等不同方式的邮件发送。

**功能点：**

* 定时邮件：指定时间发送，支持单次发送和周期发送（如每月、每季度、每年）。
* 群发邮件

## 短信服务

提供统一的短息发送接口，支持长短息。

**功能点：**

* 定时短信：指定时间发送，支持单次发送和周期发送（如每月、每季度、每年）。
* 群发短信

## 模板服务

提供基于Freemarker模板工具的统一内容生成服务，可用于格式化的动态内容输出，数据导出，电子邮件内容模板等业务。

## Tags

为特定数据打上标签，可以通过标签对数据进行分类，从而实现分类查询、汇总、统计。

tag创建后存在tag表里，全局唯一，便于统一维护和查询，业务表中增加tags字段，类型为字符串，采用逗号分隔的tagId，根据字符串长度限制一条记录关联的tag数（实际上也不可能为一条数据关联无数个标签）。

聚合：包含A或者包含B

交叉：既包含A又包含B

# 运维&实施

# 应用场景及解决方案

## 用户账号激活

通过点击激活邮件中的链接地址（有效期24小时），用户可自行激活其账号。

**业务流程：**

* 删除激活码；
* 用户状态激活
* 自动赋予默认全局角色的权限（目前包含个人事务的所有权限）；
* 自动登录；
* 跳转到主页；

## 租户开通

默认开通的用户即为app管理员，获得该app所有功能模块的权限。该用户可以邀请其他用户使用该app，并可为其配置相应的用户权限。

**业务流程：**

* 根据订单号读取订单信息（App信息，功能权限授权信息，用户信息）；
* 创建租户，将订单用户作为租户管理员加入租户成员信息中；
* 创建租户角色，将功能授权授权给该角色；
* 将角色赋予订单用户；
* 如果当前登录用户为订单用户，将租户信息和权限信息直接赋予当前用户，避免为刷新权限而需要重复登录；
* 标识订单为已开通；

## 租户注销

由租户承租人（创建者）主动注销承租的某个工作空间，注销后租户中所有成员（包括创建人、管理员、成员）将失去对该空间的访问权限和数据操作权限。

**业务流程：**

* 检查用户是否为承租人（创建人），只有承租人才能注销
* 读取租户成员信息，删除他们的租户权限，并删除成员信息，使其完全退出。（注：暂时不删除业务数据）
* 设置租户空间状态为注销（逻辑删除，保留数据）。

## 用户登录

### 非租户

### 租户管理员

租户管理员拥有所租赁app的相应功能模块（这些模块由系统管理员分配而不是全部）的权限。其可在当前租赁空间中为其他用户分配范围内的权限。

**流程：**

1. （前段流程同非租户）；
2. 获取租户表的信息（tenantId，是否租户管理员，租赁期限等）并注入到登录对象中；
3. 获取其所拥有的角色权限及所在组拥有的角色权限，并注入到登录对象中；

### 租户成员

**流程：**

1. （前段流程同非租户）；
2. 获取租户表的信息（tenantId，是否租户管理员，租赁期限等）并注入到登录对象中；
3. 获取其所拥有的角色权限及所在组拥有的角色权限，并注入到登录对象中；

## 邀请用户

注：只有租户拥有者或者租户管理员才有权限邀请用户。

**流程：**

1. 输入目标用户的邮件地址，选择一个角色给该用户（目标用户获得该角色对应权限）；
2. 系统发送邀请邮件给目标用户，目标用户可通过点击邮件中的超链直接访问网站；
3. 如果目标用户是注册用户，则额外发一条用户消息给该用户（UserMessage），目标用户可通过消息中的链接直接开通；

## 加入邀请(邮件)

注：开通受邀时，如果用户账号已属于某个租户，则无法加入。

**流程：**

1. 点击邀请邮件中的链接；
2. 系统判断该用户是注册用户还是非注册用户；
   1. 如果是注册用户，则跳转到确认页面，输入登录密码，激活邀请并自动登录。
   2. 如果是非注册用户，则跳转到注册页面（帐号自动填入邮件地址），用户进行注册，注册成功后自动开通邀请；

## 切换工作空间

受限于数据隔离要求，用户一次只能在一个工作空间范围内工作，当一个用户成为多个租户空间的成员后，需要设定当前使用哪个工作空间，并可以随时在多个工作空间中切换。

**流程：**

1. 点击切换工作空间；
2. 在下拉列表中选择当前使用的工作空间；
3. 确认

# 功能模块设计

## 后台系统管理

### 参数字典

对一些常用的静态枚举类数据（如：性别字典[男、女]，城市字典[上海、北京、无锡]）进行统一的管理，实现数据值与显示名称的映射。用户在界面操作时以选择框的形式展现。

### 在线用户管理

对登录到系统的客户端会话的统一管理，可监控并强制注销某些非法的客户端登陆。

用户登录、退出时，通过OnlineOperatorListener监听器监听HttpSession的状态变化，从而获取到Session中保存的登陆操作员信息，并将其缓存到OnlineManager管理器中。应用程序可以通过获取的OnlineManager默认实例来获取相关的操作功能，可以获取某个操作员的在线情况，可以获取所有在线操作员的列表，还可以强制某个操作员下线。

**注意：**

#### 强制用户下线

强制用户下线实际是通过一个客户端请求去控制另一个客户端，由于Web客户端是无状态的，这里操作的本质是通过为受控客户端的会话中注入控制指令，在该客户端下次访问时触发下线操作。

具体流程：

1. 通过OnlineManager获得登录用户列表信息；
2. 通过UUID获得某个会话下的用户信息；
3. 设置该会话用户的状态为强制下线(forceOffline)；
4. 在用户下次请求时，通过会话管理过滤器获得当前用户信息，判断强制下线标志，若为强制下线，则清除Session中的用户信息，并注销session；同时如果用户设置了自动登录，则还需清空其cookie中的自动登录令牌；

### 租户开通管理

当用户租用了某个app时，由系统自动或者由管理员为其开通相应功能的权限。

### 数据清理

对于一些历史数据或者系统无法自动清理的数据，由后台定时的根据规则进行清理，防止垃圾数据的产生和堆积。

|  |  |  |
| --- | --- | --- |
| 数据对象 | 清理规则 | 注释 |
| ActivationCode | 每日凌晨删除过期的 | 激活码为一次性数据，用过即需删除。  也可采用非InnoDB，无需事务支持 |
|  |  |  |

## 前台

### 基础模块

#### 用户注册

用户填写注册信息后提交，系统记录下该用户的相关信息，作为登录凭证。用户注册成功后，会通过系统邮件发送一封帐户激活的邮件，让用户通过激活链接激活帐户方能登录系统。

激活邮件的关键在于激活码的生成。为避免恶意用户不断注册无效帐户，激活链接中的激活码与注册用户信息绑定，只有匹配的激活码才能激活用户账户。

#### 用户登录

#### 用户档案

#### 忘记密码

用户忘记密码时可通过填写注册时使用的电子邮箱地址接收找回密码的邮件。该邮件中包含一个URL，通过UUID到数据库中匹配记录，数据库中记录了请求的过期时间（expiredDatetime），过期则需要重新发送邮件。

#### 用户登出

#### 自动登录

注：目前的token仅存在于客户端的cookie中，仅通过算法加密保障安全性；若需要进一步加强安全性，可在服务器端也存储一份，用于数据核对。

用户在登录时选择“下次自动登录”可实现必须输入用户名密码而自动登录到系统中。实现机制为在客户端浏览器的cookie中记录用户的登录信息。

cookie值实现：根据自动登录时客户端IP，用户登录名，加密后的密码，创建时间构造token，token将使用DES加密并使用BASE64编码，最后存放到cookie中。

自动登录实现：检查是否存在自动登录的cookie，若存在则把value进行解码，获得token中的属性值，根据属性值检查登录的有效性，若有效则读取用户信息并自动登录，若无效则清除cookie中的值，并跳转到登录页面，告诉用户自动登录已过期，需要重新登录。

**自动登录失效规则：**

* 客户端IP变更：请求的IP与token中的不一致。（可增强安全性，但是会影响用户体验，如家里到公司，IP变化了，但仍旧是同一台机器，默认不启用）；
* 用户状态不一致：用户被锁定、删除等无法登录的状态时，自动登录也将失效；
* 密码修改：数据库中密码修改后与cookie中缓存的密码不一致；
* 过有效期：根据创建时间和过期时间计算有效期，超过则失效（使用cookie时可用cookie本身的机制，这里用于加强安全性，防篡改）；

#### 邀请用户

用户可通过给其他用户发出邀请，要求其他用户加入到自己的租户空间，实现功能共享和数据共享。邀请方式是通过输入目标用户的邮件地址。

#### 共享APP（暂未实现）

注：考虑当用户邀请其他用户形成组织后，是否自动让其他用户获得当前用户租用的app使用权限。

目前认为自动共享，无需用户设置权限，今后如有需要则增加用户手动开通的功能。如果不开通，两个人之间只能分享个人事务中的功能，比如互相发站内信，TODO协作。

### APP租用

#### 租用APP

在APP列表中，根据需要选择租用某个APP，选择租用方式，付费后APP将自动开通。

#### 已购APP

查看已购APP的列表，每个APP的授权期限，可付费延期或付费升级更多服务或者空间。

### Dashboard

以小组件的模式，可将主要功能模块的简化版数据由单一界面呈现，方便用户在日常使用中，通过该单一界面方便的处理最新的，重要的，或者只有其关心的数据。

Dashboard是用户进入系统后的默认界面，内部的小组件完全由用户自定义。如：老板的Dashboard中主要以报表模块为主，销售的Dashboard中主要以客户管理模块为主。

### 消息中心

汇聚各个模块产生的，用户关心的消息。如“系统消息”，“团队消息”，“被关注消息”，“回复消息”等。消息内容需支持HTML格式，支持超链跳转到具体模块。

#### 短信中心

记录某个租户收发短信的详细日志

### 个人事务管理

#### 站内信

用户可发送简单的留言给其他用户。

#### 记录本

#### TODO List

代办事项，GTD理念，简单。用户通过指定TODO的状态来区分TODO。另外，通过todoDatetime可按时间序列过滤和排序TODO

功能点：

* 到期提醒；
* 导入/导出；

#### 联系人管理

目前仅实现单表的固定字段管理，后续可通过属性表实现扩展性、个性化。

#### 高级搜索（暂未实现）

单一界面全文搜索其他功能模块关键字，列表方式呈现。

### 租户系统管理

#### 租户管理

显示当前所上在租户的详细信息，包括租户名称、创建人、创建时间、租赁期限，功能集等，成员数，访问次数等统计信息。可修改一些全局配置，如名称，邮件服务器信息，短信服务器信息，文件上传下载等，可对UI做一定程度的自定义，并可对租户进行锁定、注销、续费、升级等操作。

功能点：

#### 组织架构

用户自行创建符合团队或企业需要的组织架构，一个成员可以属于一个或多个组。

#### 成员管理

某些用户的创建和管理不是通过前台用户注册模块，而是以企业为单位，由企业的系统管理员统一创建和管理。这些用户由于同属同一个企业或组织，具有一些相同的特性（如员工编号，企业邮箱等）。作为企业内用户管理，其类似企业邮箱的管理，不同企业间相互隔离，不同企业间的用户名可以重复，应支持类似superadmin，admin之类的用户名。

注：目前采用Email作为登录名，通过邮件地址可直接实现。

# 团队协作系统

## 项目管理

项目管理

## 任务管理

团队成员协作完成任务。可以为自己或他人指派任务，不指派特定人的任务将放入任务池，由人认领。

## 主题讨论

主题讨论

# CRM子系统

## 客户管理

### 客户信息管理

### 联系人信息管理

### 往来记录

记录每次与客户往来的记录（如：电话往来，客户拜访），用于跟踪销售行为和客户行为。往来记录由移动终端或者网页手工录入。

### 客户关怀

主动的客户关怀，可以为不同客户设置客户关怀计划（如：定期关怀，纪念日关怀），可以由系统通过短信、邮件方式关怀，也可由人员线下关怀后将关怀情况手工录入到系统中。

## 销售管理

### 销售机会

录入可能的销售机会，指派给某个员工。

### 客户池

客户池是一种全新的压力型销售管理工具，主要特点是每个销售人员可以领取一定数量内的客户池客户，在期限内必须达成签约或回款，否则客户就会自动回归客户池重新分配给其他销售人员

## 客服管理

### 咨询投诉

统一管理客户的咨询和投诉。

数据来源有：手工录入，CMS系统。

# 企业门户网站子系统(CMS)

发布企业公告、企业新闻，通过信息公开和信息交互，展现企业实力，加强客户交流。CRM是内部客户关系管理，CMS则是外部客户关系管理。

# 统计报表子系统

综合报表和年终报表，可以从公司角度统计公司的销售、采购、库存信息，通过年终报表可以了解公司的整年销售情况，以及和去年相比，根据这些统计报表制定公司的发展规划；

# 附注

## 参数字典表清单

### 通用

#### 性别（0002）

* 0：未知
* 1：男
* 2：女

#### 语言（0003）

* en\_US：英语
* zh\_CN：简体中文

#### 时区（0004）

* -1：-1
* 0：0
* 1：+1
* …
* 8：+8

#### 主题（0005）

* default：默认
* office：office

### 用户表

#### 用户类型（1001）

* 1：个人用户
* 2：企业用户

#### 用户状态（1002）

* 0：未审核
* 1：正常
* 2：注销
* 3：删除
* 4：锁定

### 站内信

#### 类型（2001）

* 1：系统短信
* 2：用户短信

### TODO

#### 状态（2101）

* -2：推迟
* -1：取消
* 1：未开始
* 2：正在进行
* 3：完成

### 日程计划

### 客户信息

#### 客户等级（3001）

* 1：1星
* 2：2星
* 3：3星
* 4：4星
* 5：5星

#### 状态（3002）

* 1：潜在
* 2：有意向
* 3：失败
* 4：已流失
* 5：已成交
* 6：维护
* 7：重点维护

### 项目

#### 状态（4001）

* -2：中止
* -1：取消
* 1：计划
* 2：立项
* 3：启动
* 4：进行中
* 5：完成

### 任务

#### 状态（4101）

* -2：中止
* -1：取消
* 0：创建
* 1：进行中
* 2：暂停
* 3：完成

#### 优先级（4102）

* 0：Suggestion
* 1：Minor
* 2：Major
* 3：Critical
* 4：Blocked