电子发票凭证管理系统-平台服务体系结构设计文档

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| 草稿 | Zrz |  | 2016-10-12 | 开始合并所有的设计文档 |
| 变更 | Zrz |  | 2016-10-14 | 版本1.1A、B，需求相关变更 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 引言

## 目的

本系统用于支持电子发票凭证管理系统中【平台服务】这个主要的服务端业务过程集，平台服务属于电子发票凭证管理系统的子系统（服务端构件簇），以中间件的形式封装电子票凭据核心领域的诸构件，采用消息集成、数据集成实现与企业端、数据提供商的交互 。本文档旨在描述业务系统的组成，功能细节，以及实现细节的各个构件及连接子。

## 定义、首字母缩写词和缩略语

|  |  |
| --- | --- |
| **定义、首字母缩写词和缩略语** | **描 述** |
| DEMO、原型 | 系统在设计之前有静态页面描述的原型系统，页面的展示效果和交互效果请以原型为准 |
| 规格说明 | 指代《E票管理系统\_产品需求文档\_V2.0\_20161008》 |
|  |  |
|  |  |

## 阅读对象

|  |  |
| --- | --- |
| **阅读对象** | **阅读建议** |
| 需求 | 概要设计：请重点阅读第5章节内容，核实与外部系统对接的前期准备及约定是否已完成 |
| 编码 | 除了需要开发的功能点，请重点阅读第5章节中相关的内容 |
|  |  |
|  |  |

## 参考资料

|  |  |
| --- | --- |
| **名 称** | **描 述** |
| 电子发票凭证管理系统\_产品需求文档\_V1.0\_Build\_01\_20160729 |  |
| E票管理系统\_产品需求文档\_V2.0\_20161008 |  |
| 电子发票凭证管理系统V1原型 |  |

## 文档概述

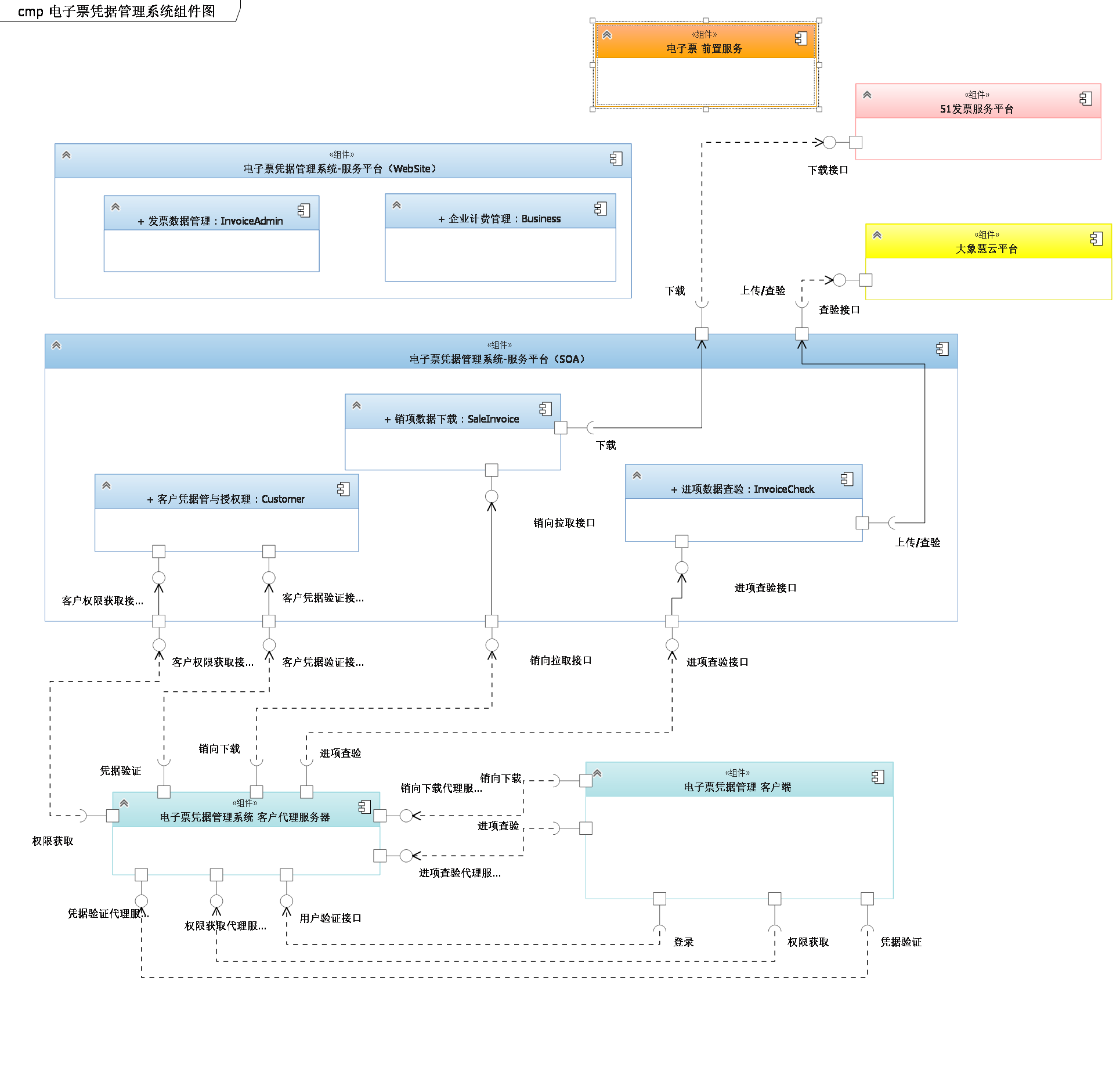
第2章节主要描述系统的整体结构

第3,4章节主要对需求进行分析，并得出设计的相关约束和主要的设计策略

第5章节主要描述系统架构相关的设计内容

# 系统概述

## 功能模块划分



1. 系统总共划分为五个组件：

A自助终端功能；

B社区服务云平台功能；

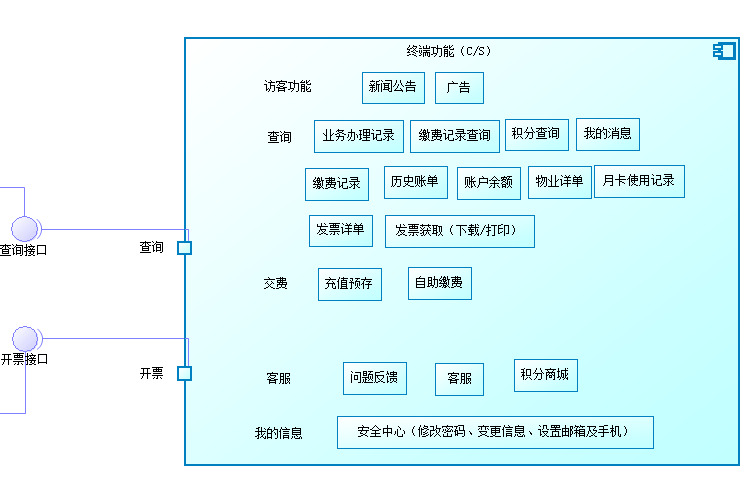
C电子票前置服务；

D物业管理软件；

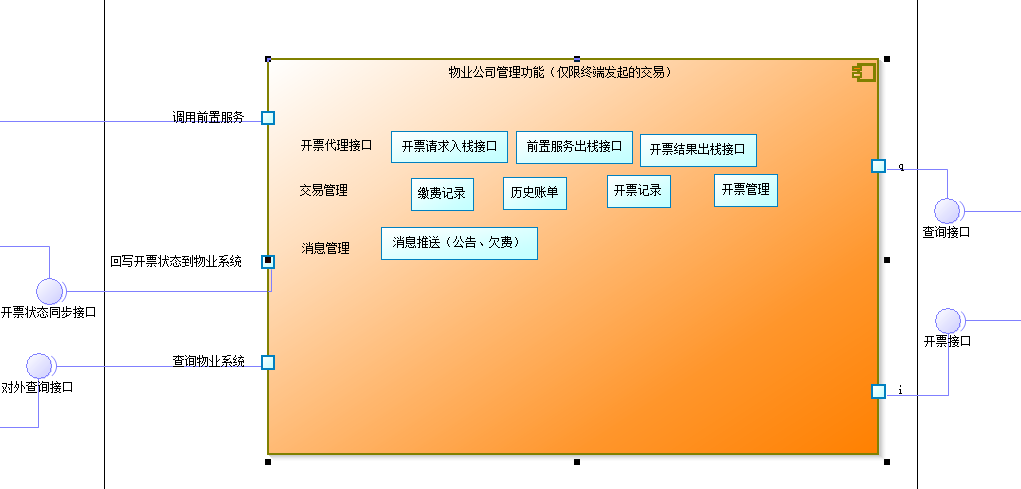
E物业停车场道闸管理软件。

其中D和E为客户既有系统；C 已由深圳航信电子票项目组实现。

1. 自助终端功能主要包括：
2. 访客功能
3. 查询
4. 交费
5. 客服
6. 用户中心

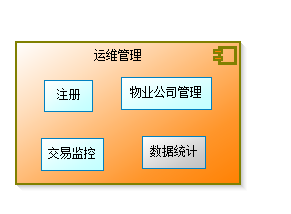


1. 社区服务云平台功能功能主要包括：
2. 开票代理接口
3. 交易管理
4. 消息管理

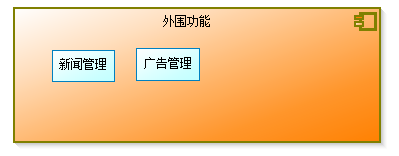


1. 运维管理功能主要包括支持系统运行相关的管理的监控功能。
2. 所有物业公司的基础字典，除了物业公司自己维护之外，客服需要权限进行操作。
3. 运维控制所有的用户注册和公司注册功能。
4. 运维还包括交易数据的监控以及异常处理功能。

系统上线阶段，可能的情况是，基础数据的初始化工作是由物业公司提供，或者从其他系统获取，通过运维的功能初始化到系统中。物业公司仅处理数据的核对和变更工作。



1. 外围功能包括的是网站结构中带有的功能，主要包括
2. 新闻动态。
3. 论坛，暂不实现
4. 广告管理



## 系统集成

1. 物业公司需要维护的信息需要提供导入的接口，以方便物业公司对信息的维护工作；
2. 物业停车场道闸管理系统需提供查询和缴费接口；
3. 物业公司物管系统需提供常规业务查询接口，以便后台管理系统/自助终端获取业务信息；

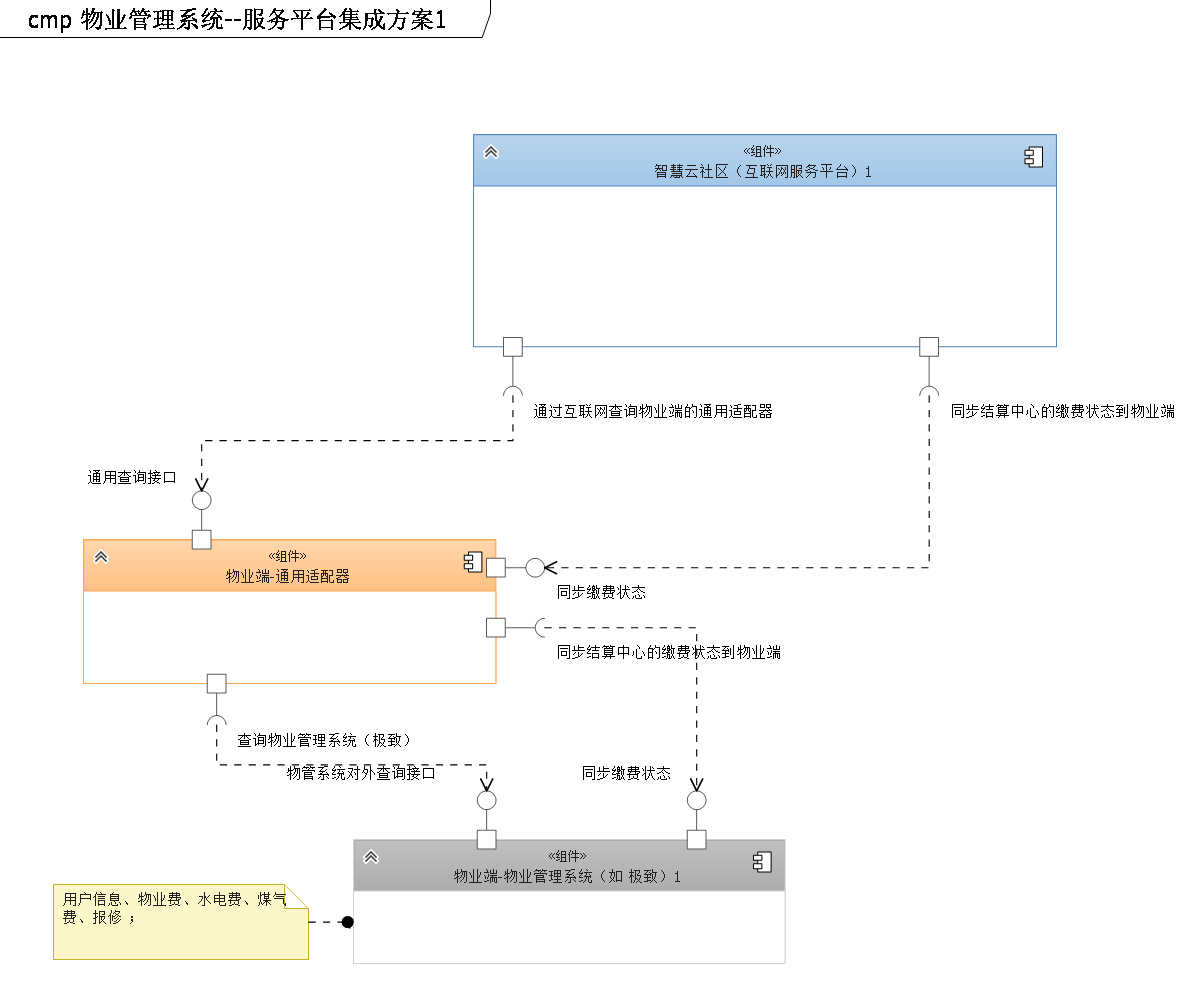
由于“服务平台”与“物业系统”间的集成在业务上的不确定性始终未能消除，此处增加若干备选方案(此文档同时降级为A版——待评审版)，请需求确认后评审确定具体方案!

这里涉及到两个方向的同步：

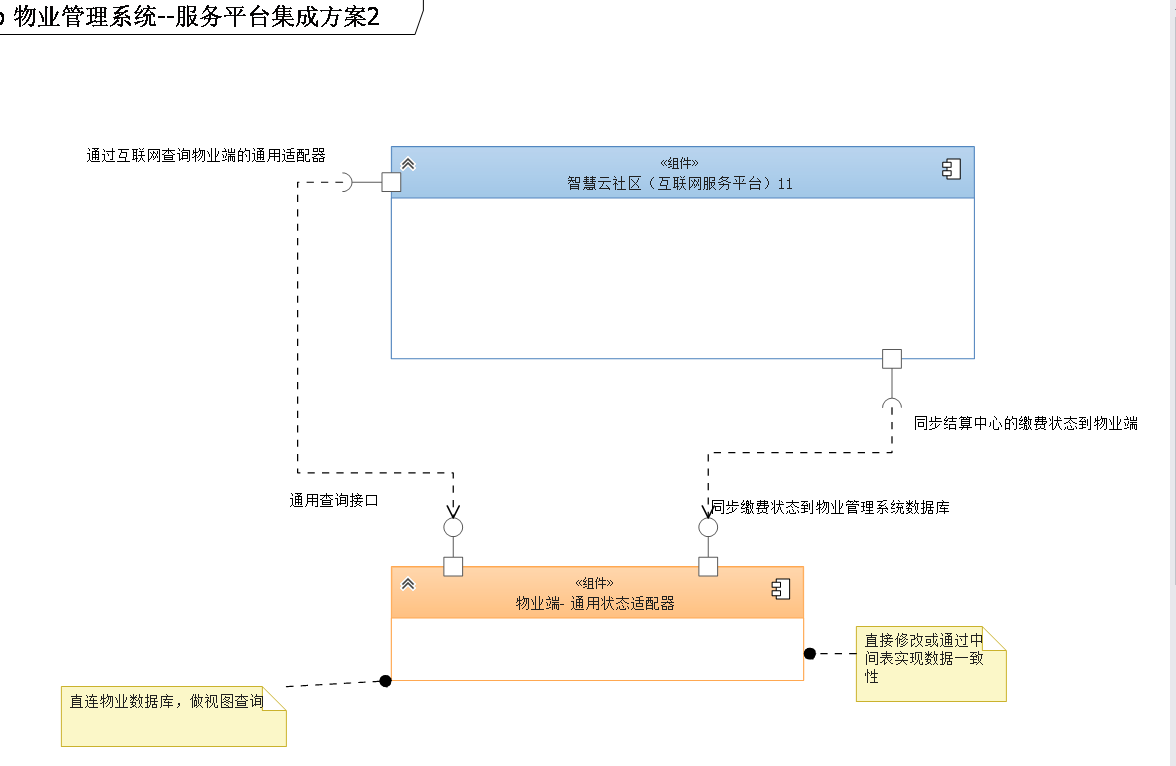
* + - 1. 物业管理系统（如极致）业务初态（欠费、待办等）到云端服务平台的同步；
      2. 云端服务平台业务更新状态（已缴、取消、完成等）到物业管理系统的同步。

方案1 :

此方案属于消息集成的范畴，每次查询数据量不宜过大



方案2：

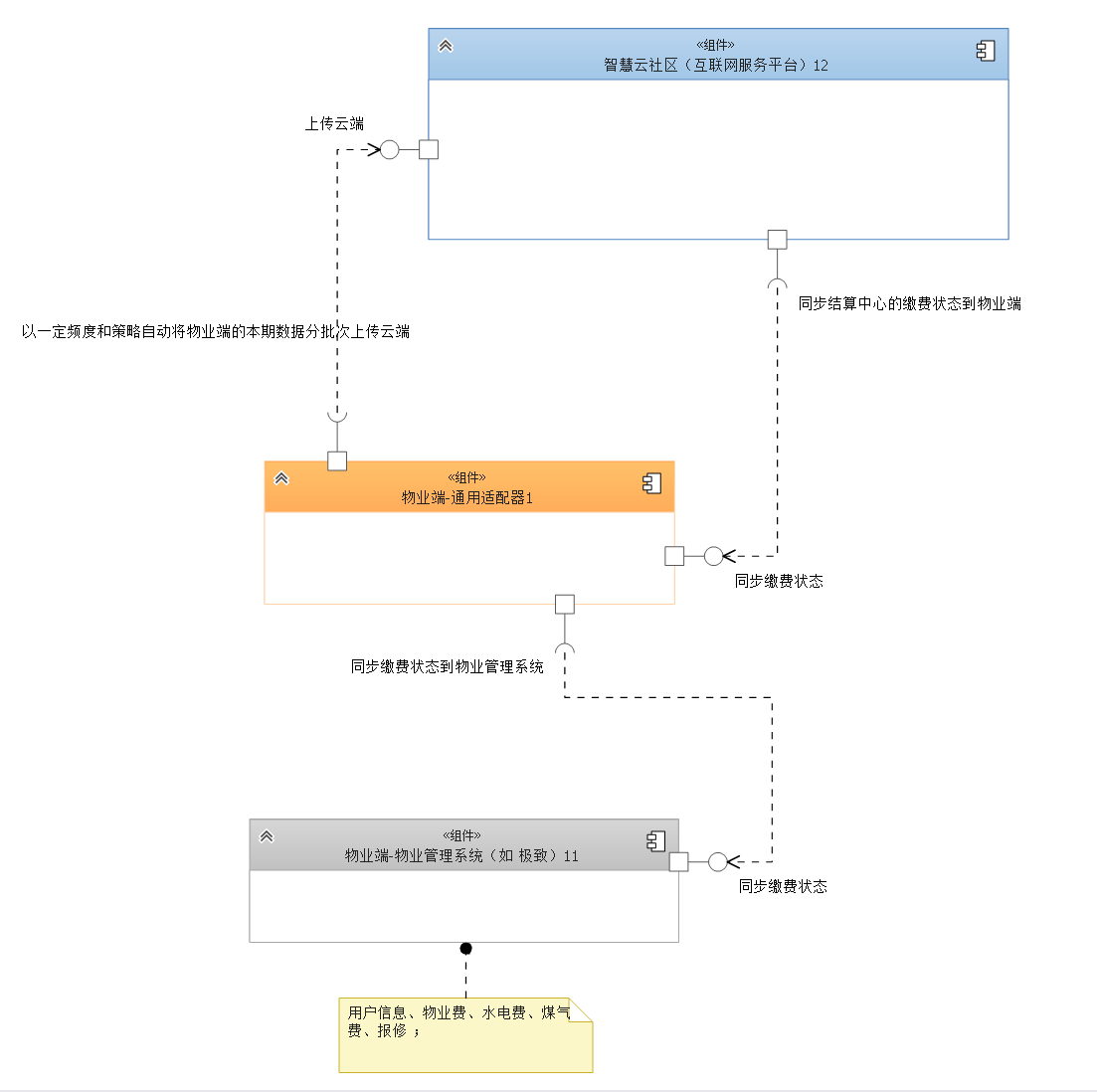


此方案中，适配器与物业管理软件共享数据库，业务对接时需要物业管理软件开发方提供数据字典，适配器通过视图查询，同时直接修改或通过中间表实现数据一致性 。

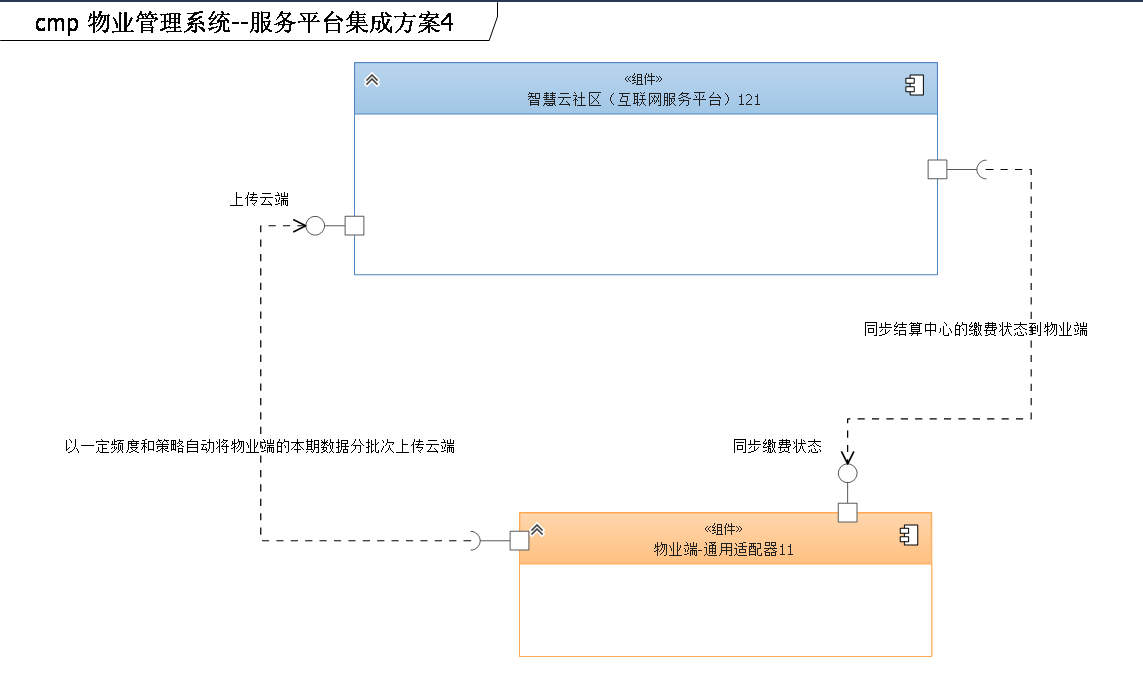
*以上两种集成方案都是云端主动抓取型，特点是: 每次查询数据量小，但查询频度高；*

*对网络的限制条件是：物业端需要具有一个公网IP。*

方案3： 客户端脉冲同步（消息集成）



方案4： 客户端脉冲同步（直连物业数据库或中间表同步）



以上两种方案取消了网络限制条件，即不需要物业端具有公网IP ；但由于其非实时性，物业管理系统实际业务状态与云端状态存在同步延迟，且每个批次上传的信息量较大，需在业务上考量其可行性！

备注: 虽可通过Oracle Change Notification实现差量式的数据同步，但此手法显然对物业管理系统的数据库选型有要求。

# 设计约束

## 3.1需求约束

|  |  |
| --- | --- |
| **约束** | **描 述** |
| **功能约束** | 需要提供菜单、功能、按钮级权限控制 |
| **功能约束** | 需要支持对企业管理员授权、收权、停用；企业管理员对企业操作人员的注册、授权； |
| **功能约束** | 需要支持对数据行按指定字段值范围，进行授权使用，包括查看、修改、删除、审批、确认等，常用的如公司信息、物业费明细管理、折扣率或优惠额等 |
| **功能约束** | 平台管理员只能管理、授权用户企业管理员、平台操作员权限，无业务操作权限 |
| **功能约束** | 系统必须提供完善的审计功能，对系统关键数据的每一次增加、修改和删除都能记录相应的修改时间、操作人和修改前的数据记录 |
| **功能约束** | 需要英语、汉语（简体 、繁体）支持，包括界面、内容、提示预警等所有信息；一期不支持除英、汉(简体、繁体)之外的语言，但需要能方便扩展更多语言的支持 |
| **功能约束** | 需要跨浏览器支持，包括IE8及以上，Chrome，360和腾讯浏览器；一期不支持火狐浏览器、Opera浏览器，但需方便扩展更多浏览器支持 |
| **功能约束** | 平台支持超级管理员与平台管理员角色，平台超极管理员只能管理、授权平台管理员，无业务操作权限 |
| **~~功能约束~~** | ~~支持根据不同语言使用不同的日期格式~~ |
| **用户界面约束** | 系统发现用户提交有误信息，必须以弹出窗口的形式明确提示用户错误的原因，并把界面控制焦点置于发生错误的控件对象上。 |
| **用户界面约束** | 系统支持操作员登录后，不超过三次鼠标点击，即可访问业务所需功能 |
| **安全性约束** | 系统在解决跨站攻击、注入攻击等系统性安全问题上，具体标准参照《OWASP Top10–2013》 |
| **安全性约束** | 用户名全系统不重复，字母数字组合，不区分大小写，长度不小于5位；  密码由字母数字组合，区分大小写，长度不小于5位，采用MD5或更高级别加密； |
| **安全性约束** | 3次密码输入错误，要求输入验证码；10次以上错误，锁定用户名； |
| **可扩展性约束** | 需要支持原有逻辑扩展，在接口不变的情况，能方便增加新的逻辑实现。  需要支持新增逻辑在不影响原有业务、代码情况下，能增加新业务、新逻辑的处理分支  需要支持原逻辑再封装，支持在原逻辑封装基础上，进行再封装、不同逻辑组合。 |
| **稳定性约束** | 系统可以通过运行更多的实例或者采用分布式处理来支持更多的用户。可以通过线性的增加硬件设备、实例个数或者分布式处理点来处理更多的数据量，能更好的在不增加响应时间的前提下支持更多的业务处理。 |

## 3.2隐含约束

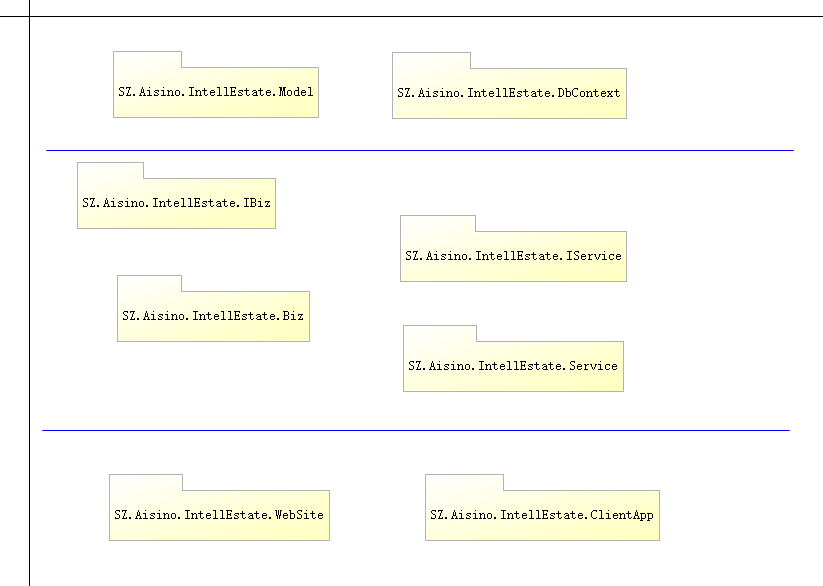
|  |  |
| --- | --- |
| **约束** | **描 述** |
| **功能约束** | 需求中提出的价格涉及多种货币，系统需要支持多币种（人民币、港币）的相关功能。 |
| **功能约束** | 需求中提出的报表部分，要求用户可自定义和扩展，因此系统需要支持自定义报表功能 |
| **功能约束** | 基础字典的管理需要同时支持通用字典和物业公司专用字典，物业公司在可以使用通用字典，专用字典，或者是专用字典+通用字典（专用字典中未定义的） |
| **功能约束** | 需求中未涉及运维相关的功能，但在某些逻辑中描述了运维的操作。因此系统需要有运维相关的功能。 |
| **性能约束** | 系统中使用的频度很高的数据需要有缓存机制，以增加页面的响应速度 |
|  |  |
|  |  |

# 设计策略

|  |  |
| --- | --- |
| **策 略** | **描 述** |
| **扩展策略** | 逻辑封装，面向接口编程  预计经常变化的内容采用XML配置文档将可变内容外置，扩展时不需要修改代码  Business层细化为两层，领域服务层（Biz）和服务facade层（Service）。其中Biz层基于领域模型划分为各个业务领域，封装具体的业务实现；Service层将Biz层的逻辑组织为大粒度的API，并有针对性地以 SOAP 或 Restful 的方式对外发布。 |
| **复用策略** | 与第三方系统对接信息采用XML配置，需要定义数据转换的处理接口，对接信息内容的生成要求可方便的进行扩展，适用于不同物业管理系统等其他需要交互的系统，可重用。 |
| **折衷策略** | 业务数据在遇到性能和功能冲突时，采用“分时”策略处理，将复杂操作拆分成多个简单操作  报表功能在遇到性能和功能冲突时，优先于功能，并通过SQL优化处理 |
| **数据关联策略** | 用户使用账户进行关联，不区分大小写  业务数据统一使用主键（Sql Server）或序列（Oracle）关联 |

# 总体设计

## 封装规范



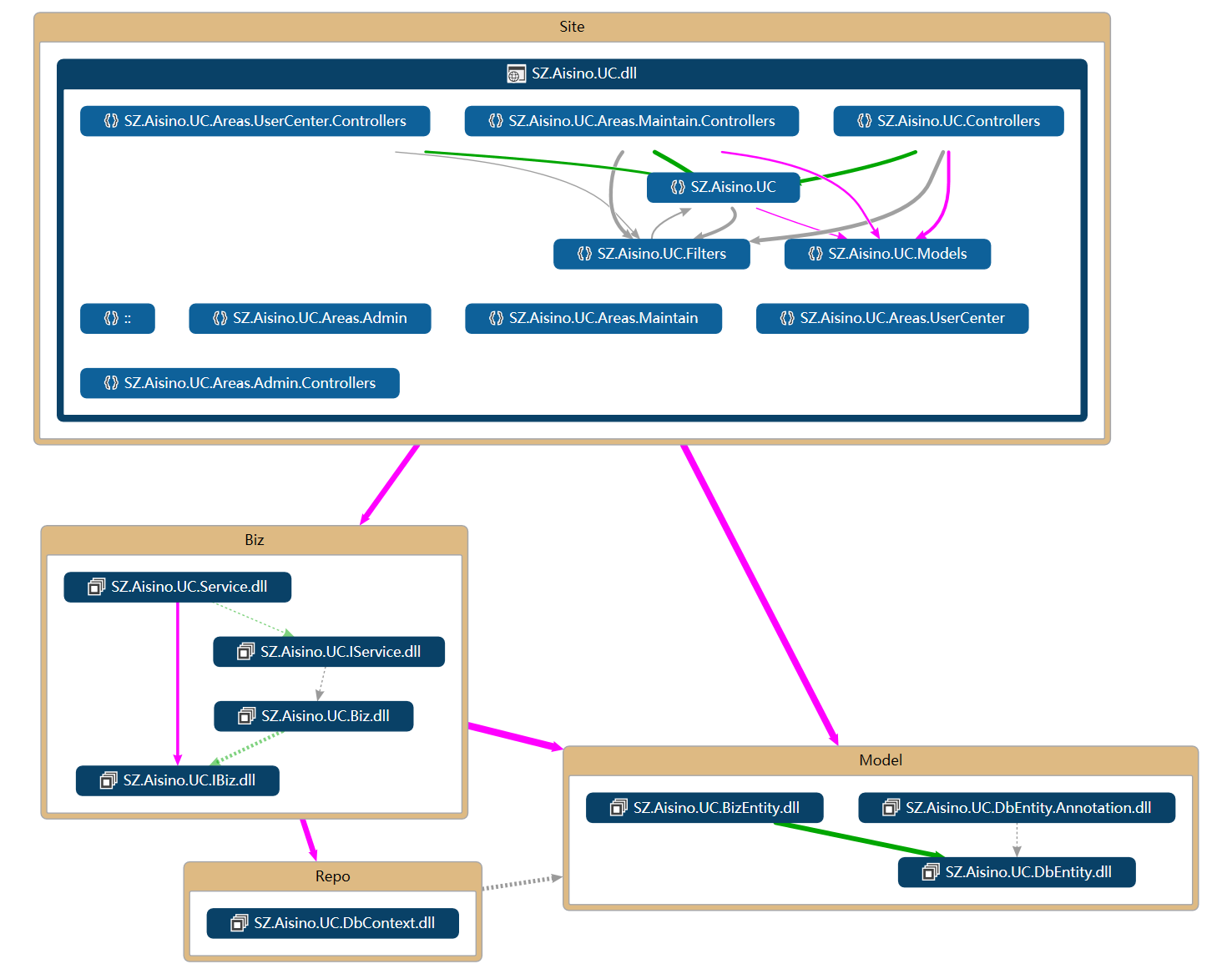
封装方案主体参考重构逻辑，三层架构。Model层和DbContext处理数据的映射和增删改查操作；Biz层和Service实现领域模型，Biz层封装可重用的逻辑代码，根据逻辑实现的不同分别封装不同的DLL，Service层对应大粒度的业务过程的封装。最终的应用层UI和逻辑分离。

### 接口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 所属系统 | 接口 | 说明 | |
| 智慧云社区（互联网服务平台） | SZ.Aisino.IntellEstate.IQueryService. | 智慧云社区查询接口：互联网服务平台对外部系统有限开放的查询接口 | |
| SZ.Aisino.IntellEstate.IEinvoiceService. | 统一开票接口：互联网服务平台对外部系统有限开放的电子票开票代理服务接口 | |
| 物业端-通用状态适配器 |  | **详见：2.2 接口部分提供的四种方案** | |
| 电子票前置服务 |  | 深圳航信电子票前置服务接口 | |
| 物业停车场道闸系统（第三方 ） | 三方待协商 | 查询接口（月卡有效期、月卡用户信息、月卡充值历史记录） | |
| 三方待协商 | 月卡充值缴费接口  （月卡基准费率配置在“公司基础信息维护模块”中实现） | |
|  |  | |
| 物业管理系统（第三方 如极致） | 三方待协商 | **详见：2.2 接口部分提供的四种方案，请需求人员根据客户对接意愿酌情考量：** | |
| 方案1 | 对外查询接口(读)同步缴费状态接口 （写） |
| 方案2 | 数据字典视图 （读） 直接修改或通过中间表实现数据一致性（写） |
| 方案3 | 数据字典视图 （读）同步缴费状态接口 （写） |
| 方案4 | 数据字典视图 （读）直接修改或通过中间表实现数据一致性（写） |
|  |  | |

### 逻辑层级关系

#### SZ.Aisino.IntellEstate的层级关系



此处以服务平台-权限子系统为例说明：

1. 在Model层增加业务实体（BizEntity）定义，将业务实体作为数据迁移对象（DTO）来封装前台接口，以利于前台接口的易用性和扩展性。
2. Repo（仓储层）和Model层（表字段映射）继续保持分离。
3. 业务逻辑分开打包，并用接口支持扩展。
4. 高层组件所有针对Biz的依赖一律采用倒置原则设计； 通过依赖注入，插件式加载所需的操作实例。

### 类库命名规范

1. 类库命名格式定义为“SzAisino.<项目代码>.<类库分层>.<类库子分类>”。最大层级不超5 层。

项目代码暂定：IntellEstate

1. 程序集/包名称采用“驼峰”格式。
2. 类名称，方法名称和属性名称采用“驼峰”格式，类名称，属性、方法名称首字大写，字段名称首字小写。
3. 接口名称固定大写“I”字头；但如过领域模型的核心类首字母为I，则接口前缀“I\_”,如 I\_InformationSyncService
4. 函数/方法命名时采用“动词+名词”组合方式。

### 数据库设计规范

Oracle：

1. 数据表名称和字段名称采用全大写，“名词+(动词)”组合，单词之间使用下划线分割。例如订单主表可命名为“ORDER\_HEAD”，对应的跟踪表可命名为“ORDER\_HEAD\_TRACE”.
2. 禁止使用中文汉语拼音或缩写来命名表、字段、主外键等一切数据库对象。
3. 数据表命名不要带有 HIS，TEMP，TMP，BAK，BACK等容易混淆的关键字。
4. 所有的数据表都需要带有唯一的SEQ关键字，并作为主键处理，除非特殊需要。SEQ的命名规则为“表名\_SEQ”。
5. 所有的数据表都需要带有时间戳字段和操作人字段。
6. 不允许使用触发器。
7. 一个数据表不要使用超过一个的BOLB（CLOB）类型字段。
8. 字符型的字段统一使用“VARCHAR2”格式，长度设置为预估长度的2-5倍。尽量不超过2000长度。
9. 所有的公司名称，街道名称等涉及公司名称的字段，统一长度为VARCHAR2(2000)

SqlServer：

1. 数据表名称和字段名称采用驼峰命名，并根据表所对应的领域（Domain），尽量保证以领域名开始；名称一律采用复数形式，例如订单主表可命名为“Orders”，对应的明细表可命名为“OrderDetails”.
2. 禁止使用中文汉语拼音或缩写来命名表、字段、主外键等一切数据库对象。
3. 数据表命名不要带有关键字。
4. 所有的数据表都需要带有唯一的标识字段，并作为主键处理，除非特殊需要。SEQ的命名规则为“<表名单数形式>ID”，如：Orders表的主键字段命名为：OrderID
5. 所有的数据表都需要带有时间戳字段和操作人字段,关键表时间戳需分化为创建和修改两个字段，如：CreateDate和ModifyDate 。
6. 不允许使用触发器。
7. 字符型的字段统一使用“nvarchar(n)”格式，长度设置为预估长度的2~5倍。尽量不超过2000字符长度, 超过4000字符请使用ntext 。
8. 所有的公司名称，街道名称等涉及公司名称的字段，统一长度为nvarchar (2000)

### 数据库核心实体关系图



## 多语种支持解决方案

### 需求说明

需求提出需要对系统支持多语种方案，包括UI界面，提示信息和内容三个部分。

### 解决方案

备注：解决方案提供了.NET平台的具体实现策略，其他平台可根据自身特点参考或独立实现：

#### UI界面的多语种支持

UI界面的多语种支持有多个解决方案：1）采用.net本身的语言包功能，在编码时绑定资源文件； 2）采用分站点开发，不同的语言单独部署不同的站点； 3）采用数据库的编码对照方式处理。

三种方案都能实现UI界面的多语种支持，相比较而言，使用语言包的方案和本地文件有关，在程序发布时需配合自动发布实现一致的集群部署。而采用不同站点的方案,UI维护的工作量会成倍增加，不利于功能的扩展和维护。采用数据库编码方式灵活性不佳，亦不予考虑。

如果团队基于ASP.NET MVC中间件开发，推荐基于元数据（Metadata）配合资源字典来解决多语言问题，可以自定义元数据提供程序来具体实现之：

public class ResDataAnnotationsModelMetadataProvider : DataAnnotationsModelMetadataProvider {

       private ResourceManager ResMgr;

       public ResDataAnnotationsModelMetadataProvider(ResourceManager resMgr) {

           if (resMgr == null)

               throw new ArgumentNullException("resMgr");

           this.ResMgr = resMgr;

       }

       private string GetString(Type containerType, string propertyName) {

           var key = string.Format(**"{0}{1}\_{2}\_DisplayName", containerType.Namespace.Replace(".", ""), containerType.Name, propertyName**);

           return this.ResMgr.GetString(key);

       }

       /// <summary>

       /// 重写生成元数据的方式

       /// 只有在页面上来通过Lambda表达式表述的字段，在渲染时才会使用元数据，才能支持多语言转换

       /// (该过滤为了减少元数据的中英文转换次数)

       /// </summary>

       /// <param name="attributes"></param>

       /// <param name="containerType"></param>

       /// <param name="modelAccessor"></param>

       /// <param name="modelType"></param>

       /// <param name="propertyName"></param>

       /// <returns></returns>

       protected override ModelMetadata CreateMetadata(IEnumerable<Attribute> attributes, Type containerType, Func<object> modelAccessor, Type modelType, string propertyName) {

           if (containerType != null) {

    //动态类型/代理类 EF

               var cType = containerType.Assembly.IsDynamic ? containerType.BaseType : containerType;

               var value = this.GetString(cType, propertyName);

               if (!string.IsNullOrWhiteSpace(v)) {

                   var display = new DisplayAttribute() {

                       Name = value

                   };

                   var attrs = attributes.ToList();

**attrs.RemoveAll(a => a is DisplayAttribute);**

**attrs.Add(display);**

                   return base.CreateMetadata(attrs, containerType, modelAccessor, modelType, propertyName);

               }

           }

           return base.CreateMetadata(attributes, containerType, modelAccessor, modelType, propertyName);

       }

   }

以下编码体现了多语言接口的设计：

public interface ILang

{

        ModelMetadataProvider GetMetadataProvider();

        string GetByKey(string key);

}

public class Lang : ILang

   {

       public ModelMetadataProvider GetMetadataProvider()

       {

           return new ResDataAnnotationsModelMetadataProvider(EntityRes.ResourceManager);

       }

       public string GetByKey(string key)

       {

           return EntityRes.ResourceManager.GetString(key);

       }

   }

多语言模块的初始化工作在Global.asax.cs中完成：

public class MvcApplication : System.Web.HttpApplication

   {

       protected void Application\_Start()

       {

          ……

ILang lang = DependencyResolver.Current.GetService<ILang>();

          ModelMetadataProviders.Current = lang.GetMetadataProvider();

……

       }

   }

此后，只要在资源字典中存在相应的键名，使用HtmlHelper的DisplayFor即自动支持多语言功能。

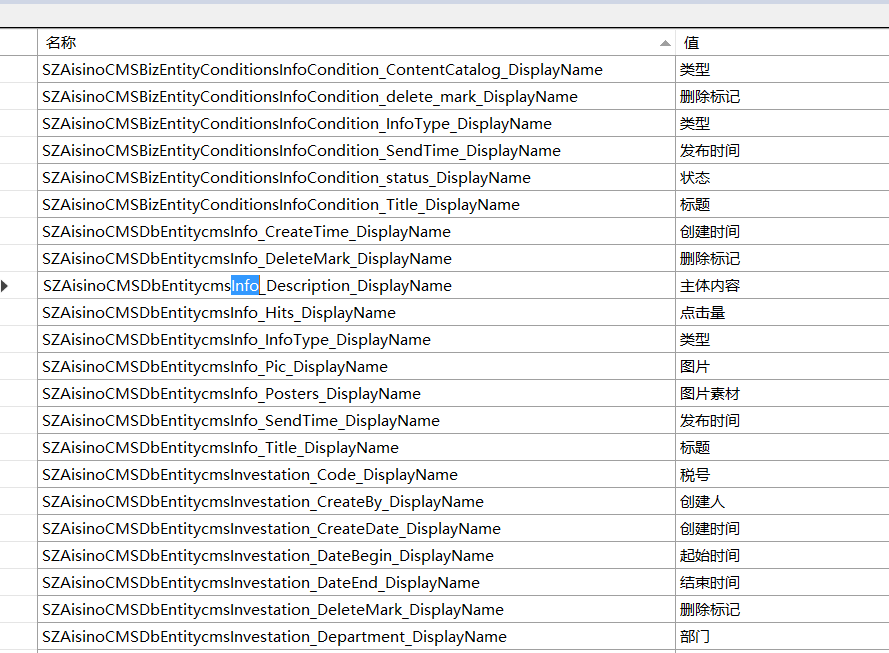
#### 相关问题

多语种的页面布局有一定的影响，开发界面时需要测试多个语种的显示效果，避免语种切换导致的版面问题。

### 资源字典设计

以下是资源字典的一个示例，请结合自定义元数据提供程序理解键的命名原则：

EntityRes.resx：



## 多币种支持解决方案

### 需求说明

系统需要支持币种处理，包括本位币和统计报表时的币种转换。

### 方案说明

1. 在所有使用价格的位置增加币种参数，“币种 + 价格”合并体现完整的价格信息。
2. 币种来源自每家物业公司的“本位币”全局参数配置。
3. 所有显示价格，金额的页面同时显示金额和币种，格式为“200USD”。
4. 每家物业公司独立维护自己的汇率转换关系，在需要使用的环节根据每家物业公司的汇率和设定进行货币转换。

### 数据库设计 （以ORACLE为例）



1. “CURRENCY\_RATE<汇率转换表>”用于保存货币转换的汇率信息，每家物业公司维护自己使用的汇率转换关系。
2. 汇率带有有效起至时间控制，在一个时间点应该只存在一套转换关系（输入数据时控制）。实际使用汇率时根据业务的时间查找汇率表获取对应汇率进行处理。
3. 汇率数据采用标记删除，避免已经使用的汇率找不到数据根源。

## 多浏览器支持解决方案

### 需求说明

系统要求支持浏览器兼容，支持IE8-10，Chrome，360和QQ浏览器。浏览器兼容问题主要集中在CSS样式的处理和JS脚本的跨浏览器兼容。

### 方案说明

1. HTML原型在制作时需要注意处理样式相关内容的浏览器兼容问题。
2. 页面的样式修饰全部要求使用CSS样式表控制，不要在元素上直接修饰。以方便在出现问题时统一修改。
3. JS脚本的兼容问题，需要注意对以下JS脚本的浏览器兼容：
4. 弹出窗口，系统的弹出窗口需要使用层弹出模式，不使用原有的Show Module方式。主要目的是避免浏览器对弹出窗口的拦截而导致功能不可用。因此建议在系统框架搭建的过程中，优先编写或者改造开源的弹出窗口控件。
5. 日期控件，系统需要使用到日期的录入控件，建议修改原有的日期控件或者测试开源的日期控件，在框架搭建阶段确定下能支持浏览器兼容的日期控件。
6. Message窗口，系统的Message消息不在使用JS的Alert函数处理，需要使用弹出层的方式，建议在框架阶段预先封装好消息提示的相关控件。
7. 菜单的显示效果，系统的菜单定义为下拉层的方式显示，相关的JS脚本需要预先进行浏览器兼容处理。建议在框架阶段确定一套系统使用的JS程序包，其他组件可JS脚本的编写基于这个包进行开发，避免不同的同时使用不同的程序包带来的浏览器兼容问题。

特别需要使用到的其他JS脚本和控件，在实际编码过程中需要进行浏览器兼容检查。

## 权限控制方案

### 方案说明

* 数据权限控制体系，以“User\_Data\_Func<用户数据权限>”数据表为核心，所有的权限控制都最终绑定到一个用户账户上。通过SQL语法实现控制逻辑。
* 某些设置在生成语法时，需要同时生成对多个功能点，多个用户账户的配置。这部分内容通过模板配置的方式实现批量生成。
* 权限控制分为三类，查询类，操作类和录入控制类
* 所有的查询操作，在查询数据时，都需要询问“User\_Data\_Func <用户数据权限>”获取到附加的SQL语法，以实现各种控制逻辑需要的效果。
* 所有的操作（Action）在执行时需要将操作的业务数据送到接口，与“User\_Data\_Func <用户数据权限>”中的条件进行验证，以确定是否允许操作。
* 录入控制的权限，设置内容保存在“User\_Data\_Func <用户数据权限>”中，实现部分由于涉及到的响应方式不同，需要在各个具体的功能中编码实现。
* 权限控制的重点在于所有的功能点都接受“User\_Data\_Func <用户数据权限>”的控制，无论现阶段是否有对应的权限设置对应到这个功能点。之后如果有新的权限控制需求，或者是原有的权限控制需求有改变的，仅需要变更规则写入“User\_Data\_Func <用户数据权限>”的方式，即可扩展对应的功能。

## 异常和日志方案

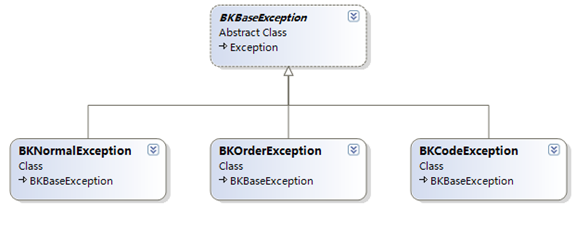
### 功能说明

异常和日志接口用于异常捕获，以及常规的业务日子记录。

### 数据库变更

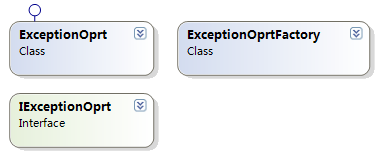
1. “OPERATION\_TYPE<操作类型>”日志记录的简略标题，用于区分一组相关的日志或者是为订阅日志提供分类。
2. “LEVELS<信息类型>”信息的分类，定义为4级“1-DEBUG 2-INFO 3-WARN 4-ERROR”。必要字段。
3. “INFORMATION<信息描述>”消息的主体，具体的内容在功能点的详细设计文档中进行描述。
4. “OBJECT\_TABLE<操作表名>，OBJECT\_SEQ<操作记录号>”用于特定数据的跟踪，目前仅在对“BUSINESS\_ORDER<订单表>”相关的日志中使用。后续可能扩展到其他需要关注的表。
5. “ORDER\_NO”在记录订单相关的日志时的必要信息，用于用户在追踪某一个订单时，作为查询条件使用。PS：此信息可能转换成Booking No，用于追踪订单的后续状态。
6. “OWNER\_COMPANY\_SEQ<物业公司>，OPRT\_CLIENT\_SEQ<操作终端>，PAY\_ CUSTOMER \_SEQ<业主>”用于登记日志的相关对象，这部分信息影响限定用户查询日志时的数据范围，后续根据实际的需求，可能作为索引使用。
7. “STATUS<数据状态>”取值为 “10-有效数据 -10-无效数据 -20-删除数据”。初始插入时为“10-有效数据”，其他两个状态值预留，目前暂不使用。

### 接口定义



在Exception对象基础上扩展了一层对象：

* “BKBaseException”类继承自“Exception”对象，抽象类，作为系统异常的基类。
* “BKNormalException”常规异常类，用于登记“非特定”的日志和异常。
* “BKOrderException”订单异常类，用于登记“订单”相关的日志和异常。和常规异常类的区别在于登记时会增加订单相关信息的采集。
* “BKCodeException”代码异常类，用于登记代码错误，常规用于Catch捕获中使用。



ExceprionOprt对象用于控制具体的日志写入操作，系统内部的异常或者是Exception对象都可以经过此类进行异常处理。

### 日志说明格式

在设计文档中的日志说明定义为以下格式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 标题 | LEVEL | 内容 | 接口 | 备注 |
| 订单撤销 | INFO  ERROR  DEBUG  WARNING |  | 常规异常  订单异常 | OWNER\_SEQ填写当前物业公司；  objName = “HTWY” |

## 缓存方案

### 功能说明

本章节用于描述缓存模块在系统中缓存的具体设计及使用场景。

### 缓存方案概要

备注：解决方案提供了.NET平台的具体实现策略，其他平台可根据自身特点参考或独立实现：

1. 缓存的主体可以是dataTable，也可以是实体集合，请根据自身仓储机制的具体实现选择，
2. 缓存推荐基于微软企业库AppFabric实现。

以下编码体现了缓存的接口设计：

public class AppFabricCaching : ICaching {

      private static DataCacheFactory Factory = null;

      private static DataCache Cache = null;

      static AppFabricCaching() {

          var cacheName = ConfigurationManager.AppSettings.Get("AppFabricCaching.Name");

          Factory = new DataCacheFactory();

          Cache = string.IsNullOrWhiteSpace(cacheName) ? Factory.GetDefaultCache() : Factory.GetCache(cacheName);

      }

      public T Get<T>(string key) where T : class {

          return (T)Cache.Get(key);

      }

      public void Set<T>(string key, T value) where T : class {

          Cache.Put(key, value);

      }

      public void Set<T>(CacheSet config, string key, T value) where T : class {

          if (config == null)

              throw new ArgumentNullException("missing config");

          if (config.Timeout <= TimeSpan.Zero)

              config.Timeout = TimeSpan.FromMinutes(10);

          // AppFabric Caching 没有相关API指定某项的优先级,只能通过加大过期时间来实现.

          if (config.NotRemoveable)

              config.Timeout = TimeSpan.FromDays(360);

          if (!string.IsNullOrEmpty(config.Region)) {

              Cache.Put(key, value, config.Timeout, config.Region);

          } else {

              Cache.Put(key, value, config.Timeout);

          }

      }

      public void Remove(string key) {

       Cache.Remove(key);

      }

  }

1. 缓存的数据采用“首次使用加载”机制，即在缓存首次被请求时方从数据库初始化。
2. AppFabric分布式缓存采用区域策略，这里使用缺省区域：default

<appSettings>

  <add key="AppFabricCaching.Name" value="default" />

……

</appSettings>

1. 对于使用Oracle作为存储单元的方案，缓存的刷新推荐使用Oracle的通知机制实现，需要使用“ODP.NET”的指定版本配合。
2. 需要缓存的数据定义

|  |  |  |
| --- | --- | --- |
| 缓存数据表 | KEY值 | 说明 |
| DICT\_CODE | DICT\_CODE\_{OWNER\_SEQ} | 字典代码表 |
| SYSTEM\_UI\_LANGUAGE | SYSTEM\_UI\_LANGUAGE | 语言包 |
| DICT\_LOCATION\_AREA | DICT\_ LOCATION\_AREA | 地区代码表 |
| DICT\_ESTATE\_STREET | DICT\_ESTATE\_STREET | 社区街道代码表 |

## 分布式会话

### 功能说明

服务平台各个子系统间需要支持单点登录，结合该系统集群部署及负载均衡的方案，推荐采用分布式会话实现同一顶级域名下个子站点的会话共享。

### 解决方案概要

备注：解决方案提供了.NET平台的具体实现策略，其他平台可根据自身特点参考或独立实现：

* + - 1. 推荐基于微软企业库AppFabric实现。
      2. 以下编码定义了一个非常简明的会话工具类，注意示例中采用了静态类实现：

public static class SessionHelper

{

    public static void Set<TSessionKey>(this TSessionKey key, object value) where TSessionKey : struct

    {

        var session = HttpContextHelper.Current.Session;

        session[key.ToString()] = value;

    }

    public static void Set(string key, object value)

    {

        var session = HttpContextHelper.Current.Session;

        session[key] = value;

    }

    public static TValue Get<TSessionKey, TValue>(this TSessionKey key) where TSessionKey : struct

    {

        var session = HttpContextHelper.Current.Session;

        object value = session[key.ToString()];

        return (TValue)value;

    }

    public static TValue Get<TValue>(string key)

    {

        var session = HttpContextHelper.Current.Session;

        object value = session[key];

        return (TValue)value;

    }

    public static void Remove<TSessionKey>(this TSessionKey key) where TSessionKey : struct

    {

        var session = HttpContextHelper.Current.Session;

        session.Remove(key.ToString());

    }

    public static void Remove(string key)

    {

        var session = HttpContextHelper.Current.Session;

        session.Remove(key);

    }

    public static void Clear()

    {

        var session = HttpContextHelper.Current.Session;

        session.Clear();

    }

}

以下是消费此工具类的示例：

var menus = SessionHelper.Get<Dictionary<string, IEnumerable<MenuItem>>>(SessionKeys.Menus.ToString());

……

menus.Set(appCode, appMenus);

SessionKeys.Menus.Set(menus);

以下是AppFabric的参考配置：

<configuration>

  <configSections>

    <section name="dataCacheClient" type="Microsoft.ApplicationServer.Caching.DataCacheClientSection,Microsoft.ApplicationServer.Caching.Core, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" allowLocation="true" allowDefinition="Everywhere" />

 <dataCacheClient>

    <hosts>

      <host name="WIN12-STATION.szhx.com" cachePort="22233" />

    </hosts>

   </dataCacheClient>

  <system.web>

    <sessionState sessionIDManagerType=" SZAisino.Common.SessionIDManager, SZAisino.Common" timeout="30" mode="Custom" cookieName="ASP.NET\_SessionId\_Global" customProvider="AppFabricCacheSessionStoreProvider" cookieless="false">

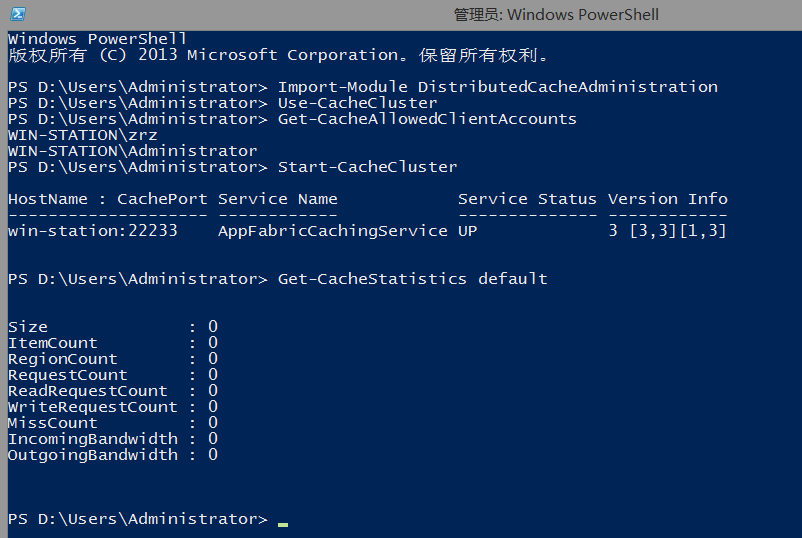
      <providers>

        <add name="AppFabricCacheSessionStoreProvider" type="Microsoft.ApplicationServer.Caching.DataCacheSessionStoreProvider, Microsoft.ApplicationServer.Caching.Client, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" cacheName="default" sharedId="OnlineSiteSession" />

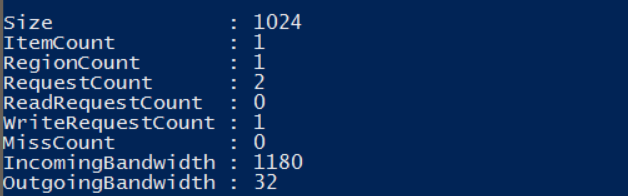
      </providers>

    </sessionState>

……  
以下为缓存服务器启动命令及运行状态：



有会话后：



# 服务平台详细设计

备注：

请外包团队基于《需求规格说明书》和本文档（《体系结构概要设计》）自行撰写。