云计算-多租户架构技术

-系统通道培训系列课程-王俏

美团 meituan.com

大众点评 dianping.com

11:25:44

云计算-多租户架构技术

-系统通道培训系列课程-王俏

美团 meituan.com　大众点评 dianping.com

讲师简介

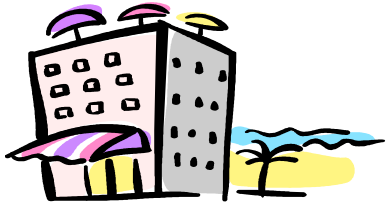
王俏，拥有计算机本科和MBA学位，2015年加入原点评，系统通道委员，现任美团点评-企业平台研发部基础架构产品负责人，在SaaS、PaaS云领域10年经验。

重点项目经验：

- 八百客SaaS与PaaS平台总监
- 建设银行新一代系统架构经理
- 华为售前云平台总架构师
- HP SSSP云平台负责人

目录

一个旅馆的例子



1个旅馆，
出租给多个人

换个角度，多个人共租用了1旅馆的空间（资源），旅馆正在面对“多租户”。

1个空间（资源）多人共用，如何划分房间（资源分配）？

本例中，我们从居住空间和床铺两个方面考量。

大通铺方案

不划分，全部共用。

旅客共享居住空间，床铺。

木板隔断方案

用木板隔离。一个隔断中容纳几名旅客。

几名旅客共享居住空间，每名旅客独享床铺。

水泥墙隔断方案

用水泥墙隔离。一个隔断中只容纳一名旅客。

每名旅客独享居住空间，床铺。

一个旅馆的例子-分析

我们来评价一下按照不同划分方案划分的结果：

		大通铺	木板隔离	水泥墙隔离	释意
容纳旅客数		50	20	10	进行分隔时必然会消耗空间，不同方案消耗空间不同，使得等量空间**最终容纳旅客数不同**。
旅客体验	私密性	差	中	好	旅客的行动是否会被其他旅客知晓**决定了私密性与安全性**。分差、中、好三个等级，对应私密性由低到高。本例中，多人共享居住空间还是单人独享居住空间决定了私密性和安全性。
	舒适性	差	中	好	居住空间如何分享，床铺如何分享共同决定了舒适性。二者均独享的方式**舒适性**最好。均共享的舒适性最低。
住宿价格（元/每天）		20	50	100	本例假设旅馆满员，每天计划总收入1000元。住宿价格=1000/容纳旅客数

云计算中的多租户

“多租户”是“云计算”的基本属性之一。云计算的三种服务层次——SaaS、PaaS和IaaS均体现了对“多租户”不同的支持。

	出租的资源	举例说明
SaaS	软件的使用权。	电子邮件系统，CRM、ERP 、网店等，或许企业用户数量不同、功能模块不同、时长不同收费不同
PaaS	软件开发平台资源（如开发支撑系列工具，应用存储空间，运行容器，平台服务等等）。	服务模块不同、API吞吐量不同、时长不同而收费不同
IaaS	硬件基础设施（如CPU、内存，存储，IP，网络设备等等）。	某租户拥有2颗CPU，8G内存，80G硬盘，10IP，2负载均衡器，创建主机数量不限，配置和时长不同而费用不同

多租户的概念

- **多租户技术**：（英语：multi-tenancy technology）或称多重租赁技术，是一种软件架构技术，它是在探讨与实现如何于多用户的环境下共用相同的系统或程序组件，并且仍可确保各用户间数据的隔离性。

- **价值**：降低供应商的实施、研发、运维综合成本，缩短交付周期，便于集成，便于流程与技术的**标准化**，便于业务与数据分析、便于构建平台化与生态

- **目的**：共用相同的软、硬件系统或程序组件，**快速复制方案**
- **难点**：**租户多，交付快，成本低，差异大**

多租户技术的历史发展

企业邮箱、视频和IM通信、ERP、CRM、HR系统等开始出现多租户，即**SaaS**（Software-as-a-Service）系统可能会运行在数台不同的机器上。这一时期还兴起了虚拟主机租用，类似于目前的**IaaS**。

许多公司为了要使用更多的运算资源，向持有大型主机（Mainframe）的供应商租用一部份的运算资源，而这些用户经常会用到相同的应用程序，当时会以用户在登录系统时输入的数据来决定用户的帐户ID，基于这个ID，Mainframe的供应商即可利用此ID来计算运算的资源使用量，包含CPU，存储器与软盘或磁带等，这个作法也被SAP公司用在其R/1到R/3的产品线。

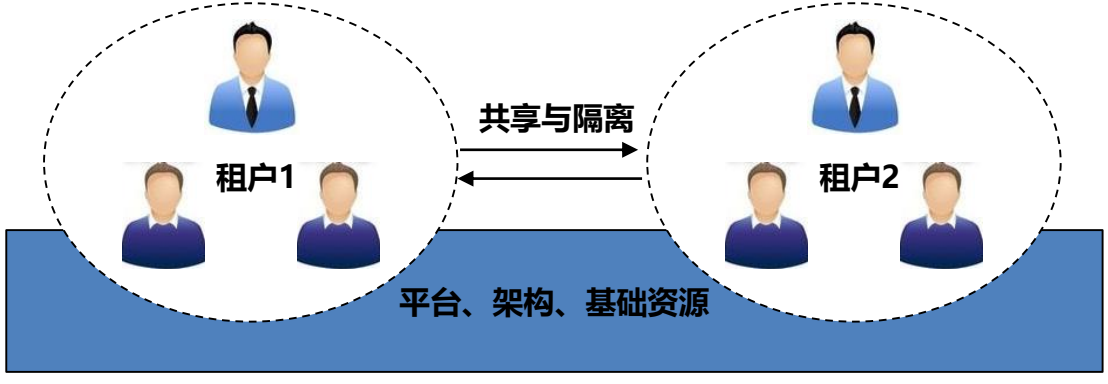
出现服务平台作为一种服务提供的商业模式。例如Amazon、谷歌、salesforce。通过网络进行程序提供的服务称之为**PaaS**(Platform-as-a-Service)。同时，随着电子商务的发展，各种网店开始兴起，成为SaaS模式普遍化的另一种形式。云计算的业务模式和市场日趋成熟。

软件类多租户发展轨迹


租户多，交付快，成本低，差异大
SaaS可配置化
+PaaS+应用托管
+应用商店与生态化
SaaS差异可配置化
+PaaS
相对固定功能的
SaaS

多租户技术对美团大众点评的价值

- 客户、子公司、BG可以基于共享平台和架构进行差异化应用自研，降低自研难度、缩短实施周期，快速复制最佳方案到租户方
- 消除各种数据孤岛，解决端对端集成过多的问题
- 解决原来应用过多，架构多变带来的运维复杂的问题
- 支持租户间相对灵活的应用、数据共享与隔离
- 提高系统资源的利用率，提高系统可用能力
- 可以在利润丰厚、竞争对手均不强大的B端云系统领域确立领先地位

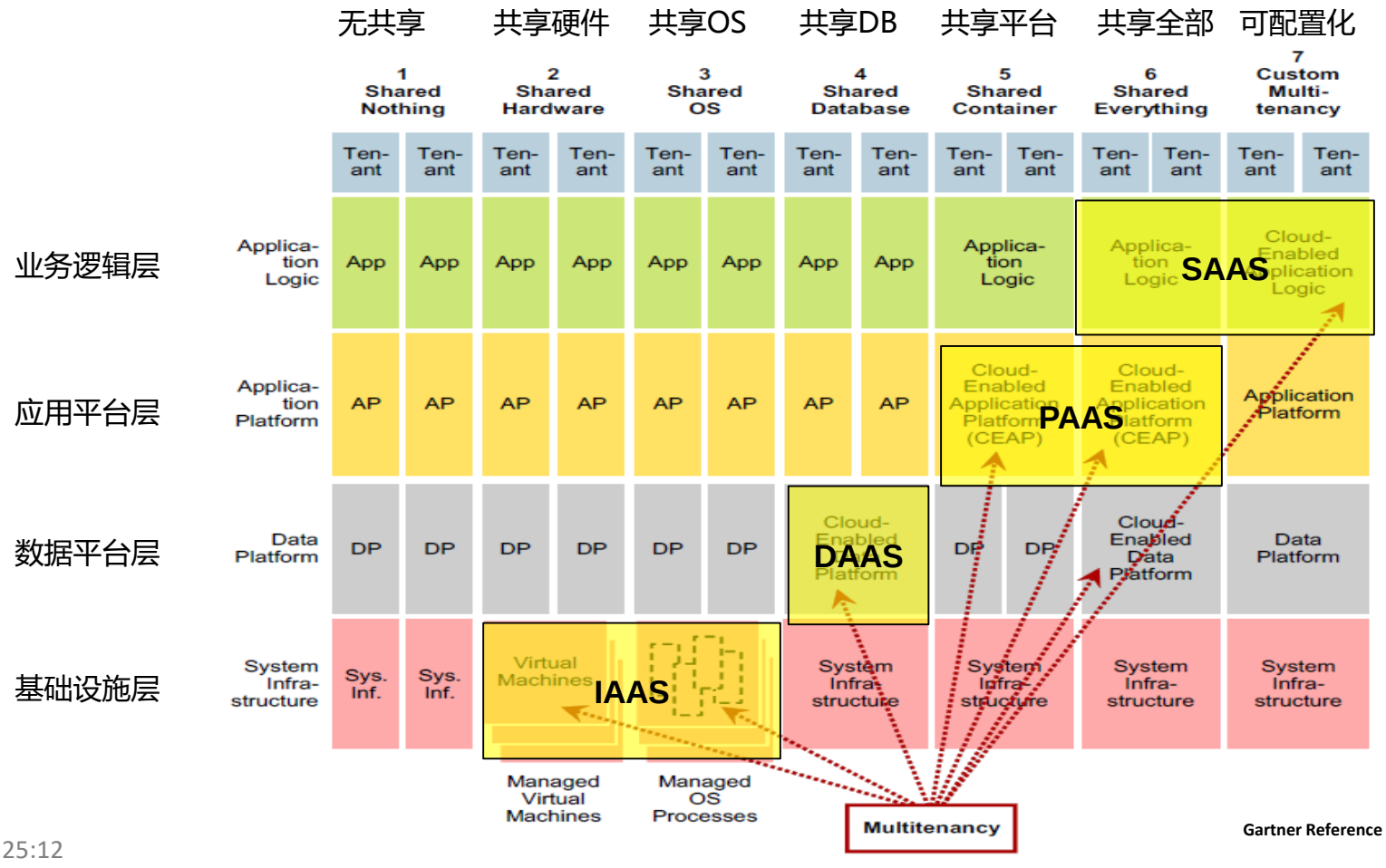


例如：

- 可租用的网店
- 可租用的ERP、供应链系统、CRM、销售、客服等系统、视频和IM等
- 企业平台相关的管理系统
- 各种研发工具

目录

高德纳Gartner多租户类型划分



Gartner Reference Model for Elasticity and Multitenancy

Gartner多租户模型与业内现状分析

级别	2-共享硬件	3-共享OS	4-共享DB	5-共享平台	6-共享全部	7-可配置化
业务逻辑层				共享	共享 SAAS	共享
应用平台层				云共享 PAAS	云共享	共享
数据平台层			云共享 DAAS	物理或实例隔离	云共享	共享
基础设施层	硬件共享 IAAS	云共享	共享	共享	共享	共享
典型案例	阿里云，美团云	阿里云，美团云	阿里云，美团云	阿里云，美团云，谷歌，亚马逊、Salesforce	企业邮箱，天猫、京东、微信、zoho、Salesforce	天猫、Salesforce、workday、LinkedIn
优点	提高资源利用率降低成本	提高资源利用率降低成本	提高资源利用率，降低成本	利用成熟的平台构建业务应用	利用成熟平台构建的业务应用可租用	各提供者无编码、高度可配置化的、独立构建的差异化业务应用
缺点	业务应用要自建应用交付周期长	业务应用要自建应用交付周期长	应用要自建，应用交付周期长	需要编码实现应用	业务需求满足度较低	业务抽象难度太大业务满足度提高困难

Salesforce、阿里、腾讯等顶级公司共同点，差不多都是先做6，再做5、7，最后提供2、3、4

思考题

- 为何SaaS市场规模最大，供应商数量很多，且供应商规模大小不一？
- 为何IaaS供应商数量不多，且母公司企业规模都较大？
- 为何PaaS市场规模最小，供应商数量也很少？
- 自己所在的产品线有哪些多租户的需求？采用哪种类型更合适？

目录

- SaaS、PaaS和多租户的概念和价值
- 多租户的分类与适用场景
- **组件化多租户系统架构如何设计？**
- 存量系统如何改造为多租户？
- 多租户系统的API如何设计？

多租户的识别与落地实施步骤

1、识别与策略选择

- 租户数量
- 租户间业务差异化
- 业务流程的复杂度
- 使用方自研能力
- 供应方研发资源投入
- 存量系统改造周期
- 供应方研发能力
- 供应方业务建模能力

2、架构与设计

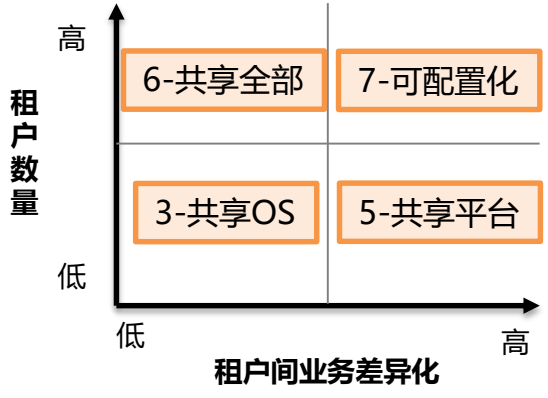
- 租户许可证与交付模式设计
- 组织结构设计
- 权限结构的设计
- 租户资源分配的设计
- 应用模块分析
- 数据结构分析
- 部署架构分析

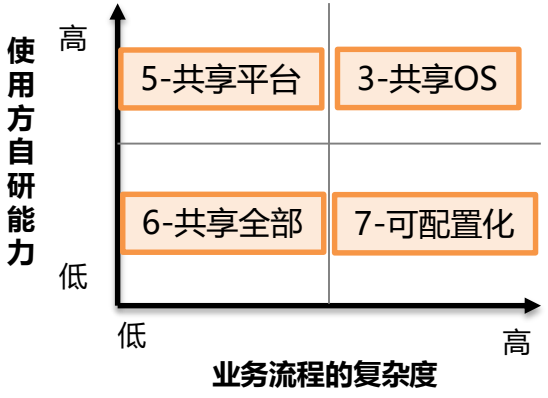
3、研发上线

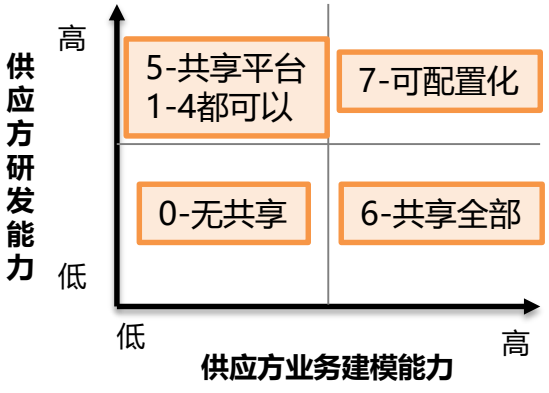
- 租户许可证的研发
- 租户模式与数据隔离测试
- 租户资源分配的测试
- 【存量系统】处理逻辑的改造
- 【存量系统】数据结构改造
- 【存量系统】部署结构的改造

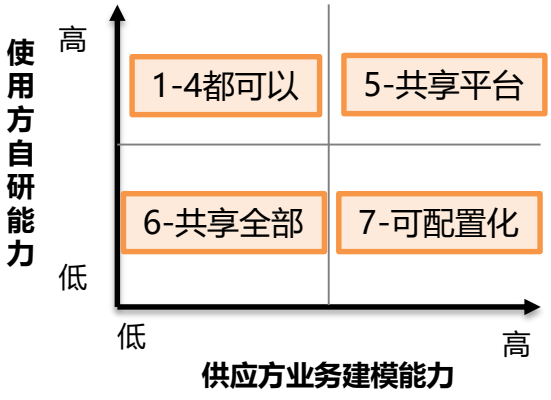
注：此实施步骤更适用与SaaS和PaaS

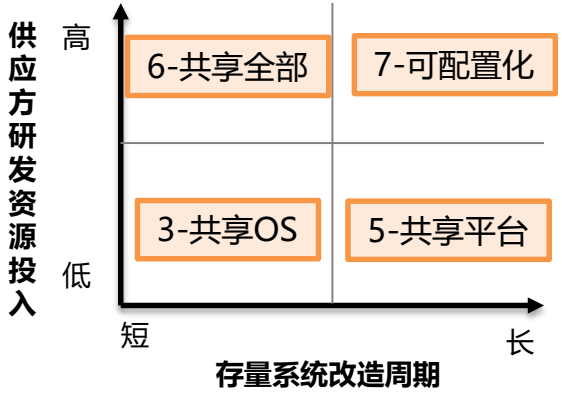
多租户识别与策略选择


租户数量
高
低
6-共享全部
7-可配置化
3-共享OS
5-共享平台
低
高
租户间业务差异化

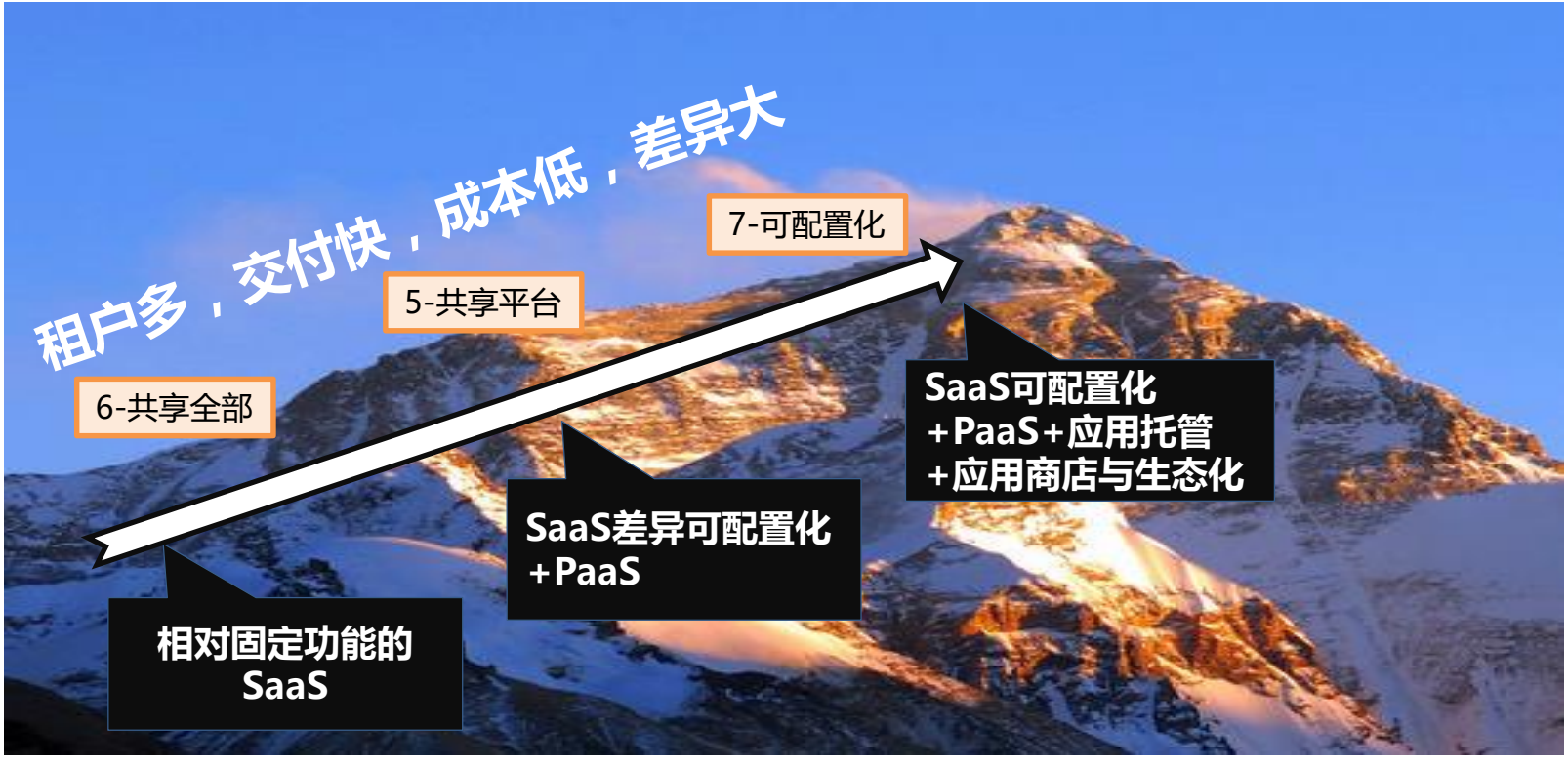

使用方自研能力
高
低
5-共享平台
3-共享OS
6-共享全部
7-可配置化
低
高
业务流程的复杂度


供应方研发能力
高
低
5-共享平台
1-4都可以
7-可配置化
0-无共享
6-共享全部
低
高
供应方业务建模能力


使用方自研能力
高
低
1-4都可以
5-共享平台
6-共享全部
7-可配置化
低
高
供应方业务建模能力

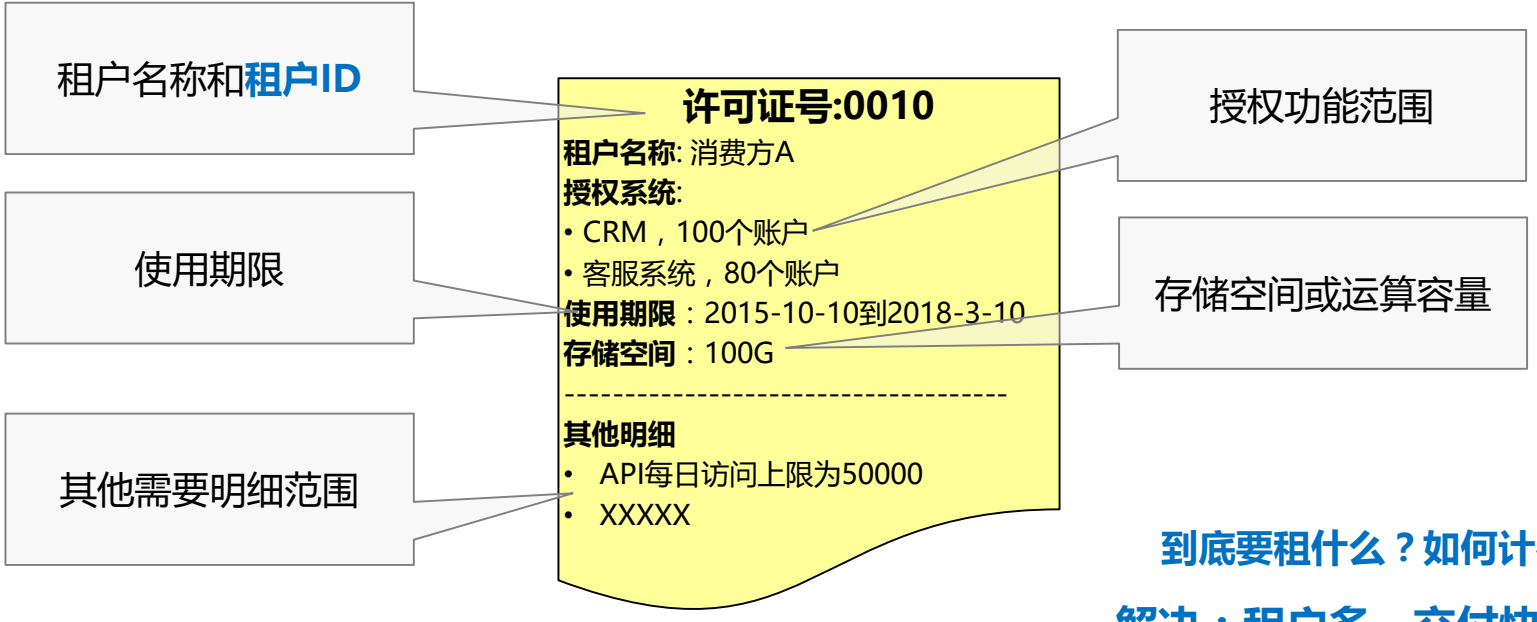

供应方研发资源投入
高
低
6-共享全部
7-可配置化
3-共享OS
5-共享平台
短
长
存量系统改造周期

策略分析验证了发展轨迹



多租户许可证与交付模式设计

每一个租户都需要由供应商发一个使用许可证，许可证用于在云系统里面进行相关的**授权能力的控制**。许可证发放后，系统一般**自动或半自动**生效各种相关授权。有些系统还需要响应的**实施顾问进行配置**才能使用。



到底要租什么？如何计费？

解决：租户多，交付快，差异大

许可证和应用模块的作用域

业务系统注册与描述

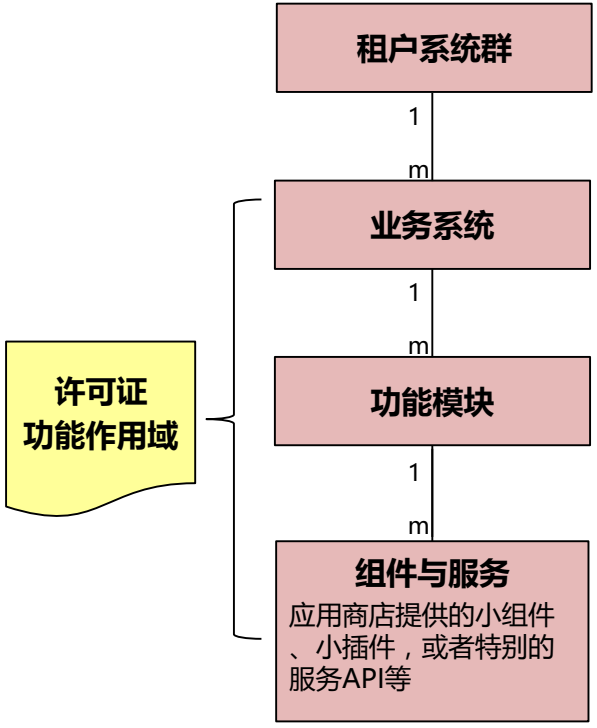
系统ID	系统名称	系统描述
XX	XX	XX
XX	XX	XX
XX	XX	XX

功能模块注册与描述

所属系统系统ID	功能模块ID	功能模块名称	功能模块描述
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX

组件与服务的注册与描述

所属系统系统ID	功能模块ID	组件ID	组件名称	组件描述
XX	XX	XX	XX	XX
XX	XX	XX	XX	XX
XX	XX	XX	XX	XX

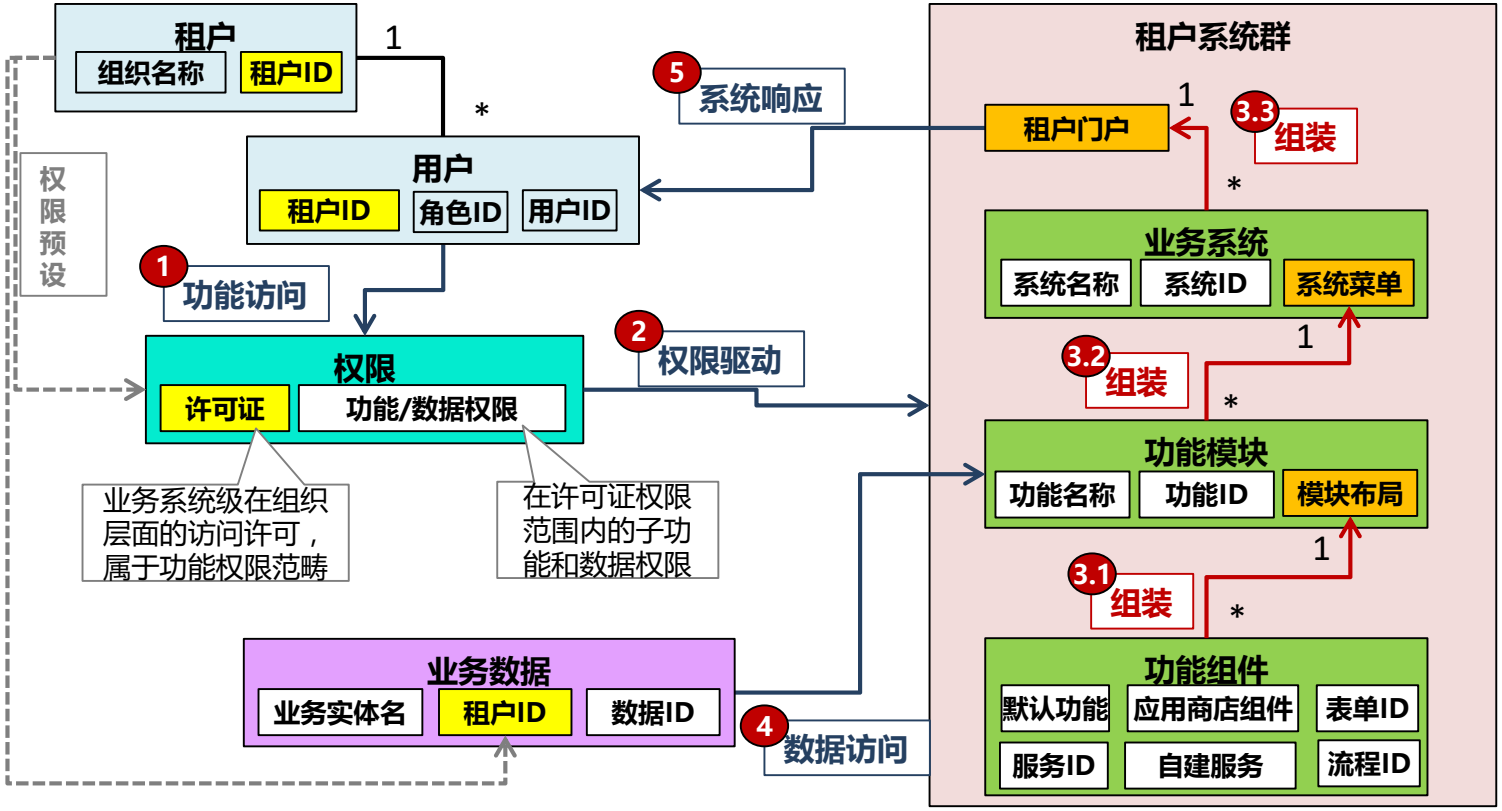


就是供应方的可供应家底盘点！
颗粒度越小，组合灵活度越高！

许可证驱动下的应用组装

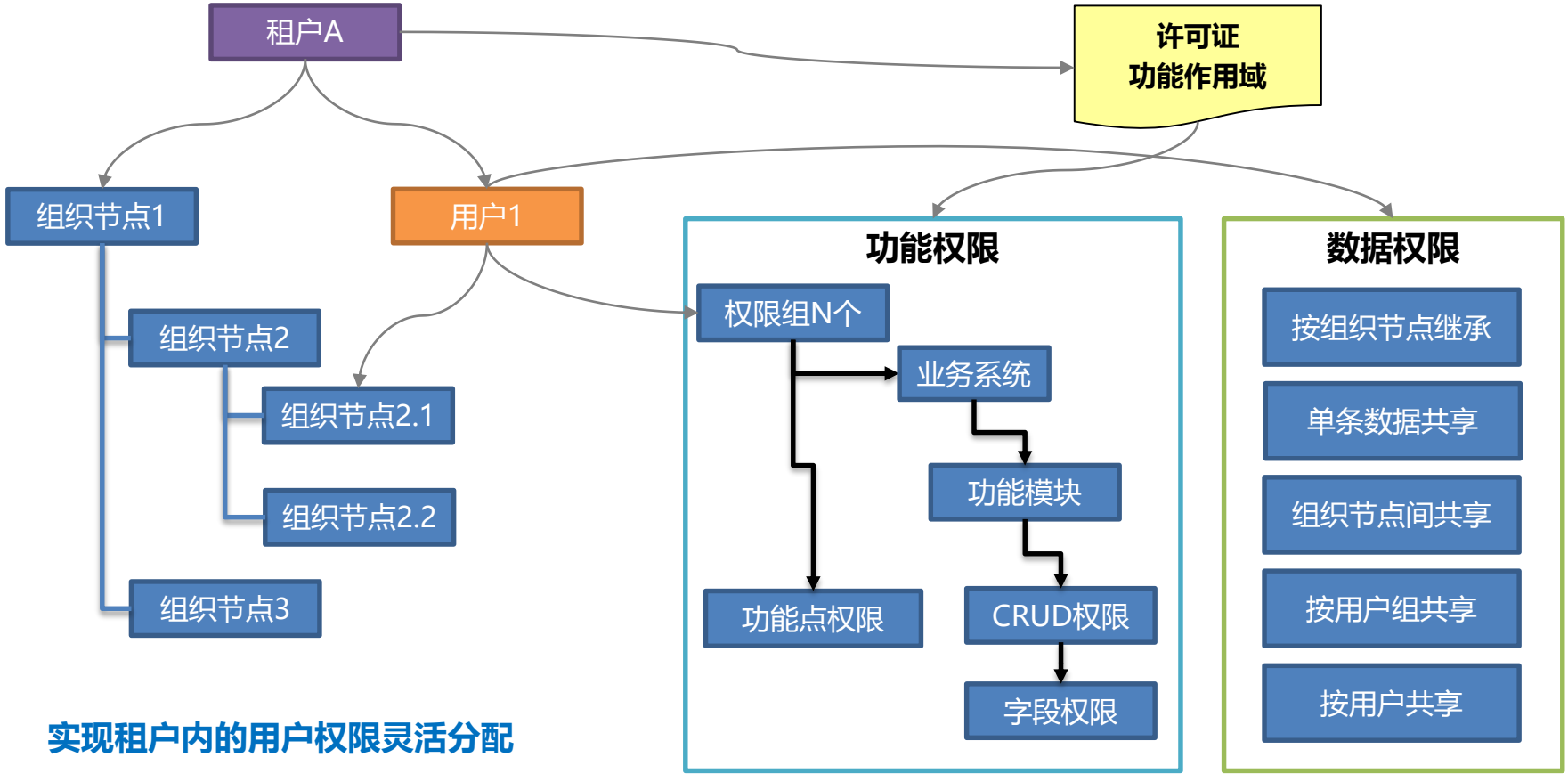
应用组装分为预制组装和动态组装

解决：交付快，成本低，差异大



实现交付的DIY，按需租用

多租户常见的组织结构与权限设计



实现租户内的用户权限灵活分配

多租户数据结构设计-数据隔离

在数据字段上加租户ID

workflow_data

租户ID	功能ID	功能字段1
XX	XX	XX
XX	XX	XX
XX	XX	XX

customer_data

租户ID	数据ID	业务字段1
XX	XX	XX
XX	XX	XX
XX	XX	XX

优点：设计简单、实现容易、利于升级、利于维护
缺点：不能体现较复杂的租户差异化需求
适用：IM、邮箱、网店等业务结构一致且业务逻辑简单的系统

在数据表上加租户ID

workflow_data_00234

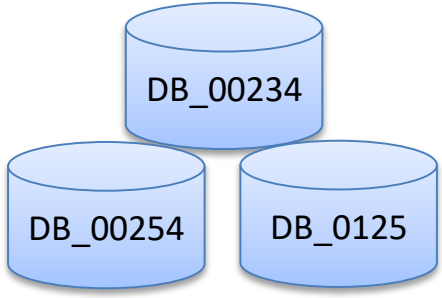
数据ID	业务字段1	业务字段2
XX	XX	
XX	XX	
XX	XX	

customer_data_00234

数据ID	业务字段1	业务字段2
XX	XX	
XX	XX	
XX	XX	

优点：升级难度适度、可以体现较复杂的业务差异化、利于差异备份和数据抽取隔离
缺点：实现难度较高，所有访问表时都要加上租户ID
适用：业务有一定的共性，但还有部分业务差异化可配置，CRM、客服、销售管理之类的系统

在数据库实例加租户ID

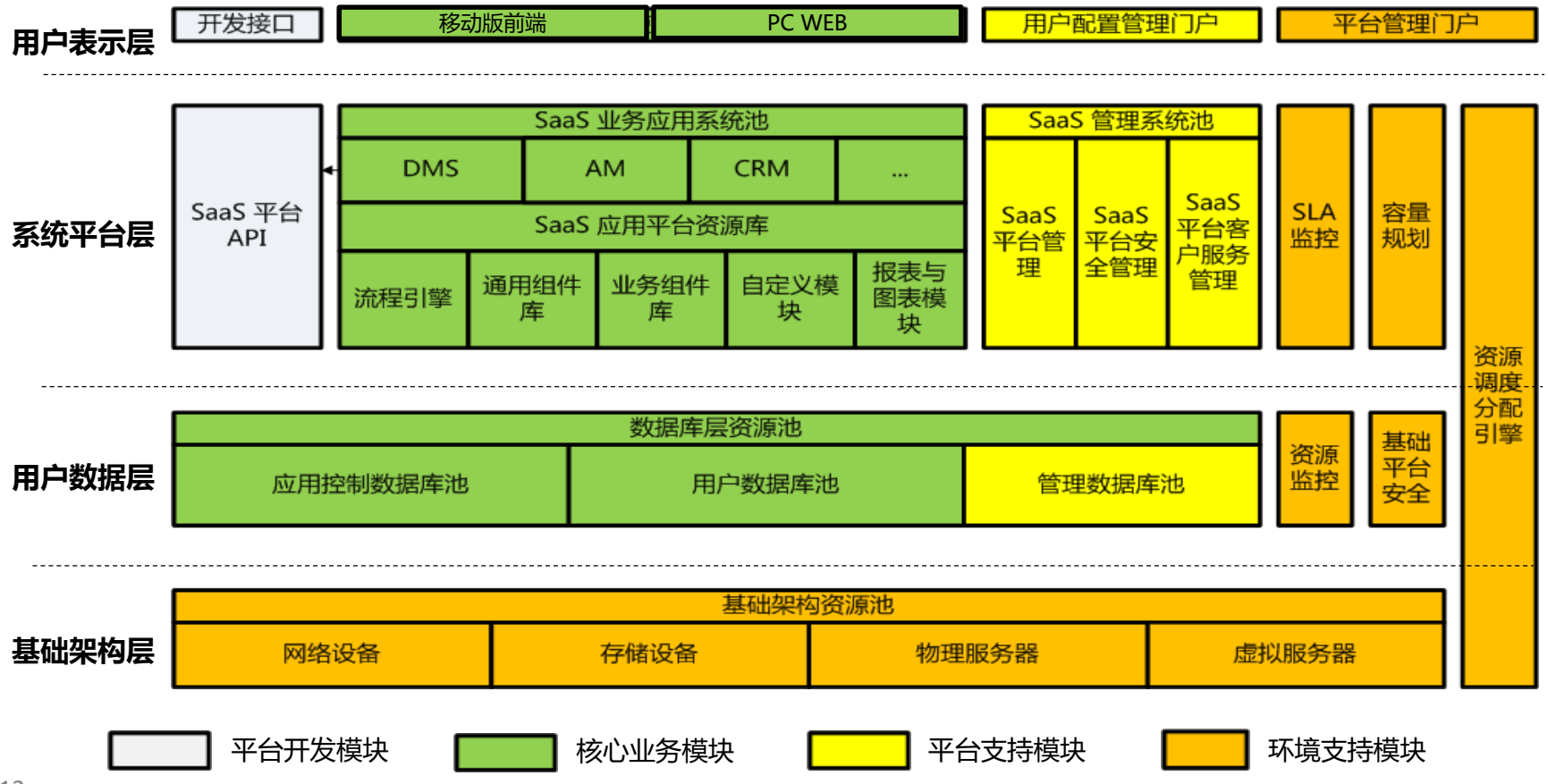


三种隔离方式可以混合使用！
解决：租户多，交付快，成本低，差异大

优点：快速实施，相互干扰小，可以实现复杂的业务差异化，可单独增值服务
缺点：资源利用率不高，成本高，租户数量多以后升级困难
适用：大租户，业务差异化很高的租户

注意：还包括**数据文件、缓存**等任何有分租户存储的都用类似方式加**租户ID**

典型的SaaS多租户架构

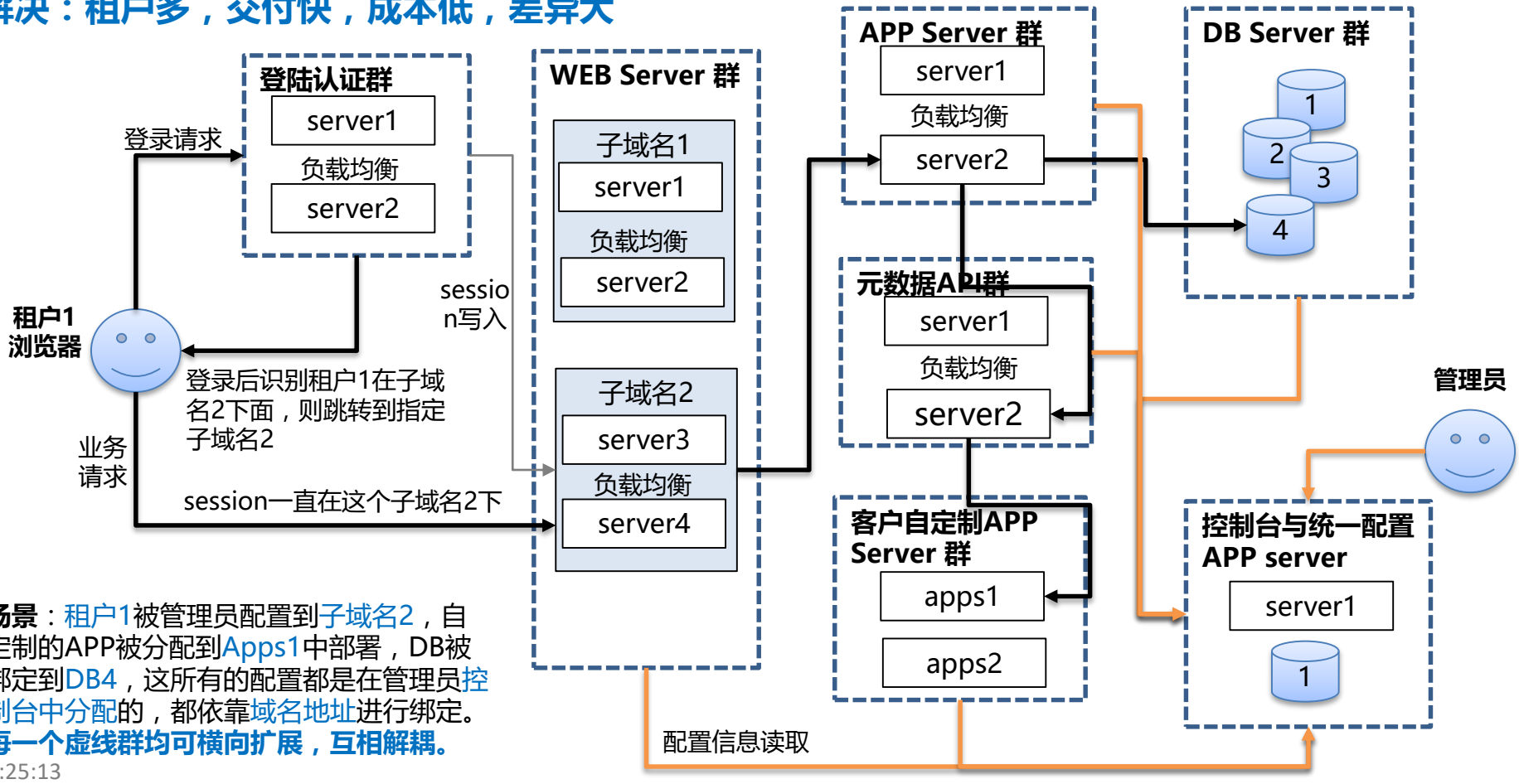


多租户架构下的开发方式

	（1）完全一套代码	（2）租户代码包	（3）应用插件方式
实现要点	•仅一套代码程序 •大量配置文件 •大量的开关程序 •前端渲染程序多且复杂 •极致的服务化实现 •技术标准化要求高 •动态生成的JS脚本多 •需要使用规则引擎等很多软编码技术 •研发团队人员少而精	•核心代码是一套 •每个租户可单独定制代码包 •代码标准化要求较高 •研发人员水平可参差不齐 •代码包和应用部署关系清晰	•核心代码是一套 •核心功能尽量完备 •热插拔的租户插件部署 •服务化和接口标准要求高 •数据隔离、松耦合的应用在容器中执行且隔离
缺点	•程序逻辑复杂 •性能优化难度大 •复杂度太高的业务逻辑实现困难	•代码版本控制困难 •开发团队人员较多 •程序集成难度大 •代码只能放在供应商侧	•应用托管技术构建需要成本
优点	•综合研发成本最低 •代码版本控制容易	•复杂度高的逻辑容易实现 •容易单独发包升级 •性能优化容易	•租户间的应用完全松耦合 •按需定制功能插件 •可由使用方自主开发插件 •可实现应用商店、应用托管

多租户系统通常的部署模式

解决：租户多，交付快，成本低，差异大



场景：租户1被管理员配置到子域名2，自定制的APP被分配到Apps1中部署，DB被绑定到DB4，这所有的配置都是在管理员控制台中分配的，都依靠域名地址进行绑定。

每一个虚线群均可横向扩展，互相解耦。

思考题

- 电商网店平台的多租户特点是什么？设计与研发上要注意什么？
- CRM等企业管理系统的多租户特点是什么？设计与研发上要注意什么？

目录

存量系统的常见问题

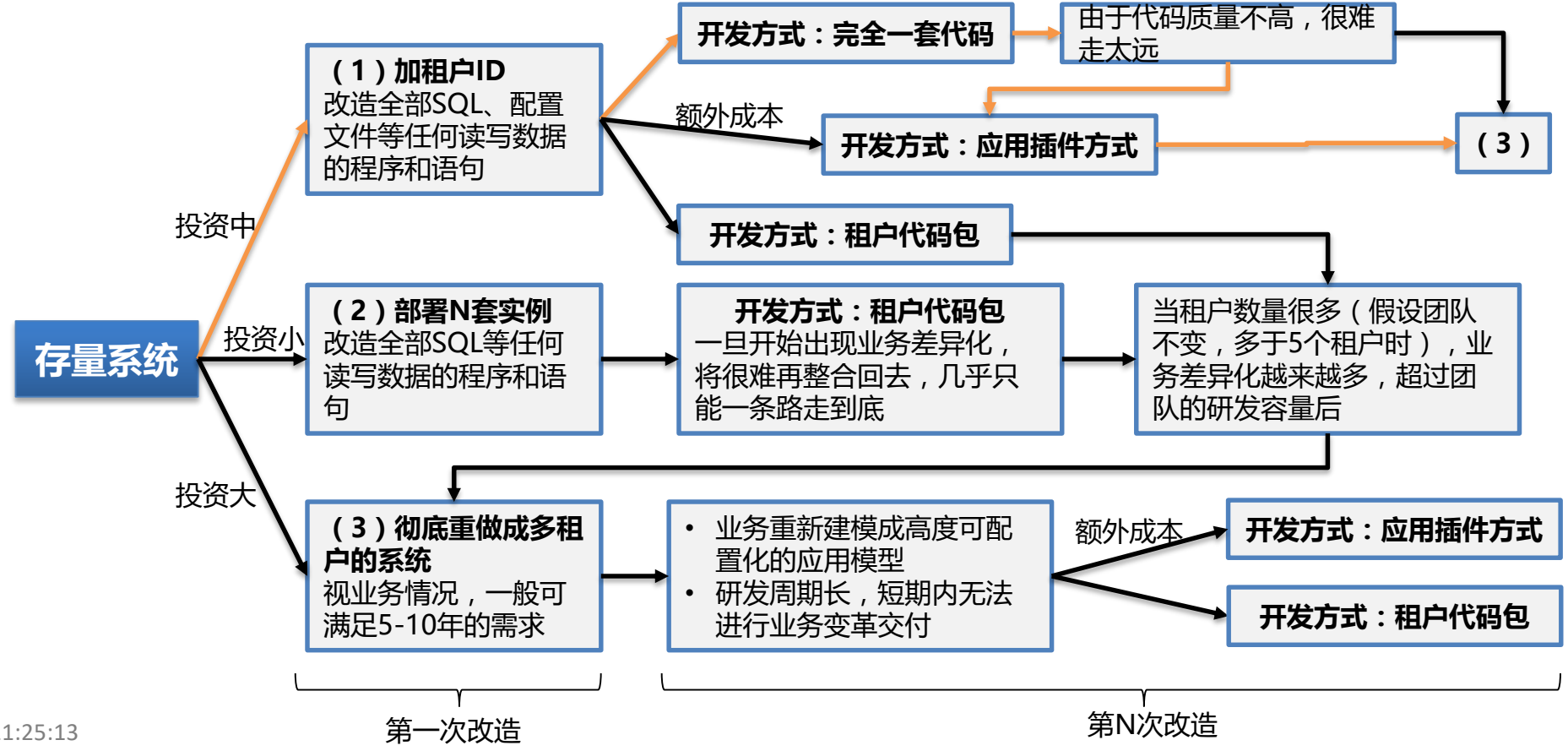
- 存量系统数据量较多
- 数据结构可能比较随意，标准化程度不一定高
- 各种配置项比较随意
- 功能模块、服务的切分颗粒度不一致
- 代码很多、且很多代码可能新人从来没有改过
- 性能扩展困难
- 部署方式难改变

存量系统的常见问题应对思路

问题	应对思路
存量系统数据量较多	如果需要按租户拆分，先加租户ID，再进行数据合并和拆分。 如果租户变多不会影响性能，则不用做特别的优化，直接加租户ID。
数据结构可能比较随意，标准化程度不一定高	必须梳理清楚全部数据结构，否则无法进行下一步
各种配置项比较随意	必须梳理清楚全部数据结构，否则无法进行下一步
功能模块、服务的切分颗粒度不一致	如果不做功能权限的控制，则不用做梳理。 如果要做功能权限控制，则必须梳理
代码很多、且很多代码可能新人从来没有改过	必须理解全部代码，否则无法进行下一步
性能扩展困难	最简单的方式是分实例部署；要么就先优化、重构再进行下一步
部署方式难改变	是否影响多租户改造，如果不影响则尽量不改变

存量系统改造多租户的长期应对策略

橙色路径是大多数情况下的推荐路径；纯新系统推荐路径3，Salesforce是直接选了路径3。
举例：某国际第一的通信设备公司，订单有200不同版本，500个差异化字段，



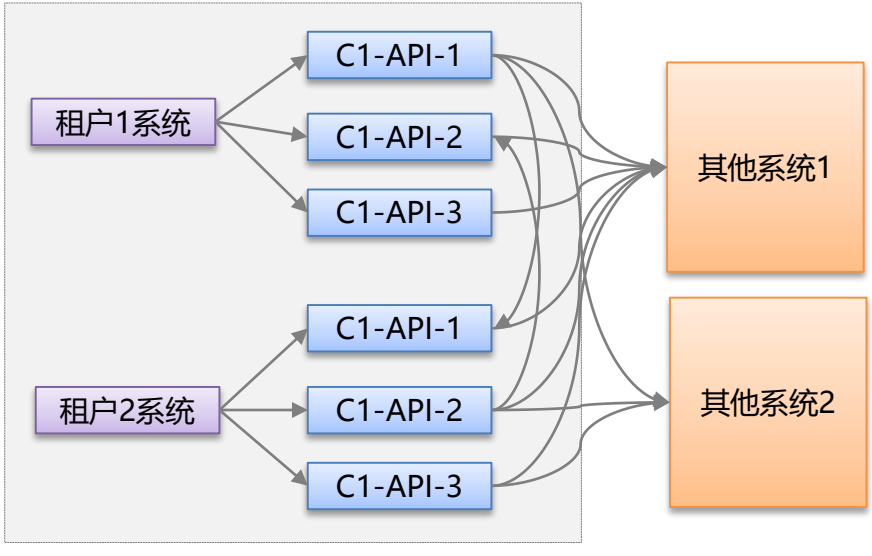
目录

- SaaS、PaaS和多租户的概念和价值
- 多租户的分类与适用场景
- 组件化多租户系统架构如何设计？
- 存量系统如何改造为多租户？
- 多租户系统的API如何设计？

基于元数据模式的多租户系统服务

元数据是描述数据的数据，这种基于元数据的服务API可以**自己描述自己的结构**，且**高度集中化、原子化的服务**、完全SOA方式构建。Salesforce、Workday均采用这种服务API模式。

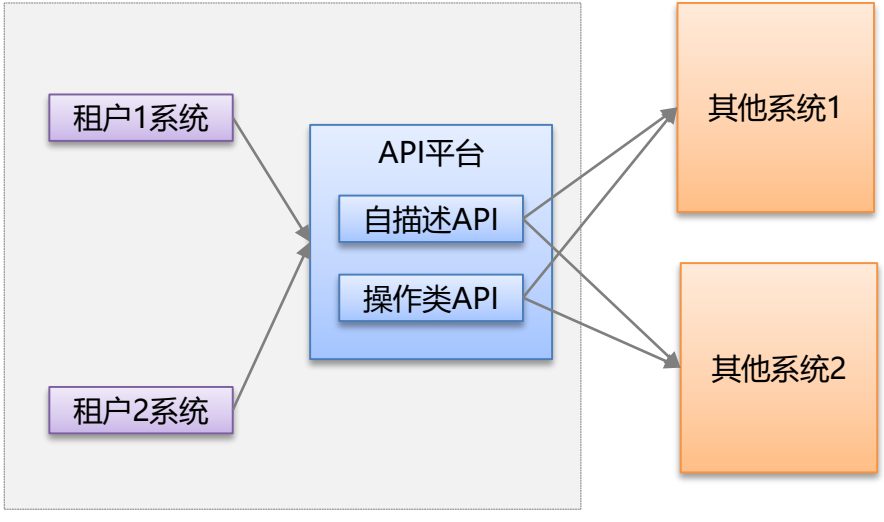
普通的API构建方式



接口数量成倍增加，接口相互调用几何级增加。
举例：某银行有4万多个接口

解决：租户多，交付快，成本低，差异大

基于元数据的API



不论租户数量如何增加，发布的接口永远总是2套固定的API；且不同租户对接同一系统时，只需开发一套对外调用即可，但必须登录握手时传入**租户ID**

调用示例：SaveResult[] = connection.create(sObject[] sObjects);

元数据模式下数据的分层实现

通过VO视图对象、BO业务对象、PO物理对象进行元数据模式下的数据字段级权限控制、租户间的数据模型隔离

VO视图对象

用户A的读写权限下

字段	业务验证规则
客户编号	XXX
客户名称	XXX
客户电话	XXX
客户行业	XXX
客户分类	XXX

用户B的读写权限下

字段	业务验证规则
客户名称	XXX
客户电话	XXX
客户行业	XXX

通过权限控制，不同的用户在BO基础上可读写的字段不同，从而生成了不同的VO，即客户界面字段

用户权限

BO业务对象

某租户的客户数据模型

字段	类型
客户编号	string
客户名称	string
客户电话	string
客户行业	string
客户分类	string

业务数据API表现的数据结构是BO，读写数据都被封装为BO，依靠BO来维护租户间不同的数据结构与完整性。对于API消费方是屏蔽PO，达到数据的安全性

转换与隔离

PO物理对象

字段	类型	长度
租户ID	int	16
主键id	int	16
客户编号	Varchar	32
客户名称	Varchar	50
客户电话	Varchar	32
客户行业ID	int	32
客户分类ID	int	16
字段	类型	长度
租户ID	int	16
分类ID	int	16
分类名称	string	32
字段	类型	长度
租户ID	int	16
行业Id	Varchar	32
行业名称	text	8000

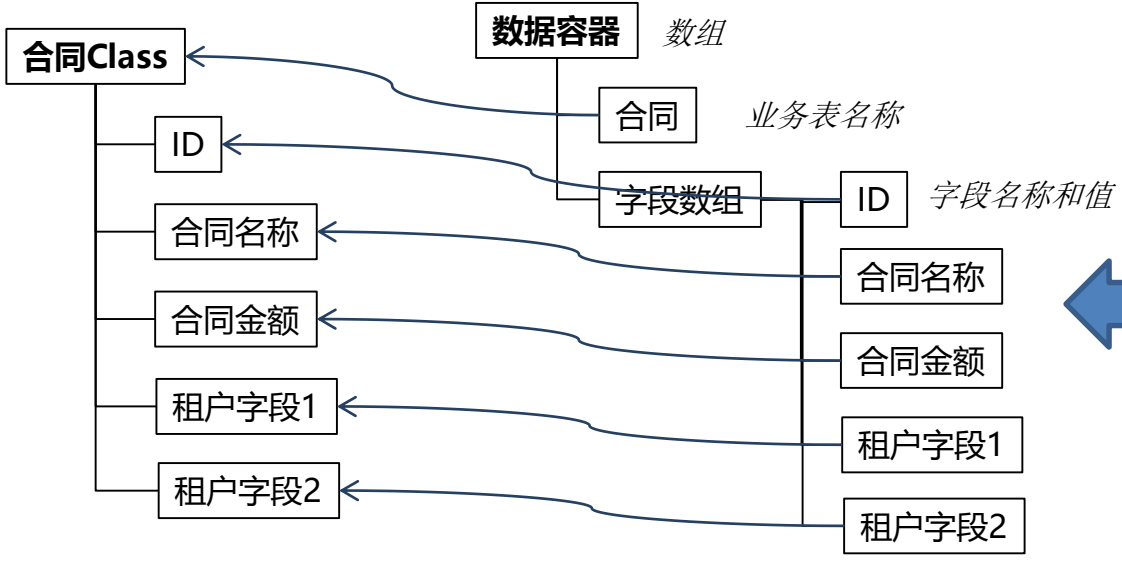
元数据服务的数据传输原理

核心点是将固定的数据结构变为数组方式，自描述API其实就是在描述这个数据的字段相关属性。这种灵活的结构可以方便的将不同租户业务数据的兼容装载、甚至可以兼容所有业务数据模型。
技术上可用**XML、Json、其他键值对、普通数组**等数据格式进行承载描述。

普通的数据对象

元数据模式的数据对象

自描述API

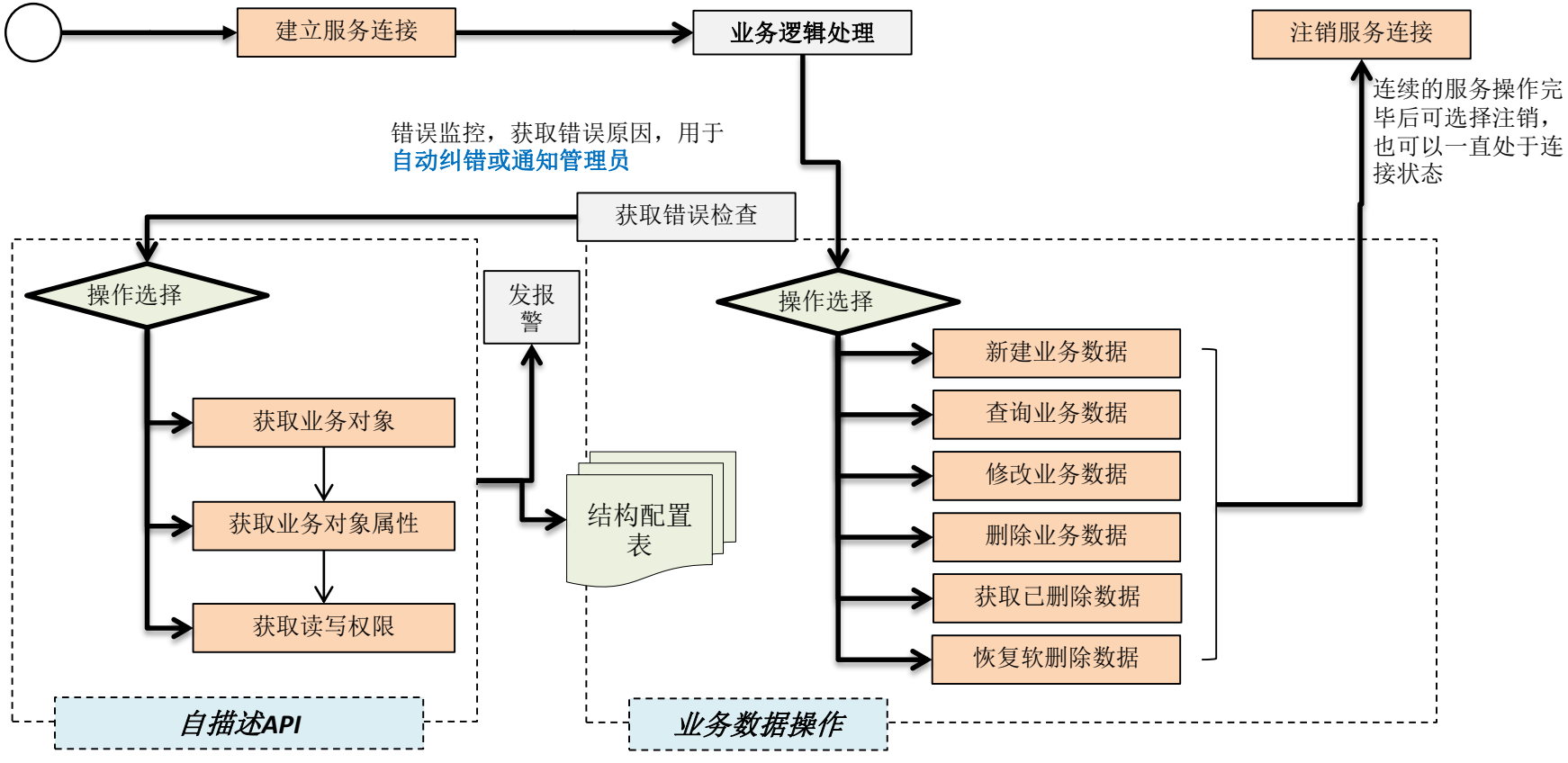


业务表名称	字段名称	数据类型
合同	ID	INT(8)
合同	合同名称	String(32)
合同	合同金额	Number(16)
合同	租户字段1	String(32)
合同	租户字段1	String(128)

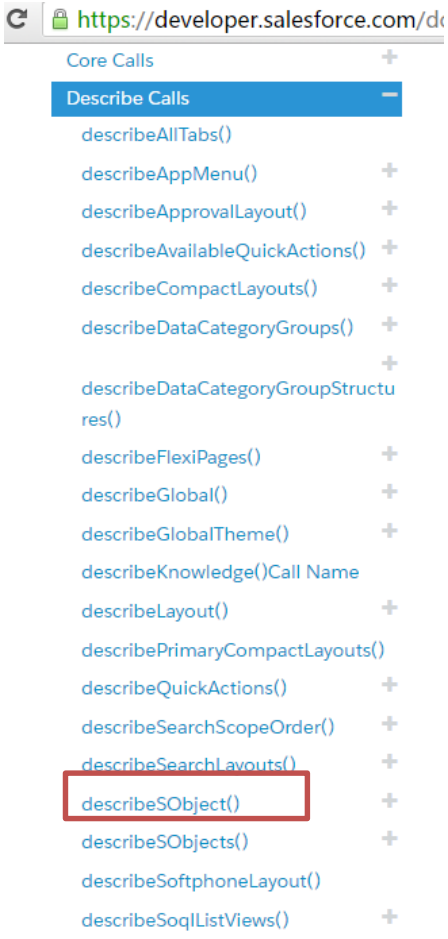
业务数据可以直接保存Key-Value模式

业务数据ID	租户ID	对象ID	属性ID	值
323111	332	5663	004	zhongguo
323111	332	5663	003	riben
23232	334	3456	012	meiguo

典型元数据服务使用场景

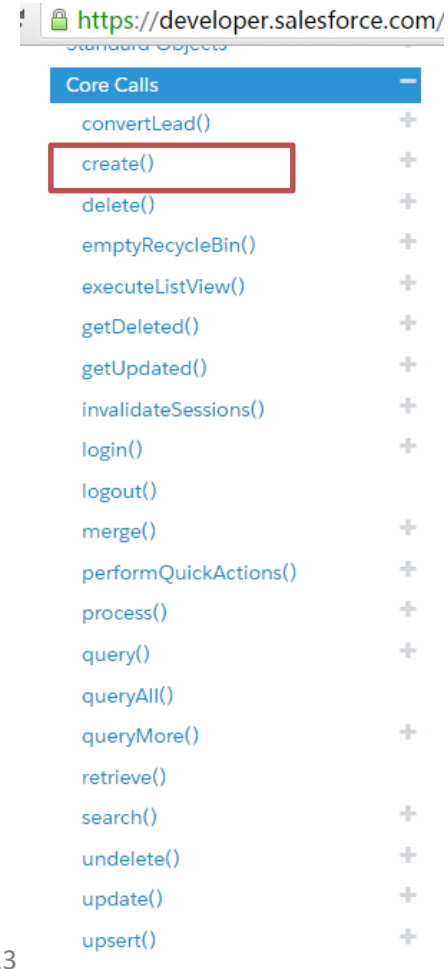


元数据服务代码示例-自描述业务对象




```apex
public void describeSObjectSample() {
    try {
        // Make the describe call
        DescribeSObjectResult describeSObjectResult =
            connection.describeSObject("Account");

        // Get sObject metadata
        if (describeSObjectResult != null) {
          System.out.println("sObject name: " +
                  describeSObjectResult.getName());
        if (describeSObjectResult.isCreateable())
            System.out.println("Createable");

        // Get the fields
        Field[] fields = describeSObjectResult.getFields();
        System.out.println("Has " + fields.length + " fields");

        // Iterate through each field and gets its properties
        for (int i = 0; i < fields.length; i++) {
          Field field = fields[i];
          System.out.println("Field name: " + field.getName());
          System.out.println("Field label: " + field.getLabel());

          // If this is a picklist field, show the picklist values
          if (field.getType().equals(FieldType.picklist)) {
              PicklistEntry[] picklistValues =
                  field.getPicklistValues();
              if (picklistValues != null) {
                System.out.println("Picklist values: ");
                for (int j = 0; j < picklistValues.length; j++) {
                  if (picklistValues[j].getLabel() != null) {
                    System.out.println("\tItem: " +
                        picklistValues[j].getLabel()
                    );
                  }
                }
              }
          }
```

元数据服务代码示例-业务数据创建




```apex
public String[] createRecords() {
   // Create two accounts
   String[] result = new String[2];
   Account account1 = new Account();
   Account account2 = new Account();

   // Set some fields on the account object
   account1.setName("The Brick Hut");
   account1.setBillingStreet("403 McAdoo St");
   account1.setBillingCity("Truth or Consequences");
   account1.setBillingState("NM");
   account1.setBillingPostalCode("87901");
   account1.setBillingCountry("US");
   // Required Name field is not being set on account2,
   // so this record should fail during create.
   // account2.setName("Camp One Creations");
   account2.setBillingStreet("25800 Arnold Dr");
   account2.setBillingCity("Sonoma");
   account2.setBillingState("CA");
   account2.setBillingPostalCode("95476");
   account2.setBillingCountry("US");
   Account[] accounts = { account1, account2 };

   try {
      // Call create() to add the accounts
      SaveResult[] saveResults = connection.create(accounts);
      // Iterate through the results.
      // There should be one successful creation
      // and one failed creation.
      for (int i = 0; i < saveResults.length; i++) {
         if (saveResults[i].isSuccess()) {
            System.out.println("Successfully created Account ID: "
                  + saveResults[i].getId());
            result[i] = saveResults[i].getId();
         } else {
            System.out.println("Error: could not create Account "
                  + "for array element " + i + ".");
            System.out.println("   The error reported was: "
                  + saveResults[i].getErrors()[0].getMessage() + "\n");
            result[i] = saveResults[i].getId();
```

多租户长远发展战略原则总结

原则	解释
业务规范化建模 业务抽象能力持续提升	业务不规范化建模，就无从元数据化，也无法进行业务模型的抽象和持续提升、应用和服务也难解耦，最终是有限的业务规模，无法快速复制成功经验，无法满足更多的差异大的客户需求
元数据化驱动	元数据化本质就是标准化的落地方式之一，多家顶级saas公司的经验，不进行元数据化，无穷无尽的代码和设计几乎无法维护，路不会走太远即会夭折或崩溃；有了元数据化，技术标准化、功能可配置化是顺水推舟一样。
应用、服务的解耦和颗粒度控制	没有解耦和颗粒度，按需购买就成了空话，性能扩展也只是在烧程序猿的脑子，没有建立在庞大易扩展的基础设施之上。许可证、权限控制也将随着解耦和颗粒度而变得异常容易。

最终解决：租户多，交付快，成本低，差异大

请继续关注系统通道课程，后面有专门课程介绍业务标准化、技术标准化和元数据化实现方法

谢谢各位！