**数据交换协议指南**

——Anycmd

2014.10.12

目录

[1 信息（Info） 3](#_Toc400884979)

[2 本体（Ontology） 3](#_Toc400884980)

[3 分类器（Classifer） 5](#_Toc400884981)

[树、层次、分形、监管、组织、构造定律 5](#_Toc400884982)

[有序化 5](#_Toc400884983)

[一致性 6](#_Toc400884984)

[具体化 6](#_Toc400884985)

[总结 7](#_Toc400884986)

[4 数据模式（Data Schema） 8](#_Toc400884987)

[5 关系（Relationship） 9](#_Toc400884988)

[6 信息格式（InfoFormat） 10](#_Toc400884989)

[7 本体码（OntologyCode） 10](#_Toc400884990)

[8 本体元素（Element） 11](#_Toc400884991)

[9 动作类型（ActionCode） 11](#_Toc400884992)

[10 信息标识（InfoID） 12](#_Toc400884993)

[10.1 单列Guid标识策略 12](#_Toc400884994)

[10.2 多列联合信息标识策略 12](#_Toc400884995)

[11 实体（Entity） 13](#_Toc400884996)

[12 命令（Command） 14](#_Toc400884997)

[12.1 命令的领域模型（Domain Model） 14](#_Toc400884998)

[12.2 命令传输模型（Command DTO） 15](#_Toc400884999)

[13 并发和处理机制 16](#_Toc400885000)

[14 消息（Message） 17](#_Toc400885001)

[15 响应（Response） 17](#_Toc400885002)

[16 状态码（StateCode） 18](#_Toc400885003)

为了帮助理解，特将PrivilegeBigram（权限二元组）这样的实体替换为‘教师’、‘学生’这样的更加贴近生活的不那么抽象的实体。从中也可以看到Anycmd的权限数据交换模块其实是一个通用的模块，不仅仅用来交换权限数据，任何结构化的数据都可以用它来交换。

注意：这里的接口只是用来数据交换的，中心节点和各个业务节点间通过这个接口来交换权限数据到各自本地。业务节点永远只使用交换到自己本地的权限数据，远端不存在供应用节点调用以验证权限的这样的接口。Anycmd是分布式的。

由于“数据交换平台”本身的抽象性复杂性，所以要说清楚它的接口不可避免的需要借助一些领域“词汇”。

“词汇”是事物的指代，所指代的事物根据所处的上下文可能是实体事物如“教师”、“服务器”也可能是逻辑事物如“服务端”。像“教师”和“服务器”这样的词汇是比较容易认同的，但对于师生基础库中的某些词汇来说事情可能并非这样，因为“数据交换”这件事并不是一个我们在日常生活中经常接触到的领域。

数据交换平台的架构和实现是一件复杂的事情，所以我们需要将这个领域大问题拆分为具体的小问题，而每一个小问题必须有一个“词汇”进行标识从而形成“领域语言”，否则将无法沟通和写进文档。每一个词汇的定义都经过了反复的权衡和推敲，已尽力做到与行业共用词汇保持一致。理想情况下如果读者脑中贮存的“词汇”的内涵是行业共用词汇的公认解释的话，读者完全可以凭借自己的知识结构和师生数据交换平台概念体系的一致性这两点而直接阅读本平台的任何文档。

## 信息（Info）

本文中会反复出现“信息”概念，如“InfoID（信息标识）”、“InfoVaue（信息值）”、“InfoString（信息字符串）”、“InfoStringConverter（信息字符串转化器）”等。为了尽力帮助阅读，本节会详述“师生数据交换平台”的领域专家是基于什么来标定信息概念的。

信息与数据和信号有些不同，“信息”二字的下面隐含了“翻译”这件事情，也就是说“能翻译”的数据和信号才是信息。比如，这里书写一个字符“1”读者能知道它是什么意思吗？读者看到“1”只是收到了一个视觉“信号”，如果交换系统不告诉你这里的字符“1”是性别“男”的意思的话恐怕字符“1”对你来说就只是一个无意义的视觉信号罢了。收到字符“1”并将它识别为性别“男”这就是“翻译”。

信号被翻译成已知的事物才能成为信息。而“翻译”是什么？是“映射”，激进一下，不妨把信息直接定义为“映射”，信息是：抽象到抽象的映射，抽象到存在的映射，存在到存在的映射，信息就是映射。

那么A被映射到B，B被映射到C，C再被映射到A，这里的映射是不是信息？是。如果这些映射不是信息，那么我们如何知道这是一个闭合的映射环的？我们之所以能够识别出这些映射是否有意义是因为我们有“知识”和“智慧”。“知识”是什么呢？

信息是映射。而“知识”是选择映射路径的能力。比如“今天天气预报说明天有雨，于是小明取消了明天晒被子的计划”这就是“知识”。小明收到了明天有雨的信号，然后在头脑中做了一系列的映射“时间映射、下雨和水映射、水和湿映射、湿和被子映射、湿被子和睡觉不舒服映射 等”关键是在这一系列映射后小明做出了“明天不晒被子”的映射，从而“明天的雨水无法映射到小明的被子”小明选择了映射的路径，选择映射路径的能力就是“知识”。

数据交换进程中所进行的一切活动都是事先设定好的“映射”并无“知识”和“智慧”，有智慧的是“人”，数据交换平台将数据收集过来，然后站在平台外部的“人”使用这些数据进行“决策（选择映射路径）”：比如，领导看到某个老师各种条件都不错头脑中考虑了一下是否将这个老师与“教育标兵”映射。

整个协议的设计和任何相关文档的书写都遵循了这里对“信息”概念的界定。

## 本体（Ontology）

在计算机科学与信息科学领域，本体是指一种“形式化的，对于共享概念体系的明确而又详细的说明”。本体提供的是一种共享词表，也就是特定领域之中那些存在着的对象类型或概念及其属性和相互关系；或者说，本体就是一种特殊类型的术语集，具有结构化的特点，且更加适合于在计算机系统之中使用；或者说，本体实际上就是对特定领域之中某套概念及其相互之间关系的形式化表达（formal representation）。本体是人们以自己兴趣领域的知识为素材，运用信息科学的本体论原理而编写出来的作品（artifacts）。本体一般可以用来针对该领域的属性进行推理，亦可用于定义该领域（也就是对该领域进行建模）。此外，有时人们也会将“本体”称为“本体论”。

——以上文字摘自[维基百科](http://zh.wikipedia.org/wiki/%E6%9C%AC%E4%BD%93_(%E4%BF%A1%E6%81%AF%E7%A7%91%E5%AD%A6))

**本体具体是什么？**

本体是领域、是边界、是讨论问题时设定的上下文。如果“教师”是一个本体，那么“教师”这个本体（领域、边界、上下文）下有些什么呢？

有教师作为一个“人”的基本属性、有教师作为一个“教育人”的基本属性，有教师在“电子世界”的属性，这些都是教师的“**数据属性**”。这里的“人”、“教育人”、“电子世界”是什么？它们也是本体，“基础库系统”也是一个本体，当“教师”本体和“基础库系统”本体作用的时候基础库系统又赋予了教师一些“系统属性”，这些系统属性与教师的其它属性没有任何本质差别。

本体下还有**“行为属性”**，东直门小学新招聘了一个老师，“招聘”就是“行为”，基础库并非教育大而全系统，所以基础库不会给教师赋予“招聘”动作，但基础库会给教师赋予“创建”动作，因为“创建”实体记录是在“基础库”领域边界下有意义的概念，这个动作可以使用一个编码为“Create”的字符串来让计算机理解，同理“更新”教师的信息，“删除”教师这都是教师本体下的行为。

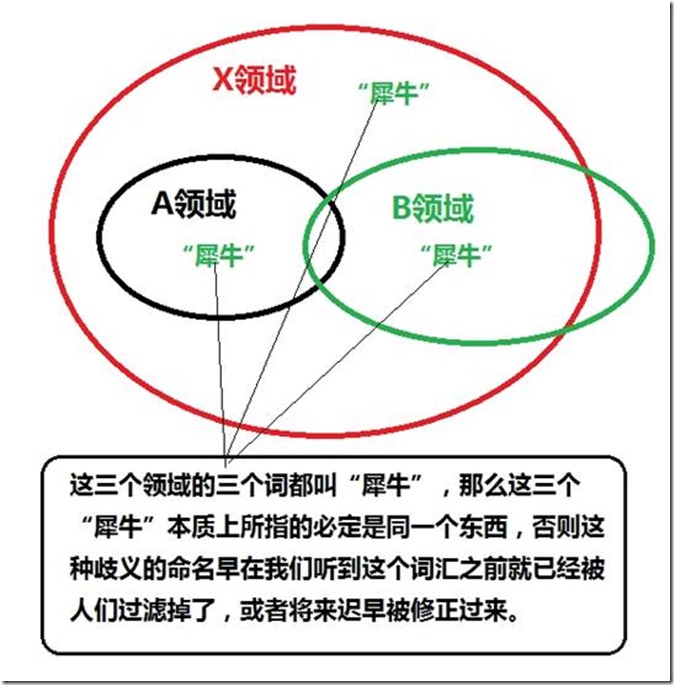
本体下还有**“数据合法性”**这样的概念，上面我们说教师有各种“数据属性”，那么“数据合法性”就是教师本体下的另一个自然概念，这种验证数据合法性的规则称作**“数据规约”**。

本体下还有**“行为控制”**这样的概念。上面说道“教师本体”下有行为属性，这些行为不是任何系统任何用户都能实施的，那么“行为控制”就是教师本体下的另一个自然概念。ACL是控制在“行为/运动/变化”上的，这一点一定要明确。“教师”本体界定了一个领域、边界和上下文，在这个本体下不仅有“数据属性”、“行为属性”、“数据合法性”、“行为控制”等概念，还有很多其它概念，而本文档所关注的是“教师”“数据库”“数据交换”这三个本体交集出来的概念子集。

**本体有层级性：**

本体的层级性是什么？比如教师是人。教师是个本体，人也是个本体。教师本体是从人本体继承而来的，这就是它们的层级关系。这是“本体的目录”，这种层级关系有时候需要考虑有时候无需考虑。如果需要考虑层级的话则这种层级关系可以体现在“本体编码”上，编码的时候给个偏移。比如，如果人本体被编码为“ren”，那么教师本体可以编码为“ren.js”，同理学生本体可编码为”ren.xs”。对本体进行组织会把问题变得相当复杂，所以基础库暂不对本体进行组织，教师本体编码为“js”学生本体编码为“xs”，也就是说暂将本体码视作“字典”，字典与目录的唯一区别是没有层级。倘若后续需要为本体引入层级目录的话，那么只需在本体码上做文章或者在“学生本体机”、“教师本体机”、“家长本体机”、“账户本体机”等等本体机系统的前端放置“本体目录机”就可以了，这不涉及对原有架构的变更。

## 分类器（Classifer）



### 树、层次、分形、监管、组织、构造定律

人类文明近万年无数人维护着的知识结构体是个相当良好的树形的。任何一个分支在使用到任何一个词汇时引用的本意都不与它的父节点冲突，既然这样那么只要我们曾经系统的熟悉过整棵树上某一个分支或者部分分支的话就可以通过上溯和下钻来容易的熟悉其它分支的知识。

### 有序化

熵是无序程度的度量。而知识是在无序中发现的规律，知识树用来组织和标识这些知识。知识树是我们对世界的建模。研究熵的目的应该也是为了发现无序中的有序，应用知识按照自然规律去改造无序按照意愿重新排列它们。知识树能够良好的记录我们认知到的知识，知识树能够良好的建模这个世界可能是因为这个世界本来就是个树结构。

### 一致性

学习生物课本的时候我把“组织”理解的太简单了。人类的知识结构组织的是十分良好的，比如“组织”这个词，它在任何领域的本意一定都是一样的，如果在整棵人类知识树的某个节点下对“组织”这个词的解释和它的父节点不一致的话，这种不一致往往会在我们第一次听到这个词之前就被过滤掉了，或将来迟早被纠正过来。

### 具体化

**什么是目录？**

目录是对具体本体下的资源进行单元划分。这种划分单元的方式有一个明显的特点是——目录具有层级关系。需要注意的是：一个资源至多属于“一种”目录的“一个节点”，不可能属于多个节点，属于多个节点的那种不叫目录。一个资源可能具有“多种”目录属性但一个属性只有一个取值。

**什么是类别？**

类别跟目录的唯一不同是类别没有层级关系。一个资源同样至多属于“一种”分类法中的“一个节点”，不可能属于多个节点，属于多个节点的那种不叫类别。类别就是对整个资源集按照某种分类标准进行枚举。如果我们按照人类的性别对人类进行分类的话，那么一个人要么是男的要么是女的，人类的分类法不可能会去把一个人既标记为男的又标记为女的。

**什么是标签？**

标签不分层级。一个资源可以有多个标签。

**什么是组？**

一个组是一个地图，整个森林中的树木是一个集合，树木类型的一个组则是一张记录树木位置的坐标图，图上的树木是整个森林的一个子集。

用户绘制一张记录树木的位置的坐标图，然后可以把这张图交给某个工人，从而工人按图索骥去伐木。

不在图上的树木可能是不允许砍伐的。Group跟手工仓库的区别是一个Group里的资源的类型都是一样的，而手工仓库里的资源可以不是同一类型的

（手工仓库中的一条资源记录需要ObjectType+ObjectID两个字段来标识而组只需一个字段）。

不添加岗位模型，岗位是一种有目录的工作组。可以为岗位授权，既然岗位是工作组，

那么自动已经可以授权。不添加专门的职务模型。职务是字典，通过系统字典表现职务。

**举例：**

淘宝的商品“类目”是个什么东西？这可能是淘宝自创的概念，如果你进一步了解会发现淘宝的类目概念还分“前台类目”和“后台类目”，前台类目就是“标签”，后台类目就是“类别”。前台类目多对多，后台类目多对一。

教师的目录是个什么东西？政府作为一个大的行政单位，它有很多很多资源，为了对这些资源进行良好的管理政府划分了很多“行政单位”。同样，教育部也对教育单位做了单元划分，区县、学校、电教馆、教科所等这些就是教育目录。一个老师必定是属于且只属于一个目录的，基础库直接使用区县、学校、电教馆等这些自然的目录来组织师生资源。

在工作的开展上，一个老师有可能效力不只一个学校，但基础库不是教育ERP系统，在基础库中一个老师至多属于一个目录，并且这个目录在基础库看来是对“数据”进行组织的目录，基础库不是一个大而全的系统，它所关注的是数据的准确性、可靠性等，它所关注的是“数据”和“数据交换”而具体业务应用需要由专门的应用系统去做。

基础库本身并不去理解、解释教师、学生、家长等本体以及各个本体下面的任何应用概念，基础库唯一理解和解释的是基础库系统赋给本体们的“系统属性”。基础库不知道“统一身份认证系统”的登录名是个什么概念，也不知道“一线通系统”关注的手机号码是个什么概念，也不知道“教师培训系统”关注的最高学历是个什么概念……，基础库只知道这些数据项的模式是什么，规约是什么，谁能对某个数据项干什么谁不能干什么……谁初始化的某个数据项，谁最后修改的这个数据项，谁修改过这个数据项并修改成了什么……。基础库负责数据的准确性、可靠性、安全性，基础库负责精细、高效地记录历史，基础库为应用系统提供数据服务。

### 总结

系统中一般有三次分类，三次分类从整体上看就是一种目录。

三次分类是：

1 划分资源类型，此为第一次分类，一个资源类型实体对应的是一种资源的全部数据集；

2 划分资源元素，此为第二次分类，这次分类是区分资源的属性，每一个属性对应定义一个值域。

3 划分资源元素的值域单元，此为第三次分类，本次分类是对整个属性值域进行单元划分，终极的单元是不划分，即每一个取值就是一个分类节点（在计算机世界中只有“离散”不存在“连续性”这个概念）。

由于基本上所有的值都是可以比较大小的，从而第三次分类往往会基于算数运算划分单元。

对与系统内部来说三次分类基本足够了

单纯的一次分类称作分类，把多次分类从整体上看的话如果这些分类具有偏移量则就是目录。

目录上不封顶下不封底！

整套人类的知识就是一个目录，而这个目录上的每一个节点就是一次分类。

所有的问题最终划归为：

1 如何组织数据；

2 如何表现数据的运动。

**模型 = 数据 + 运动。**

只要用劲理解了“目录”则部门、岗位这两个概念就弄明白了，机构的本质就是目录，而岗位是具有目录属性的工作组。

**机构：**毫无疑问它是目录节点；

**岗位：**岗位是绑定到目录的工作组；

**工作组：**工作组是跨越目录的资源集，这个资源组中“具有主体”这种类型的资源元素，具有目录属性的工作组的意思是该组中的资源被约束为只能来自本目录和它的下级目录。

**组：**资源集，这个资源集中不一定只有一种类型的资源。

**简单组：**单一类型的资源集；

**职位：**不是目录。职位属于分类或字典，职位不是绑定到目录的，职位没有目录属性。不同目录的人员可能拥有相同的职位。

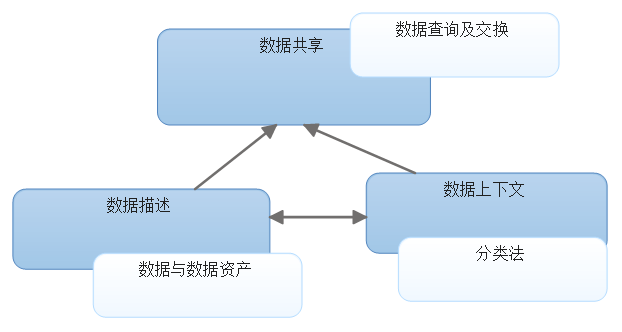
**人员：**普通的资源。

所有的资源为了方便管理都可以扩展个目录属性，人员往往扩展有目录属性。

## 数据模式（Data Schema）

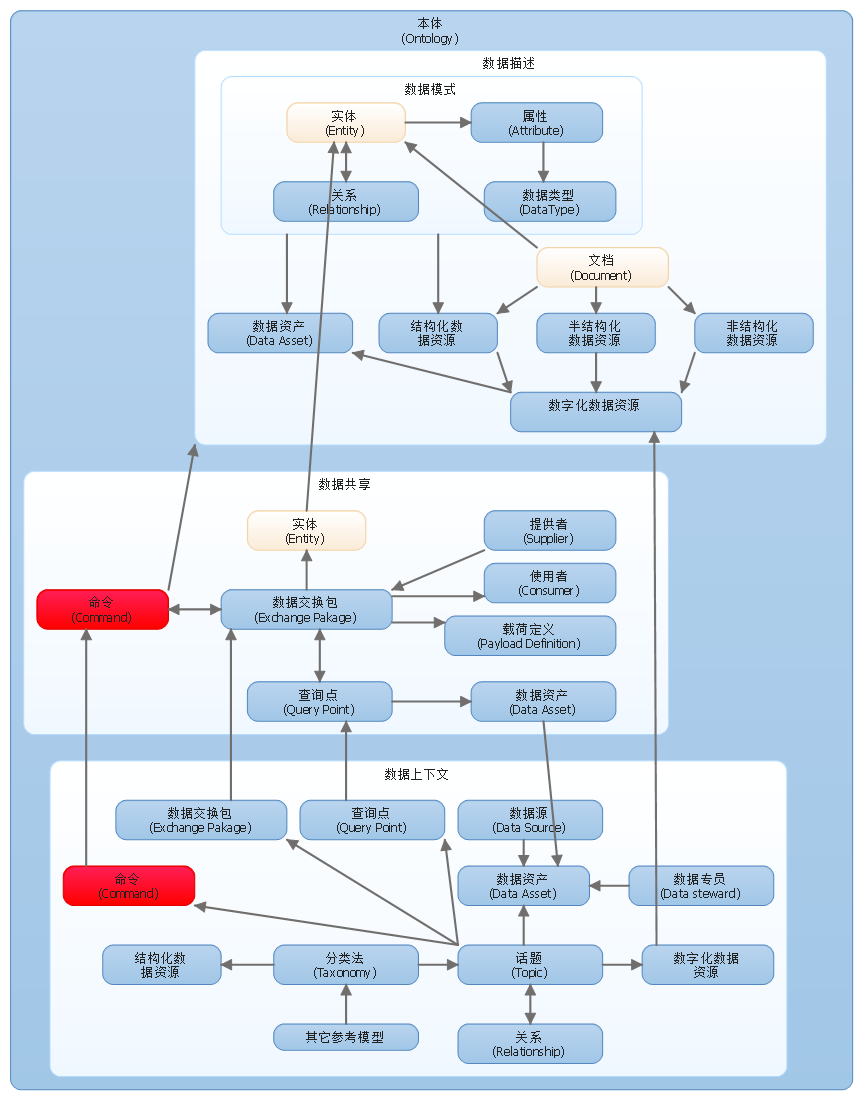
数据交换指的是在不同节点之间经常性发生的针对于固定模式或需求的数据的往来交互事务。本节就是用以定义师生实体模型的固定模式的。

**首先**，本体（**Ontology**）是一个概念框架，给出一套词汇标识一套概念，这些词汇就是术语。本体本身也需要标识，比如我们说“物理学”，物理学这三个字就标识了本体。对于本体这个形而上学的东西读者没必要纠结，只需知道在数据交换平台中计算机是使用编码来标识本体的就可以了。如“JS（教师）”标识一个本体，“XS（学生）”标识一个本体，在教师本体概念框架下“当前状态”指的是教师的状态，而在学生本体概念框架下“当前状态”指的是学生的状态，再比如：在教师本体框架下有“所教学科”这样的概念但在学生本体下没有，而在学生本体下会有“家长联系电话”这样的概念但在教师本体下没有。数据交换平台使用固定的唯一的编码来识别每一个本体，将教师编码为JS，学生编码为XS，这种编码是不区分大小写的字符串，JS和XS就是“本体码（OntologyCode）”，关于本体码在下文有专门的一章可进一步了解。本体完整的涵盖一类实体的所有属性，而不同的节点对同一类实体所关注的属性往往不同，每一个节点所关注的通常是一类实体所有属性的一个子集，这个子集我们称为主题（**Subject**）或者话题（**Topic**），有时也称作数据上下文（**Data Context**），如教育一线通系统关心JS的手机号码而不关心教师的出生年月，教育一线通的上下文就是“发手机短信”这件事，数据上下文采用分类法进行识别。



**其次**，实体上有众多属性，如教师的性别、出生日期、身份证件号、所属目录等，这类属性我们称为“元素（**Element**）”或者“属性（**Attribute**）”，为了表明它从属于一个本体才有意义的性质通常也称作“本体元素”，同样平台使用唯一的字符串编码来识别每一个本体元素，这个编码我们称作本体元素码（ElementCode）。既然本体元素是对实体数据项的描述那么为什么不直接称为“数据项”而要使用“本体元素”这个新词呢？这是因为，本体元素上封装的不仅仅是实体的数据项，还有其它一些类别的信息，如本体元素级权限等。本体元素在下文也有专门的一节来描述。**而**，数据模式是在本体下使用实体、属性、关系、数据类型定义的固定的数据结构。本体元素编码了实体属性，定义了数据项的类型、值域、是否可空等一系列数据属性。师生实体的数据模式由中心节点的一个服务定义[GetOntologys](/Api/EDIContext/Ontology/GetAll)

下面用一张图来描述本节提到的概念之间的关系。



## 关系（Relationship）

关系也是一种实体，它由相关的两个实体和关系的方向三者唯一标识。在师生基础数据领域中教师和学生是有关联关系的，如学生在不同年级师从不同的老师，同样一个老师可以教授很多学生，教师和学生之间具有多对多的关系。类似这样的业务关系已经天然存在而无需添加技术性的外键字段去实现。

## 信息格式（InfoFormat）

目前支持两种信息格式：json和xml。本交换协议基于json和xml格式增加了一个限定，即json和xml格式的数据必须是扁平的，扁平在这里解释为不具有层级关系或者说不具有复杂类型值。该约束目的在于将json和xml格式的数据限制为简单的键值对形式，键为本体元素码。

信息标识和信息值均是json或xml格式的扁平的数据，举例

{‘id’:’ E872A51E-6440-4C49-B5FB-7A1D1CAE4B28’}

和

<InfoIDValue><id> E872A51E-6440-4C49-B5FB-7A1D1CAE4B28</id></InfoIDValue>

是信息标识，

{‘SJHM’:’15261855522’,’XM’:’李白’}和<InfoIDValue><InfoID><SJHM>15261855522</SJHM><XM>李白</XM></InfoID></InfoIDValue>

也是信息标识，但

{‘XBM’:’1’, *‘LXFS’:{‘SJHM’:’15261855225’,’Email’:’test@dd.com’}*}

和

<InfoIDValue><XBM>1</XBM>*<LXFS ><SJHM>15261855225</SJHM><Email>test@dd.com</Email></LXFS>*</InfoIDValue>不是信息标识，因为它们不是扁平的键值对形式（注意红色部分）。信息值同理。

* **信息键不区分大小写**

信息字符串（InfoString）中的信息键是不区分大小写的。下面两个信息字符串意义相同：

{XM:’李白’,XBM:’01’}

{xm:’李白’,Xbm:’01’}

* **XML格式的约定**

合法的xml必须有（强制）根节点，xml格式的信息标识和信息值是需要根节点的，根节点约定（非强制）命名为InfoIDItems和InfoValueItems或InfoItems。

从上面说明信息格式的例子中可以发现有“Id、SJHM、XM、XBM”等几个编码，它们是本体元素码。本体元素文档地址在本体码文档中[GetOntologys](file:///D:\Api\EDIContext\Ontology\GetAll)

* **信息格式的可扩展性**

信息格式的扩展方向有两个：1是支持增加新的信息格式；2是支持充血已有的信息格式。

1支持增加新的信息格式

虽然平台只支持json和xml两种信息格式，但是支持添加新的信息格式的。每一种信息格式对应有自己的信息字符串转化器（InfoStringConverter）插件，添加新的格式只需编写相应的信息字符串转化器插件即可。

虽然如此但开放数据交换领域应用最广泛的格式只有json和xml，平台架构时将信息格式设计为可扩展的是为了延长本交换平台架构的生命。

2充血已有的信息格式

我们知道，该版本的信息格式被定义为简单的扁平的键值对形式。如果有合理的理由它是可以扩展出层级结构的。

## 本体码（OntologyCode）

本体是实体（Entity）的分类，本体码用以标识一类实体（Entity）。如教师是一类实体（Entity），学生是一类实体（Entity），它们分别对应教师（记为JS）本体和学生（记为XS）本体。“JS”和“XS”就是本体码。在本体文档中可以查到所有的本体码。本体文档地址[GetOntologys](file:///D:\Api\EDIContext\Ontology\GetAll)

## 本体元素（Element）

本体元素是对实体数据项的描述对象，它描述了数据项的类型、值域、可空性等一系列与数据交换相关的属性。值得说明的是，有些数据项的值域是一系列枚举值，如性别。教育部在《中华人民共和国教育行业标准》中针对教育行业中有意义的每一个业务概念做了定义，并对数据项的取值做了详细的枚举。这种可枚举类别的数据在师生信息库系统中被称作信息字典（InfoDic），字典的每一项取值称作信息字典项（InfoDicItem）。



信息字典可以通过中心节点的一个服务获得[GetInfoDics](file:///D:\Api\EDIContext\InfoDic\GetAll)

## 动作类型（ActionCode）

动作码是字符串枚举，目前其枚举值包括且仅包括：create、update、delete、get、head。动作类型字符串不区分大小写。下面一一表述它们的功用：

**Create：**用于请求创建给定信息标识和值的实体（Entity）记录。其中，信息标识用于检索要创建的实体（Entity）记录是否已经存在。

**Update：**用于请求更改给定的信息标识标识的实体（Entity）记录。

**Delete：**用于请求删除给定的信息标识标识的实体（Entity）记录。

**Get：**用于请求获取给定的信息标识标识的实体（Entity）记录。

**Head：**用于请求获取给定的信息标识标识的来自远端的建议信息标识。主要用在以多列联合信息标识换取单列Guid信息标识的场景。

通常情况下，命令接收节点无需存储类型为get或head的命令。

注意：动作类型是命令的必选字段，在本文和其它文档中有时会以具体“动作类型+命令”的形式来分类命令，称作：create类型命令、head类型命令、get型命令、update型命令、delete型命令。

## 信息标识（InfoID）

首先，标识是一个范畴很大的概念，为了说清问题我们需要把这个概念放在一个具体的领域边界中。本文档需要对业务标识、存储标识、信息标识这三个概念做鉴别性定义，这些定义所基于的领域边界是“师生基础数据”。当我们在本文和相关的其它文档中提到这三种标识时它们分别指的是：

**业务标识** 像教师的身份证件号，学生的学籍号这样的可以充当标识的字段我们称作业务标识。

**存储标识** 各业务系统持久化教师和学生数据时所使用的标识，如数据库中教师表的主键。

**信息标识** 信息标识用于数据交换，它在具体实例上可能和业务标识重叠也可能和存储标识重叠但它与这两者所基于的观察角度不同。

信息标识是符合“信息格式”章节所定义的格式的字符串。信息标识策略有两个：“单列Guid信息标识”和“多列联合信息标识”，后者的实现基于前者。

理想的情况下对于各种本体类别的实体（Entity）都能找到合理的单列业务标识作为单列信息标识，如对于教师，可以采用身份证号码作为信息标识；对于学生，可以采用学籍号作为信息标识，对于员工可以采用工号作为信息标识，对于上路的汽车可以使用车牌号作为信息标识。但现实中往往并没有一套现成的这种数据可供导入中心节点，并且很多将要与中心节点对接和有潜在对接需求的业务系统在设计时并没有考虑类似身份证号这样的业务标识，虽然如此但它们应该都会有一套自己的标识——如存储标识，虽然各系统自己的存储标识不是像身份证件号这样超越具体系统而有意义的业务标识。

本系统的数据交换协议引入了联合信息标识以应对难以找到单列信息标识的现实，与多列联合信息标识同时引入的还有单列Guid标识策略。多列联合信息标识策略以Guid标识策略为基础。

### 单列Guid标识策略

单列Guid标识策略为命令的信息标识（InfoID）字段引入了一个约定，约定Guid标识字段对应的本体元素码为“Id”，即如果节点收到信息标识为

”{‘id’:’ E872A51E-6440-4C49-B5FB-7A1D1CAE4B28’}”或

“{‘id’:’ E872A51E-6440-4C49-B5FB-7A1D1CAE4B28’,’XM’:’张三’}”

的命令则认为该命令使用的是Guid标识策略，只要信息标识中有编码为“Id”的字段则忽略其它信息标识字段（如忽略绿色部分的’XM’:’张三’）。

由中心端为每一个教师和学生分配一个唯一的Guid，各应用系统获取中心端的实体（Entity）记录把中心端的单列Guid标识与自己的本地实体（Entity）标识配对起来。很明显，把各节点的本地标识与中心节点的标识配对起来是一项绕不过的繁杂工作，必需一套合理的策略来应对这个问题，多列联合信息标识策略就是用来处理这个问题的。

### 多列联合信息标识策略

正如上面所说，单列Guid标识策略可以简单唯一的标识一条记录，但却带来了一个新的难题，就是如何和由谁来制作把各节点的本地实体（Entity）标识与中心节点的单列Guid信息标识映射起来的字典？多列联合信息标识策略就是用来解决这个问题的。简单的说多列联合信息标识策略就是“以多列联合信息标识换取单列Guid信息标识的策略”。如，客户节点向中心节点发送信息标识为”{‘SJHM’:’15261855522’,’XM’:’李白’}”动作类型为“head”的命令，中心节点根据收到的命令返回BetterInfoID字段值为” E872A51E-6440-4C49-B5FB-7A1D1CAE4B28”的建议信息标识的响应结果，客户节点收到建议的单列Guid信息标识后把该标识记录下来并用于建立自己的本地实体（Entity）标识与中心端的单列Guid信息标识映射的映射字典，这就是以多列联合信息标识换取单列Guid信息标识的过程。前面说多列联合信息标识策略是基于单列Guid标识策略的就体现在这里。多列联合信息标识换单列Guid信息标识的流程如下：

客户节点向中心节点发送信息标识形如

{‘SJHM’:’15261855522’,’XM’:’李白’}

动作类型为”Head”的命令消息

中心节点收到多列联合信息标识后根据多列信息标识指定的条件尝试到中心数据库获取两条Guid标识

如果取得两条Guid标识则返回UnUnique状态码（参见状态码表）

如果可以唯一确定一个如

”E872A51E-6440-4C49-B5FB-7A1D1CAE4B28

”的单列Guid标识则将 E872A51E-6440-4C49-B5FB-7A1D1CAE4B28

放进响应模型的BetterInfoID字段返回给请求节点

根据多列联合信息标识获取单列Guid建议信息标识的过程有时也称配对实体的过程。

理想的情况下基于多列联合信息标识策略的客户节点运行一段时间后应可以为大部分实体（Entity）建立起标识映射字典了。但现实上例外总是会存在的，对于余下的配不上对的实体（Entity）就需要人工参与处理了。

## 实体（Entity）

实体（Entity）就是交换平台所面向的业务数据，是要交换的数据，如教师、学生。实体（Entity）在各节点有自己的存储标识，这个标识我们称为节点本地的实体（Entity）标识，记为LocalEntityID。

实体是具有相同属性描述的对象(人、地点、事物)。每一个实体都是唯一的，一个实体和另一个实体是不同的，所以在各种系统中都会为实体分配一个唯一的标识。但在系统的不同层次可能会对业务上同一个实体采用不同的信息标识，如在基于面向对象模式构建的系统中程序为每一个实体对象分配唯一的指针；在关系数据库系统中会为每一个实体记录分配唯一的主键值等这都是实体标识。从中我们可以看出，实体标识是有上下文的：关系数据库的上下文是“存取”所以其主键是存取标识，内存中的指针是内存块的索引标识。那么数据交换实体标识的上下文是“数据交换”。

在Anycmd系统中，“数据交换实体”指的是教师（JS）和学生（XS）这两类本体类别的记录。而“数据交换实体标识”指的是由中心系统所分配的整个交换平台中唯一的Guid编码。因为本系统是数据交换系统，它的领域上下文是“数据交换”，所以当我们省略“数据交换”这个限定词而只说“实体”两字时它指的是“数据交换实体”。

## 命令（Command）

命令在结构上比实体文档多了“ActionCode（动作码）”和“CommandTicks（命令时间戳）”字段，除此之外“命令”和“实体文档”两者在结构上再无区别。

简单的说数据交换的过程就是节点把自己本地的操作封装为命令对象并以消息的形式传输到相邻节点，然后在相邻节点成功执行从而使本地操作的效果成功同步到远端节点的过程。所以命令是整个过程中的重要概念。这一节用形式化的语言表述命令是什么。

### 命令的领域模型（Domain Model）

#### 定义

从广义的角度说：命令属于操作，是操作实例。即，命令是对“什么”进行“怎样的”操作。如“把身份证件号码为4114251988035465的教师的性别修改为男”就是一条命令，但“修改教师”却不是命令因为它没有指定被修改的实体对象实例。为了便于理解，可以将命令与关系数据库的数据操作语句相对应，对于师生数据库数据交换平台来说可以将命令简单的理解为对关系数据库数据操作语句的对象级封装，这个说法仅仅是为帮助理解命令概念，两者并非同一概念。这种广义的描述非常抽象，下面我们换个视角从数据交换领域的角度形式化的描述一下。

从数据交换领域的角度说：命令是对业务操作的描述，是在节点间传递的数据和结构，通过一个命令实例可清楚的描述一次操作的具体内容。如果读取一条命令的内容并链接成方便阅读的话的话，则可以表述为：哪一个节点在什么时间对哪一个教师或学生进行了什么操作，若该次操作是修改操作，还可以进一步读取到具体被修改的字段以及修改后的新值。让我们用一个表格来说明命令的数据结构，如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 必选 | 类型 | 备注 |
| RequestID | 是 | String | 本地命令标识 |
| InfoID | 是 | String | 信息标识。不同节点间根据信息标识映射彼此的实体。 |
| LocalTicks | 是 | Int64 | 命令在本地生成时的时间戳。（时间戳协议见下文） |
| ActionCode | 是 | String | 动作类型 |
| NodeID | 是 | Guid | 节点标识 |
| OntologyCode | 是 | String | 本体码 |
| InfoFormat | 是 | String | 信息格式 |
| InfoValue | 否 | String | 信息值，当且仅当ActionCode为head或get或delete时可空。 |

#### 时间戳（Ticks）

假定要与数据交换平台对接的系统和潜在的将来要与中心系统对接的系统中基于.NET平台开发的最多，所以：

本协议出现的时间戳约定为.NET平台下的Utc时间戳，如DateTime.UtcNow.Ticks。请注意：其值是起始于0001-01-01 00:00:00.000的日期和时间的计时周期数，非.NET平台请注意转化。

#### 命令的原子性

命令的信息标识唯一标识一个实体，命令的执行最多影响一个实体，命令的执行结果要么成功要么失败没有中间态。举例：有一条update类型的命令，它要求把信息标识为{Id:’ 672434A4-D4ED-47D2-AFDB-C0FE02CDF14B’}的教师的性别码改为110（非法性别字典值）和把出生年月改为1984-10-01（合法的），节点接收本条命令时发现性别码是非法的则整条命令就记为失败了，不存在性别码更新失败而出生年月更新成功的状态。

#### 命令的非序列性和序列性

命令的非序列性是指：针对不同实体（Entity）的命令之间没有先后顺序；

命令的序列性指：针对同一实体（Entity）的命令之间具有先后顺序。

命令的先后顺序由命令的时间戳指定，该时间戳作为命令传输对象的一部分传输到远端节点，远端节点在整体上按照接收命令的顺序执行命令但在局部上按照命令时间戳顺序执行命令，远端节点应保证每一个客户节点发送来的命令均是按照各自的命令时间戳顺序执行的。

### 命令传输模型（Command DTO）

本协议是数据交换协议，命令不会呆在节点本地，通常会被发送到相邻的其它节点。这就需要把命令的领域模型转化为传输模型了。

对于计算机网络来说Http被划分在应用层，但对于本数据交换协议来说Http处在传输层，是命令的一种传输途径。Http是基于请求响应模式架构的，那么基于知识的重用命令有命令请求模型和命令响应模型：

#### 请求模型（Request Data Model）

模型名：CommandRequest

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **必选** | **类型及范围** | **说明** |
| **RequestID** | Ture | String | 发送节点本地命令标识。远端节点根据该标识反馈信息。 |
| **InfoID** | Ture | String | 信息标识，符合信息标识格式要求的字符串。它承载的可以是单列Guid信息标识也可以是多列联合信息标识。关于信息标识概念的定义参见其它章节。 |
| **LocalTicks** | Ture | Int64 | 本条命令在本地的生成时间戳。 |
| **ActionCode** | Ture | String | 命令动作类型。其为字符串枚举，包括且仅包括create、update、delete、head、get五个取值。否则会收到原因短语为InvalidActionCode的反馈。 |
| **InfoFormat** | True | String | 实体（Entity）值格式字符串枚举，目前取值包括且仅包括json、xml，否则收到原因短语为InvalidInfoFormat的反馈。 |
| **InfoValue** | Fasle | String | 当且仅当ActionCode为get或head或delete时InfoValue可空。  该字段承载的是信息值，信息值是扁平的键值对形式的字符串。如json信息格式：  {“infoID”:”411899”,“xm”:”李白”}  或xml信息格式：  <js><infoID>411899</infoID><xm>李白</xm></js>信息值的具体定义参见其它章节 |
| **OntologyCode** | True | String | 本体码。教师对应JS，学生对应XS |

#### 响应模型（Response Data Model）

模型名：CommandResponse

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **必选** | **类型及范围** | **说明** |
| **RequestID** | Ture | String | 发送节点本地命令标识。远端节点根据该标识反馈信息。 |
| **StateCode** | Ture | Int | 对应于命令请求模型的反馈状态码，其取值参见《命令状态码表》 |
| **ReasonPhrase** | Ture | String | 原因短语，相应状态码的可读性形式。 |
| **Description** | False | String | 描述文本。对消息反馈状态码和原因短语做进一步详细描述。如，对原因短语为NoPermission的状态做“没有如下本体元素的update权限：ZZJGM,SJHM,SFZJH”的描述。 |
| **ServerTicks** | Ture | Int64 | 服务器时间戳，标识服务器接收消息的时间。 |
| **BetterInfoID** | False | String | 建议信息ID，当原因短语为BetterInfoID时有值。 |
| **InfoValue** | Fasle | String | 当且仅当ActionCode为get时有值。其承载的是对应的get请求命令的信息标识标识的信息内容。  该字段承载的是信息值，信息值是扁平的键值对形式的字符串。如json信息格式：  {“infoID”:”411899”,“xm”:”李白”}  或xml信息格式：  <js><infoID>411899</infoID><xm>李白</xm></js>信息值的具体定义参见其它章节 |

## 并发和处理机制

并发肯定存在。当多个节点具有同一本体的同一本体元素的修改权限的时候就可能存在并发。如节点一发送一条命令请求把信息标识为”{‘id’:’ E872A51E-6440-4C49-B5FB-7A1D1CAE4B28’}”的教师的性别修改为“女”，同时节点二请求把该教师的性别修改为“男”此时就会引发并发。目前，本平台采用了乐观方式处理并发，即序列地接收命令，按照成功接收命令的先后顺序执行命令。

按照命令的接收顺序执行命令的确有些简单粗暴，但如果合理分配本体元素权限和合理分配实体（Entity）集权限的话（通过目录划分实体（Entity）集权限）并发是可以避免的。目前的这种处理并发的方式是临时的，后续会扩展本数据交换协议引入良好的并发控制策略，简单介绍如下：

我们知道命令模型的InfoValue字段是一个键值对字典，其键是本体元素码其值是对应的修改后的新值，只需要扩展出一个OldInfoValue字段，该字段跟InfoValue一样是键值对字典但该字典的值是修改前的旧值。有了新值和旧值远端就可以很好的控制并发了。

## 消息（Message）

消息是数据传输模型。命令是以消息的形式从一个节点发送到另一个节点的。一条消息中可以承载多条独立的命令。每一条消息和每一条失败的命令的发送都必须收到相应的响应状态。消息模型的数据结构如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **必选** | **类型及范围** | **说明** |
| **Commands** | Ture | CommandRequest对象列表 | 命令请求模型的数据结构见上文 |
| **Token** | Ture | TokenData | 令牌对象 |

从上表看到，消息模型主要承载一个命令请求模型列表。但除了承载命令外还承载有一个令牌对象，令牌对象有appID、ticks、tokenString三个字段，它们用于验证节点身份。令牌对象数据协议如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | 必选 | 类型 | 说明 |
| **TokenString** | Ture | string | 令牌字符串 |
| **AppID** | Ture | Guid | 应用系统标识，在此为发送节点标识 |
| **Ticks** | Ture | int64 | 时间戳 |

令牌的算法协议参见接口文档。

## 响应（Response）

响应协议是节点间的接口、合同，是每个签订合同的节点（对接的节点）必须遵守的协议规范。

远端返回类型为CommandResponse的响应结果，该类型主要封装了消息和命令的反馈状态，关于状态码协议请参考本文相关章节。CommandResponse的数据结构如下表所示：

CommandResponse数据结构：

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **必选** | **类型及范围** | **说明** |
| **StateCode** | Ture | Int | 对应于请求消息的反馈状态码，其取值参见《命令状态码表》 |
| **ReasonPhrase** | Ture | String | 原因短语，相应状态码的可读性形式。 |
| **Description** | False | String | 描述文本。对消息反馈状态码和原因短语做进一步详细描述。如，对原因短语为InvalidNodeID的状态做“未知的节点”的描述。 |
| **ServerTicks** | Ture | Int64 | 服务器返回的时间戳。表示服务器收到该消息的时间。 |
| **CommandsResponse** | Ture | CommandResponse类型对象列表 | 命令结果列表针对消息中的每一条失败的命令做反馈。CommandResponse类型见上面的传输模型章节。 |

1. 在节点间传播的每一条消息必须收到相应的响应状态；
2. 在节点间传播的每一条不成功的命令必须收到相应的响应状态码；

一条消息中可以包含多条命令，远端节点返回的状态中要包含对应于消息的状态码和对应于每一条失败命令的状态码。

注意：这里强调了远端节点必须为每一条失败的命令返回响应状态但没有说为每一条命令返回响应状态。我们知道一条消息可以承载一批命令，也就是说远端节点可以不保证为整批中成功接收的命令返回状态但必须保证为不成功接收的命令返回状态。

1. 中心节点为接收成功的每一条由多列联合信息标识标识的非create命令返回一一对应的状态和建议信息标识（BetterInfoID）。

本条说的是，如果客户节点发送的命令使用的不是单列Guid信息标识的并且可以成功换得单列Guid信息标识的话，远端节点保证会为每一条这样的命令返回一一对应的响应状态。

1. 当且仅当命令类型为get时返回模型的InfoValue字段有值。
2. 远端返回的信息所使用的信息格式必须与收到的命令所使用的信息格式保持一致。关于信息格式请参见相关章节。

通常情况下来自同一节点的命令仅具有一种格式，本协议的存在使得不要求来自同一节点的命令必须使用一种的信息格式。

总结：每一条消息和每一条失败的命令的发送都必须收到对应的状态码。另外，每一条可以换得建议信息标识的命令也必定在响应模型中对应有状态码。

一条消息中包含多条命令，所以消息的响应模型中包含一个命令响应模型列表。

## 状态码（StateCode）

状态码用来描述命令和消息的交换状态。是数据交换平台协议定义中的一个重要部分。

状态码在结构上包括三个字段，由两个必选字段和一个可选字段组成：编码（StateCode）和原因短语（ReasonPharse）是必须的，状态描述（Description）是可选的。各节点处理下列状态码的方式相同。

stateCode:200, reasonPharse:’Ok’, Description:’接收成功’

stateCode:200, reasonPharse:’Ok’, Description:’执行成功’

stateCode:200, reasonPharse:’Ok’, Description:’干的不错，恭喜你成功了’

下面是状态码表局部，用以帮助说明状态码的数据结构，要查看完整的状态码表请转到[数据交换状态码表](/StateCode/StateCodes)为准。

如果您熟悉Http协议的状态码的话，下面的状态码和Http状态码具有相同的分类规律，这个相似性不是偶然的而是有意为之的。状态码取值在[100-200}左闭右开区间表示info，[200-300}区间表示success，[400-500}区间表示fail，500表示error（服务节点内部逻辑异常）。

Info [100-200}success [200-300}error[400-500]

|  |  |  |  |
| --- | --- | --- | --- |
| **Http状态码（无意义）** | **StateCode（必选）** | **ReasonPhrase（可选）** | **Description（可空备注）** |
| 200 | 0 | None | 未定义 |
| 200 | 1 | InvalidApiVersion | 非法的版本号 |
| 200 | 2 | InvalidRequestType | 非法的请求类型 |
| … | … | … | … |
| … | … | … | … |
| … | … | … | … |
| 200 | 111 | InvalidPageIndex | 非法的分页索引 |
| 200 | 112 | InvalidPageSize | 非法的分页尺寸 |
| 200 | 200 | Ok | 成功 |
| 200 | 201 | ReceiveOk | 接收成功。 |
| 200 | 202 | ExecuteOk | 执行成功。 |
| 200 | 204 | ToAudit | 待审计 |
| 200 | 205 | AuditApproved | 审核通过 |
| 200 | 400 | Fail | 失败 |
| 200 | 401 | ReceiveFail | 接收失败。 |
| 200 | 402 | ExecuteFail | 执行失败。 |
| 200 | 409 | InvalidElementCode | 非法的本体元素码。 |
| … | … | … | … |
| … | … | … | … |
| … | … | … | … |
| 200 | 413 | InvalidActionCode | 非法的动作码。 |
| 200 | 419 | AuditUnapproved | 审核未通过。 |
| 200 | 500 | InternalServerError | 内部服务器错误。 |