**数据交换接口文档**

WebApi接口协议 v1

Anycmd

2014.10.12

# 目录

[目录 1](#_Toc400885386)

[1 前言 2](#_Toc400885387)

[2 采用技术与术语 2](#_Toc400885388)

[3 接口技术参考方案 3](#_Toc400885389)

[4 概念 3](#_Toc400885390)

[本体（Ontology） 3](#_Toc400885391)

[本体元素（Element） 3](#_Toc400885392)

[实体（Entity） 3](#_Toc400885393)

[本体动作（Action） 3](#_Toc400885394)

[字典（InfoDic） 3](#_Toc400885395)

[目录（Catalog） 4](#_Toc400885396)

[编码 4](#_Toc400885397)

[5 接口的分类 4](#_Toc400885398)

[6 接口和协议 5](#_Toc400885399)

[6.1. 时间戳协议 5](#_Toc400885400)

[6.2. 证书协议 5](#_Toc400885401)

[6.1.1. 证书数据协议 5](#_Toc400885402)

[6.1.2. 签名证书算法 6](#_Toc400885403)

[6.1.3. OAuth签名算法 8](#_Toc400885404)

[6.1.4. 令牌证书算法（不推荐） 8](#_Toc400885405)

[6.3. 字符串大小写协议 9](#_Toc400885406)

[6.4. 服务测试接口 9](#_Toc400885407)

[6.5. 单条命令接口 10](#_Toc400885408)

# 前言

该接口说明书是依据HTTP规范，针对某市师生数据交换平台的实际情况而制定的。在没有特殊要求的情况下，各对接节点必须满足本说明书的规定，严格遵循本规范中的接口数据规范，并在无特殊情况下采用本规范所提供的参考技术方案。

相关文档：[《数据交换协议指南》](http://www.anycmd.com/Content/Doc/数据交换协议指南.pdf)

# 采用技术与术语

本规范建议在无特殊情况下，接口采用WebApi技术。

下列缩略语适用于本规范：

Json 一种轻量级的数据交换格式。

Xml 可扩展标记语言 (Extensible Markup Language, XML)。

WebApi 网络应用程序接口。

# 接口技术参考方案

本接口技术方案基于Web Api技术。

**技术约定** 接口实现过程中对技术约定如下：

1. 基于HTTP 1.1协议传输；
2. 为确保内容传输的安全性，各相邻节点需要互相进行身份验证
3. 本接口过程采用UTF-8编码；

# 概念

本节解释必要的概念，因为本套接口的每一个参数都是有语义的，知道下面的概念可以帮助使用接口。一下子读不懂下面的概念没有关系，可以跳过概念直接转到接口章节，必要时再回头即可。

本体（Ontology）

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”。进一步的了解可查询维基百科或百度百科，本协议中使用“本体”这个词汇时使用的是百科上的内涵。

本体元素（Element）

“教师”二字标识了一个本体，当A告诉B“张老师去年是教语文的今年教数学了”，B说“我跟他是大学同学，他是数学系的”，A说“原来如此”。这两个沟通中的人能够互相明白对方的意思首先是因为“老师”二字界定了本体，“张老师”三字定位了“实体”。而“教语文的”“教数学的”“数学系”是张老师的“属性值”，而“属性”在此命名为“本体元素”，如教师本体有“所教学科”、“学历”、“专业”、“从教年月”等本体元素。

实体（Entity）

实体是具体本体下的一个具体事物，这个事物可能存在物理世界的真实映射也可以是完全虚构的事物。实体有一个重要属性是必须可以“标识”，也就是说必定可以区分出两个实体的不同。在师生基础数据库中每一个教师是一个教师实体，每一个学生是一个学生实体。师生基础中心库为每一个教师和学生实体分配唯一的编号，这个编号就是实体的唯一标识，各业务系统通过该标识与中心系统交换信息。

本体动作（Action）

动作用以定义可以面向具体本体做些什么。如，可以创建教师、可以修改教师的信息、可以删除教师，所以教师本体上定义有编码为“Create”、“Update”、“Delete”的动作。动作是依赖于本体的，动作语义的解释依赖于本体。如果本体是“文档”则动作编码为“Create”、“Update”、“Delete”不再合适，“Upload”、“Download”、“Compress”、“UnCompress”更合适。

字典（InfoDic）

有些本体元素在实体上的取值不是任意的。当本体是“人”时，人有“民族”这个本体元素，本体元素“民族”的“数据类型”是字典型的。“教师”本体是“人”本体的一个子类，张老师是一个“实体人”，张老师的“民族属性”取值就不是任意的而是由教育部的“民族”字典限定的。

目录（Catalog）

目录用以对具体本体的实体集进行单元划分。对于师生数据交换平台的“教师”和“学生”本体来说两者的目录巧合是一样的。师生实体集的组织单元是“区县”、“学校”、“电教馆”、“教科所”等这样的具有一定程度的稳定性的行政、企业、事业单位。目录和字典一样具有可枚举性质，整个某市大约有上千个教育性质的目录，但目录与字典有一个重要的不同：目录具有层级性质，这体现在目录的编码上。本数据交换平台的目录来源于国家学籍系统。

编码

计算机不擅长处理像“张老师”、“教语文的”、“他是数学系的”这样的信息。所以为了计算机化需要设计一种更利于计算机理解的语言。基本上各行各业都有国家级的相关编码标准。

1. 将教师本体编码为“JS”、学生本体编码为“XS”、测试本体编码为“JSTest”。
2. 本体元素编码来自教育部相关标准，本体元素类型和字典编码来自教育部标准。
3. 目录编码来自国家学籍系统。

# 接口的分类

上一节表述了一些概念，这些概念确立了一个良好的上下文。而这一节基于这个上下文做接口设计层面上的说明。本套接口基于消息面向协议设计，其与基于远程过程调用面向功能设计的接口的一个明显不同点就是入门门槛高，但进了门则会一片坦途。

**基于消息面向协议的接口设计是支持师生基础库未来发展的一项重要措施。**

基础库对外提供的接口对于操作一条实体记录来说是功能完备的。以后服务端不应打破一扇窗户对外提供任何面向功能的接口，什么时候打破这个窗户什么时候转向失败的朝向。

面向对象的行为仅有两种：Command和Query（暂不支持），不存在第三种情况。本协议将命令进一步分三种：Action（行动）、Command（命令）、Event（事件）。三者在数据传输对象的结构上不做进一步区分，三者结构完全相同。

**时间戳：**三者均具有时间戳属性，不同的是“行动”的时间戳来自当前的宇宙上下文；命令的时间戳由客户端填入，但命令的具体执行时间由服务端决定；事件的时间戳往往是过去的也由客户端填入。

**执行性质：**服务端立即执行请求类型为Action的命令，随意执行请求类型为Command的命令（“随意”指何时执行由服务端自定，如延迟到晚上执行），如何处理请求类型为Event的命令也由服务端自主决定（面向EventSourceType、EventSubjectCode和StateCode编程）。

# 接口和协议

本文档主要定义节点的Web Api方法和相关数据结构，并不强调Http Request和Response文档的完整性，文档中出现的有些是Http文档片段。关于Http协议层的东西应由程序员自己把握，但本文会给出调用样例以帮助程序员编程。

为什么使用javascript语言书写示例？

本文中的示例是使用javascript语言书写的。有这样几个原因：1示例是必要的，因为示例是采用真实的计算机语言书写的对读者（程序员）来说没有歧义；2本接口文档是独立于平台和语言的，将示例绑定到具体的平台不太合适，但不管是哪个平台的开发人员都熟悉javascript。3javascript除了作为一种无歧义的语言来使用外还有一个重要的特性就是它可以直接运行在浏览器环境下从而可以提供在线示例。

**服务地址：**http://anycmd.com/api

## 时间戳协议

调查显示将要对接的业务系统和潜在的有对接需求的业务系统中基于.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。

## 证书协议

### 证书数据协议

本文中所列服务方法的调用大都需要传入一个证书对象。证书对象有CredentialType、ClientType、ClientID、UserType、UserName、Password、Ticks七个字段，它们用于验证节点身份。其数据协议如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **字段** | **必选** | **类型** | **说明** |
| CredentialType | True | String | 证书类型。字符串枚举，取值token（不推荐）或signature（推荐）。其它类型以后支持。 |
| ClientType | True | String | 客户端类型。取值必须是node。其它类型以后支持。客户端往往是个计算机程序，如“一线通业务节点”就是基础库的一个客户端。 |
| ClientID | True | String | 客户端标识。 |
| UserType | False | String | 用户类型。用户往往是个人。当CredentialType为OAuth时有意义，OAuth暂不支持。 |
| UserName | False | string | 用户名。当CredentialType为OAuth时有意义，OAuth暂不支持。 |
| Password | Ture | String | 1，当CredentialType取值为signature时为使用“[签名算法](#_签名算法)”计算得到的签名字符串。  2，当CredentialType取值token时为使用[令牌算法](#_令牌算法)计算得到的令牌字符串。  3，待支持OAuth证书协议时再做解释。 |
| Ticks | Ture | int64 | 证书时间戳。 |

下文对于每一个接口方法的描述中都有“**是否需要身份认证**”一项，如果这项的值是“是”则表示该服务方法的调用需传入证书对象，“否”表示不需要传入证书对象。

### 签名证书算法

主要用在客户端类型为node的场景。secretKey为您的Client对接时得到的密钥。

// 使用HMACSHA1算法加密

private static string ComputeHash(string orignalString, string secretKey) {

    if (orignalString == null) {

        throw new ArgumentNullException("orignalString");

    }

    var hmacsha1 = new HMACSHA1();

    hmacsha1.Key = Encoding.ASCII.GetBytes(secretKey);

    byte[] hashBytes = hmacsha1.ComputeHash(Encoding.ASCII.GetBytes(orignalString));

    return Convert.ToBase64String(hashBytes); // 编码为Base64

}

// 签名

private static void Sign(IMessage message, string secretKey) {

    if (message == null) {

        throw new ArgumentNullException("message");

    }

    if (message.Credential == null) {

        throw new InvalidOperationException("无效的命令对象：未设置证书对象。");

    }

    CredentialData credential = message.Credential;

    // 将签名结果通过证书对象的Password字段传送

    credential.Password = ComputeHash(message.ToOrignalString(), secretKey);

}

/// <summary>

/// 将给定的命令消息转化为原始签名字符串

/// </summary>

/// <param name="message">命令消息</param>

/// <returns></returns>

public static string ToOrignalString(this IMessage message, ICredentialData credential) {

    if (message == null) {

        return string.Empty;

    }

    if (credential == null) {

        return string.Empty;

    }

    StringBuilder sb = new StringBuilder();

    // 证书

    sb.Append("CredentialType=").Append(credential.CredentialType);

    sb.Append("&SignatureMethod=").Append(credential.SignatureMethod);

    sb.Append("&ClientID=").Append(credential.ClientID);

    sb.Append("&ClientType=").Append(credential.ClientType);

    sb.Append("&UserName=").Append(credential.UserName);

    sb.Append("&UserType=").Append(credential.UserType);

    sb.Append("&Ticks=").Append(credential.Ticks);

    // 命令

    sb.Append("&Version=").Append(message.Version);

    sb.Append("&RequestID=").Append(message.RequestID);

    sb.Append("&RequestType=").Append(message.RequestType);

    sb.Append("&ActionCode=").Append(message.ActionCode);

    sb.Append("&ResultItemKey=");

    if (message.ResultItemKey != null) {

        string resultItem = string.Empty;

        int l = resultItem.Length;

        foreach (var item in message.ResultItemKey) {

            if (resultItem.Length != l) {

                resultItem += ",";

            }

            resultItem += item;

        }

    }

    sb.Append("&OntologyCode=").Append(message.OntologyCode);

    sb.Append("&EventSourceType=").Append(message.EventSourceType);

    sb.Append("&EventSubjectCode=").Append(message.EventSubjectCode);

    sb.Append("&EventStateCode=").Append(message.EventStateCode);

    sb.Append("&EventReasonPhrase=").Append(message.EventReasonPhrase);

    sb.Append("&InfoID=");

    if (message.InfoID != null) {

        foreach (var item in message.InfoID) {

            if (item != null) {

                sb.Append("&").Append("InfoID\_").Append(item.Key).Append("=").Append(item.Value);

            }

        }

    }

    sb.Append("&InfoValue=");

    if (message.InfoValue != null) {

        foreach (var item in message.InfoValue) {

            if (item != null) {

                sb.Append("&").Append("InfoValue\_").Append(item.Key).Append("=").Append(item.Value);

            }

        }

    }

    sb.Append("&LocalTicks=").Append(message.LocalTicks);

    sb.Append("&Initiator=").Append(message.Initiator);

    sb.Append("&IsDumb=").Append(message.IsDumb);

    // 忽略大小写

    return sb.ToString().ToLower();

}

### OAuth签名算法

同[签名证书算法](#_签名证书算法)。唯一不同是secretKey为客户端密钥和用户授权的访问密钥的组合，组合规则是以“&”连接：“client secretKey**&**oauth access key”。

oauth access key的获取流程见**基础库OAuth文档**。

### 令牌证书算法（不推荐）

/// <summary>

/// 计算并返回令牌字符串

/// </summary>

/// <param name="appID">公钥</param>

/// <param name="ticks"></param>

/// <param name="secretKey">私钥</param>

/// <returns></returns>

public static string Token(string appID, Int64 ticks, string secretKey) {

    var s = (string.Format("{0}{1}{2}", appID, ticks.ToString(), secretKey)).ToLower();// 转化为小写

    MD5CryptoServiceProvider crypto = new MD5CryptoServiceProvider();

    byte[] bytes = crypto.ComputeHash(Encoding.UTF8.GetBytes(s));

    StringBuilder sb = new StringBuilder();

    foreach (byte num in bytes) {

        sb.AppendFormat("{0:x2}", num);

    }

    return sb.ToString();

}

## 字符串大小写协议

如不明确说明，本文档中的所有服务所接收的字符串类型的参数和字符串类型的返回值均是不区分大小写的。如，“单条命令接口”所接收的参数列表中的名为“OntologyCode”的参数取值为“JS”和“js”必须等效。

## 服务测试接口

各节点都需要提供。相关文档[《数据交换协议状态码表》](http://www.anycmd.com/StateCode/StateCodes)

**服务名：**/api/IsAlive

**服务类型：**WebApi

**HTTP请求方式：**Any

**支持格式：**JSON、XML

**是否需要身份认证：**否

**请求参数：**Path表示可以从Url上传值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Parameter** | **Data Type** | **Required** | **Description** |
| Version | path | string | Yes | 版本标识。取值：v1 |

**注意事项：**无

**调用样例：**本调用样例使用javascript作为描述语言，基于jQuery类库实现。

$.ajax({

url: '/api/AnyIsAlive',

data: JSON.stringify({

version: 'v1'

}),

type: "GET",// Any Http Method

contentType: "application/json; charset=utf8",

success: function (data) {

showResult(JSON.stringify(data));

}

});

**返回结果：**

|  |  |  |
| --- | --- | --- |
| **字段名** | **类型** | **备注** |
| IsAlive | Boolean | 是否可用。True标识可用，False不可用。 |
| StateCode | Int32 | 数据交换协议状态码。 |
| ReasonPhrase | String | 数据交换协议原因短语。 |
| Description | String | 数据交换协议状态码描述。 |
| ServerID | String | 服务器标识。 |
| ServerTicks | Int64 | 服务器时间戳。 |

**JSON示例**

**JSON请求文档：**

GET http://localhost:20140/api/AnyIsAlive?version=v1 HTTP/1.1

Host: localhost:20140

User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0

Accept: application/json, text/javascript, \*/\*; q=0.01

Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3

Accept-Encoding: gzip, deflate

X-Requested-With: XMLHttpRequest

Referer: http://localhost:20140/

Connection: keep-alive

**JSON响应文档：**

HTTP/1.1 200 OK

Cache-Control: private

Content-Type: application/json; charset=utf-8

Server: Microsoft-IIS/8.0

X-Powered-By: ServiceStack/3.969 Win32NT/.NET

X-AspNet-Version: 4.0.30319

X-Powered-By: ASP.NET

Date: Wed, 30 Oct 2013 06:25:52 GMT

Content-Length: 151

{"IsAlive":true,"StateCode":200,"ReasonPhrase":"Ok","Description":"服务可用","ServerTicks":635187111527557402,"ServerID":"Windows8"}

## 单条命令接口

由服务节点提供给客户节点。

Command：一个封装了用于处理客户端请求的逻辑对象。这些对象可以立即执行，为推迟处理而进入队列、保存和记录日志。

Command Message：标识客户端需要调用的逻辑操作的消息。命令消息还为这些操作提供输入数据。

相关文档[《数据交换协议状态码表》](http://www.anycmd.com/StateCode/StateCodes)[《数据交换协议指南》](http://www.anycmd.com/Content/Doc/%E6%95%B0%E6%8D%AE%E4%BA%A4%E6%8D%A2%E5%8D%8F%E8%AE%AE%E6%8C%87%E5%8D%97.pdf)

**服务名：**/api/AnyMessage

**服务类型：**WebApi

**HTTP请求方式：**Any

**支持格式：**JSON、XML

**是否需要身份认证：**是

**请求参数：**Path表示可以从Url上传值。支持测试性调用、探索性调用，错误请求的响应中带有引导性信息，支持在试验中掌握该接口的用法。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Parameter** | **Data Type** | **Required** | **Description** |
| Version | path | string | Yes | 版本标识。取值：v1 |
| IsDumb | path | Boolean | No | 是否是哑的。默认为False。True表示是哑的，False表示不是哑的。哑命令不会影响服务端实体的状态， 但哑命令与对应的非哑命令一样经过相同的流程路径。客户节点程序员可以通过发送哑命令的方式测试对应的非哑命令。 |
| Credential | path | CredentialData | Yes | 证书。证书对象在结构上有CredentialType、ClientType、UserName、Password、Ticks五个属性，它们用于验证节点身份。 |
| RequestType | path | string | Yes | 请求类型。必须是Action或Command或Event。Action接收立即执行，Command何时执行由服务端自定但必须执行， 如何处理Event由服务端自定，面向EventSourceType、EventSubjectCode、EventStateCode编程。 |
| OntologyCode | path | string | Yes | 本体码。教师对应JS，学生对应XS，测试对应JSTest。 |
| ActionCode | path | string | Yes | 动作码。教师、学生两个本体的动作码巧合相同。目前均是Create、Update、Delete、Get、Head五个取值。 |
| EventSourceType | path | string | No | 事件源类型。必须是Command或Entity。Command事件源类型的事件用以告诉远端节点它发送过来的命令在我端的处理状态。 Entity事件源类型的事件用以告诉远端节点我端的实体发生了某个事件，我端的实体在远端有对应的实体。 |
| EventSubjectCode | path | string | No | 事件主题码。主题码为点号分割的层级结构。 编码为“StateCodeChanged”的主题基本包括命令的所有事件，而“StateCodeChanged.Audit”编码的是审核事件。 |
| RequestID | path | string | Yes | 请求标识。当RequestType为Event且EventSourceType为Command时是远端节点向我分发命令时使用的RequestID。 |
| LocalTicks | path | string | Yes | 本地时间戳。对于Event该时间是事件在客户端发生的时间而对于Action和Command该时间戳的意义由客户端自由定义。 |
| EventStateCode | path | int | No | 状态码。状态码参见《数据交换协议状态码表》。 |
| EventReasonPhrase | path | string | No | 原因短语。状态码参见《数据交换协议状态码表》。 |
| Initiator | path | string | No | 命令发起人。Action、Command、Event发起人，如果客户端本地未记录发起人则使用该字段委托中心端代为记录。 注意：中心端记录下了它收到的每一条影响实体状态的命令，但暂未开放给应用节点查询。 |
| ResultItemKey | path | string[] | No | 本体元素码数组，指示当前命令的ActionInfoResult响应项。对于get型命令来说null或空数组表示返回所有当前client有权get的字段， 对于非get型命令来说null或空数组表示不返回ActionInfoResult值。 |
| InfoID | path | KeyValue[] | Yes | 信息标识。Key、Value键值对数组，键为本体元素码。 |
| InfoValue | path | KeyValue[] | No | 信息值。Key、Value键值对数组，键为本体元素码。 |

**响应结果：**

和请求参数结构上完全一样。

**注意事项：**虽然本服务支持所有Http Method但建议首选Post请求方式。

**调用样例：**

$.ajax({

url: '/api/AnyMessage',

data: JSON.stringify({

version: 'v1',

requestType: 'command',

actionCode: 'update',

ontologyCode: "JSTest",

requestID: '@Guid.NewGuid()',

credential: { userName: '@userName', password: '@tokenString', ticks: '@ticks' },

localTicks: '@DateTime.UtcNow.Ticks',

infoID: [{ key: 'id', value: '0008E9A4-CC11-48FB-9B1C-C72D4795AEDF' }],

infoValue: [{ key: "XM", value: "张三" }]

}),

type: "POST",// Any Http Method

contentType: "application/json; charset=utf8",

success: function (data) {

showResult(JSON.stringify(data));

}

});

Http请求文档

POST http://localhost:20140/api/Command HTTP/1.1

Host: localhost:20140

User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0

Accept: \*/\*

Accept-Language: zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3

Accept-Encoding: gzip, deflate

Content-Type: application/json; charset=UTF-8

X-Requested-With: XMLHttpRequest

Referer: http://localhost:20140/

Content-Length: 458

Connection: keep-alive

Pragma: no-cache

Cache-Control: no-cache

{"version":"v1","requestType":"command","requestID":"3e91b9fa-e13f-41df-a07c-bbd92daf245b","credential":{"userName":"b2628a53-d014-41cd-8725-9014bb917fe8","password":"667ab186706b620048eb0e3c9c901f90","ticks":"635187161551849010"},"infoID":[{"key":"id","value":"0008E9A4-CC11-48FB-9B1C-C72D4795AEDF"}],"localTicks":"635187161551849010","actionCode":"update","actionInfoResult":[{"key":"XM","value":"张三"}],"ontologyCode":"JSTest"}

Http响应文档