MBAPI模型——向统一标准化的努力

**摘要**

本文描述了一个api接口的统一模型。该技术正在被众多大厂证实可用于大规模开放api系统的建造。但是迄今api还没有一个跨厂商跨平台的标准概念模型，对其使用和相关含义的确定造成很多不必要的混乱。MBAPI（Message Base API）模型就是希望整理现今各大厂、各平台、各领域的api模型，以及研究原型中的思想，找出共同点，从而寻求解决目前这种混乱的情况。它致力于成为api标准未来发展的基础。MBAPI模型是一个内涵丰富且开放的技术，即将吸引着众多用户，研究人员和产品厂商对它投入大量精力。MBAPI模型专注于api已成共识的方面，从功能上对其进行若干级别的划分：TODO。每一级别是不断累积的，每一级别是在上一级别的基础上增加新的要求而成。本文中还将列出MBAPI中暂不包含，但十分重要的其它api特性。

# 介绍

本文计划制订一个api的标准参考模型。api（应用程序编程接口）是一个有着悠久历史的技术。虽然近几年来云计算的出现，引起了云api这个热点话题，但api的基本概念已经作为设计应用程序编程接口的方法而使用了数十年。api标准的缺失使得出现了形形式式的实现，阻碍了api的先进性的发挥。api这个词对于很多刚开始设计api的设计师来说可能尚停留在“知其名”的阶段，从而被设计和实现出的api往往经不住推敲。面向api编程的应用开发人员每天对着各种不同风格的api工作是难以心情愉悦。本文试图改变这种状况，定义一个拿来即可使用级别的api最佳实践。即使当前web上的api尚未统一，掌握了这套概念框架，也可以使作为面向api编程的客户端程序员的您站在一个足够的高度统一的看待它们。

# 概述

简单的说，应用程序编程接口的设计就是给出一套约束，使Client和Server可以使用这套约束表达和理解意愿。 “表达和理解意愿”更进一步可以更简短的表达为“传递信息【信息】”。Client和Server使用基于预先定义的语法书写的表述【Message】进行沟通或交互。

# 准备

## Message

Message的分类

### 命令

### 行动

### 事件

## 本体

### 本体元素

### 本体动作

## 资源

### 实体

## 信息字典

## 组织结构

## 信息编码

# api的型

接口不能随便提供，一开始就要关注“治理”。

我们如何表达意愿的呢？有以下要素：

* + 首先， 本体是什么 Ontology
  + 其次， 要干什么 Action
  + 三， 定位目标实体
  + 四， 提供输入

## Command模式

来看看四人帮的Command模式是怎么说的：

**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。**

### **看看sql**

### 看看nosql

### 看看http

### 看看我们以前写的方法

如果提供不出案例那就看看LogOn(string loginName,string passworkd)吧

Account

Login

## Query模式

有没有需求？

oData

别急，等待

先走其它途径，暂时不要提供接口。

## **总结**

行为的分类：Command、Query

Command再分类：Action、Command、Event

执行性质

InfoID、InfoValue

RPC、Message Base

规定如何完整的表述一件事情并为“做”提供完整的输入，只是完成了“形”的定义。只是走完了形式化这一步，但还有一步。

**第二步是填充那个“形”，是“交换”、是“映射”、是“信息”。**

# api的瓤

## 信息

## 其它

ACL

Rbac

1Action;2ElementAction;3ClientAction;4ClientElementAction;5OrganizationAction;6EntityAction,7EntityElementAction;8EntityClientAction;9EntityClientElementAction;10引入时间维；11引入访问计数。

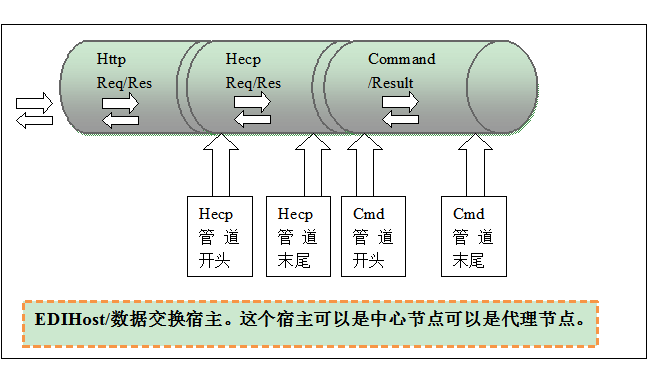
1. Message
2. Command
3. Action
4. Event
5. Message的其它问题

# 交互

## 消息

Restful

Request/Response



认证

使用Http作为应用协议吗

Http和WebService有区别吗

Socket

# 参考架构

## Hecp

转到vs解决方案

原则、模式、实践

## 传输层

## 本体机

本体服务器

其它

CQRS说的是Command和Query分离。它把Action分类为两种：Command和Query。并且明确的说“不存在第三种”。

其实博客园的博主也是分作两类的：一类是男博主，一种是女博主。不存在第三种博主。将博客园的博主按照男女分类为两种的分类法比将Action分类为Command和Query好理解多了吧。

上面的分类也可以表述为：Action分两种，一种是Command，一种是“非”Command；博客园的博主分两种，一种是男博主，一种是“非”男博主。

什么是Command？什么是Query？不必追求100%正确，借助大家熟知的“方法/Method来”来形式化Command和Query的话。那么，所有我们书写过的“无返回值”的方法都是Command，所有我们书写过的“有返回值”的方法都是Query。“有无返回值”跟“是非男女”比较像了吧。比如这里书写一个void SignIn(string loginName, string password)，这个方法就属于Command，而AccountInfo GetAccount(string loginName)则属于Query。那么要是有人书写了AccountInfo Login(string loginNamem, string password)方法怎么办？可以将它看作是Command和Query的混合，先Command再Query。

在实现CQRS的框架中我们是如何模型化Command的呢？是这样的，比如对于上面的void SignIn(string loginName, string password)命令方法可以模型化为下面的样子：

public class SignInAccountCommand : Command{

public SignInAccountCommand(string loginName, string password){

this.LoginName = loginName;

this.Password = password;

}

public string LoginName{get; private set;}

public string Password{get; private set;}

}

SignInAccountCommand跟SignIn方法完全一样，都是完整的描述一件事情，并为‘做’提供完整的输入。loginName和password即是输入。SignInAccountCommand成功对象化了SignIn方法并且丝毫没有损失信息。值得说明的是：Command中的输入和Event中的输出都应是只读的不可变的”。

下面我们看看“四人帮”的Command模式是怎么说的：

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。

Command完整的描述了一件事情并为做提供了完整的输入。当我们使用命令式编程语言比如C#写程序的时候就是在书写一个个的命令。比如我们的rdb中有个Function表，有个引用了Function表的ManagedFunction表，还有个引用了ManagedFunction表的RoleFunction表。当我们在数据访问层或者应用服务层使用C#书写删除Function实体的逻辑的时候是怎么做的呢？可以使用如下三个命令来表达：DeleteRoleFunctionCommand，DeleteManagedFunctionCommand和DeleteFunctionCommand。将这三个命令按照顺序执行就可以了。

public class SignInAccountCommand : Command{

public SignInAccountCommand(string loginName, string password){

this.LoginName = loginName;

this.Password = password;

}

public string LoginName{get; private set;}

public string Password{get; private set;}

}

跟

void SignIn(string loginName, string password)是等效的。但是为每一个命令方法都创建一个command class看起来不太像是最佳实践。应该有更好的等效的表达命令的方式。可不可以这样？引入几个简单的概念，然后基于对这些简单概念的理解给出一个“形”。使用这个形应可以表达任何命令。现在我们可以将SignIn方法表述成这样：

ontology=”account”

verb=”signIn”

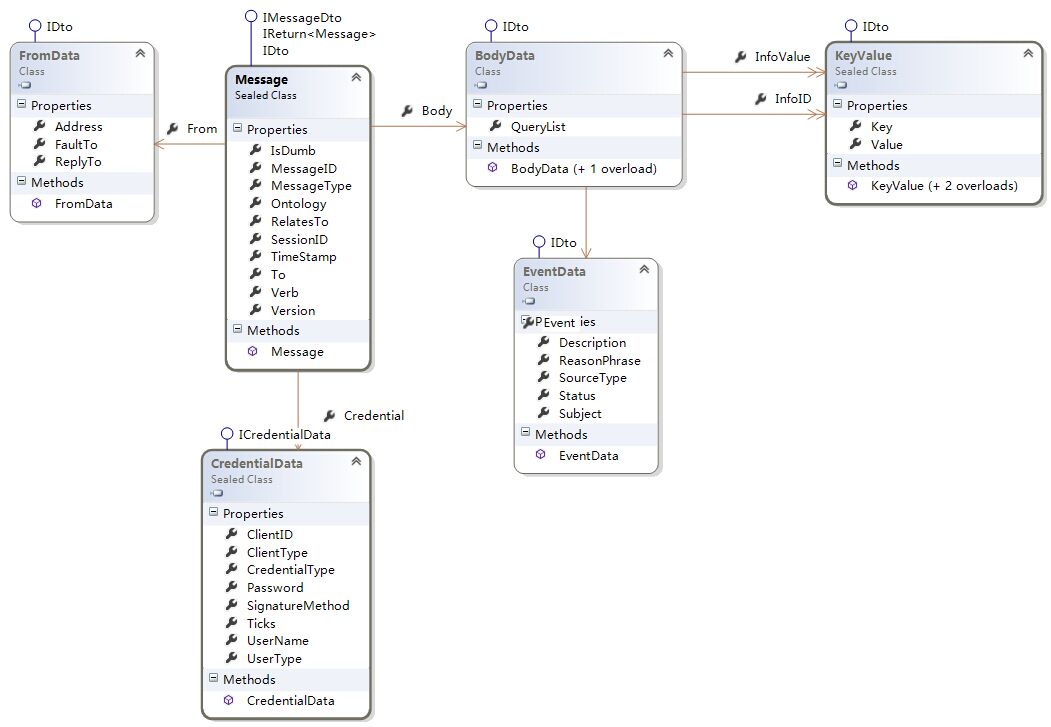
infoID=[{‘key’:’loginName’,’value’:’anycmd’}]

infoValue=[{‘key’:’loginName’,’value’:’anycmd’},{‘key’:’password’,’value’:’123456’}]

看来是可以的，因为通过以上四行我们同样完整的描述了SignIn命令并为执行提供了完整的输入。上面这种形式也并不太难理解，对比一下http的请求文档两者没有本质的区别。Ontology + infoID = uri，verb = httpMethod，infoValue = httpContent。

既然使用http也能描述命令为什么还要提出新的形式呢？[这个](http://www.cnblogs.com/xuefly/p/3428505.html)不成熟的随笔上有回答。

Ontology、verb、infoID、infoValue是Command的要素，除此之外还有一些辅助元素，如下图所示：



采用上图的“形”填入承载“信息”的数据是可以描述宇宙万事万物的。

Message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）。

服务器只有一个收发这三种Message的服务就可以了：

