

北京银行ESB技术规范培训



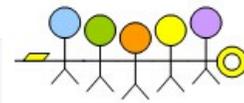
内容提纲

- 概述
- ESB整体架构
- 报文数据定义
- 服务消费者的接入
- 服务消费者的接出
- 服务组合原则



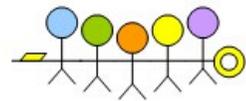
概述

- 对ESB的接口进行约定，以协调各方的开发团队。
- 对相关技术和编程方法进行介绍，以便于各团队的应用开发。
- 内容包括：整体架构、报文格式、服务消费者的接入和服务提供者的接出。



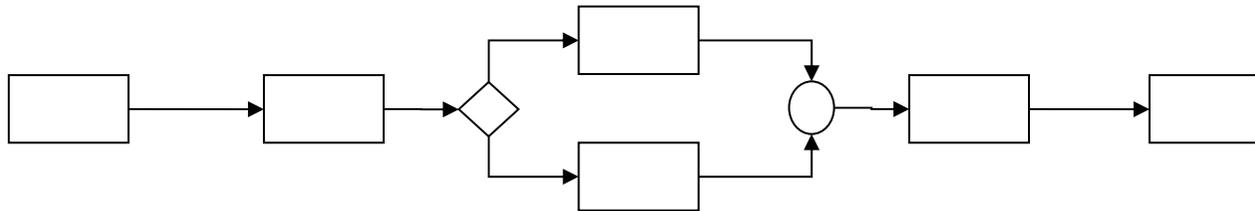
整体架构

- SOA架构包括服务消费者、ESB、服务提供者三个部分。
- **服务消费者** 是对于ESB服务的请求发起方，包括网银，电话银行，一户通，ATM等渠道。
- **服务提供者** 对服务进行真实业务处理的系统，包括AS400核心，基金、三方存管等。
- **ESB** 在中间提供服务路由和调度，所有对服务的访问都需要通过ESB进行。

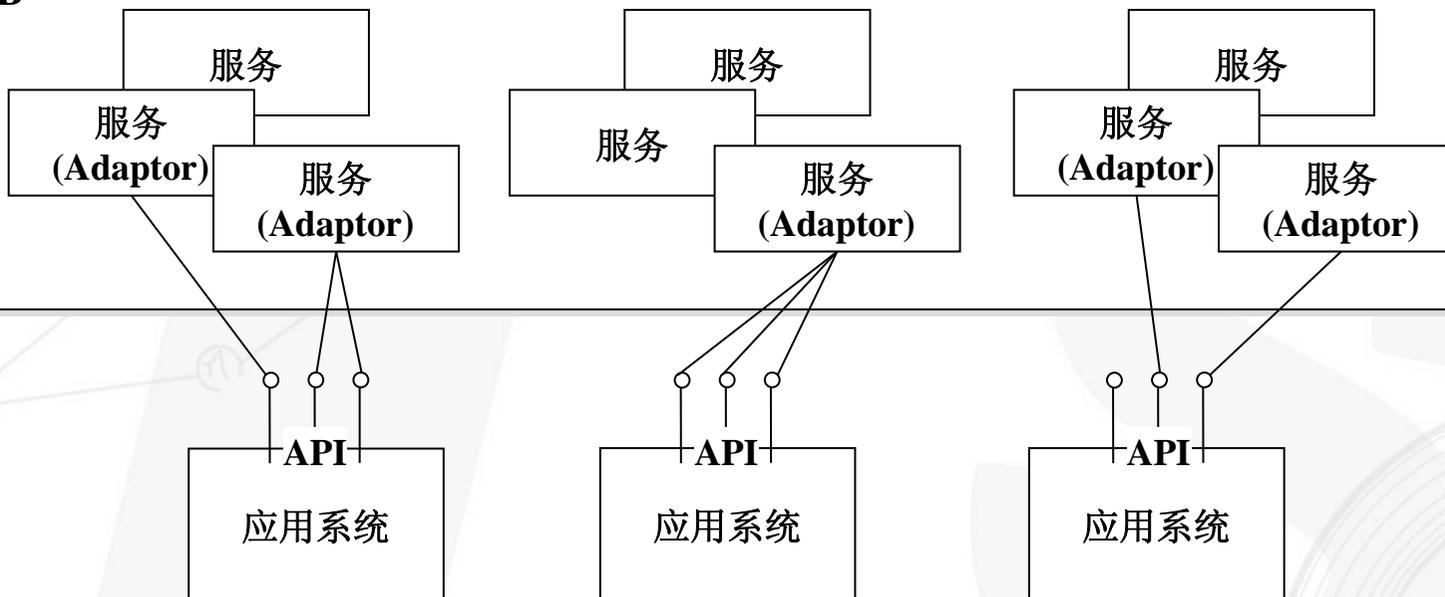


整体架构

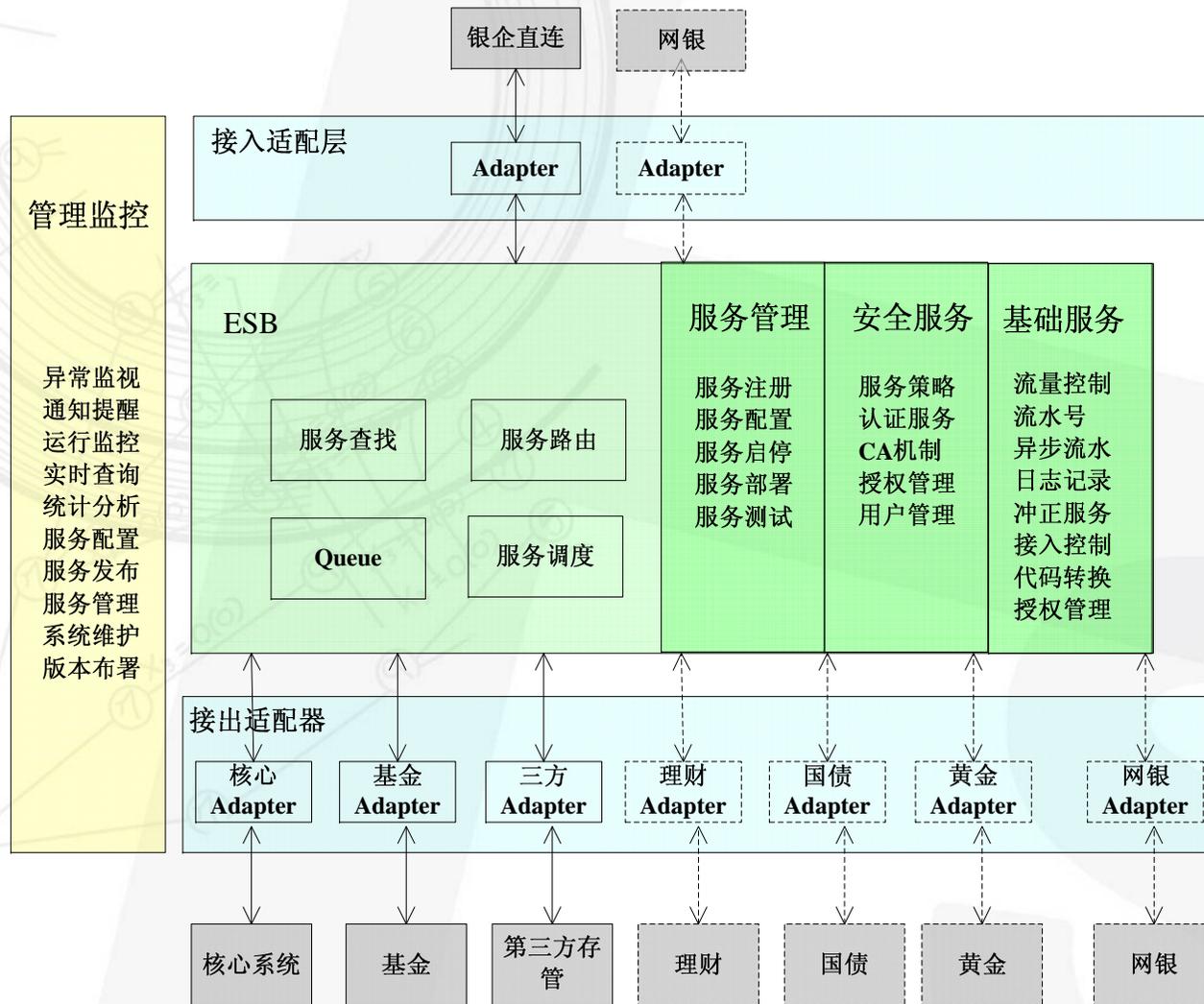
业务流程



ESB



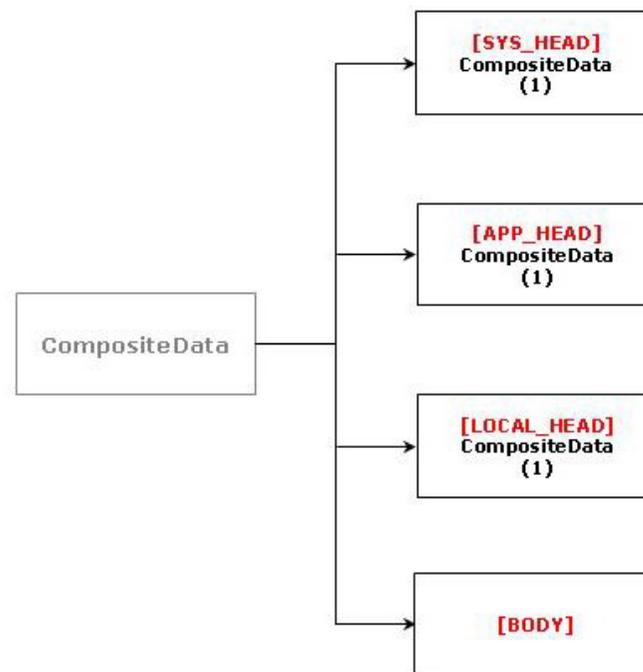
北京银行的集成方案架构

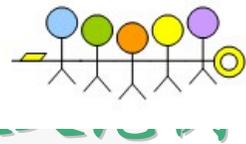




报文数据结构定义

- 系统头：包含交易的一些共有属性。
- 应用报文头：当交易需要做多页查询时，将使用应用报文头来描述查询的公共数据，是否需要多页查询由具体服务接口约定。
- 本地头：当交易发生本地授权时，使用该结构上送本地授权信息。
- 报文体：存放交易相关数据。





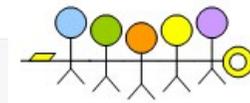
```
<?xml version="1.0" encoding="UTF-8"?>
<service>
  <body>
    <data name="SYS_HEAD">
      <struct> <data name="TRAN_CODE"><field scale="0" type="string"
        length="10">0161</field></data>
    </struct></data>
    <data name="BODY">
      <struct>
        <data name="CARDNO"><field scale="0" type="string" length="10">6221560100093072</field></data>
        <data name="OPER_CODE"><field scale="0" type="string" length="10">00426</field></data>
        <data name="CURR"><field scale="0" type="string" length="10">242</field></data>
        <data name="BANK_ACCT_NO"><field scale="0" type="string"
          length="10">6029070000001690</field></data>
        <data name="BANK_NO"><field scale="0" type="string" length="10">001</field></data>
        <data name="START_DATE"><field scale="0" type="string" length="10">20080701</field></data>
        <data name="TRAN_CODE"><field scale="0" type="string" length="32">2637</field></data>
        <data name="ACCT1"><field scale="0" type="string" length="10">0016600246589</field></data>
        <data name="END_DATE"><field scale="0" type="string" length="10">20080806</field></data>
      </struct>
    </data>
    <data name="APP_HEAD">
      <struct><data name="PGUP_OR_PGDN"><field scale="0" type="string"
        length="1">1</field></data></struct>
    </data>
    <data name="LOCAL_HEAD"><struct /></data>
  </body>
</service>
```



报文头的说明



北京银行报文头规范.doc



com.dc.eai.data.CompositeData中的接口

public CompositeData(): 构造函数

public void addField(String name, Field field): 添加域

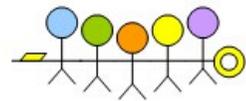
public void addArray(String name, Array array) : 添加数组

public void addStruct(String name, CompositeData struct) : 添加结构体

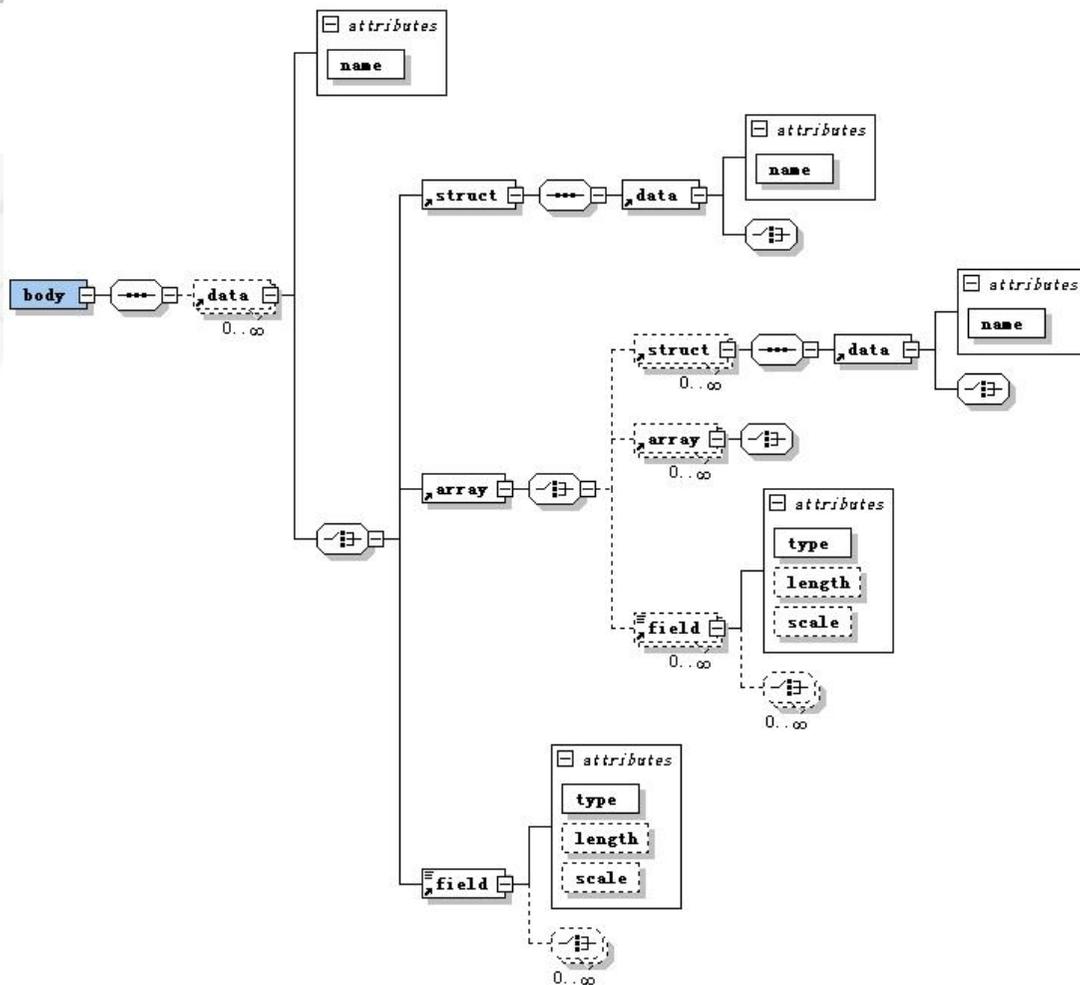
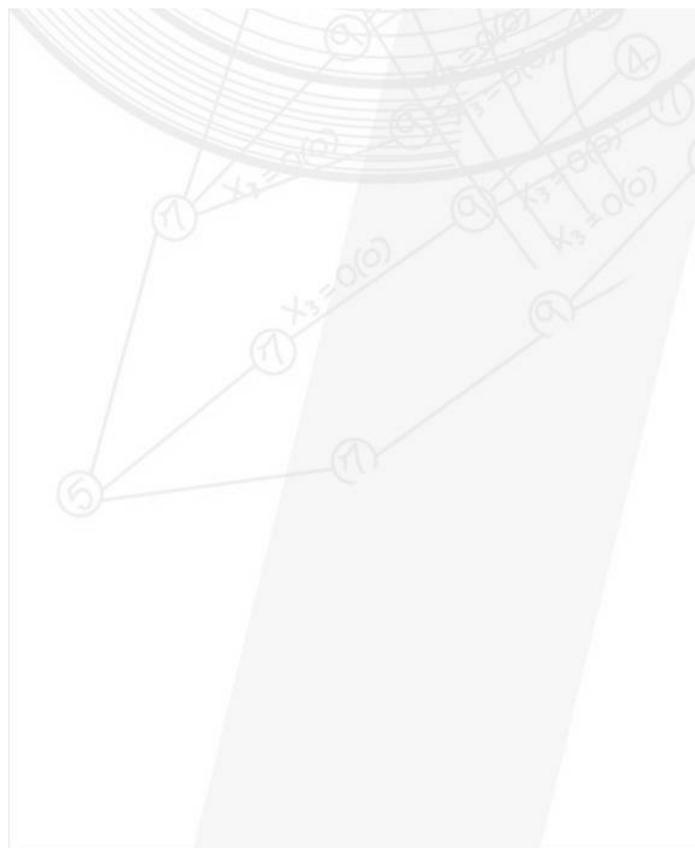
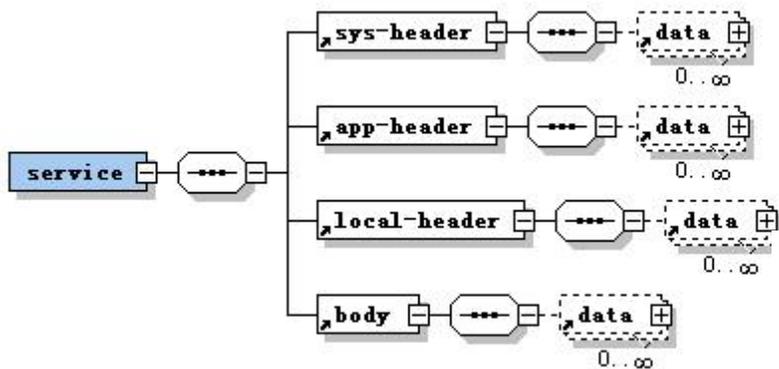
public Field getField(String name) : 获取域

public Array getArray(String name) : 获取数组

public CompositeData getStruct(String name) : 获取结构体



XML报文结构





XML与CompositeData之间的转换

com.dc.eai.conv.packconv.StandardCBSIPackageConverter

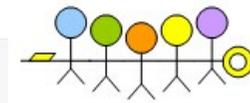
- 标准报文打包拆包 XML格式报文与CompositeData之间相互转换

Constructor Summary

[StandardCBSIPackageConverter](#) ()

Method Summary

void	pack (OutputPacket packet, CompositeData data, IOConfig config) 组包处理
void	unpack (InputPacket packet, CompositeData data, IOConfig config) 拆包处理



● com.dc.eai.conv.Packet

Constructor Summary

[Packet](#) ()

Method Summary

void	advance (int size) : 增加当前的偏移值, 不检查参数的合法性, 由调用者检查
byte[]	getBuff () : 获取缓冲区数据
int	getLength () : 获取总的的数据长度
int	getOffset () : 获取当前处理的偏移值
void	setBuff (byte[] buff)
void	setOffset (int offset) : 设置当前的偏移值, 不检查参数的合法性, 由调用者检查



● com.dc.eai.conv.InputPacket

Constructor Summary

[InputPacket](#)(byte[] buff)

Method Summary

boolean	exceed (int size)
---------	-----------------------------------

Methods inherited from class com.dc.eai.conv.[Packet](#)
[advance](#), [getBuff](#), [getLength](#), [getOffset](#), [setBuff](#),
[setOffset](#)



● com.dc.eai.conv.OutputPacket

Constructor Summary

[OutputPacket](#) ()

[OutputPacket](#) (int init_size)

[OutputPacket](#) (int init_size, int inc_size)

构造函数 **init_size** 用于最初创建缓冲区的大小 **inc_size** 用于当缓冲区不够时每次至少增加的大小

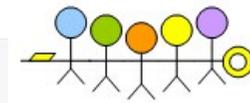
Method Summary

void [ensure](#) (int size)

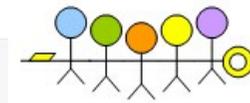
确保在 **offset** 后至少能够容纳 **size** 字节的大小

Methods inherited from class com.dc.eai.conv.[Packet](#)

[advance](#), [getBuff](#), [getLength](#), [getOffset](#), [setBuff](#), [setOffset](#)



```
CompositeData req_data = new CompositeData();
CompositeData sys_head = new CompositeData();
//设置相关的域...
CompositeData app_head = new CompositeData();
//设置相关的域...
CompositeData local_head = new CompositeData();
//设置相关的域...
CompositeData body = new CompositeData();
//设置相关的域
Adapter adapter = new Adapter();
req_data.addStruct(Adapter.SYS_HEAD, sys_head);
req_data.addStruct(Adapter.APP_HEAD, app_head);
req_data.addStruct(Adapter.LOCAL_HEAD, local_head);
req_data.addStruct(Adapter.BODY, body);
//设置IP、端口和超时时间
//进行交易请求，返回响应结果
CompositeData rsp_data = adapter.docomm(req_data);
//取响应系统头,获取相关获。。。
CompostieData sp=rsp_data.getSruct(Adapter.SYS_HEAD);
//取响应应用头,获取相关获。。。
CompostieData ap=rsp_data.getSruct(Adapter.APP_HEAD);
//取响应本地头,获取相关获。。。
CompostieData lp=rsp_data.getSruct(Adapter.LOCAL_HEAD);
//取响应报文体,获取相关获。。。
CompostieData bd=rsp_data.getSruct(Adapter.BODY);
//可在rsp_data中查找对应的响应信息
```



XML到CompositeData的转换

```
byte[] requestData = .....
```

```
StandardCBSIPackageConverter spc = new StandardCBSIPackageConverter();
```

```
CompositeData cd = new CompositeData();
```

```
InputPacket ip = new InputPacket(requestData);
```

```
spc.unpack(ip, cd, null);
```

ESB服务消费者接入规范

服务消费者接入

- 服务消费者通过向**ESB**发送**BOBSD**报文来调用服务，完成业务处理。
- 已有的、不可以改动的渠道接入**ESB**，需要开发**Adapter**，进行报文转换，将原有的报文格式转换为标准的**BOBSD**报文。

系统接入原则

- 新建渠道系统接入ESB系统，需要遵照BOBSD的规范接入。
- 已有渠道调用ESB发布新的服务，通过ESB连接后端服务系统，需要遵照BOBSD的规范接入。
- 新建业务服务系统通过ESB对外发布服务，遵照BOBSD规范。
- 已有服务系统调用ESB发布的服务，根据系统调整范围的大小、工作量和分析结论作相应处理：
 - 改动工作量较小，改动较容易系统遵照BOBSD规范接入；
 - 改动工作量较大，对系统的影响很大则按原有服务系统的报文规范。

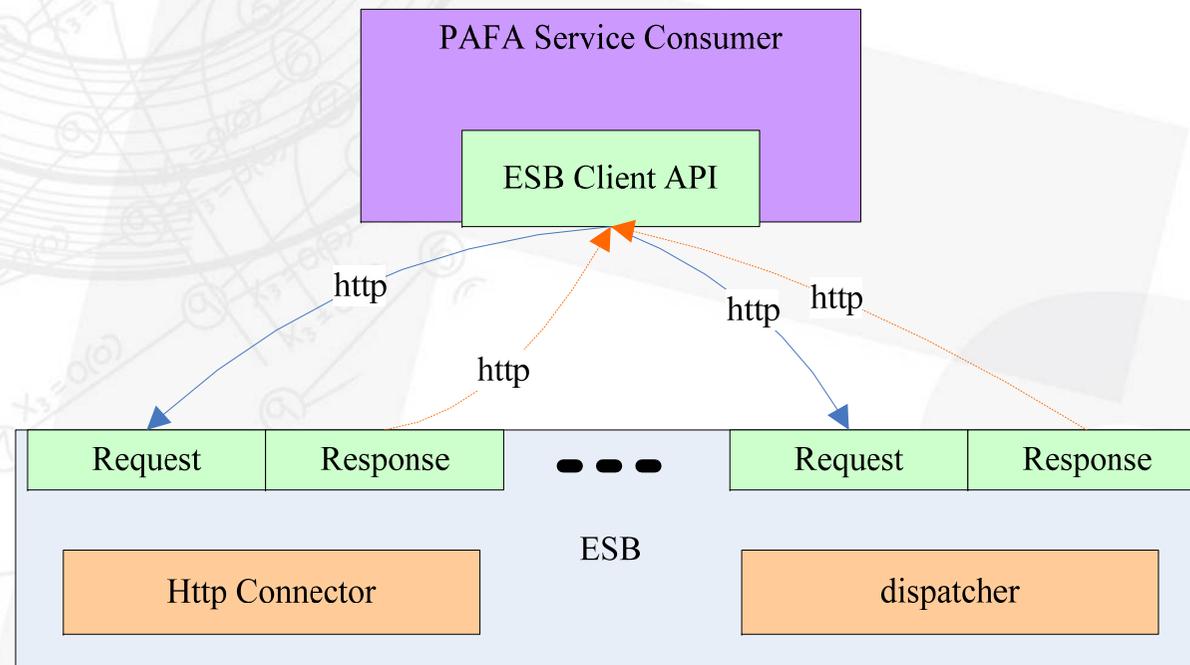


新开发的服务消费者接入规范

- **连接方式：**服务消费者通过HTTP协议向ESB间发送服务请求。
- **发送报文和响应报文均为为xml格式的报文。**
- **ESB提供服务请求客户端API，以屏蔽通讯协议。**

JAVA平台接入规范

- PAFA系统通过调用ESB Client API实现与ESB间通讯处理，ESB提供请求API屏蔽通讯协议和报文格式的技术细节，提高开发效率。

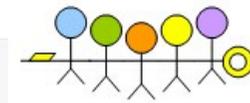




服务消费者客户端API

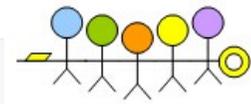
com.dcfes.esb.client.ESBClient接口:

- **public CompositeData request(CompositeData request) throws TimeOutException;**
以CompositeData格式发送请求
- **public byte[] request(byte[] request) throws TimeOutException;**
以xml格式发送请求

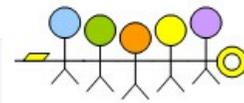


服务调用流程

1. 构建请求组合数据CompositeData。
 - ① 实例化请求组合数据CompositeData
 - ② 构建系统头，应用头，本地头组合数据（CompositeData），根据规范分别对系统头，应用头，本地头进行赋值。
 - ③ 创建报文体组合数据，根据输入报文接口进行赋值，添加到请求组合数据中。
2. 调用request(CompositeData req)发送交易请求，得到响应报文。
3. 调用CompositeData的相应方法读取响应信息，进行业务处理。



服务提供者的接出



服务提供者的接出

- 服务提供者必须提供API以便ESB调用，以完成业务处理；
- ESB在调用服务提供者的API时，服务提供者应该完成BOBSD约定的业务处理。
- 或者按照BOBSD，对于服务规范所定义的标准报文进行响应。



ESB中的一些约定

- 提供系统级别和应用级别超时处理机制。
- 新建服务系统字符编码统一采用 UTF-8。

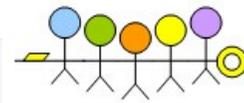


服务组合规范



组合服务的定义

- 组合服务：在服务总线上基于简单原子服务，定义出来功能更丰富的另外一个服务出来，组合服务拥有与简单原子服务不同的服务代码和服务接口。
- 组合形式：组合形式包括原子服务的串行、并行和基于简单规则的组合。
- 组合服务的实现：可以在服务总线上基于组合服务功能进行组合。



组合服务的组合原则

- ESB作为整个银行的信息高速公路，为了保障高效运转，**尽可能不要在ESB上进行服务组合。**
- 考虑到服务超时、运行效率，被组合的服务不宜过多，一般以3-4个为宜。
- 组合服务的业务规则要尽可能简单。
- 组合服务的操作数据为接口输入数据、原子服务的返回数据，组合服务不会从任何数据库中提取数据，也不会将数据存储在持久化存储中。
- 在组合服务实现时，必须有完善的事务保障机制，并在服务组合时，考虑清楚事务如何保障。



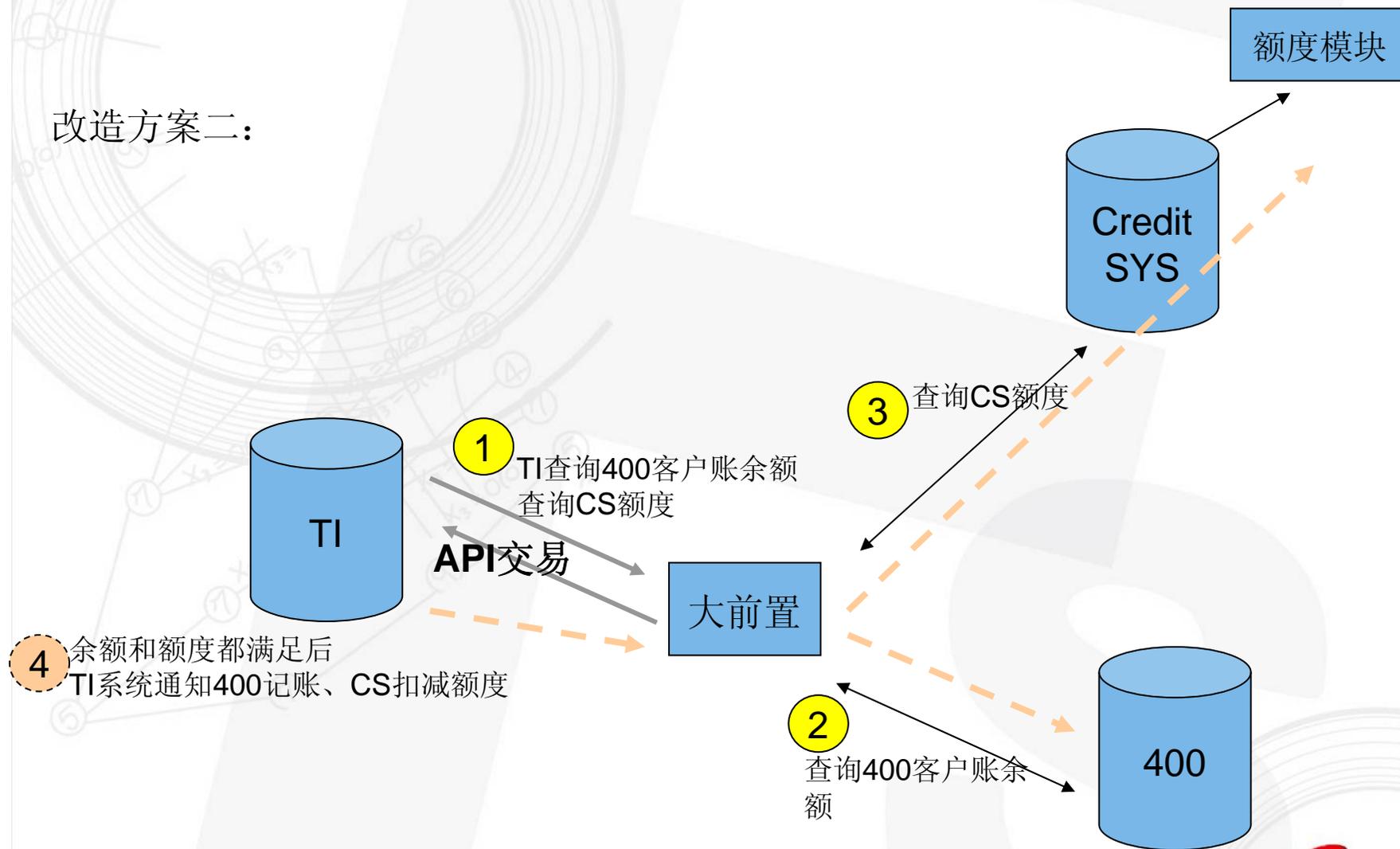
组合服务实现时需要考虑的问题

- 服务超时：
 - 组合服务实现需要仔细考虑服务的超时间设置，前后台系统均要对该超时时间认可。
 - 同时组合服务需要处理前端调用者超时，还在运行的服务如何清理的问题。
- 事务处理：
 - 事务处理的保障以冲正服务为基础实现
 - 不要在组合服务中与后台服务系统的事务处理紧密耦合

组合服务案例分析

第一版：信贷系统三期---TI系统相关业务额度实时控制

改造方案二：



说明:TI系统向大前置发送查询客户账余额和额度的请求,大前置向400查询余额,400返回结果,如果条件满足,大前置向CS查询额度,如果条件满足,CS占用额度并返回大前置结果,大前置再将两方结果返回给TI,在余额和额度都满足的情况下,向大前置发送记帐和扣减额度指令,大前置通知CS扣减额度,通知400记账。



方案可能会遇到的问题

● 问题

- 在并发情况下，用户甲查询了CS额度或者400余额，在发送发送记帐和扣减额度指令前，用户乙对同一账户CS额度和余额做了修改，导致用户甲的扣减动作失败。
- 在网络异常情况下，CS额度扣减成功，但是响应无法返回，对于ESB或者TI系统，都只能让整个交易失败。CS扣减的额度如何补偿是一个问题。

● 原因

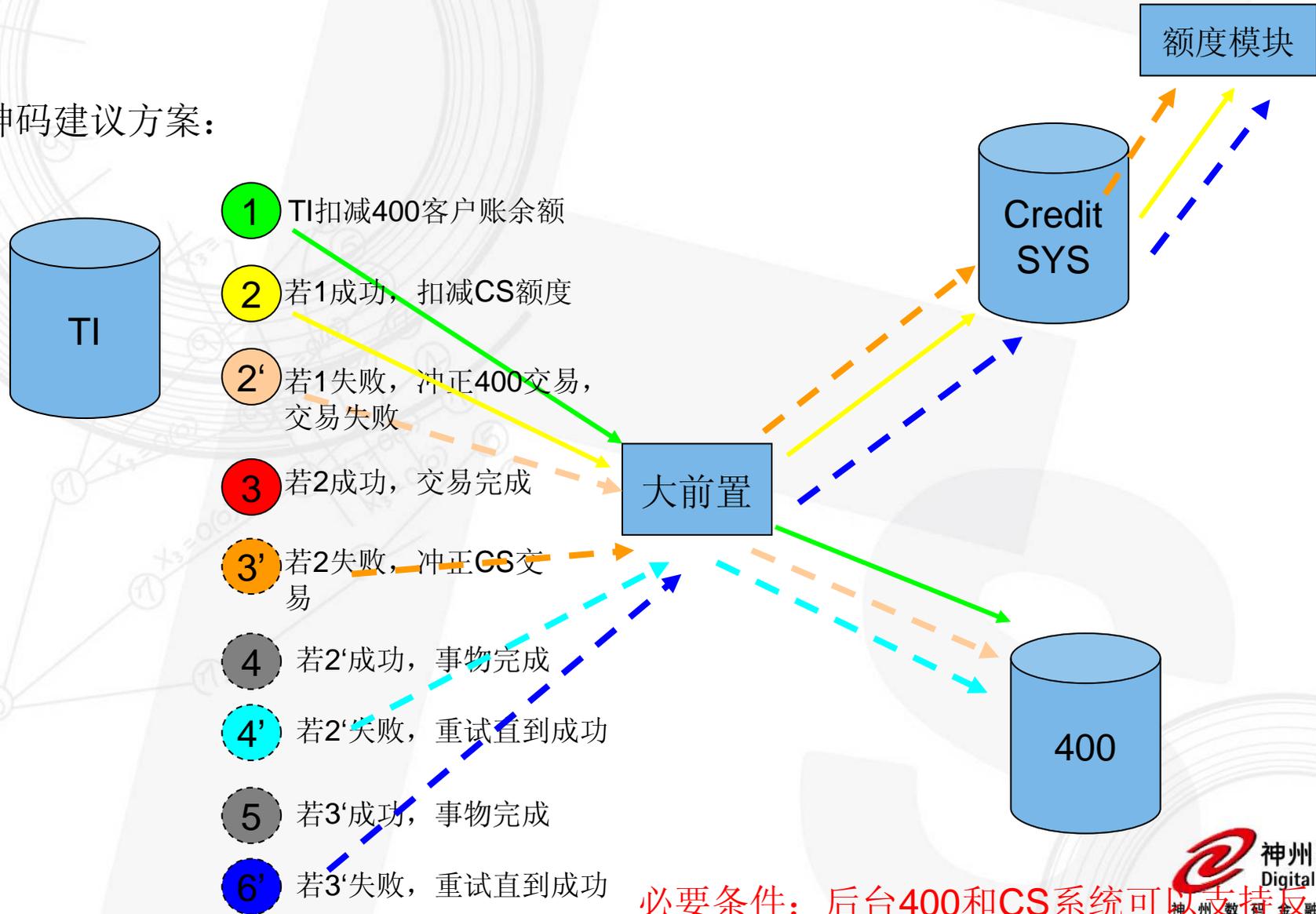
- 没有一个全局、分布式、基于SOA的事务管理机制。

● 更多的问题

- 若不实现全局事物管理机制，在问题2总是会存在。
- 若有了全局事务处理机制，系统面临较大的改造。

第二版：信贷系统三期---TI系统相关业务额度实时控制

神码建议方案：

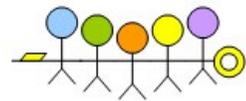


必要条件：后台400和CS系统可以支持反复冲正。



其他方案

- 方案：
 - 在ESB上定义新的服务，完成400余额扣除和CS额度扣减。
- 条件
 - **ESB记录流水**
 - **ESB完成自动冲正**
- 问题
 - **ESB功能需要增强**



第三版：TI系统相关业务额度实时控制

●TI发起查询交易

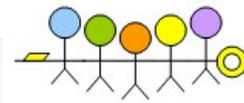
- TI发送查询交易给前置，前置同时发给主机和CS。如果主机和CS都成功，则返回给TI成功，如果有一方失败，则返回给TI失败。

●向CS系统的查询交易会锁住CS的额度

●TI发起记账交易

- TI发送记账交易给前置，前置先发送给主机，如果主机记帐成功，返回给前置，前置则将主机返回报文发给CS，如果CS返回给前置成功，则前置返回TI成功，如果CS返回给前置失败，则返回给TI失败，返回错误信息可标识出主机记帐成功，CS不成功（此情况，建议TI能够按记帐成功进行后续处理，CS单独进行手工变更处理）；

- 如果主机记账失败，前置直接返回失败给TI，返回错误信息可标识出主机和CS都不成功。



第三版方案存在的问题

- CS查询服务会锁定CS额度，强制客户端参与CS系统的事务管理，事务边界划分存在问题。
- 若CS查询服务的冲正失败，CS额度会一直锁定，会严重影响系统的使用
- 查询服务和扣减服务存在耦合
- 查询之后再扣减，在并发情况下一样会存在问题，客户承诺只有一个人调用该服务不能成为程序设计的基础。