



云原生

DDD架
构

金融级

Biz-SIP

金融级云原生业务中台

Biz-SIP是一套基于领域驱动设计（DDD），用于快速构建金融级云原生架构的服务整合中间件，包含了在金融场景里锤炼出来的最佳实践。

START



Work

Business

Project

Completion

目录 CONTENTS

01. 产品介绍

02. 功能介绍

03. 产品价值

04. 业务场景



PART 01

产 品 介 绍

Biz-SIP金融级云原生业务中台



云原生

可快速搭建云原生微服务体系，快速开发更具可靠性和扩展性、更加易于维护的云原生应用



DDD架构

领域驱动设计（DDD）打通架构与设计，帮助业务中台落地，加速企业规模化持续创新

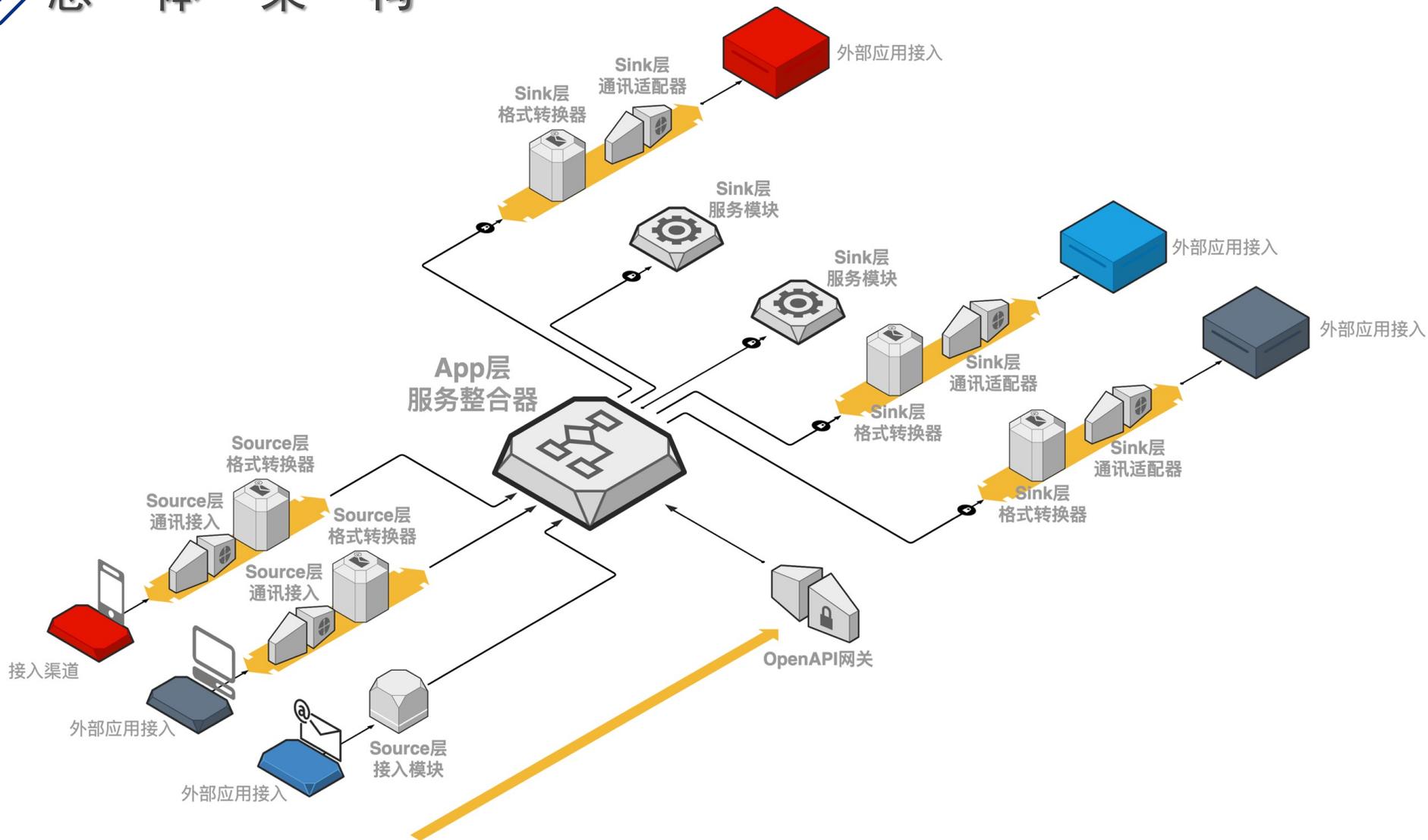


金融级

经历过大规模金融场景的锤炼，包含构建金融级云原生架构所需的能力，让用户更加专注于业务开发

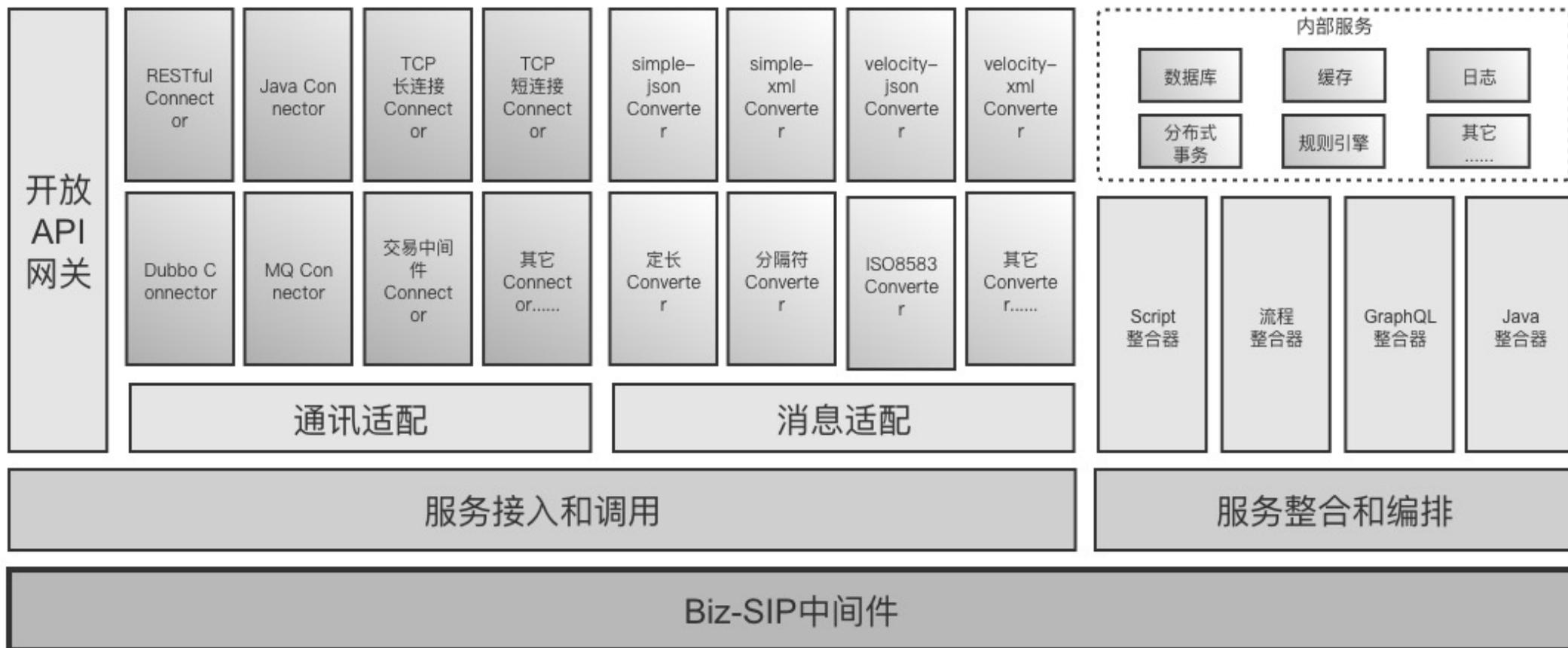


总体架构





逻辑架构

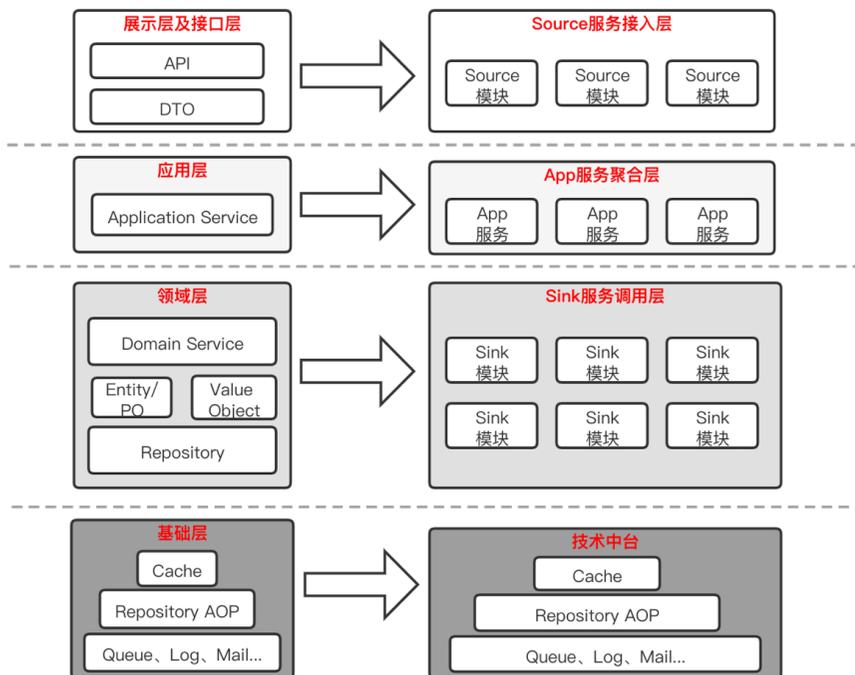


PART 02

功 能 介 绍

DDD架构

DDD
四层架构



Biz-SIP
三层架构
+
技术中台

0
1

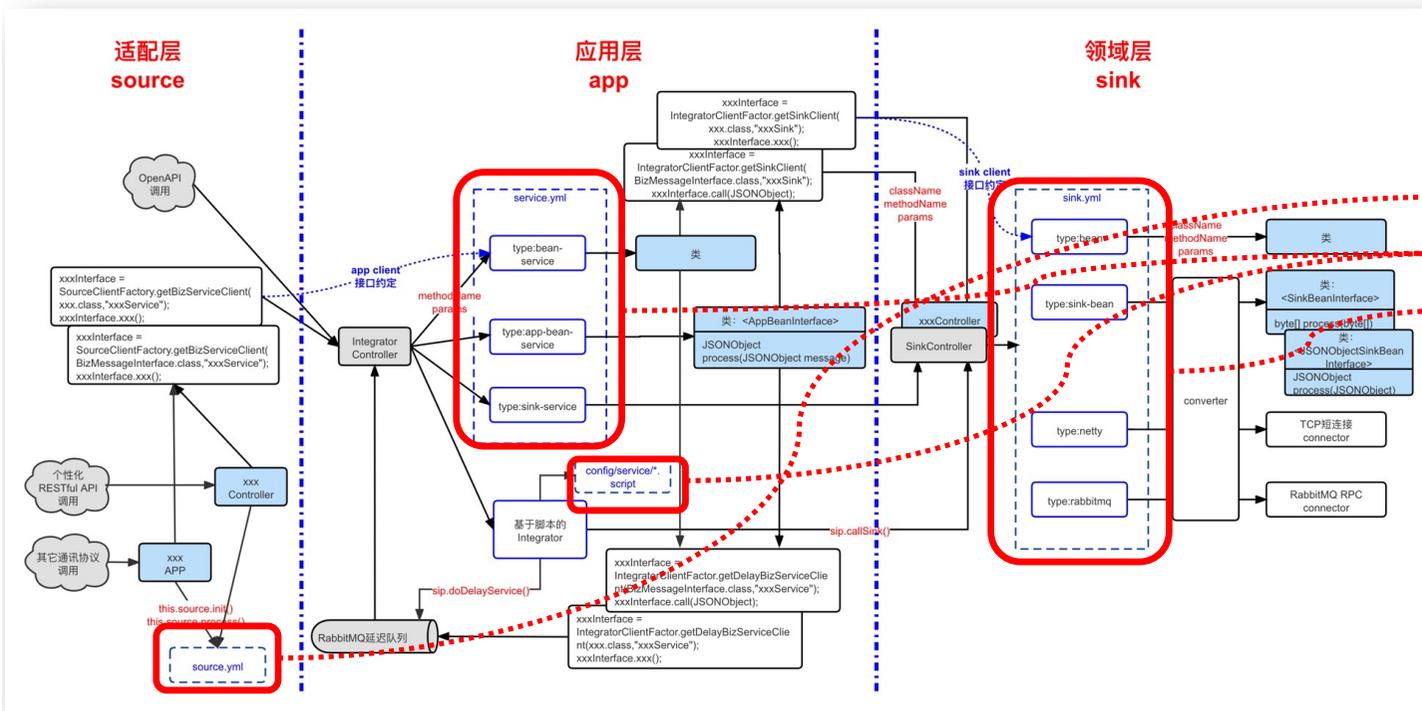
DDD架构打通架构与设计，帮助业务中台落地，加速企业规模化持续创新

0
2

Biz-SIP实现了DDD架构的平台化实现和落地



基于配置+代码框架的快速开发平台



配置文件（蓝色虚框）

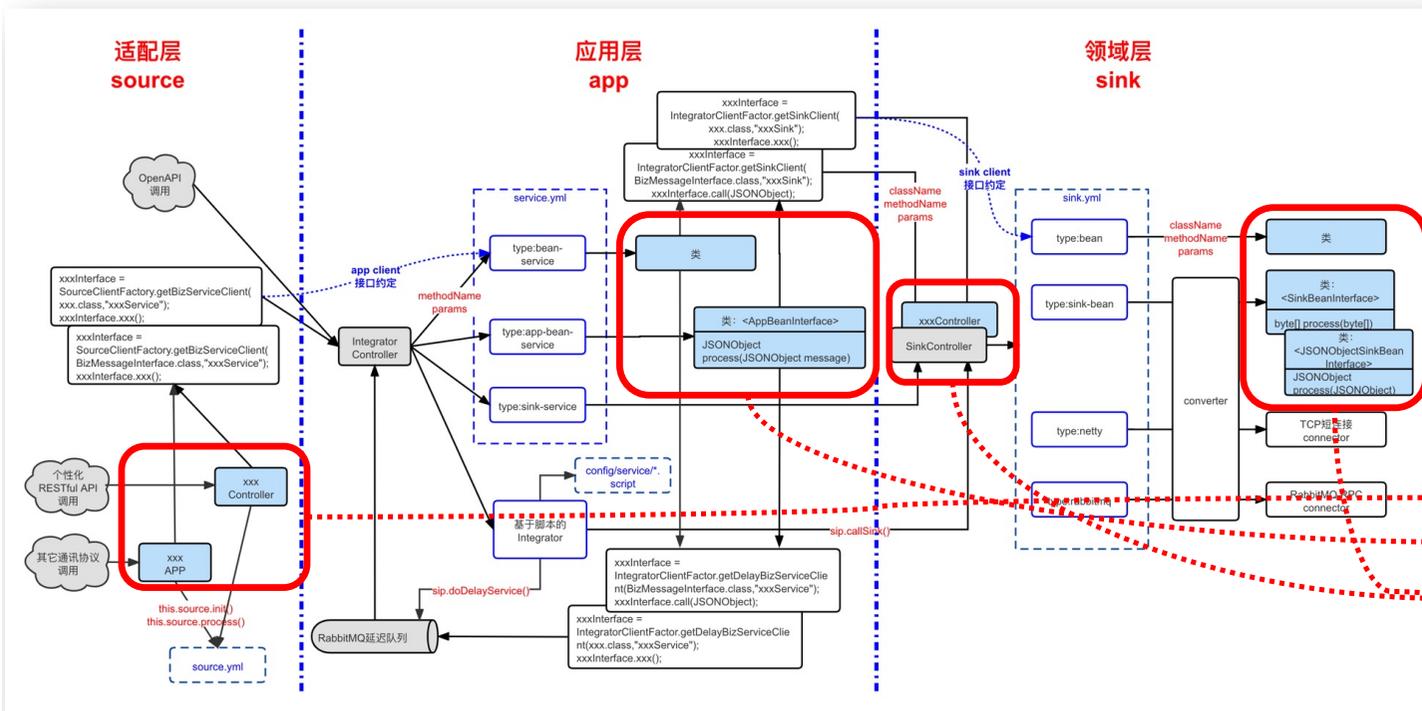
- Source层：通讯适配、消息格式转换
- App层：服务编排、服务接口校验
- Sink层：通讯适配、消息格式转换

模块代码（蓝底黑框）

- Source层：source 模块 starter
- App层：app 服务模块 starter
- Sink层：sink 模块 starter



基于配置+代码框架的快速开发平台



配置文件（蓝色虚框）

Source层：通讯适配、消息格式转换
App层：服务编排、服务接口校验
Sink层：通讯适配、消息格式转换

模块代码（蓝底黑框）

Source层：source 模块 starter
App层：app 服务模块 starter
Sink层：sink 模块 starter



基于配置+代码框架的快速开发平台

1

Source、App、Sink的配置

source.yml

定义source服务接入端口、
消息类型和服务调用约定

```
source.yml
1  id: source1
2  converter:
3    type: simple-json
4  service:
5    service-rules:
6      - predicate: '#{data[serviceId] != null}'
7        rule: '#{data[serviceId]}'
8      - predicate:
9        rule: source1/error
10
11 id: source4
12 converter:
13   type: simple-xml
```

service.yml

定义app服务类型和参数

```
service.yml
1  bizServiceId: sink/customer
2    type: sink-service
3  sinkId: customer-sink
4
5  bizServiceId: sink/account
6    type: sink-service
7  sinkId: account-sink
8
9  bizServiceId: app/personal
10   type: bean-service
11   className: com.xbank.app.service.PersonalAppService
12
13 bizServiceId: sink/payment1
14   type: sink-service
15   sinkId: payment1-sink
```

sink.yml

定义sink服务接入端口、消
息类型和通知适配参数

```
sink.yml
109 id: netty
110   type: rest
111   url: http://bizsip-sample-sink/netty
112   converter:
113     type: simple-json
114   connector:
115     type: netty
116     host: 127.0.0.1
117     port: 10001
118
119 id: sink9
120   type: rest
121   url: http://bizsip-sample-sink/sink9
122   converter:
123     type: simple-json
124   connector:
125     type: rabbitmq
126     route-key: sink9
127
128 id: sink10
129   type: rest
130   url: http://bizsip-sample-sink/sink10
```



基于配置+代码框架的快速开发平台

2

格式转换、服务校验和服务编排的配置

格式转换：
定义消息类型，以及解包和打包的配置

```
1 - name: sex
2   length: 1
3 - name: accountNo
4   length: 8
5 unpack-functions:
6   - name: trim
7 - name: accountName
8   length: 10
9 unpack-functions:
10  - name: trim
11 - name: balance
12  length: 10
13 pack-functions:
14  - name: decimalFormat
15    args:
16    - "###,###.00"
```

服务校验：
定义app服务的域校验和服务级校验

```
1 field-check-rules:
2 - field: email
3   rule: isEmail
4   message: '不是邮箱地址: {}'
5 - field: sex
6   rule: notEmpty
7   message: '不能为空'
8 - field: mobile
9   rule: isMatchRegex
10  args:
11  - '^([1][3,4,5,6,7,8,9][0-9]{9})$'
12  message: '不是手机号 {}'
13 field-check-mode: one
14 service-check-rules:
15 - script: if(data.sex == '1')
16   {return '性别不符合!'};
17  message: '额度超限'
18 service-check-mode: one
```

服务编排：
通过脚本实现服务的编排（除了脚本外，还有通过Java代码实现的服务编排）

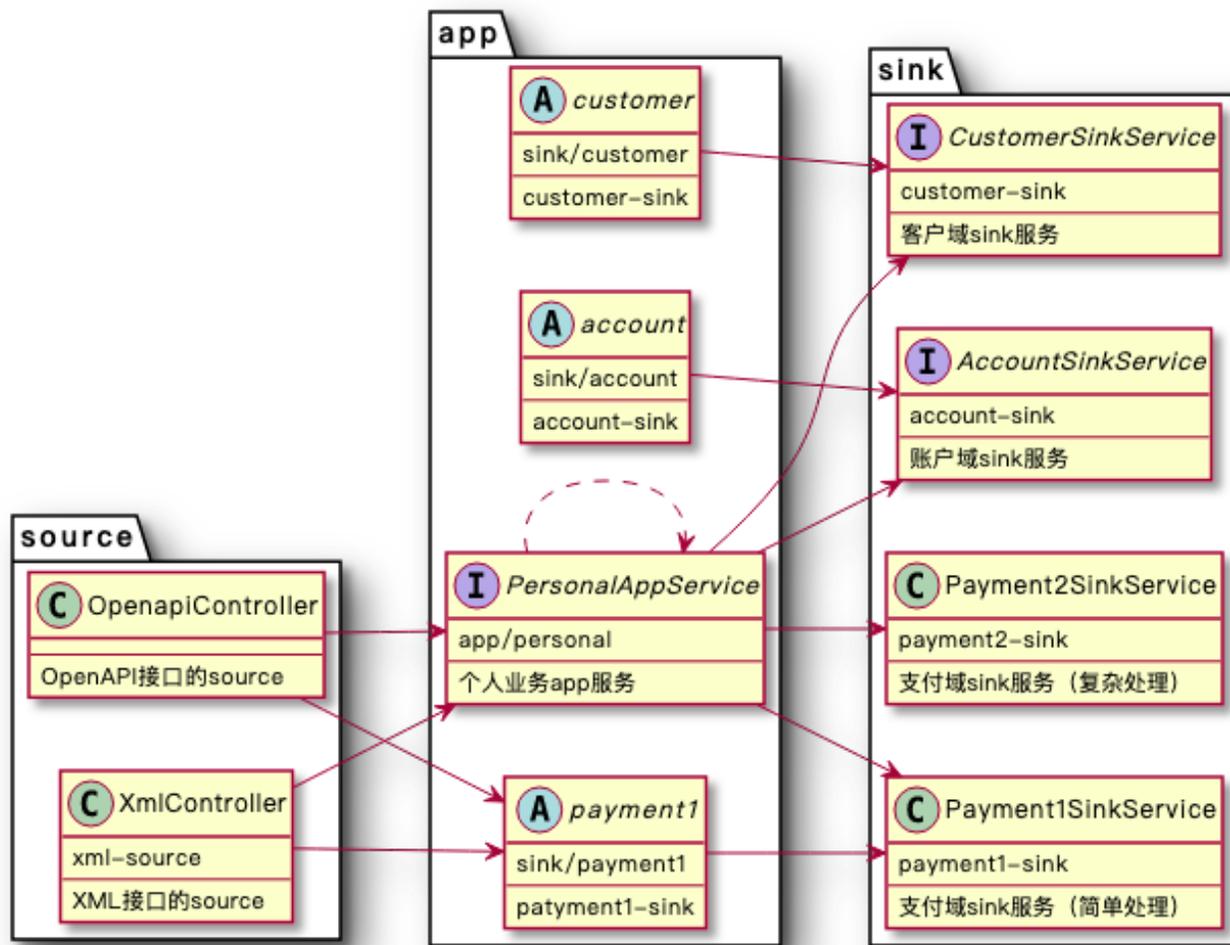
```
1 var result = sip.callSink("sink1",bizmessage.data)
2 if (result.code == 0) {
3   bizmessage.data.accountName = result.data.accountName;
4 }
5 else {
6   var error_result = {
7     code: result.code,
8     message: result.message
9   }
10  return error_result;
11 }
12 result = sip.callSink("sink2",bizmessage.data)
13 if (result.code == 0) {
14   bizmessage.data.balance = result.data.balance;
15 }
16 else {
17   var error_result = {
18     code: result.code,
19     message: result.message
20   }
21  return error_result;
22 }
23 return bizmessage.data;
```



基于配置+代码框架的快速开发平台

3

平台提供支撑DDD框架适配层、应用层和领域服务层的三层代码框架





基于配置+代码框架的快速开发平台

4

Source层服务类实现app服务接口调用

```
21 @RestController
22 @RequestMapping("/personal")
23 public class OpenapiController {
24     private PersonalAppInterface personalAppInterface = SourceClientFactory
25         .getBizServiceClient(PersonalAppInterface.class, bizServiceId: "app/personal");
26     private BizMessageInterface payment1SinkInterface = SourceClientFactory
27         .getBizServiceClient(BizMessageInterface.class, bizServiceId: "sink/payment1");
28
29     @GetMapping(value = "/getCustomerAndAccountList")
30     public CustomerAndAccountList getCustomerAndAccountList(String customerId) {
31         return this.personalAppInterface.getCustomerAndAccountList(customerId);
32     }
33
34     @GetMapping(value = "/getAccountListByCustomerId")
35     public List<Account> getAccountListByCustomerId(String customerId) {
36         return this.personalAppInterface.getAccountListByCustomerId(customerId);
37     }
38
39     @GetMapping(value = "/getCustomer")
40     public Customer getCustomer(String customerId) {
41         return this.personalAppInterface.getCustomer(customerId);
42     }
43
44     @GetMapping(value = "/getCustomerAndSaf2Payment2")
45     public Customer getCustomerAndSaf2Payment2(String tranCode, String customerId) throws BizMessageException {
46         return this.personalAppInterface.getCustomerAndSaf2Payment2(tranCode, customerId);
47     }
48 }
```

定义app服务调用接口

调用app服务接口



基于配置+代码框架的快速开发平台

5

App层服务类实现sink服务编排和sink服务接口调用

```
1 package com.xbank.app.service;
2
3 import ...
4
18
19 @...}
20
21 public class PersonalAppService implements PersonalAppInterface {
22     private AccountSinkInterface accountSinkInterface = IntegratorClientFactory
23         .getSinkClient(AccountSinkInterface.class, sinkId: "account-sink");
24     private CustomerSinkInterface customerSinkInterface = IntegratorClientFactory
25         .getSinkClient(CustomerSinkInterface.class, sinkId: "customer-sink");
26     private BizMessageInterface payment1SinkInterface = IntegratorClientFactory
27         .getSinkClient(BizMessageInterface.class, sinkId: "payment1-sink");
28     private BizMessageInterface payment2SinkInterface = IntegratorClientFactory
29         .getSinkClient(BizMessageInterface.class, sinkId: "payment2-sink");
30     private PersonalAppInterface personalAppDelayInterface = IntegratorClientFactory
31         .getDelayBizServiceClient(PersonalAppInterface.class, bizServiceId: "app/personal",
32             ...delayMilliseconds: 0,1000,2000,4000,8000,16000,32000);
33
34 @Override
35 public CustomerAndAccountList getCustomerAndAccountList(String customerId) {
36     Customer customer = this.customerSinkInterface.getCustomer(customerId);
37     List<Account> accountList = this.accountSinkInterface.getAccountListByCustomerId(customerId);
38     CustomerAndAccountList customerAndAccountList = new CustomerAndAccountList();
39     customerAndAccountList.setCustomer(customer);
40     customerAndAccountList.setAccountList(accountList);
41     return customerAndAccountList;
42 }
```

定义sink服务调用接口

调用sink服务接口



基于配置+代码框架的快速开发平台

6

Sink层服务类实现交易处理和第三方系统对接

```
public class AccountSinkService implements AccountSinkInterface {
    @Autowired
    private PayoutCmdExe payoutCmdExe;
    @Autowired
    private PayoutCompensationCmdExe payoutCompensationCmdExe;
    @Autowired
    private GetAccountListByCustomerIdCmdExe getAccountListByCustomerIdCmdExe;

    @Override
    public List<Account> getAccountListByCustomerId(String customerId) {
        return this.getAccountListByCustomerIdCmdExe.getAccountListByCustomerId(customerId);
    }

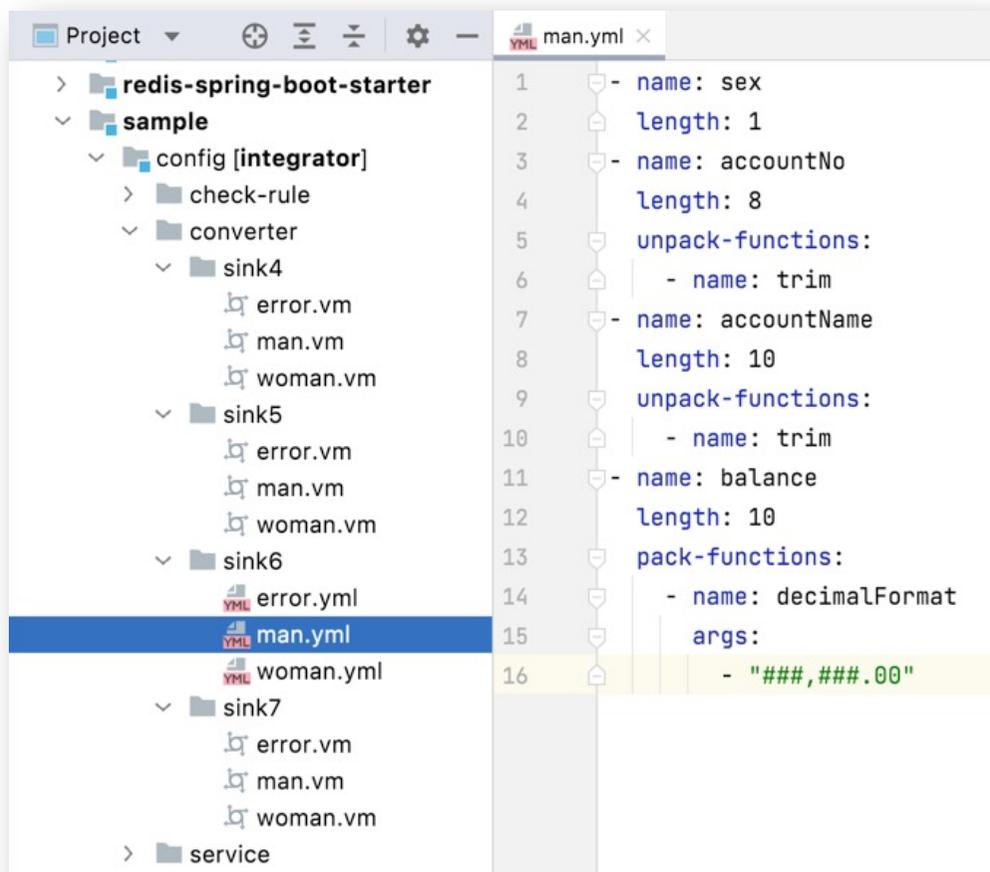
    @Override
    public Account payout(String accountId, Long amount) {
        return this.payoutCmdExe.payout(accountId, amount);
    }

    @Override
    public Account payoutCompensation(String accountId, Long amount) {
        return this.payoutCompensationCmdExe.payoutCompensation(accountId, amount);
    }
}

class Payment2SinkService implements JSONObjectSinkBeanInterface {
    @Autowired
    private TimeoutCmdExe timeoutCmdExe;
    @Autowired
    private TimeoutAndFailCmdExe timeoutAndFailCmdExe;
    @Autowired
    private TimeoutAndSuccessCmdExe timeoutAndSuccessCmdExe;
    @Autowired
    private SuccessCmdExe successCmdExe;

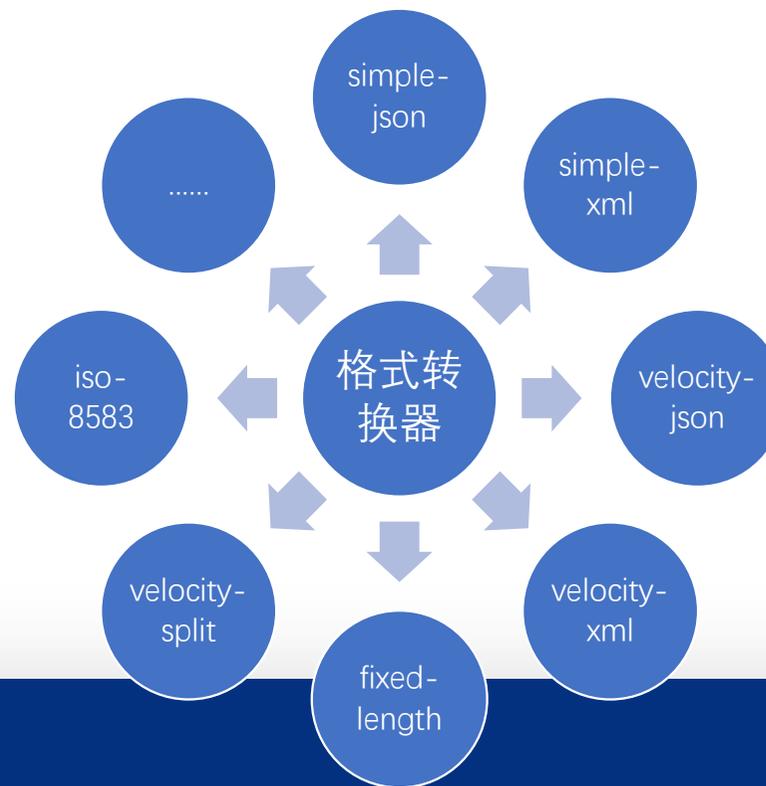
    @Override
    public JSONObject process(JSONObject jsonObject) throws BizException {
        Log.info("传入消息:\n{}", jsonObject.toString());
        AbstractSinkBeanCmdExe sinkBeanCmdExe;
        String tranMode = (String)jsonObject.get("tranMode");
        switch (tranMode) {
            case "timeout":
                // 收到交易后, 永远返回超时
                return timeoutCmdExe.execute(jsonObject);
            case "3timeout-fail":
                // 收到交易后, 前3次返回超时, 第4次返回失败码
                return timeoutAndFailCmdExe.execute(jsonObject);
            case "3timeout-success":
                // 收到交易后, 前3次返回超时, 第4次成功返回原报文
                return timeoutAndSuccessCmdExe.execute(jsonObject);
        }
    }
}
```

支持多种消息格式之间的互相转换



The screenshot shows an IDE interface with a project tree on the left and a YML configuration file named 'man.yml' open in the editor. The project tree is expanded to show a 'sample' directory containing a 'converter' sub-directory with several 'sink' folders (sink4 to sink7). Each sink folder contains VM files (error.vm, man.vm, woman.vm) and some have YML files (error.yml, man.yml, woman.yml). The 'man.yml' file is selected and its content is displayed in the editor. The YML content is as follows:

```
1  - name: sex
2    length: 1
3  - name: accountNo
4    length: 8
5    unpack-functions:
6      - name: trim
7  - name: accountName
8    length: 10
9    unpack-functions:
10     - name: trim
11  - name: balance
12    length: 10
13  pack-functions:
14    - name: decimalFormat
15      args:
16        - "###,###.00"
```



- 平台内置格式转换器，支持XML、JSON、定长、有分隔符、ISO8583等多种报文格式的解包和打包
- 插件化的格式转换器Converter，可扩展更多的报文格式

支持多种通讯模式的适配



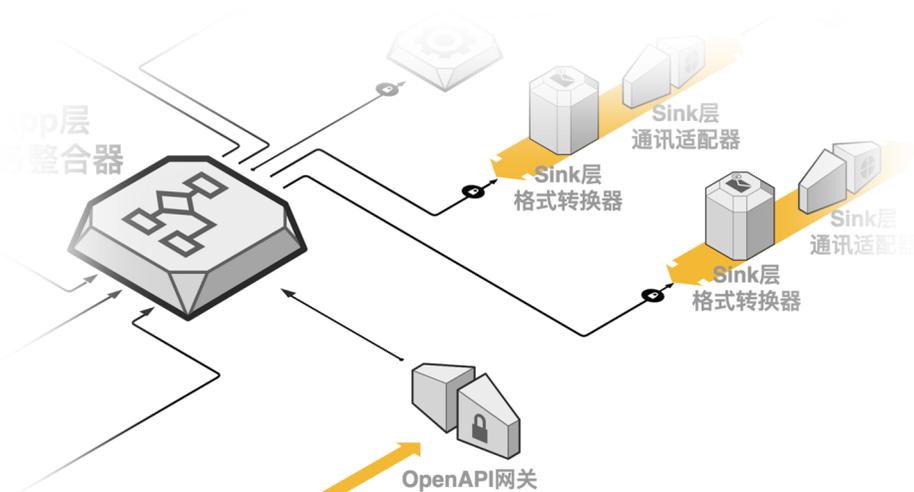
```
sink.yml
109 - id: netty
110   type: rest
111   url: http://bizsip-sample-sink/netty
112   converter:
113     type: simple-json
114   connector:
115     type: netty
116     host: 127.0.0.1
117     port: 10001
118
119 - id: sink9
120   type: rest
121   url: http://bizsip-sample-sink/sink9
122   converter:
123     type: simple-json
124   connector:
125     type: rabbitmq
126     route-key: sink9
127
128 - id: sink10
129   type: rest
130   url: http://bizsip-sample-sink/sink10
```

- 平台内置Restful、TCP（长连接、短连接）、RabbitMQ等通讯适配器，可实现0代码配置适配接入和调用
- 规范的通讯适配器开发框架，方便开发人员快速个性化开发通讯适配模块
- 插件化的通讯连接器Connector，可扩展更多的通讯适配接入和调用

OpenAPI网关

统一接口

为所有聚合服务提供统一对外的API接口标准



接口检验

实现并行的域级和服务级接口校验

全局监控

实现接口全生命周期管理，链路跟踪、应用性能、业务数据监控于一体的

安全认证

提供安全的网关认证鉴权机制

App 层支持灵活的服务编排



支持多种服务编排方式，并且服务编排引擎可实现插件化扩展

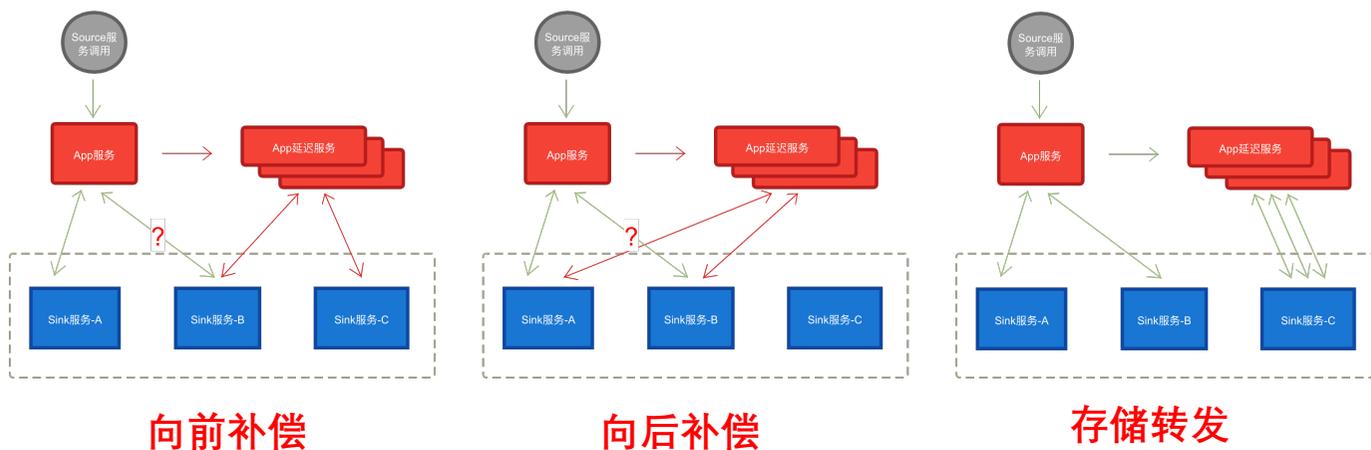
可开发基于Java的服务编排类

```
18  
19 @Override  
20  
21 public class PersonalAppService implements PersonalAppInterface {  
22     private AccountSinkInterface accountSinkInterface = IntegratorClientFactory  
23         .getSinkClient(AccountSinkInterface.class, sinkId: "account-sink");  
24     private CustomerSinkInterface customerSinkInterface = IntegratorClientFactory  
25         .getSinkClient(CustomerSinkInterface.class, sinkId: "customer-sink");  
26     private BizMessageInterface payment1SinkInterface = IntegratorClientFactory  
27         .getSinkClient(BizMessageInterface.class, sinkId: "payment1-sink");  
28     private BizMessageInterface payment2SinkInterface = IntegratorClientFactory  
29         .getSinkClient(BizMessageInterface.class, sinkId: "payment2-sink");  
30     private PersonalAppInterface personalAppDelayInterface = IntegratorClientFactory  
31         .getDelayBizServiceClient(PersonalAppInterface.class, bizServiceId: "app/pers  
32         ...delayMillisecond: 0, 1000, 2000, 4000, 8000, 16000, 32000);  
33  
34     @Override  
35     public CustomerAndAccountList getCustomerAndAccountList(String customerId) {  
36         Customer customer = this.customerSinkInterface.getCustomer(customerId);  
37         List<Account> accountList = this.accountSinkInterface.getAccountListByCustom  
38         CustomerAndAccountList customerAndAccountList = new CustomerAndAccountList();  
39         customerAndAccountList.setCustomer(customer);  
40         customerAndAccountList.setAccountList(accountList);  
41         return customerAndAccountList;  
42     }  
}
```

可编写流程脚本来实现服务编排

```
1 var result = sip.callSink("sink1", bizmessage.data)  
2 if (result.code == 0) {  
3     bizmessage.data.accountName = result.data.accountName;  
4 }  
5 else {  
6     var error_result = {  
7         code: result.code,  
8         message: result.message  
9     }  
10    return error_result;  
11 }  
12 result = sip.callSink("sink2", bizmessage.data)  
13 if (result.code == 0) {  
14     bizmessage.data.balance = result.data.balance;  
15 }  
16 else {  
17     var error_result = {  
18         code: result.code,  
19         message: result.message  
20     }  
21    return error_result;  
22 }  
23 return bizmessage.data;  
24
```

支持分布式事务



Saga 分布式事务

支持向前补偿和向后补偿
支持重试机制，重试时间及最大重试次数可定制

最大努力通知

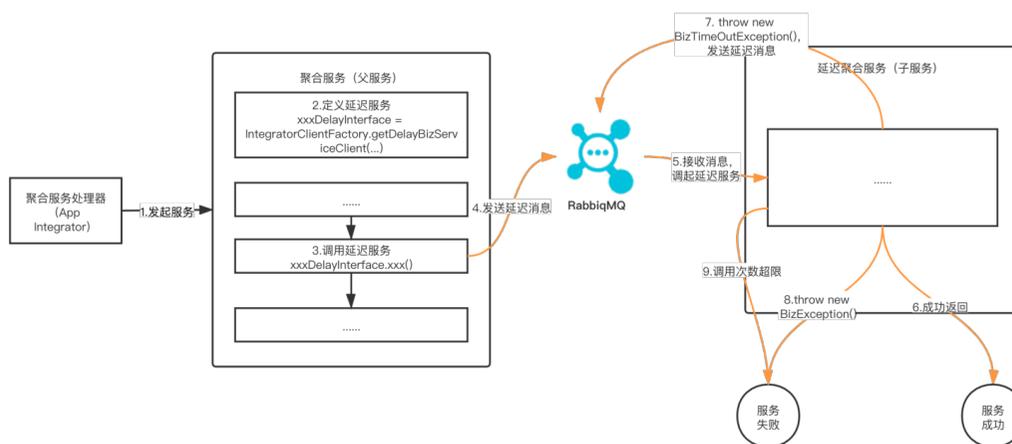
支持重试时间及最大重试次数可定制

支持同步异步服务调度

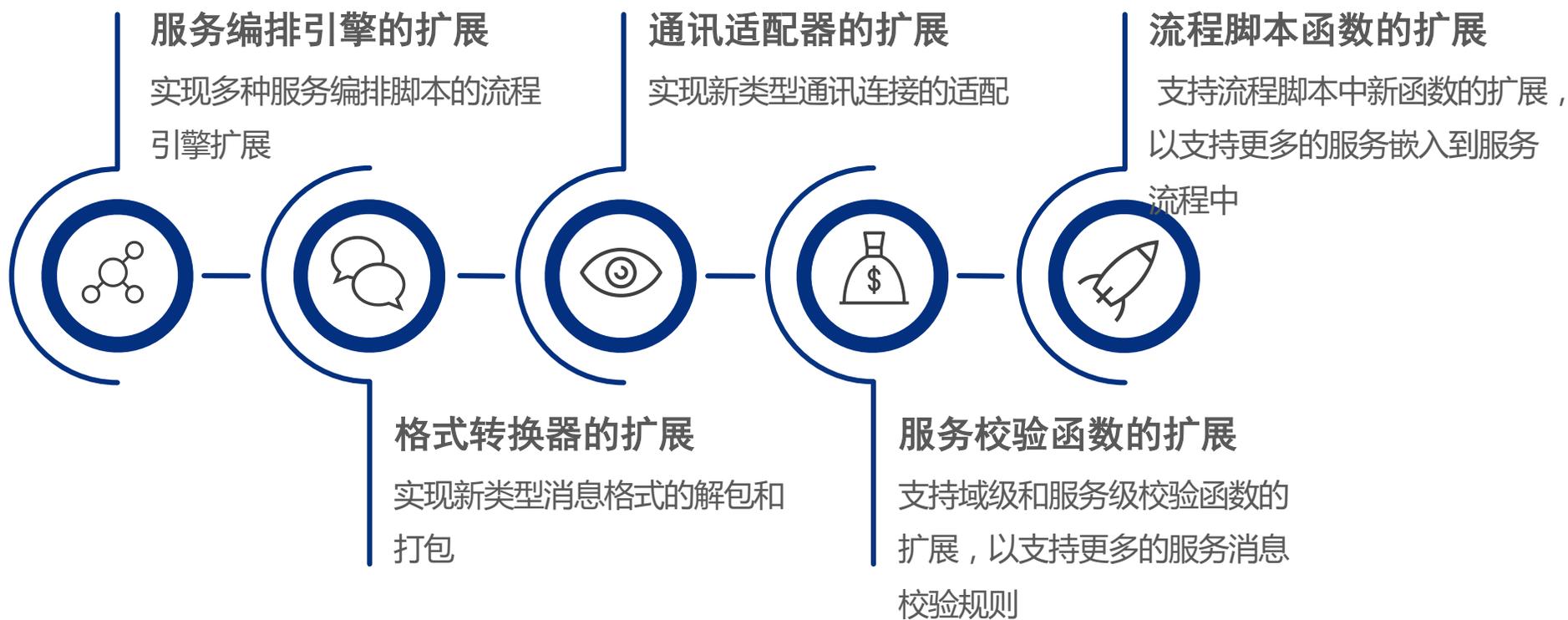
在服务编排中，支持对同步和异步 Sink 层服务的调度，满足多种场景需求。

支持交易全程监控

可订阅并处理交易成功、挂起、失败事件，并进行代码处理

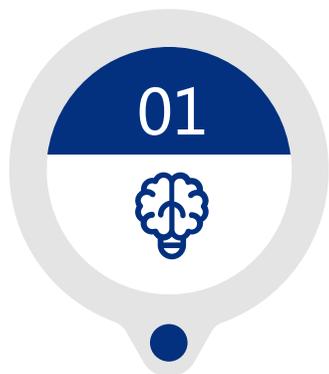


功能组件插件的扩展





支持多种部署方案



主机部署

支持基于SpringBoot轻量快速部署，以及基于SpringCloud中大型集群化系统环境部署。



Docker 部署

支持基于Docker容器的快速部署，针对对并发和高可用有一定要求的系统环境。



K8S容器部署

支持集群化部署，针对高可用、高并发的大规模系统环境部署。



Istio服务网格部署

实现基于云原生的持续交付和DevOps运维体系，支持超大规模系统环境部署。

PART 03

产 品 价 值

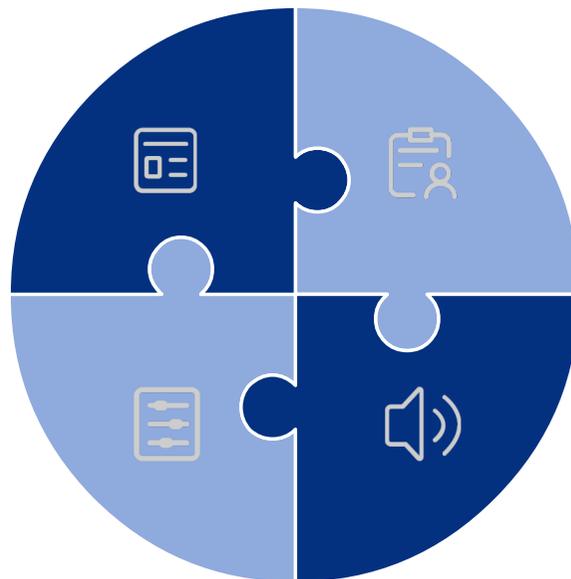
产品价值

基于主流的云原生架构

基于Spring Cloud云原生开发框架，支持Docker、Kubernetes、Istio等部署方式。

面向微服务系统的统一管理

基于DDD架构，实现可配置化的微服务编排引擎，支持分布式事务保障



遗留系统接入和微服务改造

实现消息转换适配可配置化，支持多种消息类型之间的互转

实现通讯接入适配可配置化，快速对接原有系统

实现业务视角对IT的统一管理

通过DDD分层架构，使系统更容易被理解，业务人员也可能参与到系统的架构和配置中来。

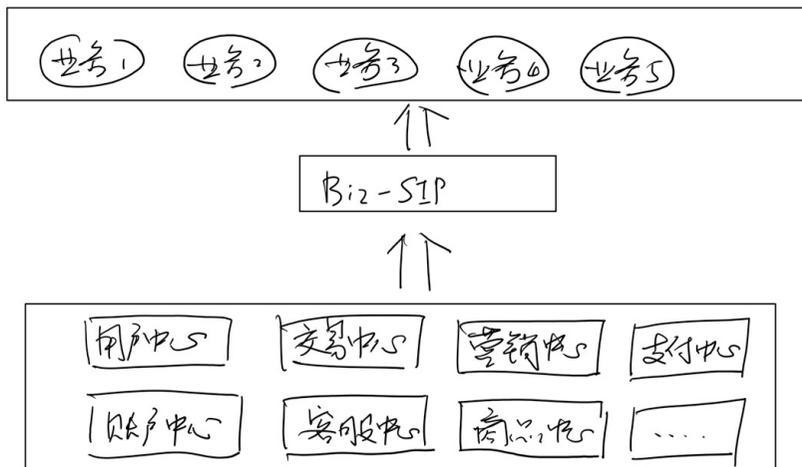
PART 04

业 务 场 景

业务中台

挑战

- 业务中台支持多个前台业务且具备业务属性的共性能力组织，构建企业共享服务中心，助力企业实现快速、低成本创新
- 如何上升到公司战略？
- 如何快速整合现有服务能力？



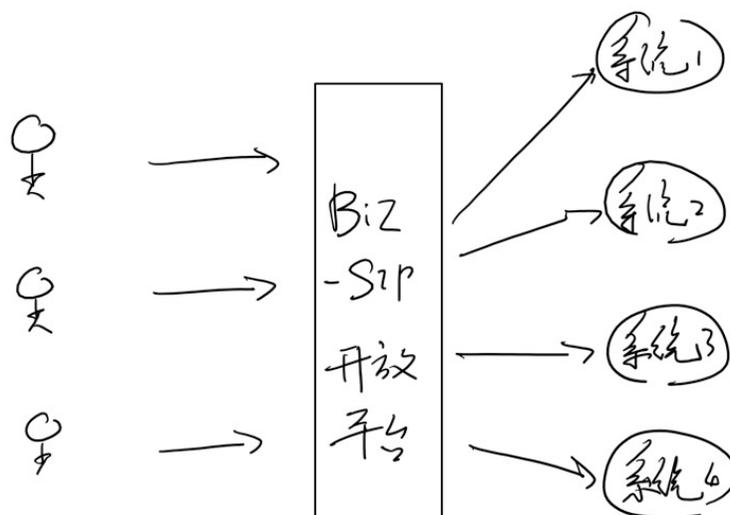
解决方案

- 基于Biz-SIP打造业务中台，整合各系统服务；
- Biz-SIP能快速适配企业遗留系统，实现快速接入；
- Open API接口统一实现对内和对外的中台服务能力；
- 基于云原生架构，满足高可用高并发，满足集中化的业务中台核心引擎。

开放平台

挑战

- 企业外部生态系统的建立，开放平台的建设是根本；
- 开放平台需要实现现有遗留系统的服务能力开放；
- 开放平台承载越来越多的业务，要求高并发高可用。



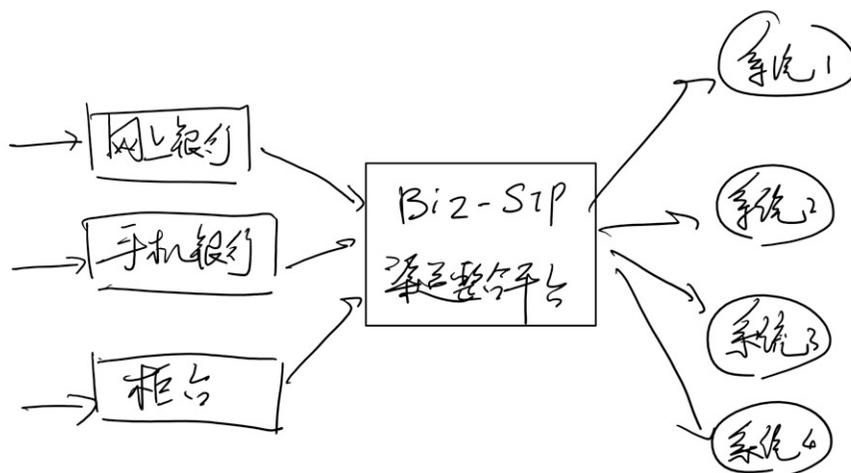
解决方案

- Open API网关实现了服务接口的统一暴露，符合开放平台接口标准；
- Biz-SIP实现了API接口的统一配置和管理，满足开放平台统一接口管理的要求；
- Biz-SIP实现了高并发、高可用的服务调用，满足开放平台应对突发流量的能力；
- Biz-SIP提供了整合现有遗留系统，实现微服务化改造的能力。

渠道整合

挑战

- 和企业现有系统服务的对接
- 统一的客户视图
- 高可用、高并发



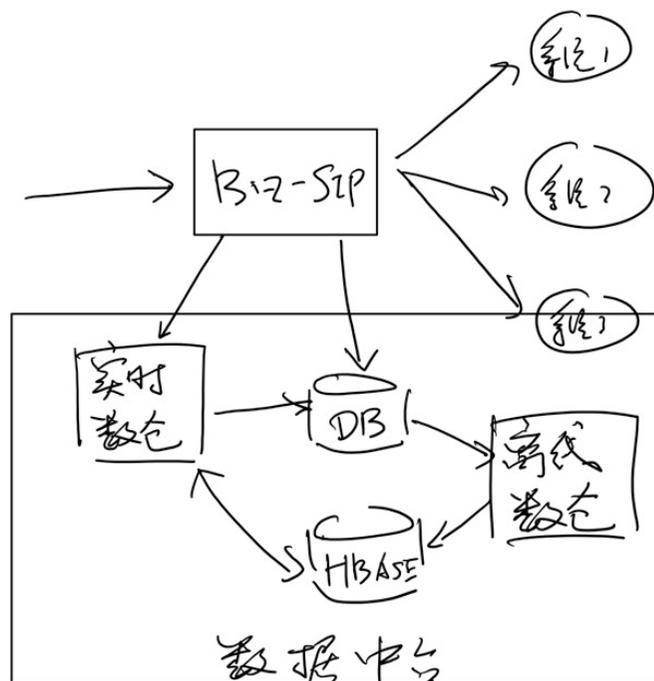
解决方案

- Biz-SIP能快速适配企业遗留系统，实现快速接入；
- 通过Open API接口，为全渠道提供统一的服务接入；
- Biz-SIP接入统一的客户中心，并融入渠道业务流程编排中，实现全渠道统一的客户视图；
- 基于云原生架构，满足高可用高并发，满足集中化的业务中台核心引擎。

数据中台

挑战

- 如何让数据中台助力业务？
- 如何使用数据中台实时数仓、离线数仓的技术能力？



解决方案

- 数据中台接入Biz-SIP，并整合在服务编排流程中，为业务提供数据存储和访问能力；
- Biz-SIP能在服务编排中通过实时流计算，接入实时数仓；
- 为原有遗留系统提供了非侵入性的数据采集接口。
- Biz-SIP内部服务接口标准，为实时数仓和离线数仓提供了标准的数据接口基础；



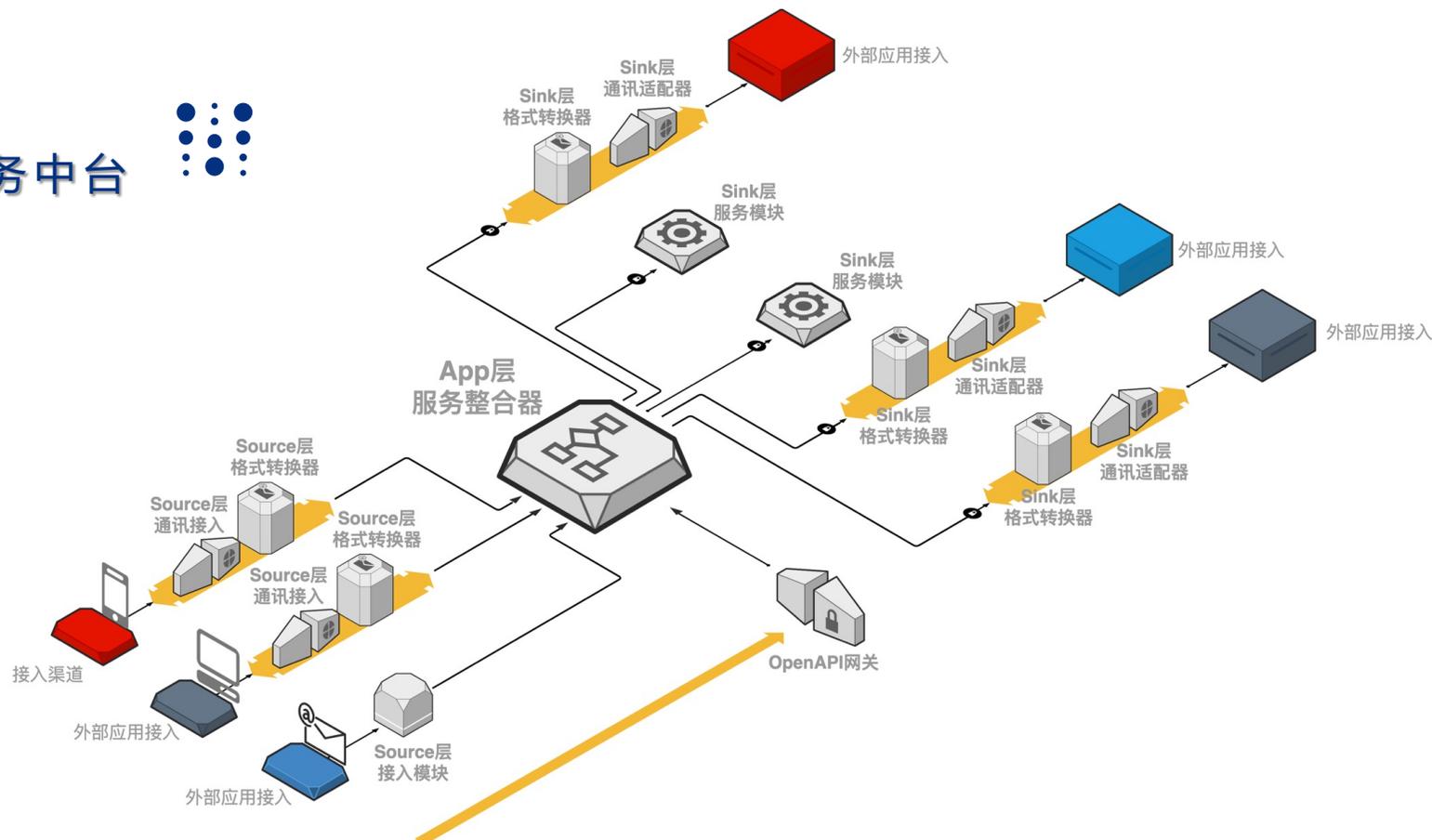
云原生

DDD架构

金融级

Biz-SIP

打造金融级云原生业务中台



END