RepChain Interface Cooperation

基于RepChain 2.0 的链外接口协同组件

RepChain DEV Team

目录

1. RepChain Interface Cooperation

基于RepChain 2.0 的链外接口协同组件

立刻进入

1.1 相关文档

RepChain 2.0 使用说明

RepChain 1.1 使用说明

RepChain 1.1 合约开发

RepChain 1.1 Dashboard

RepChain 1.0 使用说明

2. 开始

2.1 引言

> 注意
>
> 首次阅读本文档，请严格按照文档顺序进行阅读，以便可以更加全面的理解如何使用链外接口协同组件，谢谢。

> 提示
>
> 本项目主要参考了专利《一种可举证的接口协同方法及系统》，并通过RepChain区块链进行实现，提供了接入中间件的存证方式与示例。

2.1.1 一、为什么会有接口协同组件

在开发人员开发过程中，难免会遇到如下情况：

1. 调用其他系统的接口时，需要双方协调。
2. 接口提供方文档不清晰。
3. 接口调用时，由于一些原因，调用方无法正确拿到调用数据，进而双方产生矛盾却无从考证。

2.1.2 二、链外接口协同组件

接口协同组件旨在解决上述等问题而产生，并提供如下方案：

1. 规范约束了统一的IDL (接口描述语言)，给服务方在提供接口定义时提供了统一规范。
2. 服务方及调用方均需要在区块链上进行登记，登记后按照接口描述语言进行实现即可完成调用。
3. 接口调用时需要双方对数据进行签名，并存证于区块链上。保证了在接口调用中，若出现意料之外的问题时有据可查。
4. 双方存证于区块链的数据均为传输数据的HASH取值，并且由双方进行签名后存证，保证了敏感数据的安全性和可靠性。

2.1.3 三、术语

同步请求

指调用方调用服务方接口时，可直接通过当前请求获取数据。无需等待服务方推送。

异步请求

指调用方调用服务方接口时，服务方会返回一个应答信息，但应答信息中不包含调用方所需要的数据。

服务方需要通过访问调用方的接口进行数据推送，再返回调用方所需要的数据。此类场景常常出现在审批流程等业务中。

数据签名

通过RepChain注册的证书和私钥对数据进行签名，签名算法使用接口定义中所定义的算法。

数据验签

通过对签名数据的证书验证，判断数据签名是否由当前证书持有者进行签名。

交易对象

提交给区块链的数据结构，由区块链合约进行约束。与普通后台接口的接口请求参数类似，不同点在于需要签名后提交，且通过RepChain的Api提交给区块链。

2.1.4 四、 接口协同存证方式

接入中间件

2.1.5 五、存证方式

使用中间件实现接口协同存证提供了如下几种代码示例：

1. 同步请求
   - 同步单次调用服务方

     此类为最常见的场景，即请求后则得到返回数据。
   - 同步多次调用服务方

     此类场景为当数据量过大，一次请求无法获取全部所需数据时，需要多次请求服务方接口。
2. 异步请求
   - 异步单次调用请求方，请求方返回单次数据

     此类场景为当调用服务方数据时，服务方需要有流程审批等原因，无法同步返回数据。此类情况需要在接口定义时，定义应答方接口规范，并让调用方进行接口实现。
   - 异步多次调用请求方，请求方多次返回数据

     此类场景为当调用服务方数据时，服务方需要有流程审批等原因，无法同步返回数据，且数据量较大。此类情况需要在接口定义时，定义应答方接口规范，并让应答方进行接口实现。

2.2 准备工作

2.2.1 一、需要准备的环境及工具

1. Java 1.8 (必须)

建议使用`zulu-jdk`（oracle-jdk也是可以的），从官网下载，也可使用**IDEA**直接下载(File->Project Structure->SDKS，然后选择"+"，选择Download JDK)， 选择Azul-zulu-community，jdk-8。

需要配置环境变量，或者在编译器中指定当前版本jdk。

2. Maven项目构建管理工具 (必须)

Maven 是一个项目管理工具，可以对 Java 项目进行构建、依赖管理，是一个自动化构建工具。

自动化构建工具：将原材料（java、js、css、html....）->产品（可发布项目）

编译-打包-部署-测试 -> 自动构建可从官网下载并配置相关信息。

> ⚠ 注意

- 需要将Maven的配置到系统的环境变量中，以方便在终端中使用maven命令。
- 需要将同一个Maven地址配置到使用的编译器中，防止项目构建时出现其他问题。

3. Gradle项目构建管理工具 (必须)

*Gradle*是一个基于Apache Ant和Apache Maven概念的项目自动化构建开源工具。可通过示例代码中的gradlew脚本进行下载。

需要使用者先进行一定的学习。

4. IDE (必须)

编译器，可根据自己使用情况进行选择，推荐使用最新版IDEA。

5. Lombok (必须)

Java简化代码的工具，需要将Lombok插件安装到**IDE**中，具体可查看Lombok官网。

6. RepChain (必须)

需要有一套已经在运行的RepChain组件环境，若已有RepChain环境可忽略。

快速搭建方法可查看此处。

如果需要详细的RepChain文档，可查看RepChain文档。

7. RepChain Dashboard (必须)

RepChain管理控制台，请参考部署视频，文档更新中...

8. RCJava-core (必须)

与RepChain 连接的Java客户端及代码示例。

具体使用说明请查看此处。

> ⚠ 注意
>
> 此处需要使用 `mvn clean install` 方式发布到本地仓库。

9. 接入中间件客户端(必须)

- 下载代码 `middleware` 分支

```
git clone https://gitee.com/BTAJL/api-coord.git -b mid-client
```

- 进入到代码文件夹，执行脚本将代码发布到maven本地仓库

Win

```
cd .\api-coord\
.\gradlew.bat publishLibraryPublicationToMavenLocal
```

Mac/Linux

```
cd api-coord
./gradlew publishLibraryPublicationToMavenLocal
```

- 引入项目

Maven

```
<dependency>
  <groupId>repchain.interface.cooperation</groupId>
  <artifactId>Interface-Cooperation-Client</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

Gradle

```
    implementation 'repchain.interface.cooperation:Interface-Cooperation-Client:1.0-SNAPSHOT'
```

2.2.2 二、接口协同所需代码

1. RepChain主程序

注意：RepChain需要使用2. 0版本

Gitee地址：https://gitee.com/BTAJL/repchain/tree/dev_jdk13_2.0.0.0/

2. RepChain-Dashboard管理平台

未开源，需要请联系Repchain Gitee

3. RCJava-core

与RepChain相关的Java 客户端

Gitee地址：https://gitee.com/BTAJL/RCJava-core

4.Api-coord

接入中间件代码，

代码暂未开源，未经许可禁止商用。

middleware_rc2.0.0分支为中间件代码

mid-client分支为客户端代码

Gitee地址：https://gitee.com/BTAJL/api-coord

2.2.3 三、通过RepChain-Dashboard进行账号注册及授权

> 提示

请通过RepChain-Dashboard进行账号注册及授权操作，具体文档请参考RepChain-Dashboard使用文档。

2.2.4 四、OpenAPI接口定义语言

1. OpenAPI

openApi用于接口协同中接口描述，可在接口定义中起到描述约束的作用。请先了解相关定义，官网。

2. 生成OpenAPI描述JSON

如果项目中用到了Swagger-UI相关功能，可通过SwaggerUI进行接口描述内容生成。根据Swagger版本访问相对应的接口即可获取OpenAPI生成的Json数据。

例：

Swagger 2.0中，通过以下接口即可获取OpenApi的接口描述的json数据。

```
http://localhost:8081/example/v2/api-docs
```

3. 接口声明

3.1 服务定义

3.1.1 基本信息

> 提示

请先注册RepChain账号，并登录到RepChain-Dashboard管理控制台页面。

接口协同所有操作均以此为前提。

一、 总体描述

服务定义功能，是为了规范调用方和服务方的接口实现描述，所定义的内容。 其中定义了存证时和HASH取值时所需要的算法，服务中接口的描述及定义，还提供了接口定义的使用说明。 定义的内容包含基本信息，使用说明，接口定义，应答方接口定义，应答方使用说明。

二、新增服务定义并填写基础信息

- 新增演示示例

三、表单说明

服务名

约束：（必填）（不可更改）(不同版本号情况下服务名可相同)

类型：（字符串输入框）

描述：用于展示的服务名称，由用户定义，同一个服务名称可以定义多个版本号，服务名定义好后不可更改。

接口定义语言

约束：（必填）

类型：（选择）

描述：用于描述接口信息的接口定义语言，当前版本暂时只提供OpenApi。可通过Swagger-UI生成相关的接口描述的JSON数据。

内容HASH算法

约束：（必填）

类型：（选择）

描述：对传输的内容进行Hash取值的算法，当前版本暂时只提供sha256withecdsa算法。

签名算法

约束：（必填）

类型：（选择）

描述：签名算法，对传输内容的Hash值，通过证书及私钥进行数据签名，此处为数据签名需要用到的算法。当前版本只提供sha256withecdsa算法。

版本

约束：（必填）

类型：（字符串输入框）

描述：对接口定义进行版本定义，同一个服务名版本号必须不同。版本号可包含字符串。

3.1.2 使用说明定义

> 提示

请先注册RepChain账号，并登录到RepChain-Dashboard管理控制台页面。

接口协同所有操作均以此为前提。

> 注意

使用说明为编写当前服务的说明，使用说明为Markdown格式。

一、 描述

使用说明使用Markdown格式，在接口定义时，可将具体一些描述内容写入其中。

如：

1. 服务有一些特殊的接口调用方式。
2. 服务如何鉴权，鉴权流程等信息。
3. 定义方联系方式等。
4. OpenAPI接口定义语言无法描述的一些信息。

使用说明可在新建服务定义时填写，也可以在服务定义基本信息保存好以后，点击编辑进行填写。

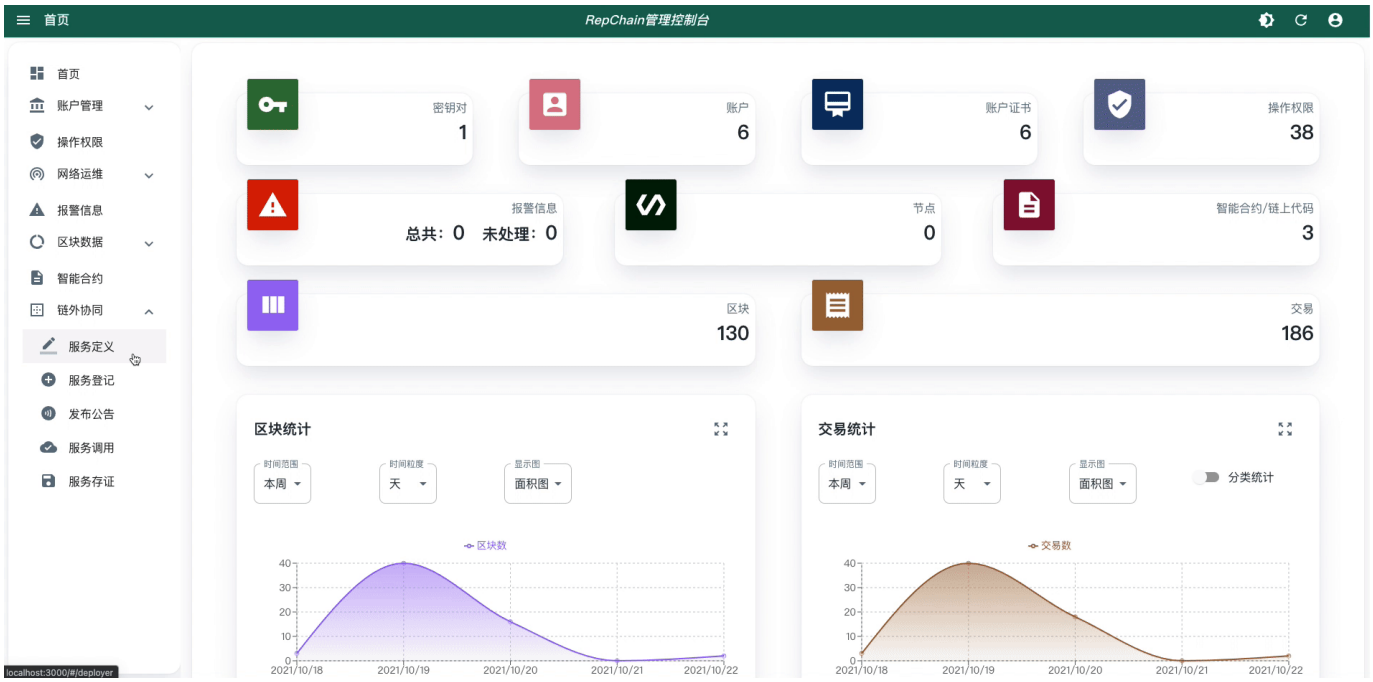
使用说明中包含一些Markdown语法示例，可删除后编写服务定义相关说明内容。

> 注意

如果服务定义为 异步 返回数据，则推荐同时也填写应答方使用说明，两者使用方法一致。

二、使用示例

左侧为输入编辑器，右侧为实时预览展示内容。



3.1.3 接口定义

> 提示

请先注册RepChain账号，并登录到RepChain-Dashboard管理控制台页面。

接口协同所有操作均以此为前提。

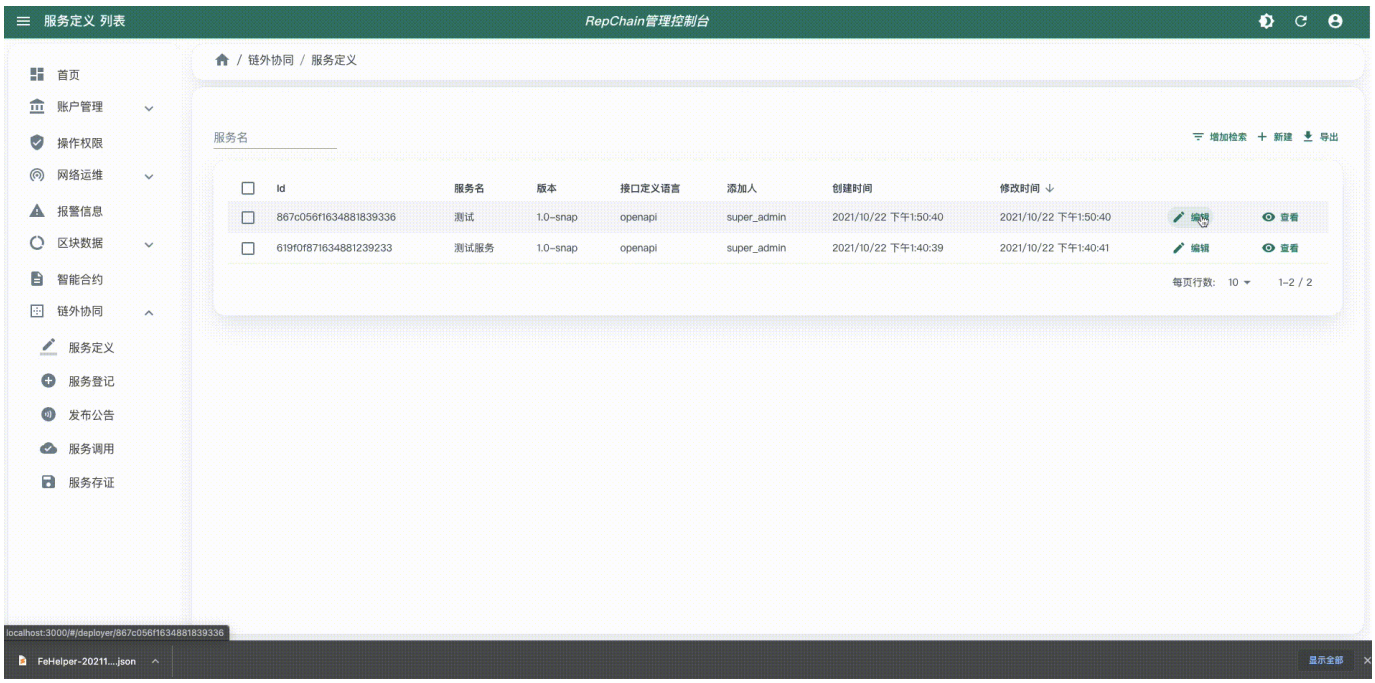
> 注意

接口定义语言使用OpenAPI，详情可点击此处查看。

一、描述

- 接口定义，用规范好的接口定义语言进行接口描述，并展示在界面中，提供其他调用及服务方进行实现。
- 当前版本接口定义语言为OpenAPI，具体描述可点击此处查看。
- 如果项目中使用了Swagger-UI，那么可以通过Swagger-UI的接口获取接口描述的JSON。将JSON复制到接口定义中即可。
- 接口定义可在服务定义创建时编写，也可以在服务定义创建好后通过编辑进行编写。
- 接口定义默认给出JSON示例，编写可根据示例修改或直接替换成已经写好的JSON。
- 如果请求为异步操作时，最好将应答方接口定义一并编写。规范调用方所需要实现的应答接口。

二、使用示例



3.2 服务登记

> 提示
>
> 请先注册RepChain账号，并登录到RepChain-Dashboard管理控制台页面。
>
> 接口协同所有操作均以此为前提。

> 注意
>
> 1. 服务登记分为服务提供方服务登记，服务调用方（调用登记）两种。
> 2. 服务登记需要填写的表单内容完全一致，只是区分提供方与调用方。
> 3. 其中，提供方可发布公告，而调用方只能查看公告，不可发布公告。
> 4. 服务登记时，需要已经在接口定义定义好相关内容，否则无法提交服务登记。

3.2.1 一、描述

服务登记和服务调用，是为了服务方和调用方进行登记存证。

存证信息包含地址，端口号及服务定义，这样调用方即可知道需要调用哪些服务，而服务方可以知道有哪些调用方会调用自己的服务。

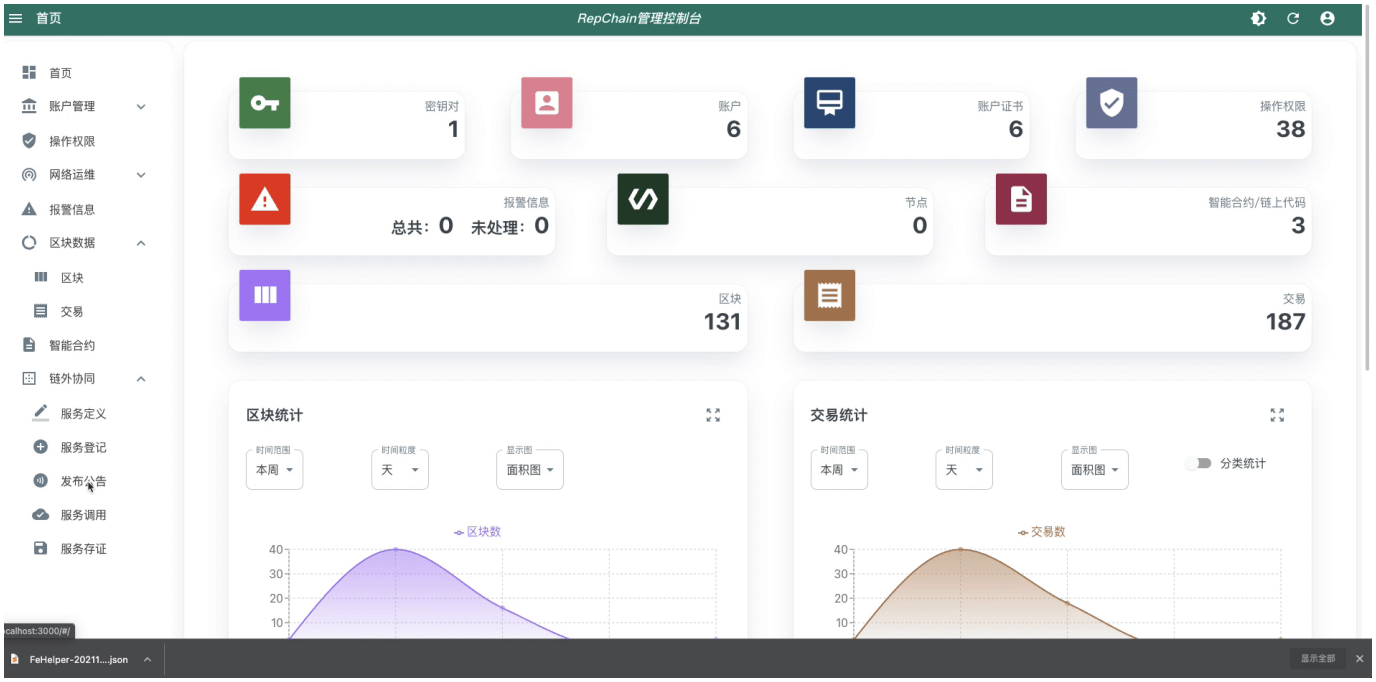
同时，由于有统一的接口定义规范约束，也防止了文档及定义不清晰的问题。服务方和调用方进行登记后，可以清晰的根据接口定义而实现相关接口及业务逻辑。

3.2.2 二、服务登记入口

如图：

- 菜单中服务登记为服务提供方登记界面。
- 菜单中服务调用为服务调用方登记界面。

3.2.3 三、使用示例



3.2.4 四、表单说明

登记名称

约束：（必填）（不可更改）（不可重复）

类型：（字符串输入框）

描述：用于展示的登记名称，由用户定义。

地址

约束：（必填）

类型：（字符串输入框）

描述：登记方的服务器地址，请确保服务方或调用方可访问到此地址。

> 注意
>
> 使用接入中间件时，需要填写接入中间件所在地址。

端口

约束：（必填）（必须为数字）

类型：（数字输入框）

描述：地址对应的端口号。

> 注意
>
> 使用接入中间件时，需要填写配置文件中 `middleware.comServer.port` 所配置的端口号。

服务名

约束：（必填）

类型：（选择框）

描述：选择需要登记的服务名，选择后调用方需要实现接口登记中接口定义的内容。若为异步接口调用，则调用方需要实现应答方接口定义的内容。

3.3 发布公告

> 提示

请先注册RepChain账号，并登录到RepChain-Dashboard管理控制台页面。

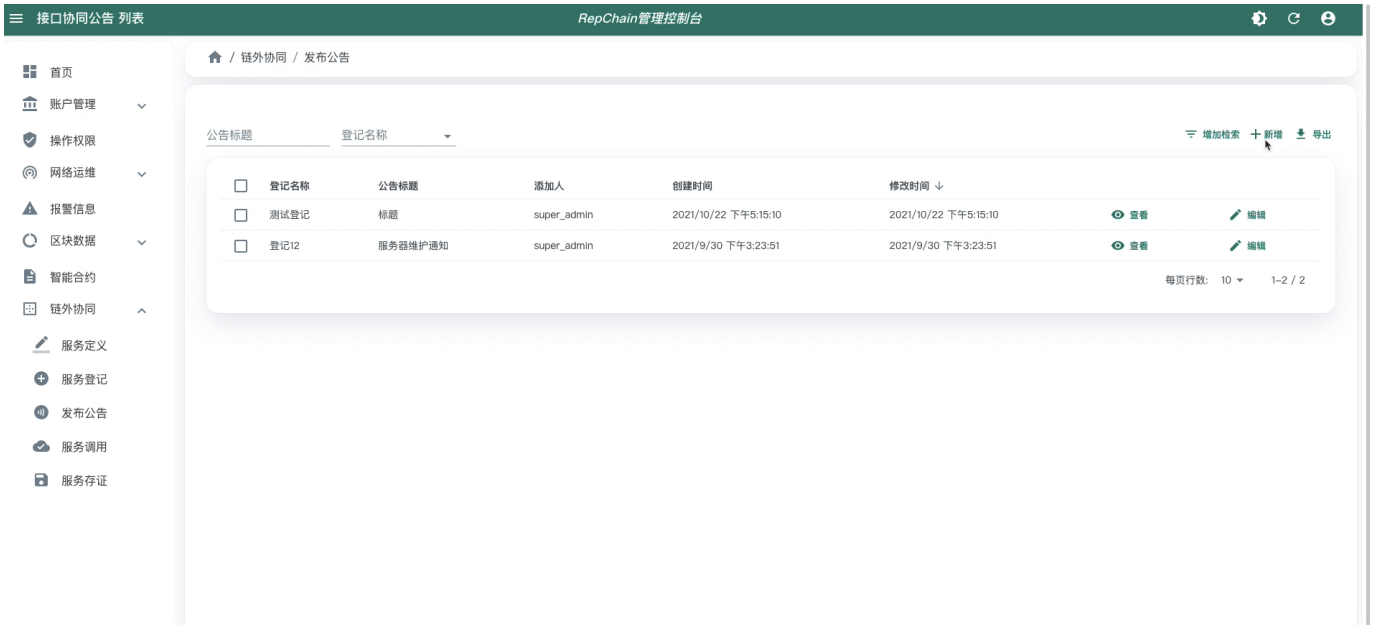
接口协同所有操作均以此为前提。

3.3.1 一、描述

发布公告功能，是为了解决服务方在一些不确定因素下（如机房维护，检修等），可以提前发布公告来写明原因及持续时长等内容，公告详细信息采用Markdown形式进行编写。

公告发布后，调用方可通过公告列表进行查看公告内容。

3.3.2 二、使用示例



3.3.3 三、表单说明

登记名称

约束：（必填）

类型：（选择框）

描述：用于指定发布那个登记的公告

公告标题

约束：（必填）

类型：（字符串输入框）

描述：用于展示在列表页中的公告标题

公告内容

约束：无

类型：（markdown输入框）

描述：用于编写具体的公告内容，Markdown格式。

4. 接口存证

4.1 前言

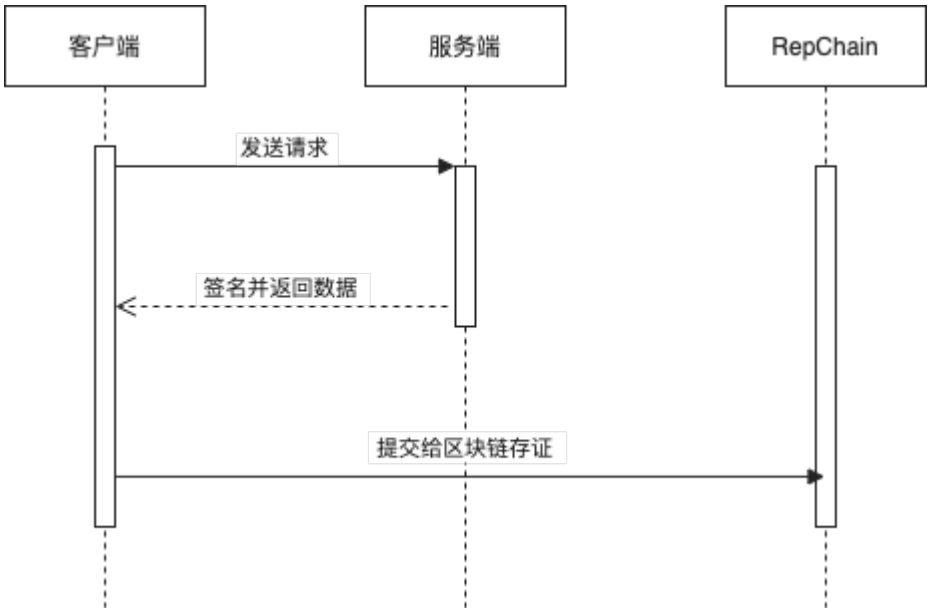
> 提示
>
> 请使用接入中间件进行接口存证，可大大减少用户代码量。

4.1.1 一、前言

接口示例中并未引用一些Web框架，如Spring等，但注释中标注出一些可以使用单例的对象及配置。若要提高代码效率等，需要使用者自己封装成Bean。

4.1.2 二、存证过程-概要

首先，存证过程概要如下，主要为三个步骤：



存证过程主要分为三个步骤：

1. 对接口发起请求；
2. 服务端对数据进行签名并返回数据；
3. 提交给RepChain进行存证。

4.1.3 三、存证过程-详细

对上面概要进行详细说明，便于理解。

1. 对接口发起请求

- 构建接口请求参数
- 对接口请求参数进行数据签名
- 构建区块链存证Header（请求头）

- 发送请求

2. 服务端对数据签名并返回数据

- 服务端接收到请求后进行业务处理，获取返回数据
- 对返回数据进行数据签名
- 将返回结果和数据签名结果一同返回给客户端

3. 提交给RepChain存证

- 获取服务端返回数据
- 构建需要存证的交易对象
- 提交存证的交易对象给RepChain
- 完成接口调用

4.1.4 四、验证权限

权限验证各个系统会存在不同的方式进行验证，此处提供一个基于RepChain身份认证的权限认证。

通过对数据签名，调用方证书进行权限认证，认证通过则调用方拥有该接口调用权限。

此验证方式前提为调用方和服务方需要进行个人证书的交换，使双方都持有对方的证书，以便对签名数据进行数据验签。

4.2 HTTP请求存证

4.2.1 同步请求

HTTP单次同步请求

> 提示
>
> 请使用接入中间件进行接口存证，可大大减少用户代码量。

一、概述

同步请求，即为客户端请求一次服务端就返回数据，没有第二次请求。

此类情况则只完成一次存证过程即可。

二、请求步骤

请求方

1. 构建请求数据
2. 对数据进行数据签名
3. 构建存证请求头
4. 发送请求数据到服务端，并接收返回结果
5. 构建交易对象
6. 提交给RepChain区块链

服务方

1. 获取请求参数及请求头信息
2. 数据验签判断权限
3. 执行业务操作获取返回数据
4. 对返回数据进行数据签名
5. 返回数据及签名信息给请求方

三、示例

> 提示
>
> 使用接入中间件进行HTTP单次同步请求。

多次同步请求

> 提示
>
> 请使用接入中间件进行接口存证，可大大减少用户代码量。

一、概述

当数据量过大，一次请求无法获取全部所需数据时，需要多次请求服务方接口。

此类情况每一次请求都需要进行一次存证。

二、请求步骤

请求方

与单次请求存证过程相似，每一次请求都需要执行单次请求的执行步骤。

服务方

服务方步骤，与HTTP单次同步服务方步骤相同。

可查看点击此处进行查看。

三、示例

> 提示
>
> 使用接入中间件进行HTTP多次同步请求。

4.2.2 异步请求

HTTP单次异步请求

一、概述

> 提示
>
> 请使用接入中间件进行接口存证，可大大减少用户代码量。

异步请求，当调用服务方数据时，服务方需要有流程审批等原因，无法同步返回数据。此类情况需要在接口定义时，定义应答方接口规范，并让调用方进行接口实现。

此类情况则只完成两次存证过程，调用方请求时需要存证，服务方应答调用方接口时同样也需要进行存证。

二、请求步骤

请求方

由于是异步请求，请求方需要发送请求，并提供应答方调用接口，用于接收应答数据。

请求：

1. 构建请求数据
2. 对数据进行数据签名
3. 构建存证请求头
4. 发送请求数据到服务端，并接收返回结果
5. 构建交易对象
6. 提交给RepChain区块链

接收应答：

1. 获取请求参数及请求头信息
2. 数据验签判断权限
3. 执行业务操作并构建返回数据结果
4. 对返回数据进行数据签名
5. 返回数据及签名信息给请求方

服务方

接收请求：

1. 获取请求参数及请求头信息
2. 数据验签判断权限
3. 将Header对象进行暂存，并构建已经收到请求的返回数据
4. 对返回数据进行数据签名
5. 返回数据及签名信息给请求方

应答调用方：

1. 从暂存数据中过去调用方发送的Header
2. 构建返回结果数据
3. 对数据进行数据签名
4. 构建存证请求头
5. 发送请求数据到应答接口，并接收返回结果
6. 构建交易对象
7. 提交给RepChain区块链

三、示例

> 提示

使用接入中间件进行HTTP单次异步请求。

HTTP多次异步请求

一、概述

> 提示
>
> 推荐使用接入中间件进行接口存证，可大大减少用户代码量。

异步请求，当调用服务方数据时，服务方需要有流程审批等原因，无法同步返回数据。此类情况需要在接口定义时，定义应答方接口规范，并让调用方进行接口实现。

此示例为最复杂的流程，即调用方进行多次异步请求，服务方进行多次异步请求应答。

由于是多次请求，代表调用方在一个方法内会进行多次请求。服务方将多次请求的数据进行持久化，然后执行业务流程。执行完成后，将调用方需要的数据通过多次应答，发送给调用方的应答接口。

此类情况则需要完成多次存证过程，调用方请求时需要进行存证，服务方应答调用方接口后同样需要存证。

二、请求步骤

请求方

由于是异步请求，请求方需要发送请求，并提供应答方调用接口，用于接收应答数据。

请求：

执行多次单次异步请求步骤，每次请求都进行存证

接收应答：

1. 获取请求参数及请求头信息
2. 数据验签判断权限
3. 执行业务操作并构建返回数据结果
4. 对返回数据进行数据签名
5. 返回数据及签名信息给请求方

服务方

接收请求：

1. 获取请求参数及请求头信息
2. 数据验签判断权限
3. 将Header对象进行暂存，并构建已经收到请求的返回数据
4. 对返回数据进行数据签名
5. 返回数据及签名信息给请求方

应答调用方：

执行多次单次应答调用步骤，每次应答都进行存证

三、示例

> 提示
>
> 使用接入中间件进行HTTP多次异步请求。

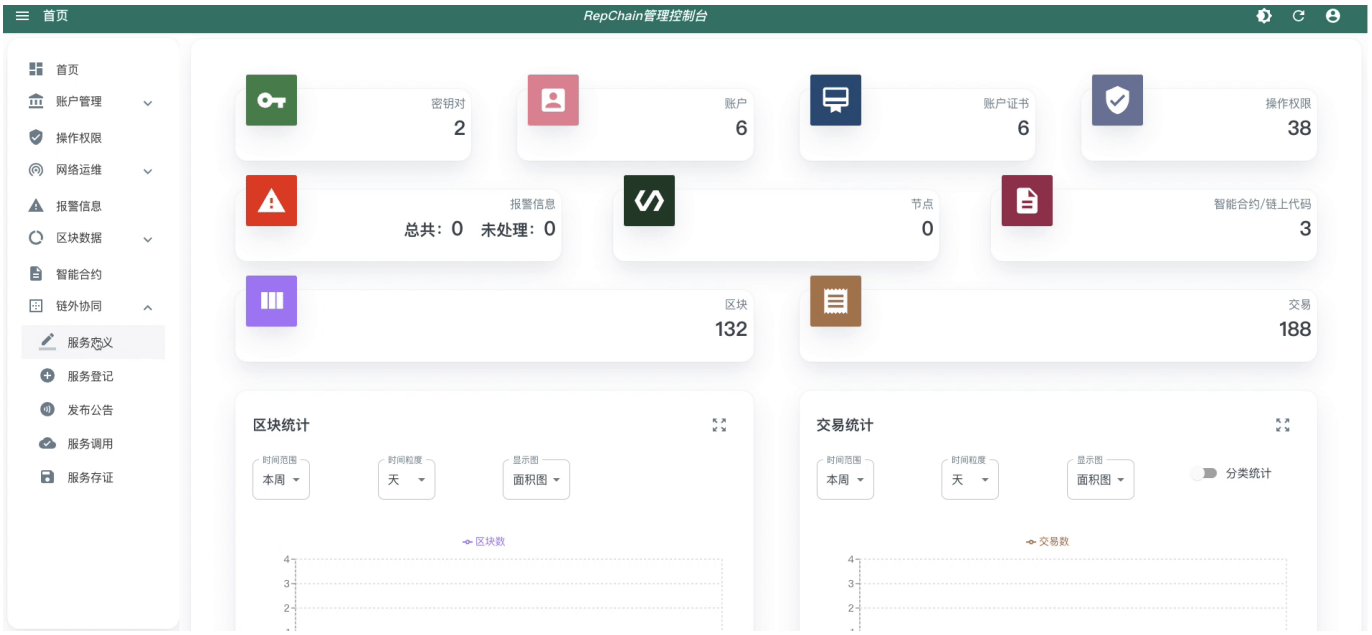
4.3 查看存证

> 提示
>
> 请先注册RepChain账号，并登录到RepChain-Dashboard管理控制台页面。
>
> 接口协同所有操作均以此为前提。

4.3.1 一、描述

查看存证，用于查看接口调用时，存证于区块链上的存证信息。

4.3.2 二、使用示例



5. 接入中间件

5.1 2.0

5.1.1 前言

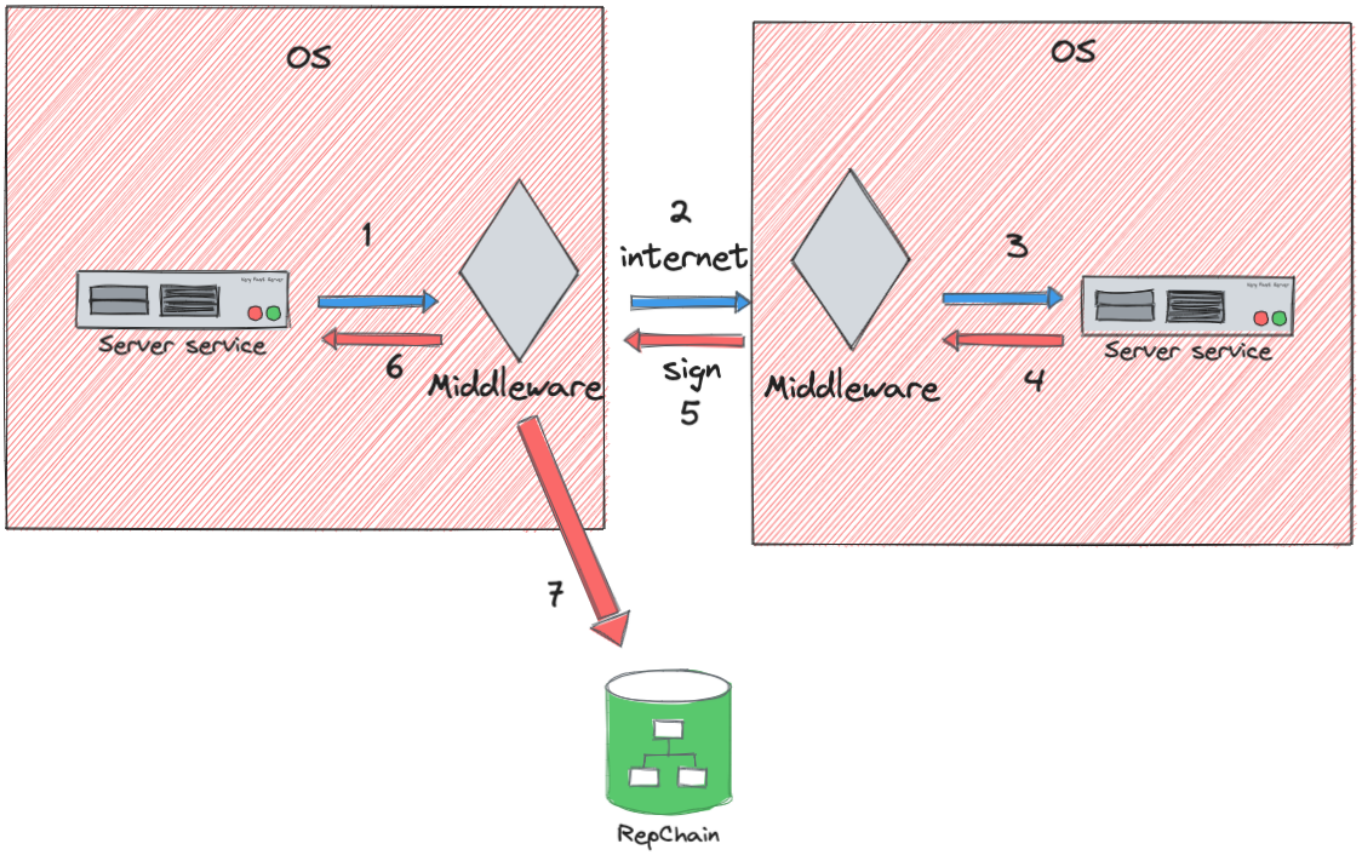
一、说明

1. 接入中间件是为了帮助用户，解决接口协同时的存证、签名问题。
2. 接入中间件简化了数据存证、证书校验、数据签名等工作。
3. 同时使用gRPC协议进行数据传输，提高了数据传输效率。
4. 提供客户端连接中间件，减少了用户调用中间件时的代码量。
5. 封装文件传输、文件下载功能，进行文件传输时完全交由中间件操作，不用再去编写文件流的相关代码。

> ⚠注意
>
> 每一个中间件对应一个宿主机，对应一套私钥和证书，中间件需要和宿主机部署在同一个操作系统中。

二、中间件基本原理

中间件基本原理如下：



提示

首先，中间件和服务部署在同一个操作系统中。

中间件基本原理：

1. 请求方发起请求给中间件；
2. 中间件收到请求数据进行签名，并将数据转发给服务方中间件；
3. 服务方中间件获取请求数据校验签名，如果校验通过则，将请求数据发送给服务方接口；
4. 服务方返回数据给服务方中间件；
5. 服务方中间件对数据进行签名，签名后将数据返回给请求方中间件；
6. 请求方中间件返回数据给请求方；
7. 请求方中间件对数据进行签名，提交给RepChain进行存证。

5.1.2 快速开始

一、安装JDK

下载jdk8，安装并根据自己的操作系统配置好环境变量。

推荐下载 zulu-jdk-1.8

二、下载中间件

根据不同系统下载中间件的压缩包

1. WINDOWS

下载 **Zip** 文件 https://gitee.com/BTAJL/api-coord/attach_files/1104691/download/Interface-Cooperation-Middleware-2.0.0-RC2.0.0.zip

2. LINUX

- 使用 **wget** 下载 **tar**

```
wget https://gitee.com/BTAJL/api-coord/attach_files/1104690/download/Interface-Cooperation-Middleware-2.0.0-RC2.0.0.tar
```

- 使用远程传输 **tar**

可以先下载 **tar** 到本地电脑上，然后通过**SSH**工具将tar包传输到服务器中。

3. MAC

下载 **Zip** 文件或下载 **Tar** 文件。

三、解压

1. WINDOWS

使用解压工具将 **Zip** 压缩包解压缩。

2. LINUX

```
tar -xvf Interface-Cooperation-Middleware-2.0.0-RC2.0.0.tar
```

3. MAC

解压方式使用上述两种任选其一即可。

四、修改配置文件

- 打开文件 `conf/application-middle.yml`
- 填写 RepChain 相关地址

```
# repchain配置文件
repchain:
  # repchain 地址
  host: 192.168.2.76:8081
  ...
```

* 填写你的服务器地址

由于中间件和宿主服务需要部署在同一个操作系统中，所以host填写127.0.0.1即可

```
...
# 中间件用于访问宿主服务的客户端配置
recClient:
  # 服务地址
  host: 127.0.0.1
  # 服务端口号
  port: 8080
  ...
```

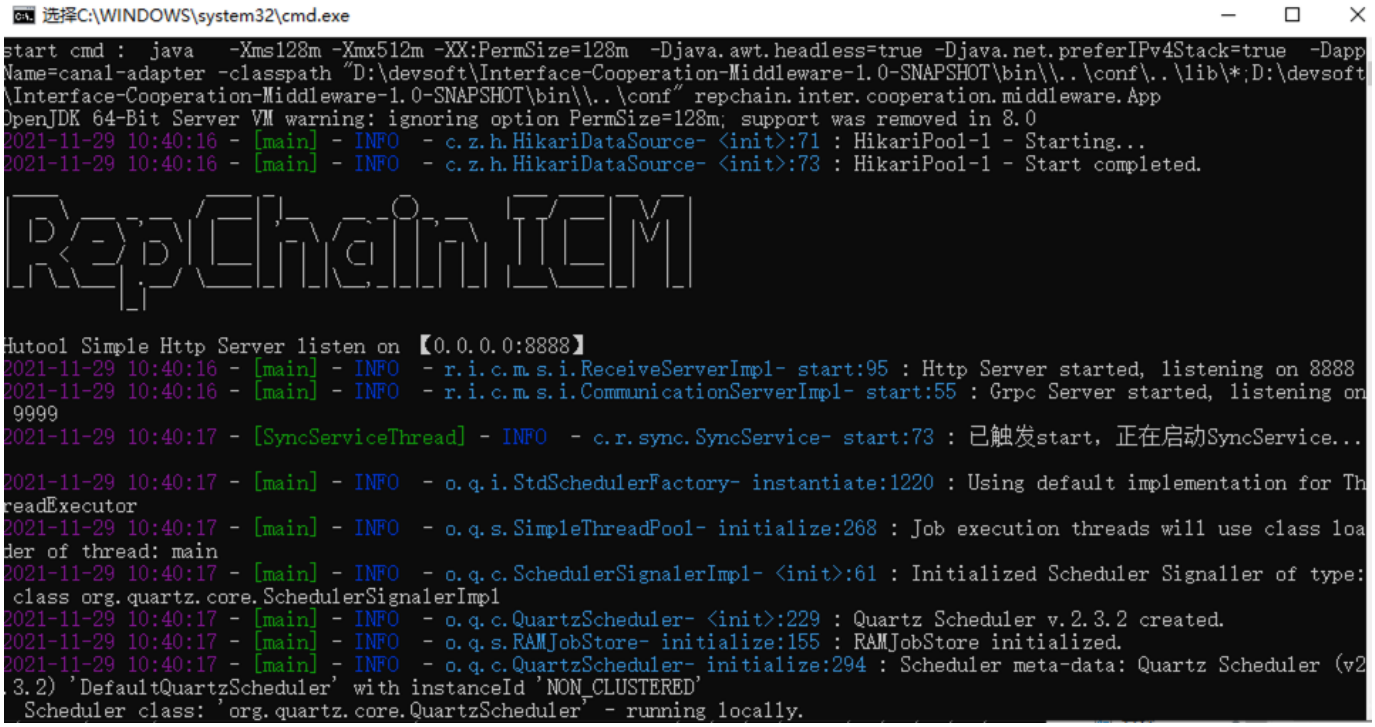
五、启动中间件

1. WINDOWS

- 终端执行启动命令

```
bin\start.cmd
```

- 或双击启动脚本 `bin/start.cmd`



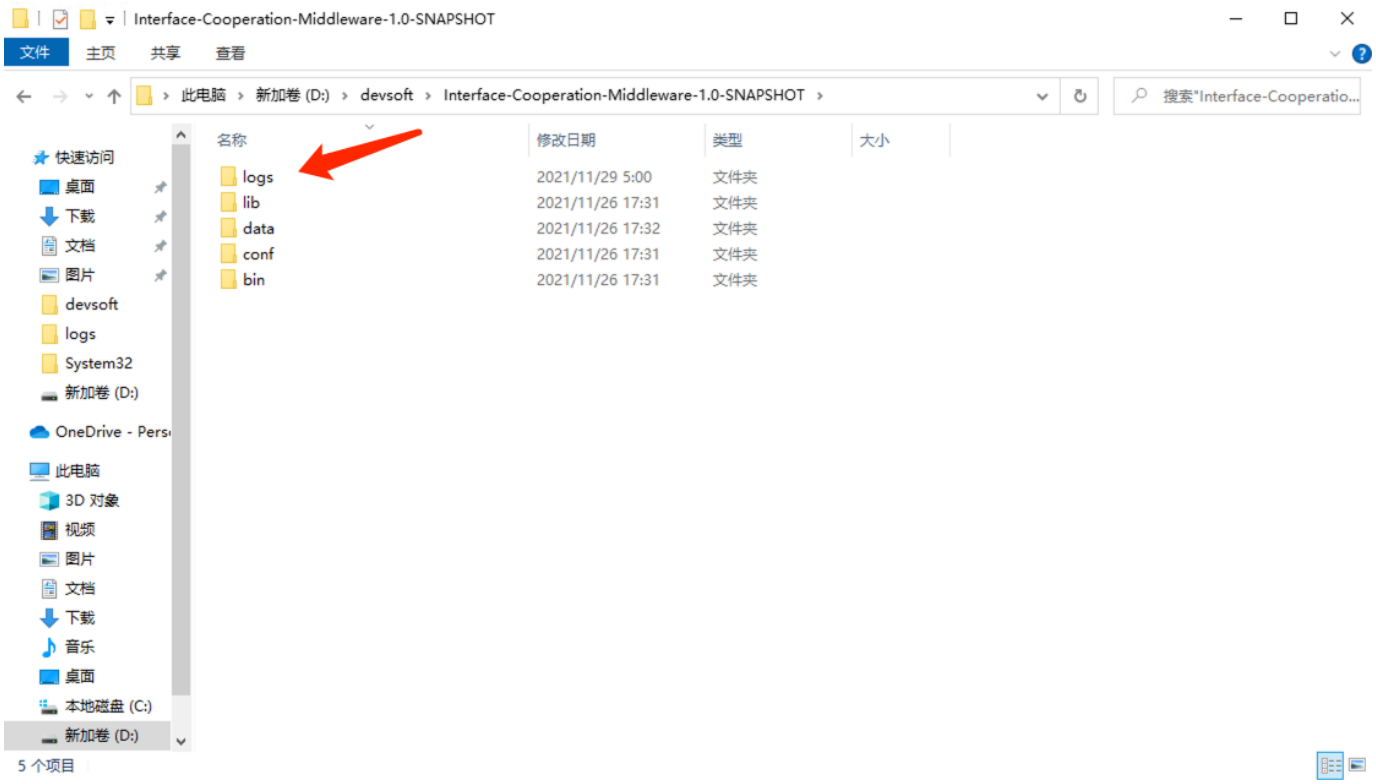
2. LINUX 或 MAC

- 终端执行启动脚本 `bin/startup.sh`

```
sh bin/startup.sh
```

六、查看日志

中间件启动后会创建 `logs` 文件夹，文件下会产生相应的日志文件。



七、停止中间件

1. WINDOWS

- 关闭终端窗口或者在终端执行 `Ctrl + C`

2. LINUX 或 MAC

- 终端执行停止脚本 `bin/shutdown.sh`

```
sh bin/shutdown.sh
```

5.1.3 用户指南

前期准备

一、安装JDK

需要在部署中间件的服务器及开发的机器上都进行安装。

参考准备工作中的Java安装

二、定义接口

服务提供方需要提供接口定义标准。

具体使用方式请参考 服务定义

三、服务登记及服务调用登记

服务登记为服务提供方进行服务登记，登记服务方的服务器地址及端口号等信息。

调用方登记同服务方法登记相同，为调用方登记服务器地址及端口号等信息。

具体登记步骤请参考 服务登记

四、部署中间件

> 提示

1. 此处部署中间件需要服务 提供方 及服务 调用方 都进行 部署 ，所有的接口协同数据传输及存证均由 中间件 进行完成。
2. 部署前，服务提供方和服务调用方需要通过线下的方式，进行证书交换。
3. 双方交换好证书后，需要将证书写入配置文件中 `repchain-services` 下，同时将双方的ID也写入其中。

1. 下载中间件，参考下载中间件

2. 解压缩压缩包，参考解压

3. 修改配置

> 注意

修改配置请按照下面配置说明进行修改，下面所述说明均为 必须 修改的选项。

配置文件中的其他的默认配置可以根据自己的需求进行修改，但是不要删除任何默认配置。

此处说明需要修改的几个 必要 的配置:

- RepChain相关配置都需要进行修改，具体如下，详细说明可参考此处。

```
# repchain配置文件
repchain:
  # repchain 地址
  host: 192.168.2.76:8081
  # 用户唯一标识
  creditCode: CGZL7XOwwa
  # 证书名称
  certName: test
  # 证书
  cert: -----BEGIN CERTIFICATE-----
    MIIBIDCBx6ADAgECAgQDNHtwMAoGCCqGSM49BAMCMA8xDTALBgNVBAMMBFRFU1Qw
    IhgPMjAyMTEwMjYwMTE0MDVaGA85OTk5MTIzMTIzNTk1OVowDzENMAsGA1UEAwwE
    VEVTVDBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABO3voNU9OV4EoDyBHQcD+az/
    E5TB1i5fRf+tTaNgjCWr6jErt/Ce21b18uwTuZXx38nvA5126S1MEpCAuuJSFymj
    DTALMAkGA1UdIwQCMAAwCgYIKoZIzj0EAwIDSAAwRQIhANjTlEKVu0O5lgEloHrh
    ZLcNMHH9S2sVzasTK5OJuXwGAiBGi2GL4W4XlWzjl94swApuv2bb7rbo2huBIzhc
    zZXMcw==
    -----END CERTIFICATE-----
  # 加密私钥
  privateKey: -----BEGIN ENCRYPTED PRIVATE KEY-----
    MIHUMD8GCSqGSIb3DQEFDTAyMBoGCSqGSIb3DQEFDDANBAi0q5jmLN61vAIBZDAU
    BggqhkiG9w0DBwQITUCFquFjnwEEgZAbZouweJPtR6zRaDhULbyzF3uaOnMErtx3
    kIguDP/qeb3wVoTW8kFuOw1jdrY5WyHGCngfpM5br6AUaEqIlBnA7rS3KZfhsxxq
    yLQcaySsfUowUE61oStDJtDgHMJE0dbxuYXjnzSQrnp9AnnX65uwUeDE6c+i7DnW
    pWoroVTXK/dQb4vWNmxmacQ6G8jmW0s=
    -----END ENCRYPTED PRIVATE KEY-----
  # 私钥设置的密码
  password: 123456
  # 签名证书对应的请求方id和服务方id可在dashboard查看
  services:
    # 用户自定义id，不能重复，必填
    - serviceId: 1
      # 调用方ID
      e_from: bb743c821635920387678
      # 调用方证书
      from_cert: -----BEGIN CERTIFICATE-----
        MIIBIDCBx6ADAgECAgQDNHtwMAoGCCqGSM49BAMCMA8xDTALBgNVBAMMBFRFU1Qw
        IhgPMjAyMTEwMjYwMTE0MDVaGA85OTk5MTIzMTIzNTk1OVowDzENMAsGA1UEAwwE
        VEVTVDBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABO3voNU9OV4EoDyBHQcD+az/
        E5TB1i5fRf+tTaNgjCWr6jErt/Ce21b18uwTuZXx38nvA5126S1MEpCAuuJSFymj
        DTALMAkGA1UdIwQCMAAwCgYIKoZIzj0EAwIDSAAwRQIhANjTlEKVu0O5lgEloHrh
        ZLcNMHH9S2sVzasTK5OJuXwGAiBGi2GL4W4XlWzjl94swApuv2bb7rbo2huBIzhc
        zZXMcw==
        -----END CERTIFICATE-----
      # 服务提供方id
      e_to: 7b5c66df1635920342945
      # 提供方证书
      to_cert: -----BEGIN CERTIFICATE-----
        MIIBIDCBx6ADAgECAgQDNHtwMAoGCCqGSM49BAMCMA8xDTALBgNVBAMMBFRFU1Qw
        IhgPMjAyMTEwMjYwMTE0MDVaGA85OTk5MTIzMTIzNTk1OVowDzENMAsGA1UEAwwE
        VEVTVDBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABO3voNU9OV4EoDyBHQcD+az/
        E5TB1i5fRf+tTaNgjCWr6jErt/Ce21b18uwTuZXx38nvA5126S1MEpCAuuJSFymj
        DTALMAkGA1UdIwQCMAAwCgYIKoZIzj0EAwIDSAAwRQIhANjTlEKVu0O5lgEloHrh
        ZLcNMHH9S2sVzasTK5OJuXwGAiBGi2GL4W4XlWzjl94swApuv2bb7rbo2huBIzhc
        zZXMcw==
        -----END CERTIFICATE-----
```

- Middleware配置，以下配置请根据服务登记或服务调用相关内容进行修改，详细说明可参考此处。

```
## 中间件配置
middleware:
  # 用来访问宿主机的服务接口的客户端
  recClient:
    host: 127.0.0.1
    port: 80
  # 提供给宿主机调用中间件的服务接口配置
  recServer:
    # 端口
    port: 8888
  # 提供给其他中间件访问的grpc服务端口，此处需要和服务登记或服务调用中提交的接口保持一致
  comServer:
    # 端口
    port: 8080
```

4. 启动中间件，参考启动停止重启

五、引入客户端

发布接入中间件客户端到本地仓库并引入代码中，参考此处。

同时，客户端也提供 **jar** 方式引入，点击 此处 下载jar包。

目录结构

一、文件结构

```
├── logs                                # 日志文件夹
├── lib                                 # 依赖包及jar包文件夹
├── data                                # 持久化数据文件夹
├── conf                                # 配置相关文件夹
├── bin                                 # 启动、停止等相关脚本文件夹
└── file                                # 传输及接收文件的文件夹，包含临时文件及持久化文件
```

二、文件夹说明

logs

用于存放日志的文件夹，由中间件首次启动时创建，可在配置文件中指定为其他文件夹。

lib

存放中间件编译后的jar包及相关依赖jar包的文件夹。

data

用于数据持久化文件夹，文件夹中包含sqlite文件，用于存入相关的元数据。

如果遇到需要迁移的情况，可以将此文件夹直接替换到迁移目录。

> 注意
>
> 只有当配置中使用默认sqlite持久化方式时，此文件夹才会保存中间件持久化数据。
>
> 如果将数据源更改为其他配置，如Mysql等，则此持久化元数据不会保存到此文件夹中。

conf

用于存放中间件配置文件，文件夹中包含一个yml格式的配置文件。

> 注意
>
> 修改配置文件后需要重启中间件才会生效。

bin

用于存放脚本，包含启动脚本、关闭脚本、重启脚本等。

file

用于存放中间件产生的临时文件，及中间传输过程中根据用户设置持久化的文件。

文件传输时会自动创建此目录，用户可以在配置文件中修改此目录的路径。

此处为默认路径。

启动停止重启

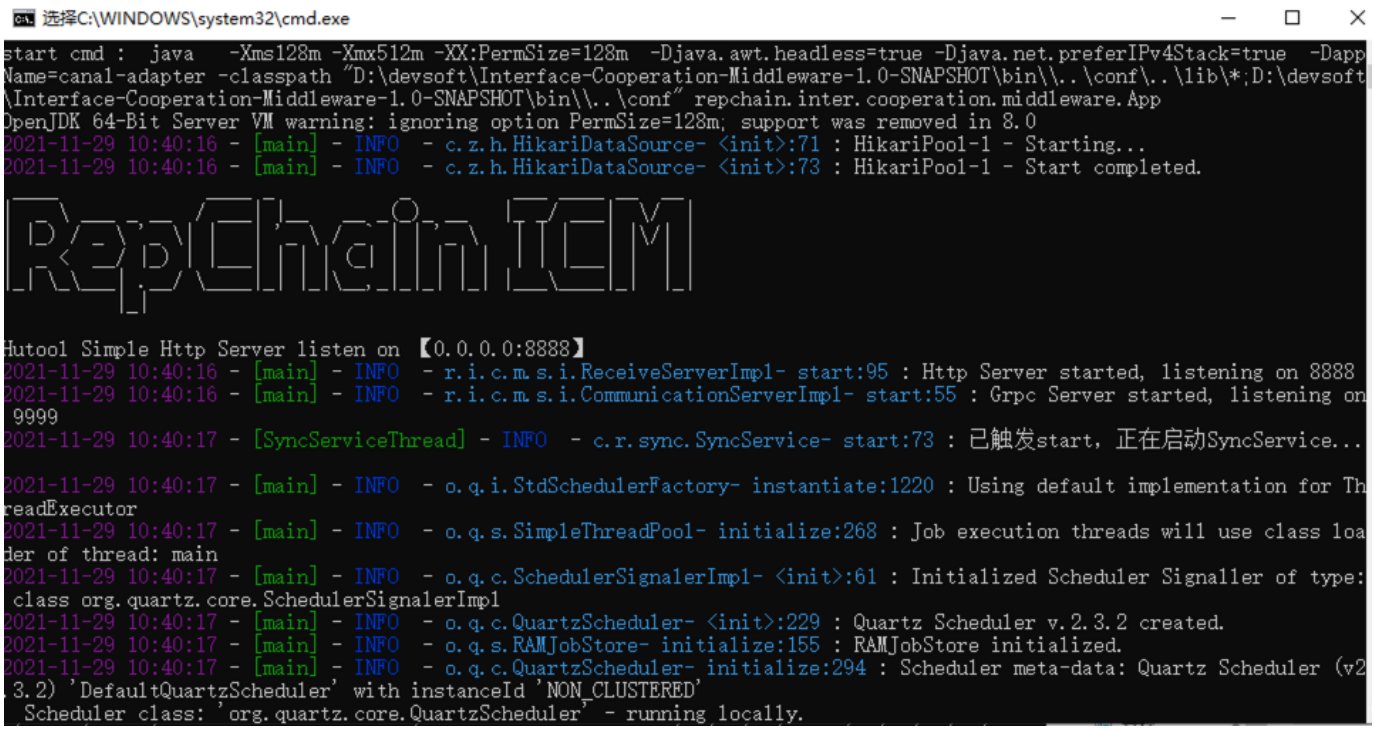
一、启动

1. Windows

- 终端执行启动命令

```
bin\start.cmd
```

- 或双击启动脚本 `bin/start.cmd`



2. Linux 或 Mac

- 终端执行启动脚本 `bin/startup.sh`

```
sh bin/startup.sh
```

二、停止

1. Windows

- 关闭终端窗口或者在终端执行 `Ctrl + C`

2. Linux 或 Mac

- 终端执行停止脚本 `bin/shutdown.sh`

```
sh bin/shutdown.sh
```

三、重启

1. Windows

- 暂不支持自动重启，请先停止中间件服务后再启动。

2. Linux 或 Mac

- 终端执行重启脚本 `bin/restart.sh`

```
sh bin/restart.sh
```

配置文件

> 注意
>
> 以下所有数据均为示例数据，请用户不要在生产环境中使用以下数据。
>
> 特别注意，一定 不要 使用示例数据中的证书及私钥！！！

一、说明

配置文件位于中间件 `conf` 目录下 `application-middle.yml` 。

二、配置说明

1. repchain

repchain及接口存证相关配置

chainNetworkId

- 描述：组网Id
- 示例：identity-net
- 约束： 必填

host

- 描述：配置中间件连接RepChain的地址
- 示例：192.168.2.76:8081
- 约束： 必填

height

- 示例：0
- 描述：初始同步的区块高度，与已经持久化的区块高度做对比，取最大值。例：如果此处配置0，已经持久化的区块高度为2，则从2开始同步区块。
- 约束： 必填

creditCode

- 示例：CGZL7XOwwa
- 描述：用户在区块链上的唯一标识。
- 约束： 必填

certName

- 示例：cert
- 描述：用户用于接口存证的证书名称。
- 约束： 必填

cert

- 示例：

```
-----BEGIN CERTIFICATE-----
MIIBIDCBx6ADAgECAgQDNHtwMAoGCCqGSM49BAMCMA8xDTALBgNVBAMMBFRFU1Qw
IhgPMjAyMTEwMjYwMTE0MDVaGA85OTk5MTIzMTIzNTk1OVowDzENMAsGA1UEAwwE
VEVTVDBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABO3voNU9OV4EoDyBHQcD+az/
E5TB1i5fRf+tTaNgjCWr6jErt/Ce21b18uwTuZXx38nvA5126S1MEpCAuuJSFymj
DTALMAkGA1UdIwQCMAAwCgYIKoZIzj0EAwIDSAAwRQIhANjTlEKVu0O5lgEloHrh
ZLcNMHH9S2sVzasTK5OJuXwGAiBGi2GL4W4XlWzjl94swApuv2bb7rbo2huBIzhc
zZXMcw==
-----END CERTIFICATE-----
```

- 描述：用户用于接口存证的证书。
- 约束： 必填

privateKey

- 示例：

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIHUMD8GCSqGSIb3DQEFDTAyMBoGCSqGSIb3DQEFDDANBAi0q5jmLN61vAIBZDAU
BggqhkiG9w0DBwQITUCFquFjnwEEgZAbZouweJPtR6zRaDhULbyzF3uaOnMErtx3
kIguDP/qeb3wVoTW8kFuOw1jdrY5WyHGCngfpM5br6AUaEqIlBnA7rS3KZfhsxxq
yLQcaySsfUowUE61oStDJtDgHMJE0dbxuYXjnzSQrnp9AnnX65uwUeDE6c+i7DnW
pWoroVTXK/dQb4vWNmxmacQ6G8jmW0s=
-----END ENCRYPTED PRIVATE KEY-----
```

- 描述：用户用于接口存证的私钥，用户的私钥请保存好，不要泄露给任何人。
- 约束： 必填

password

- 示例：123456
- 描述：用户设置的私钥的密码
- 约束： 必填

services

- 描述：用户用来设置请求方或应答方证书的配置，可设置多个，每个请求方对应的应答方都需要再此配置。

serviceId

- 示例：1
- 描述：用户自定义ID，使用客户端请求中间件时需要提供此ID来定位需要请求的服务。
- 约束： 必填

e_from

- 示例：bb743c821635920387678
- 描述：服务调用方ID，可在RepChain管理控制台-链外协同-服务调用中Id列中获取。
- 约束： 必填

from_cert

- 示例：

```
-----BEGIN CERTIFICATE-----
   MIIBIDCBx6ADAgECAgQDNHtwMAoGCCqGSM49BAMCMA8xDTALBgNVBAMMBFRFU1Qw
   IhgPMjAyMTEwMjYwMTE0MDVaGA85OTk5MTIzMTIzNTk1OVowDzENMAsGA1UEAwwE
   VEVTVDBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABO3voNU9OV4EoDyBHQcD+az/
   E5TB1i5fRf+tTaNgjCWr6jErt/Ce21b18uwTuZXx38nvA5l26S1MEpCAuuJSFymj
   DTALMAkGA1UdIwQCMAAwCgYIKoZIzj0EAwIDSAAwRQIhANjTlEKVu0O5lgEloHrh
   ZLcNMHH9S2sVzasTK5OJuXwGAiBGi2GL4W4XlWzjl94swApuv2bb7rbo2huBIzhc
   zZXMcw==
-----END CERTIFICATE-----
```

- 描述： 服务调用方证书，用于请求接口校验身份时使用。需要调用方提供。
- 约束： 必填

e_to

- 示例：7b5c66df1635920342945
- 描述：服务提供方Id，可在RepChain管理控制台-链外协同-服务登记中Id列中获取。
- 约束： 必填

to_cert

- 示例：

```
-----BEGIN CERTIFICATE-----
   MIIBIDCBx6ADAgECAgQDNHtwMAoGCCqGSM49BAMCMA8xDTALBgNVBAMMBFRFU1Qw
   IhgPMjAyMTEwMjYwMTE0MDVaGA85OTk5MTIzMTIzNTk1OVowDzENMAsGA1UEAwwE
   VEVTVDBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABO3voNU9OV4EoDyBHQcD+az/
   E5TB1i5fRf+tTaNgjCWr6jErt/Ce21b18uwTuZXx38nvA5l26S1MEpCAuuJSFymj
   DTALMAkGA1UdIwQCMAAwCgYIKoZIzj0EAwIDSAAwRQIhANjTlEKVu0O5lgEloHrh
   ZLcNMHH9S2sVzasTK5OJuXwGAiBGi2GL4W4XlWzjl94swApuv2bb7rbo2huBIzhc
   zZXMcw==
-----END CERTIFICATE-----
```

- 描述：服务提供方证书，用于异步请求应答接口时校验身份使用。需要服务提供方提供。
- 约束： 必填

2. middleware

中间件相关配置。

file

- 描述：文件配置

bufferSize

- 示例：8192
- 描述：文件缓冲大小，如果文件传输速度较慢，可以增大此参数，此参数推荐1024的倍数
- 约束： 必填

temp

- 示例：/file/tmp
- 描述：临时文件目录，此处必须填写绝对路径，默认路径为中间目录下 `file/tmp`
- 约束： 非必填

backupPath

- 示例：/file/backup
- 描述：持久化备份文件目录，此处必须填写绝对路径，默认路径为中间目录下 `file/backup`
- 约束： 非必填

log

- 描述： 日志相关配置

path

- 示例：/data/logs
- 描述：日志保存路径
- 约束： 非必填

recClient

- 描述：中间件访问用户的服务的客户端

host

- 示例：127.0.0.1
- 描述：提供服务或应答的地址，由于和宿主服务部署在同一个服务器上此处填写127.0.0.1
- 约束： 必填

port

- 示例：80
- 描述：提供服务或应答的端口
- 约束： 必填

protocol

- 示例：http
- 描述：协议，当前版本暂时只支持http
- 约束： 必填

timeout

- 示例：5000
- 描述：访问服务的超时时间，单位为毫秒
- 约束： 必填

recServer

- 描述：宿主服务调用中间件服务的相关配置，此处提供的为http服务。

port

- 示例：8888
- 描述：提供的http服务的端口
- 约束： 必填

corePoolSize

- 示例：50
- 描述：初始线程池大小
- 约束： 必填

maxPoolSize

- 示例：200
- 描述：最大线程池大小
- 约束： 必填

workQueue

- 示例：100
- 描述：队列，用来存储未执行的线程
- 约束： 必填

comClient

- 描述：中间件访问其他中间件的客户端。

timeout

- 示例：5000
- 描述：超时时间（毫秒）
- 约束： 必填

maxTotal

- 示例：8
- 描述：池中最大连接数
- 约束： 必填

minIdle

- 示例：1
- 描述：最少的空闲连接数
- 约束： 必填

maxIdle

- 示例：8
- 描述：最多的空闲连接数
- 约束：必填

maxWaitMillis

- 示例：-1
- 描述：当连接池资源耗尽时, 调用者最大阻塞的时间, 超时时抛出异常 单位:毫秒数
- 约束：必填

lifo

- 示例：true
- 描述：连接池存放池化对象方式, true放在空闲队列最前面, false放在空闲队列最后
- 约束：必填

minEvictableIdleTimeMillis

- 示例：1800000
- 描述：连接空闲的最小时间, 达到此值后空闲连接可能会被移除, 默认即为30分钟
- 约束：必填

blockWhenExhausted

- 示例：true
- 描述：连接耗尽时是否阻塞, 默认为true
- 约束：必填

comServer

- 描述：中间件提供其他中间件访问的服务，默认使用Grpc协议。

port

- 示例：8888
- 描述：提供的http服务的端口
- 约束： 必填

corePoolSize

- 示例：50
- 描述：初始线程池大小
- 约束： 必填

maxPoolSize

- 示例：200
- 描述：最大线程池大小
- 约束： 必填

workQueue

- 示例：100
- 描述：队列，用来存储未执行的线程
- 约束： 必填

3. datasource

url

- 示例：jdbc:sqlite:data/repchain_mid.db
- 描述：此处可以填写sqlite或mysql，其他数据库暂未支持，sqlite默认填写为中间件文件夹相对路径。
- 约束： 必填

user

- 示例：test
- 描述：数据库连接用户名
- 约束： 必填

pass

- 示例：123456
- 描述：数据库连接密码
- 约束： 必填

driver

- 示例：org.sqlite.JDBC
- 描述：数据库驱动
- 约束： 必填

showSql

- 示例：true
- 描述：是否在日志中显示执行的SQL
- 约束： 必填

formatSql

- 示例：true
- 描述：是否格式化显示的SQL
- 约束： 必填

showParams

- 示例：true
- 描述：是否显示SQL参数
- 约束： 必填

sqlLevel

- 示例：debug
- 描述：打印SQL的日志等级，默认debug，可以是info、warn、error
- 约束： 必填

5.1.4 中间件客户端

开始使用

一、介绍

为了方便用户的使用，接入中间件提供了客户端的方式，让服务于接入中间件进行连接。

使用客户端方式可以更容易地连接中间件，更加规范的传输数据，减少因数据不规范而导致的问题。

目前客户端方式只提供java版本，引入方式可以参考此处。

二、简单使用

> 提示

此处提供一个简单的代码示例，此代码示例为单次同步请求操作。

且此代码示例并没有持久化请求及应答的数据。

```
        // 构建请求参数
        Map<String, Object> map = new HashMap<>(1);
        map.put("loginName", "Tom");
        // 发送请求，并获取返回结果
        InterCoResult result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间（毫秒），地址及端口号填写配置文件中 middleware-recServer 下的配置
                .create("http://localhost:8888", 50000)
                // 请求类型，根据服务定义设置
                .setHttpType(HttpType.GET)
                // 中间件中的服务id，根据yml文件中 repchain-services-serviceId 配置填写，决定请求哪个服务
                .setServiceId("1")
                // 设置服务定义访问的url
                .setUrl("/user/valid")
                // 设置传输的数据
                .setForm(map)
                // 发送数据
                .msg();
```

同步请求

一、同步单次请求

同步单次请求，发送则得到返回结果。

> 注意
>
> 在一次调用中只发送一次同步请求数据。

1. 接口描述

/user/valid 检查用户名是否可用

- 接口声明

  综合工作平台新增用户时验证账号可用性

- 接口约束

  无

- URI

  GET /user/valid

检查账号是否可用输入参数如下：

序号	参数	是否必填	参数类型	描述
1	loginName	是	String	用户名

- 请求消息

  可以用FormData格式传输，若非FormData格式且字段中还有特殊符号，需使用URLEncode编码，请求消息具体如下：

```
{
 "loginName":"XXXXXXXXX",
}
```

- 响应信息

```
{
 "code": 0,
 "data": {},
 "msg": "ActionOK"
}
```

检查账号是否可用输出结果如下：

序号	参数	参数类型	描述
1	code	Integer	返回状态码
2	msg	String	返回状态信息
3	data	Object	返回业务数据

2. 代码示例

```
        // 构建请求参数
        Map<String, Object> map = new HashMap<>(1);
        map.put("loginName", "Tom");
        // 发送请求，并获取返回结果
        InterCoResult result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间（毫秒），地址及端口号填写配置文件中 middleware-recServer 下的配置
                .create("http://localhost:8888", 50000)
                // 请求类型，根据服务定义设置
                .setHttpType(HttpType.GET)
                // 中间件中的服务id，根据yml文件中 repchain-services-serviceId 配置填写，决定请求哪个服务
                .setServiceId("1")
                // 设置服务定义访问的url
                .setUrl("/user/valid")
                // 设置传输的数据
                .setForm(map)
                // 发送数据
                .msg();
```

二、同步多次请求

同步多次请求，如果多次请求包含在同一个方法中，推荐将存证的请求cid设置成一致。

若将请求cid设置为同一个，则需要将每次请求的序号进行递增操作。

1. 接口描述

见同步单次请求中的接口描述。

2. 代码示例

```
        // 1.第一次请求
        Map<String, Object> map = new HashMap<>(1);
        map.put("loginName", "12110107bi45jh675g");
        // 传输设置
        ReqOption reqOption = new ReqOption();
        // 设置是否为最后一次请求，默认为true
        reqOption.setIsEnd(ReqOption.FALSE);
        InterCoResult result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间
                .create("http://localhost:8888", 50000)
                // 请求类型，根据接口定义设置
                .setHttpType(HttpType.GET)
                // 中间件中的服务id，根据yml文件配置填写
                .setServiceId("1")
                // 设置访问的url
                .setUrl("/user/valid")
                // 设置传输的数据
                .setForm(map)
                // 发送数据
                .msg(reqOption);
        // 2. 第二次请求
        reqOption = new ReqOption();
        // 由于是同一次请求，所以请求id需要一致，获取上一次请求id
        reqOption.setCid(result.getCid());
        // 设置请求序号，默认为1
        reqOption.setSeq(2);
        // 设置是否为最后一次请求，默认为true
        reqOption.setIsEnd(ReqOption.FALSE);
        // 设置请求参数
        map = new HashMap<>(1);
        map.put("loginName", "yewu");
        result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间
                .create("http://localhost:8888", 50000)
                // 请求类型，根据接口定义设置
                .setHttpType(HttpType.GET)
                // 中间件中的服务id，根据yml文件配置填写
                .setServiceId("1")
                // 设置访问的url
                .setUrl("/user/valid")
                // 设置传输的数据
                .setForm(map)
                // 发送数据
                .msg(reqOption);
        // 3. 第三次请求 （最后一次请求）
        reqOption = new ReqOption();
        // 由于是同一次请求，所以请求id需要一致，获取上一次请求id
        reqOption.setCid(result.getCid());
        // 设置请求序号，默认为1
        reqOption.setSeq(3);
        // 设置请求参数
        map = new HashMap<>(1);
        map.put("loginName", "test");
        result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间
                .create("http://localhost:8888", 50000)
                // 请求类型，根据接口定义设置
                .setHttpType(HttpType.GET)
                // 中间件中的服务id，根据yml文件配置填写
                .setServiceId("1")
                // 设置访问的url
                .setUrl("/user/valid")
                // 设置传输的数据
                .setForm(map)
                // 发送数据
                .msg(reqOption);
```

三、同步上传文件

> 注意

1. 上传文件的方式为，将文件的 绝对路径 发送给中间件，而不是将文件流发送给中间件。
2. 由于中间件和宿主服务部署在同一个操作系统中，所以中间件接收到文件路径后，可以直接读取到文件，并通过grpc方式发送给服务方中间件，进行文件传输。
3. 同步上传文件方式仅适用于较小的文件传输，如果文件较大，请适当增加客户端超时时间，或选择异步文件传输的方式进行文件上传。
4. 由于中间件托管了文件传输，所以在文件传输过程中可以携带参数进行传输。
5. 当前版本文件传输，单次请求只支持单文件传输。若要传输多个文件，请将多个文件压缩为一个文件进行传输，或者多次调用文件传输请求。
6. 为安全起见，当前中间件不会主动删除获取文件，请使用者使用后自行移动文件或删除文件。

1. 接口描述

/user/upload 文件上传

- 接口声明

  使用中间件客户端同步上传文件时调用该接口

- 接口约束

  无

- URI

  POST /user/upload

文件上传输入参数如下：

序号	参数	是否必填	参数类型	描述
1	editor	否	String	编辑人
2	file	否	String	文件路径

- 请求消息

  可以用FormData格式传输，若非FormData格式且字段中还有特殊符号，需使用URLEncode编码，请求消息具体如下：

```
{
 "editor":"XXXXXXXXX",
 "file":"XXXXXXXXX",
}
```

- 响应信息

```
{
 "code": 0,
 "data": {},
 "msg": "ActionOK"
}
```

文件上传输出结果如下：

序号	参数	参数类型	描述
1	code	Integer	返回状态码
2	msg	String	返回状态信息
3	data	Object	返回业务数据

2. 代码示例

```
Map<String, Object> map = new HashMap<>(1);
map.put("editor", "Jack");
InterCoResult result = MiddlewareClient
        // 填写中间件地址及端口号，及超时时间
        .create("http://localhost:8888", 50000)
        // 请求类型，根据接口定义设置
        .setHttpType(HttpType.POST)
        // 中间件中的服务id，根据yml文件配置填写
        .setServiceId("1")
        // 设置访问的url
        .setUrl("/user/upload")
        // 设置传输的数据
        .setForm(map)
        // 设置文件在传输参数中字段
        .setFileField("file")
        // 设置传输的文件
        .setFile(new File("/home/repchain/test.tar.gz"))
        // 发送数据
        .sendFile();
```

四、同步下载文件

> 注意

1. 下载文件是通过中间件进行文件下载。
2. 文件下载完成后，中间件会将下载的文件地址返回给宿主服务。
3. 宿主服务可以直接读取文件。
4. 下载的文件会写入 `InterCoResult` 对象中，`filePath` 字段下，字段下为文件的绝对路径。
5. 为安全起见，当前中间件不会主动删除获取文件，请使用者使用后自行移动文件或删除文件。

1. 接口描述

/user/download 文件下载

- 接口声明

  使用中间件客户端同步下载文件时调用该接口

- 接口约束

  无

- URI

  GET /user/download

文件下载输入参数如下：

序号	参数	是否必填	参数类型	描述
1	username	否	String	用户名

- 请求消息

  可以用FormData格式传输，若非FormData格式且字段中还有特殊符号，需使用URLEncode编码，请求消息具体如下：

```
{
 "username":"XXXXXXXXX",
}
```

- 响应信息

```
{
 "code": 0,
 "data": {},
 "msg": "ActionOK"
}
```

文件下载输出结果如下：

序号	参数	参数类型	描述
1	code	Integer	返回状态码
2	msg	String	返回状态信息
3	data	Object	返回业务数据

2. 代码示例

- 服务提供方代码示例

由于文件传输交由中间件进行托管，所以下载文件时，需要约定一个方式通知中间件文件的具体地址。

中间件约定的方式为，将文件路径写入到response的header中。

```
// 将文件绝对路径设置到header中，参数为filePath，此处需要填写服务器上文件的绝对路径
response.setHeader("filePath", "/test.zip");
```

- 调用方代码代码示例

```
        // 构造请求参数
        Map<String, Object> map = new HashMap<>(1);
        map.put("username", "Jack");
        // 发送请求，并获取返回结果
        InterCoResult result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间
                .create("http://localhost:8888", 500000)
                // 请求类型，根据接口定义设置
                .setHttpType(HttpType.GET)
                // 中间件中的服务id，根据yml文件配置填写
                .setServiceId("1")
                // 设置访问的url
                .setUrl("/user/download")
                // 设置传输的数据
                .setForm(map)
                // 发送数据
                .download();
```

异步请求

一、异步单次请求

异步调用具体说明参考HTTP异步请求概述

> 注意
>
> 在一次调用中只发送一次同步请求数据。

1. 接口描述

/user/async 获取数据列表（服务方）

- 接口声明

  服务方使用中间件客户端异步请求获取数据列表时调用该接口

- 接口约束

  无

- URI

  GET /user/async

服务方获取数据列表输入参数如下：

序号	参数	是否必填	参数类型	描述
1	pageSize	是	int	每页数量
1	pageNo	是	int	页码

- 请求消息

  可以用FormData格式传输，若非FormData格式且字段中还有特殊符号，需使用URLEncode编码，请求消息具体如下：

```
{
 "pageSize":"XXXXXXXXX",
 "pageNo":"XXXXXXXXX",
}
```

- 响应信息

```
{
 "code": 0,
 "data": {},
 "msg": "ActionOK"
}
```

服务方获取数据列表输出结果如下:

序号	参数	参数类型	描述
1	code	Integer	返回状态码
2	msg	String	返回状态信息
3	data	Object	返回业务数据

/user/callback 数据列表应答接口（调用方）

- 接口声明

  调用方使用中间件客户端异步应答服务方获取数据列表的请求时调用该接口

- 接口约束

  无

- URI

  GET /user/callback

调用方应答数据列表输入参数如下：

序号	参数	是否必填	参数类型	描述
1	data	是	Object	应答返回列表数据

- 请求消息

  可以用FormData格式传输，若非FormData格式且字段中还有特殊符号，需使用URLEncode编码，请求消息具体如下：

```
{
 "data":"XXXXXXXXX",
}
```

- 响应信息

```
{
 "code": 0,
 "data": {},
 "msg": "ActionOK"
}
```

调用方应答数据列表输出结果如下:

序号	参数	参数类型	描述
1	code	Integer	返回状态码
2	msg	String	返回状态信息
3	data	Object	返回业务数据

2. 代码示例

- 调用方代码示例

> 提示

异步请求时，必须在`ReqOption`对象中，设置`callBackUrl`和`callBackMethod`两个参数。

此处两个参数用于告诉服务方的中间件，具体的应答接口地址和应答接口的类型。

```
        // 构建请求参数
        Map<String, Object> map = new HashMap<>(1);
        map.put("pageSize", 1);
        map.put("pageNo", 10);
        // 设置请求配置对象
        ReqOption option = new ReqOption();
        // 设置为异步请求
        option.setSync(ReqOption.FALSE);
        // 异步请求结束标志设置为false
        option.setIsEnd(ReqOption.FALSE);
        // 设置异步请求应答接口地址
        option.setCallbackUrl("/callback");
        // 设置异步请求接口应答地址请求类型
        option.setCallbackMethod(HttpType.POST.toString());
        // 发送请求，并获取返回结果
        InterCoResult result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间
                .create("http://localhost:8888", 50000)
                // 请求类型，根据接口定义设置
                .setHttpType(HttpType.GET)
                // 中间件中的服务id，根据yml文件配置填写
                .setServiceId("1")
                // 设置访问的url
                .setUrl("/user/async")
                // 设置传输的数据
                .setForm(map)
                // 发送数据
                .msg(option);
```

- 服务方应答代码示例

> 提示

1. 服务方在应答时需要知道应答id。
2. 应答会随着参数一起传如接口，使用`request.getParameter("callbackId")`可接收到应答id。
3. 此处代码示例展示的为服务方应答时的代码示例。

```
        // 从数据库获取调用方请求的数据
        List<Map<String, Object>> list = baseMapper.selectAll("table_name",pageSize,pageNo);
        // 构建请求参数
        Map<String, Object> map = new HashMap<>(1);
        map.put("data", list);
        // 传输设置
        ReqOption reqOption = new ReqOption();
        // 设置为应答
        reqOption.setBReq(ReqOption.FALSE);
        // 设置为异步请求
        reqOption.setSync(ReqOption.FALSE);
        // 发送请求，并获取返回结果
        InterCoResult result = MiddlewareClient
            // 填写中间件地址及端口号，及超时时间（毫秒），地址及端口号填写配置文件中 middleware-recServer 下的配置
            .create("http://localhost:8888", 50000)
            // 设置传输的数据
            .setForm(map)
            // 设置回调id，会随着请求一起传到服务方，每次请求都是唯一的
            .setCallBackId(callbackId)
            // 发送数据
            .msg(reqOption);
```

二、异步多次请求

异步多次请求，可以参考同步多次请求，就是将异步单次操作进行多次调用。

注意

1. 异步多次请求参考同步请求时，需要设置请求为异步（sync=false），同时需要设置应答接口url(callbackUrl)及应答接口类型(CallbackMethod)。
2. 如果多次请求的应答接口均为同一个，则推荐将请求ID（cid）也设置为同一个。

三、异步上传文件

注意

1. 上传文件的方式为，将文件的 绝对路径 发送给中间件，而不是将文件流发送给中间件。
2. 由于中间件和宿主服务部署在同一个操作系统中，所以中间件接收到文件路径后，可以直接读取到文件，并通过grpc方式发送给服务方中间件，进行文件传输。
3. 由于中间件托管了文件传输，所以在文件传输过程中可以携带参数进行传输。
4. 当前版本文件传输，单次请求只支持单文件传输。若要传输多个文件，请将多个文件压缩为一个文件进行传输，或者多次调用文件传输请求。
5. 为安全起见，当前中间件不会主动删除获取文件，请使用者使用后自行移动文件或删除文件。

1. 接口描述

/user/asyncUpload 文件上传（服务方）

- 接口声明

  服务方使用中间件客户端异步上传文件时调用该接口

- 接口约束

  无

- URI

  POST /user/asyncUpload

服务方异步上传文件输入参数如下：

序号	参数	是否必填	参数类型	描述
1	editor	否	String	编辑人
2	file	否	String	文件路径

- 请求消息

  可以用FormData格式传输，若非FormData格式且字段中还有特殊符号，需使用URLEncode编码，请求消息具体如下：

```
{
 "editor":"XXXXXXXXX",
 "file":"XXXXXXXXX",
}
```

- 响应信息

```
{
 "code": 0,
 "data": {},
 "msg": "ActionOK"
}
```

服务方异步上传文件输出结果如下：

序号	参数	参数类型	描述
1	code	Integer	返回状态码
2	msg	String	返回状态信息
3	data	Object	返回业务数据

/user/callbackUpload 异步文件应答接口（调用方）

- 接口声明

  调用方使用中间件客户端异步应答服务方上传文件时调用该接口

- 接口约束

  无

- URI

  GET /user/callbackUpload

调用方应答异步上传文件输入参数如下：

序号	参数	是否必填	参数类型	描述
1	data	是	Object	应答状态数据

- 请求消息

  可以用FormData格式传输，若非FormData格式且字段中还有特殊符号，需使用URLEncode编码，请求消息具体如下：

```
{
 "data":"XXXXXXXXX",
}
```

- 响应信息

```
{
 "code": 0,
 "data": {},
 "msg": "ActionOK"
}
```

调用方应答异步上传文件输出结果如下:

序号	参数	参数类型	描述
1	code	Integer	返回状态码
2	msg	String	返回状态信息
3	data	Object	返回业务数据

2. 代码示例

- 调用方代码示例

```
        // 设置传输参数
        Map<String, Object> map = new HashMap<>(1);
        map.put("editor", "Jack");
        ReqOption option = new ReqOption();
        // 设置为异步请求
        option.setSync(ReqOption.FALSE);
        // 异步请求结束标志设置为false
        option.setIsEnd(ReqOption.FALSE);
        // 设置异步请求应答接口地址
        option.setCallbackUrl("/callbackUpload");
        // 设置异步请求接口应答地址请求类型
        option.setCallbackMethod(HttpType.GET.toString());
        // 发送请求，并获取返回结果
        InterCoResult result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间
                .create("http://localhost:8888", 500000)
                // 请求类型，根据接口定义设置
                .setHttpType(HttpType.POST)
                // 中间件中的服务id，根据yml文件配置填写
                .setServiceId("1")
                // 设置访问的url
                .setUrl("/user/asyncUpload")
                // 设置传输的数据
                .setForm(map)
                // 设置文件接受的字段
                .setFileField("file")
                // 设置需要传输的文件
                .setFile(new File("/Users/Downloads/xxx.tar.gz"))
                // 发送数据
                .sendFile(option);
```

- 服务方应答代码示例

> 提示

1. 服务方在应答时需要知道应答id。
2. 应答会随着参数一起传如接口，使用 `request.getParameter("callbakcId")` 可接收到应答id。
3. 此处代码示例展示的为服务方应答时的代码示例。
4. 此部分代码用于添加到接收文件完成后，告诉调用方已经完成文件接收，如果接收文件失败也可以通过此方式进行通知。

```
        // 构建请求参数
        Map<String, Object> map = new HashMap<>(1);
        map.put("data", "ok");
        // 传输设置
        ReqOption reqOption = new ReqOption();
        reqOption.setBReq(ReqOption.FALSE);
        reqOption.setSync(ReqOption.FALSE);
        // 发送请求，并获取返回结果
        InterCoResult result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间（毫秒），地址及端口号填写配置文件中 middleware-recServer 下的配置
                .create("http://localhost:8888", 50000)
                // 设置传输的数据
                .setForm(map)
                // 设置回调id，会随着请求一起传到服务方，每次请求都是唯一的
                .setCallBackId(callbackId)
                // 发送数据
                .msg(reqOption);
```

四、异步下载文件

> 提示

1. 异步下载文件其实是由调用方申请文件。
2. 服务方获取申请后进行审批等流程，然后将文件推送给调用方。
3. 为安全起见，当前中间件不会主动删除获取文件，请使用者使用后自行移动文件或删除文件。

1. 接口描述

/user/download 文件下载（服务方）

- 接口声明

  服务方使用中间件客户端异步下载文件时调用该接口

- 接口约束

  无

- URI

  GET /user/download

服务方异步下载文件输入参数如下：

序号	参数	是否必填	参数类型	描述
1	editor	否	String	编辑人
2	file	否	String	需要下载的文件

- 请求消息

  可以用FormData格式传输，若非FormData格式且字段中还有特殊符号，需使用URLEncode编码，请求消息具体如下：

```
{
 "editor":"XXXXXXXXX",
 "file":"XXXXXXXXX",
}
```

- 响应信息

```
{
 "code": 0,
 "data": {},
 "msg": "ActionOK"
}
```

服务方异步下载文件输出结果如下：

序号	参数	参数类型	描述
1	code	Integer	返回状态码
2	msg	String	返回状态信息
3	data	Object	返回业务数据

/user/callbackDownload 文件下载应答接口（调用方）

- 接口声明

  调用方使用中间件客户端异步应答服务方下载文件时调用该接口

- 接口约束

  无

- URI

  POST /user/callbackDownload

调用方应答异步下载文件输入参数如下：

序号	参数	是否必填	参数类型	描述
1	data	是	Object	应答状态数据

- 请求消息

  可以用FormData格式传输，若非FormData格式且字段中还有特殊符号，需使用URLEncode编码，请求消息具体如下：

```
{
 "data":"XXXXXXXXX",
}
```

- 响应信息

```
{
 "code": 0,
 "data": {},
 "msg": "ActionOK"
}
```

调用方应答异步下载文件输出结果如下:

序号	参数	参数类型	描述
1	code	Integer	返回状态码
2	msg	String	返回状态信息
3	data	Object	返回业务数据

2. 代码示例

- 调用方请求代码示例

调用方发送普通的异步消息进行请求

```
        // 构建请求参数
        Map<String, Object> map = new HashMap<>(1);
        map.put("editor", "Jack");
        map.put("file", "xxx.tar.gz");
        // 设置请求配置对象
        ReqOption option = new ReqOption();
        // 设置为异步请求
        option.setSync(ReqOption.FALSE);
        // 异步请求结束标志设置为false
        option.setIsEnd(ReqOption.FALSE);
        // 设置异步请求应答接口地址
        option.setCallbackUrl("/callbackDownload");
        // 设置异步请求接口应答地址请求类型
        option.setCallbackMethod(HttpType.POST.toString());
        // 发送请求，并获取返回结果
        InterCoResult result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间
                .create("http://localhost:8888", 50000)
                // 请求类型，根据接口定义设置
                .setHttpType(HttpType.GET)
                // 中间件中的服务id，根据yml文件配置填写
                .setServiceId("1")
                // 设置访问的url
                .setUrl("/user/asyncDownload")
                // 设置传输的数据
                .setForm(map)
                // 发送数据
                .msg(option);
```

- 服务方应答代码示例

> 注意

如果文件较大，请将超时时间适当延长。

```
        // 构建请求参数
        Map<String, Object> map = new HashMap<>(1);
        map.put("data", "ok");
        // 传输设置
        ReqOption reqOption = new ReqOption();
        // 设置为应答
        reqOption.setBReq(ReqOption.FALSE);
        // 发送请求，并获取返回结果
        InterCoResult result = MiddlewareClient
                // 填写中间件地址及端口号，及超时时间（毫秒），地址及端口号填写配置文件中 middleware-recServer 下的配置
                .create("http://localhost:8888", 50000)
                // 设置传输的数据
                .setForm(map)
                // 设置回调id，会随着请求一起传到服务方，每次请求都是唯一的
                .setCallBackId(callbackId)
                // 设置文件
                .setFile(new File("/xxx.tar.gz"))
                // 设置文件参数
                .setFileField("file")
                // 发送数据
                .sendFile(reqOption);
```

API

MIDDLEWARECLIENT

中间件客户端Api，用来让宿主服务访问中间件的客户端。

create(String host, int timeout)

- 参数
- **host**：中间件地址，例：http://127.0.0.1:8888
- **timeout**：超时时间（毫秒），例：5000
- 返回值

  返回一个 `MiddlewareClient` 实例
- 描述

  静态方法 ，用于创建一个 `MiddlewareClient` 客户端实例，指定中间件地址及超时时间。

setUrl(String url)

- 参数
- **url** ： 需要请求服务方的url。例：/user/getList
- 返回值

返回当前`MiddlewareClient`实例

- 描述

设置请求服务方的url，根据服务定义中的接口url进行填写，不需要填写地址及端口号，只需要填写url即可。

setHttpType(HttpType httpType)

- 参数
- **httpType** ： 请求类型。例：HttpType.GET
- 返回值

返回当前`MiddlewareClient`实例

- 描述

设置请求类型，请根据服务方服务定义中的接口请求类型进行填写。

setHeader(String key, String value)

- 参数
- **key** ： 请求头key值。例：x-access-id
- **value**：请求头value值。例：659426660
- 返回值

返回当前`MiddlewareClient`实例

- 描述

设置http请求头内容。

setServiceId(String serviceId)

- 参数
- **serviceId** ： 中间件中用户定义的serviceId。例：1

- 返回值

返回当前`MiddlewareClient`实例

- 描述

用户定义的serviceId，在中间件配置文件 `repchain-services-serviceId` 中定义。

setForm(Map form)

- 参数
- **form** ： 需要传输的参数。
- 返回值

返回当前`MiddlewareClient`实例

- 描述

用于请求服务方传输的参数。

setFile(File file)

- 参数
- **file** ： 需要传输的文件。例：new File("/home/test/xxx.tar")
- 返回值

返回当前`MiddlewareClient`实例

- 描述

此文件应该持久化在宿主机中。若没有持久化到宿主机，请先将文件持久化到宿主机。

此处用于获取文件的绝对路径，将文件的绝对路径传输给中间件。

setFileField(String fileField)

- 参数
- **fileField** ： 文件映射的参数字段。
- 返回值

返回当前`MiddlewareClient`实例

setCallbackId(String callbackId)

- 参数
- **callbackId** ： 异步请求的callbackID。
- 返回值

返回当前`MiddlewareClient`实例

- 描述

异步请求时会随着请求一起传过来，返回异步信息时需要携带此参数进行返回。

msg()

- 参数
- 无。

- 返回值

获取中间件返回结果对象 InterCoResult

- 描述

发送数据，简化发送数据流程，此方法通常只用于同步请求发送数据，且不需要进行数据持久化。

msg(ReqOption reqOption)

- 参数
- **reqOption** ： 请求配置项。
- 返回值

获取中间件返回结果对象InterCoResult

- 描述

发送数据，并获取返回结果，通过 ReqOption 对象进行发送数据设置，例如是否为同步请求，是否持久化等。

sendFile()

- 参数
- 无
- 返回值

获取中间件返回结果对象InterCoResult

- 描述

发送文件，此方法仅支持同步获取结果，如果文件过大请使用其他方式传输文件，或者将客户端超时时间适当延长。

使用此方法时，必须先设置 setFile 和 setFileField。

sendFile(ReqOption reqOption)

- 参数
- **reqOption** ： 请求配置项。
- 返回值

获取中间件返回结果对象InterCoResult

- 描述

发送文件，使用此方法时，必须先设置 setFile 和 setFileField。如果文件过大可在 ReqOption中设置为异步传输。

download()

- 参数
- 无
- 返回值

获取中间件返回结果对象InterCoResult

- 描述

下载文件，此方法只支持同步请求，如果文件过大，请适当延长超时时间或者使用异步请求方式。

download(ReqOption reqOption)

- 参数

- **reqOption** ： 请求配置项。
- 返回值

获取中间件返回结果对象InterCoResult

- 描述

下载文件，如果文件过大，请适当延长超时时间或者使用异步请求方式。

data(PerReq perReq)

- 参数
- **perReq** ： 查询条件参数。
- 返回值

获取中间件返回结果对象InterCoResult

- 描述

获取持久化数据接口，提供存证服务id（cid），及分页查询方法，默认可以传空对象（new PerReq（））。

REQOPTION

请求配置对象，用于配置请求数据是否持久化等。

seq

- 类型：int
- 默认：1
- 描述：用于表示同一次调用中调用的序号，默认为1，若在同一次调用中存在多次请求，请手动递增此设置。

isEnd

- 类型：int
- 默认：`ReqOption.TRUE`
- 描述：用于表示是否为最后一次请求，若在同一次调用中存在多次请求，请将非最后一次请求设置为`ReqOption.FALSE`。

bReq

- 类型：int
- 默认：`ReqOption.TRUE`
- 描述：用于表示是否为请求，如果为异步请求应答调用方时，此处需要设置为 `ReqOption.FALSE`。

callBackMethod

- 类型：String
- 默认：`ReqOption.GET`
- 描述：用于设置应答接口的方法，如：`ReqOption.GET`，`ReqOption.POST`，只有在设置为异步请求时，此选项才会生效。

callBackUrl

- 类型：String
- 默认：/
- 描述：用于设置应答接口的 url，例：/callback/list，只有在设置为异步请求时，此选项才会生效。

cid

- 类型：String
- 默认：由客户端自动生成随机ID。
- 描述：用于标识请求id，如果一次调用存在多次请求，推荐使用同一个请求id。

sync

- 类型：int
- 默认：`ReqOption.TRUE`
- 描述：是否为同步请求，默认为同步请求。

reqSave

- 类型：int
- 默认：`ReqOption.FALSE`
- 描述：是否将请求参数持久化到中间件，默认为不持久化。

resultSave

- 类型：int
- 默认：`ReqOption.FALSE`
- 描述：是否将获取的请求结果持久化到中间件数据库，默认为不持久化。

fileSave

- 类型：int
- 默认：`ReqOption.FALSE`
- 描述：是否将上传的文件或下载的文件进行备份持久化，默认为不持久化。

INTERCORESULT

客户端返回结果对象，用于接收中间件返回的数据。

code

- 类型：int
- 默认：无
- 描述：返回结果状态码，0为正确，其他为异常。

msg

- 类型：String
- 默认：无
- 描述：返回的状态消息，如果 code 不为零，则会返回错误信息。

data

- 类型：Object
- 默认：无
- 描述：请求返回的数据。

cid

- 类型：String
- 默认：无
- 描述：每次请求存证的请求ID。

filePath

- 类型：String
- 默认：无
- 描述：如果请求为下载文件，此处会返回下载文件的文件路径。

PERREQ

获取持久化数据查询对象。

cid

- 类型：String
- 默认：无
- 描述：请求存证id，可在Dashboard中请求接口存证界面查看，可以为空。

pageNo

- 类型：int
- 默认：无
- 描述：页码，从0开始，可以为空，为空时服务端默认为0。

pageSize

- 类型：int
- 默认：无
- 描述：每页数量，可以为空，为空时服务端默认为10。

5.1.5 迁移

关于迁移

> 提示
>
> 迁移前请先阅读 目录结构

迁移中间件主要需要保留的为配置文件和数据文件。

配置文件为yml文件，数据文件为data文件下db文件，如果使用时更改过数据源可忽略数据文件。

迁移时可以将中间件文件夹整体打包复制到新的服务器。

也可以重新下载中间件，将yml文件及数据文件复制到相应的文件夹。

file文件夹下为持久化的对象数据，如果修改过文件存储的目录，则可忽略此处。

如果没有修改过文件夹配置，则file文件夹也需要一同迁移，迁移后放置中间件根目录下。

5.1.6 持久化

一、如何持久化

持久化数据会存储在请求方，通过客户端的 ReqOption 来进行配置文件和数据的持久化。

二、查看持久化数据

- 通过数据库查看持数据

1. 文件持久化内容，默认位于中间件根目录 `file` 文件夹下，可在配置文件中修改默认路径
2. 数据持久化内容，默认位于中间件根目录 `data` 文件夹下的 `repchain_mid.db` 文件中，此文件为sqlite文件，可通过sqlite驱动及其他工具进行查看。可在配置文件中修改持久化方式。当前版本支持sqlite绝对路径方式和mysql持久化方式。

- 通过api查看持数据

```
InterCoResult result = MiddlewareClient
        // 填写中间件地址及端口号，及超时时间
        .create("http://localhost:8888", 50000)
        // 设置cid，pageNo，pageSize，所有参数均可以为空
        .data(PerReq.builder().cid("xxx").pageNo(0).pageSize(10).build());
```

- 通过REST查看

此处rest请求地址和端口为中间件的http服务的地址和端口。