OpenAPI SDK

您可以使用阿里云开发者工具套件（Alibaba Cloud SDK for Java），不用复杂编程即可访问Flink全托管服务。本文为您介绍如何安装并使用阿里云开发者工具套件访问Flink全托管服务。

> **重要**
>
> 2022年9月19日，我们对Flink全托管产品版本进行了升级。后续将默认使用新的API版本（V2版本）为您服务。本文档为Flink全托管旧版OpenAPI SDK使用文档（已停止维护）。建议您尽快完成API版本升级，使用新版SDK使用文档，详情请参见SDK快速入门和产品公告。

前提条件

- RAM用户已创建AccessKey。

  > **说明**
  >
  > 为避免主账号泄露AccessKey带来安全风险，建议您创建RAM用户，授予RAM用户Flink全托管相关的访问权限，再使用RAM用户的AccessKey调用SDK。相关文档请参见：
  >
  > - 创建RAM用户操作步骤，请参见创建RAM用户。
  > - Flink全托管相关访问权限，请参见手动授权（方式一）。
  > - 为RAM用户创建AccessKey，请参见创建AccessKey。

- 已安装Java环境。Alibaba Cloud SDK for Java要求使用JDK 8或更高版本。

使用限制

仅支持以下几个地域：

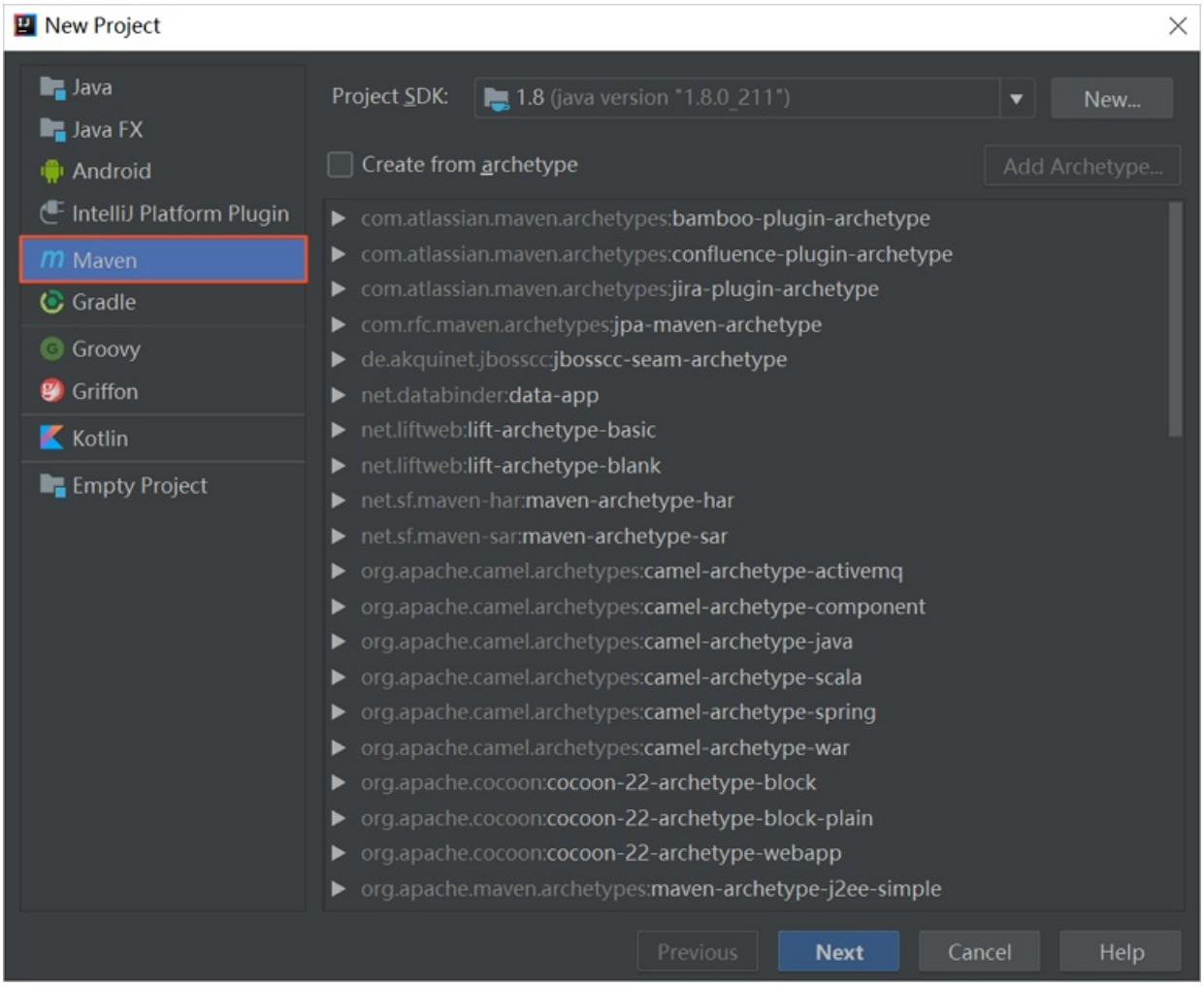
- 杭州
- 上海
- 深圳
- 北京
- 新加坡
- 张家口

> **说明**
>
> 暂不支持其他地域。

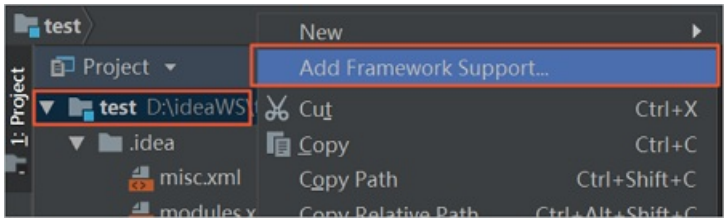
安装Alibaba Cloud SDK for Java

1. 通过以下任一方式在IDEA中配置Maven项目管理工具。
   - 使用IDEA中集成的Maven项目管理工具。
   - 访问Maven官方下载页面（Download Apache Maven）下载对应操作系统的Maven工具，手动配置Maven工具。
2. 通过以下任一方式创建Maven项目。

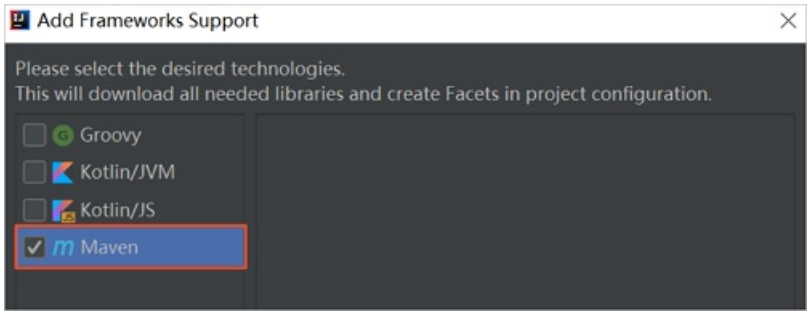
- 方式一：在IDEA中添加一个Maven项目。



- 方式二：将已有的项目转换为Maven项目。
  a. 右键单击要转换的项目，并选择**Add Framework Support...**。



  b. 选择**Maven**，并单击**OK**。



3. 在项目目录下的pom.xml文件中，添加阿里云的**aliyun-java-sdk-core**后，还需要添加Flink全托管的**aliyun-java-sdk-ververica**和**ververica-common**依赖。

```
<dependency>
   <groupId>com.aliyun</groupId>
   <artifactId>aliyun-java-sdk-core</artifactId>
   <version>4.5.3</version>
</dependency>

<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-ververica</artifactId>
  <version>1.0.8</version>
</dependency>

<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>ververica-common</artifactId>
  <version>1.0.25</version>
</dependency>
```

添加依赖后，Maven项目管理工具会自动下载相关JAR包，您可以在项目目录下的External Libraries查看已成功导入的依赖。

请求步骤

1. 配置服务所在的regionId，用户的AccessKey和Secret，来生成Client对象。

```
/*
  目前支持以下region
  cn-hangzhou
  cn-shanghai
  cn-shenzhen
  cn-beijing
  ap-southeast-1
  cn-zhangjiakou
*/
String regionId = "<regionId>";
String AccessKey = "<AccessKey>";
String Secret = "<Secret>";
IClientProfile profile = DefaultProfile.getProfile(regionId, AccessKey, Secret);
DefaultAcsClient client = new DefaultAcsClient(profile);
```

2. 初始化请求类，使用setter设置请求参数。

   以GetGlobalDeploymentDefaults请求为例。

```
GetGlobalDeploymentDefaultsRequest getGlobalDeploymentDefaultsRequest = new GetGlobalDeploymentDefaultsR
equest();
// 您可以在Flink开发控制台，工作空间详情中获取工作空间ID（workspaceId）。
// 请不要在workspaceId中直接写入工作空间名称。
String workspaceId = "<workspaceId>";
String namespace = "<namespace>";
getGlobalDeploymentDefaultsRequest.setWorkspace(workspaceId);
getGlobalDeploymentDefaultsRequest.setNamespace(namespace);
```

3. 调用并返回结果。
   - 方式一：直接print全部返回结果。

     ```
     System.out.println(client.doAction(getGlobalDeploymentDefaultsRequest).getHttpContentString());
     ```

   - 方式二：反序列化成相应的Response对象。

```
ResultModel<GetGlobalDeploymentDefaultsResp> globalDefaults = JsonUtil.toBean(client.doAction(getGloba
lDeploymentDefaultsRequest).getHttpContentString(), new TypeReference<ResultModel<GetGlobalDeploymentD
efaultsResp>>() {
    });
```

参考示例

```
String regionId = "cn-hangzhou";

IClientProfile profile = DefaultProfile.getProfile(regionId, "AccessKey", "Secret");
DefaultAcsClient client = new DefaultAcsClient(profile);

GetGlobalDeploymentDefaultsRequest getGlobalDeploymentDefaultsRequest = new GetGlobalDeploymentDefaultsRequ
est();
getGlobalDeploymentDefaultsRequest.setWorkspace(workspaceId);
getGlobalDeploymentDefaultsRequest.setNamespace(namespace);

ResultModel<GetGlobalDeploymentDefaultsResp> globalDefaults = JsonUtil.toBean(client.doAction(getGlobalDepl
oymentDefaultsRequest).getHttpContentString(), new TypeReference<ResultModel<GetGlobalDeploymentDefaultsRes
p>>() {
    });
```

全部SDK Demo

```
package sample;

import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.http.FormatType;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
import com.aliyuncs.ververica.model.v20200501.*;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.ververica.common.model.ResultModel;
import com.ververica.common.model.deployment.Artifact;
import com.ververica.common.model.deployment.Deployment;
import com.ververica.common.model.deployment.DeploymentState;
import com.ververica.common.model.namespace.Namespace;
import com.ververica.common.params.*;
import com.ververica.common.resp.*;
import com.ververica.common.util.JsonUtil;
import com.ververica.common.util.SdkUtil;

import java.util.List;


public class SdkSample {

    /**
     * 正常格式:
     * {
     * "requestId": "202008030937-4TFJR5MK2R",
     * "success": true, #为true时，结果中仅展示requestId，false时会抛出reason和message。
     * "message": "",
     * "reason": "",
     * "data": 数据内容（Object）
     * }
     */
```

```
    public static void main(String[] args) throws JsonProcessingException, ClientException {

        final String AccessKey = "<AccessKey>";
        final String Secret = "<Secret>";
        final String workspaceId = "<workspaceId>";
        final String namespace = "<namespace>";

        String regionId = "cn-hangzhou";

        IClientProfile profile = DefaultProfile.getProfile(regionId, AccessKey, Secret);
        DefaultAcsClient client = new DefaultAcsClient(profile);

        /*
         目前支持以下4个region
         ververica.cn-hangzhou.aliyuncs.com
         ververica.cn-shanghai.aliyuncs.com
         ververica.cn-shenzhen.aliyuncs.com
         ververica.cn-beijing.aliyuncs.com

         */

        String artifactFilename = "test-1.jar";//需要您手动上传flink datastream JAR包，指定获取或删除的artifact
名字。


        // List namespaces
        ListNamespacesRequest listNamespacesRequest = new ListNamespacesRequest();
        ResultModel<ListNamespacesResp> listNamespacesRespResultModel = JsonUtil.toBean(SdkUtil.getHttpCont
entString(client, listNamespacesRequest), new TypeReference<ResultModel<ListNamespacesResp>>() {
        });
        List<Namespace> namespaceList = listNamespacesRespResultModel.getData().getNamespaces();
        System.out.println(JsonUtil.toJson(namespaceList));
        String workspaceId = "";
        String namespace = "";

        if (null != namespaceList && namespaceList.size() != 0) {
            workspaceId = namespaceList.get(0).getWorkspace();
            String namespaces = namespaceList.get(0).getName();
            String[] ns = namespaces.split("/");
            namespace = ns[1];
        } else {
            /*
               没有namespace，需要购买https://realtime-compute.console.aliyun.com/#/dashboard/serverless/asi
。
             */
            System.exit(1);
        }

        //sql语法检查，成功则validationResult：VALIDATION_RESULT_VALID_INSERT_QUERY或者VALIDATION_RESULT_VALI
D_DDL_STATEMENT，其他查看errorDetails
        ValidateSqlScriptRequest validateSqlScriptRequest = new ValidateSqlScriptRequest();
        validateSqlScriptRequest.setWorkspace(workspaceId);
        validateSqlScriptRequest.setNamespace(namespace);
        ValidateSqlScriptParams validateSqlScriptParams = new ValidateSqlScriptParams();
        validateSqlScriptParams.setStatement("CREATE TABLE datagen_source ( name VARCHAR, score BIGINT ) CO
MMENT 'datagen source table' WITH ( 'connector' = 'datagen' )");
        validateSqlScriptRequest.setParamsJson(JsonUtil.toJson(validateSqlScriptParams));
        validateSqlScriptRequest.setHttpContentType(FormatType.JSON);
        ResultModel<ValidateSqlScriptResp> validateSqlScriptRespResultModel = JsonUtil.toBean(SdkUtil.getHt
tpContentString(client, validateSqlScriptRequest), new TypeReference<ResultModel<ValidateSqlScriptResp>>()
{
```

```
        });
        System.out.println(JsonUtil.toJson(validateSqlScriptRespResultModel));


        // 执行DDL操作。如果执行成功，则提示data.result='RESULT_SUCCESS|RESULT_SUCCESS_WITH_ROWS'。否则请查看mes
sage。
        ExecuteSqlScriptRequest executeSqlScriptRequest = new ExecuteSqlScriptRequest();
        executeSqlScriptRequest.setWorkspace(workspaceId);
        executeSqlScriptRequest.setNamespace(namespace);
        ExecuteSqlScriptParams executeSqlScriptParams = new ExecuteSqlScriptParams();
        executeSqlScriptParams.setStatement("create table RAN_TABLE (a varchar) with ('connector' = 'random
', 'type' = 'random');");
        executeSqlScriptRequest.setParamsJson(JsonUtil.toJson(executeSqlScriptParams));
        executeSqlScriptRequest.setHttpContentType(FormatType.JSON);
        ResultModel<ExecuteSqlScriptResp> executeSqlScriptRespResultModel = JsonUtil.toBean(SdkUtil.getHttp
ContentString(client, executeSqlScriptRequest), new TypeReference<ResultModel<ExecuteSqlScriptResp>>() {
        });
        System.out.println(JsonUtil.toJson(executeSqlScriptRespResultModel));


        //创建Deployment(SQL作业)。
        GetGlobalDeploymentDefaultsRequest getGlobalDeploymentDefaultsRequest = new GetGlobalDeploymentDefa
ultsRequest();
        getGlobalDeploymentDefaultsRequest.setWorkspace(workspaceId);
        getGlobalDeploymentDefaultsRequest.setNamespace(namespace);
        String dataStr = SdkUtil.getHttpContentString(client, getGlobalDeploymentDefaultsRequest);
        System.out.println(dataStr);
        ResultModel<GetGlobalDeploymentDefaultsResp> globalDefaults = JsonUtil.toBean(dataStr, new TypeRefe
rence<ResultModel<GetGlobalDeploymentDefaultsResp>>() {
        });
        System.out.println(JsonUtil.toJson(globalDefaults.getData()));

        Deployment.DeploymentSpec spec = globalDefaults.getData().getSpec();
        spec.setState(DeploymentState.RUNNING);

        // SQL作业。
        Artifact.SqlScriptArtifact sqlScriptArtifact = new Artifact.SqlScriptArtifact();
        sqlScriptArtifact.setSqlScript("INSERT INTO blackhole_sink SELECT UPPER(name), score FROM datagen_s
ource");
        spec.getTemplate().getSpec().setArtifact(sqlScriptArtifact);

        // 获取Artifacts列表。
        ListArtifactsRequest listArtifactsRequest = new ListArtifactsRequest();
        listArtifactsRequest.setWorkspace(workspaceId);
        listArtifactsRequest.setNamespace(namespace);
        ResultModel<ListArtifactsResp> artifacts = JsonUtil.toBean(SdkUtil.getHttpContentString(client, lis
tArtifactsRequest), new TypeReference<ResultModel<ListArtifactsResp>>() {
        });
        System.out.println(JsonUtil.toJson(artifacts));

        // 获取Artifact，指定Filename，后续创建datastream作业时使用。
        GetArtifactMetadataRequest getArtifactMetadataRequest = new GetArtifactMetadataRequest();
        getArtifactMetadataRequest.setWorkspace(workspaceId);
        getArtifactMetadataRequest.setNamespace(namespace);
        getArtifactMetadataRequest.setFilename(artifactFilename);
        dataStr = SdkUtil.getHttpContentString(client, getArtifactMetadataRequest);
        System.out.println(dataStr);
        ResultModel<GetArtifactMetadataResp> artifact = JsonUtil.toBean(dataStr, new TypeReference<ResultMo
del<GetArtifactMetadataResp>>() {
        });
        System.out.println(JsonUtil.toJson(artifact));

        // 创建DataStream作业只需要传入不同的ArtifactUri即可。
```

```
        // 创建DataStream作业只需要传入不同的Artifact即可。
        Artifact.JarArtifact jarArtifact = new Artifact.JarArtifact();
        jarArtifact.setJarUri(artifact.getData().getArtifact().getUri());


        // 获取部署目标列表。
        ListDeploymentTargetsRequest listDeploymentTargetsRequest = new ListDeploymentTargetsRequest();
        listDeploymentTargetsRequest.setWorkspace(workspaceId);
        listDeploymentTargetsRequest.setNamespace(namespace);
        ResultModel<ListDeploymentTargetsResp> deploymentTargets = JsonUtil.toBean(SdkUtil.getHttpContentSt
ring(client, listDeploymentTargetsRequest), new TypeReference<ResultModel<ListDeploymentTargetsResp>>() {
        });
        System.out.println(JsonUtil.toJson(deploymentTargets.getData().getDeploymentTargets()));

        CreateDeploymentParams createDeploymentParams = new CreateDeploymentParams();
        Deployment.DeploymentMetadata metadata = new Deployment.DeploymentMetadata();
        metadata.setName("sql-test-1");
        spec.setDeploymentTargetId(deploymentTargets.getData().getDeploymentTargets().get(0).getMetadata().
getId());
        createDeploymentParams.setMetadata(metadata);
        createDeploymentParams.setSpec(spec);

        CreateDeploymentRequest createDeploymentRequest = new CreateDeploymentRequest();
        createDeploymentRequest.setWorkspace(workspaceId);
        createDeploymentRequest.setNamespace(namespace);

        String paramsStr = JsonUtil.toJson(createDeploymentParams);
        System.out.printf("##########params:\n%s\n", paramsStr);
        createDeploymentRequest.setParamsJson(paramsStr);
        createDeploymentRequest.setHttpContentType(FormatType.JSON);
        dataStr = SdkUtil.getHttpContentString(client, createDeploymentRequest);
        System.out.println(dataStr);

        ResultModel<CreateDeploymentResp> createDeploymentRespResultModel = JsonUtil.toBean(dataStr, new Ty
peReference<ResultModel<CreateDeploymentResp>>() {
        });
        System.out.println(JsonUtil.toJson(createDeploymentRespResultModel));
        String deploymentId = createDeploymentRespResultModel.getData().getMetadata().getId();

        // 获取Deployment。
        GetDeploymentRequest getDeploymentRequest = new GetDeploymentRequest();
        getDeploymentRequest.setWorkspace(workspaceId);
        getDeploymentRequest.setNamespace(namespace);
        getDeploymentRequest.setDeploymentId(deploymentId);
        dataStr = SdkUtil.getHttpContentString(client, getDeploymentRequest);
        System.out.println(dataStr);
        ResultModel<GetDeploymentResp> getDeploymentRespResultModel = JsonUtil.toBean(dataStr, new TypeRefe
rence<ResultModel<GetDeploymentResp>>() {
        });
        Deployment deployment = getDeploymentRespResultModel.getData();
        deployment.getMetadata().getAnnotations().put("update-flag", "zdbox");
        deployment.getMetadata().getLabels().put("key2", "value2");
        deployment.getSpec().setState(DeploymentState.RUNNING);


        // 更新Deployment状态。
        UpdateDeploymentDesiredStateParams updateDeploymentDesiredStateParams = new UpdateDeploymentDesired
StateParams();
        updateDeploymentDesiredStateParams.setState(DeploymentState.CANCELLED);

        UpdateDeploymentDesiredStateRequest updateDeploymentDesiredStateRequest = new UpdateDeploymentDesir
edStateRequest();
        updateDeploymentDesiredStateRequest.setWorkspace(workspaceId);
```

```
        updateDeploymentDesiredStateRequest.setNamespace(namespace);
        updateDeploymentDesiredStateRequest.setDeploymentId(deploymentId);
        updateDeploymentDesiredStateRequest.setParamsJson(JsonUtil.toJson(updateDeploymentDesiredStateParam
s));
        updateDeploymentDesiredStateRequest.setHttpContentType(FormatType.JSON);

        dataStr = SdkUtil.getHttpContentString(client, updateDeploymentDesiredStateRequest);
        System.out.println(dataStr);
        ResultModel<UpdateDeploymentDesiredStateResp> updateDeploymentDesiredStateRespResultModel = JsonUti
l.toBean(dataStr, new TypeReference<ResultModel<UpdateDeploymentDesiredStateResp>>() {
        });
        System.out.println(JsonUtil.toJson(updateDeploymentDesiredStateRespResultModel));


        // Create savepoint：如果data为空，则说明触发失败，请查看message。
        /*
         * 成功包含data字段返回数据：{"data":{"metadata":{"createdAt":"2020-08-18T07:04:40.379926Z","jobId":"
d0d6720f-00ac-47cc-8d54-7d88a4d7b446","modifiedAt":"2020-08-18T07:04:40.379926Z","deploymentId":"0cb796fc-2
641-488e-bde8-52b2199c5747","origin":"USER_REQUEST","resourceVersion":1,"namespace":"test","annotations":{"
com.dataartisans.appmanager.controller.deployment.spec.version":"4"},"id":"c4e40dc9-69c7-4ba0-8add-5f862ecb
00ef"},"apiVersion":"v1","kind":"Savepoint","spec":{"savepointLocation":"oss://qiqi-zp/vvp/flink-savepoints
/namespaces/test/deployments/0cb796fc-2641-488e-bde8-52b2199c5747/c4e40dc9-69c7-4ba0-8add-5f862ecb00ef"},"s
tatus":{"state":"STARTED"}},"requestId":"188B1FFC-2729-458E-A404-D3CD5D972DA0"}
         * 失败无data字段，并且有message：{"RequestId":"B545121D-ED4C-4E71-9D76-CBD510A04E5B","HostId":"verve
rica-share.cn-shanghai.aliyuncs.com","Code":"BadRequest","Message":"No active job for a deployment."}
         */

        CreateSavepointRequest createSavepointRequest = new CreateSavepointRequest();
        createSavepointRequest.setWorkspace(workspaceId);
        createSavepointRequest.setNamespace(namespace);
        CreateSavepointParams createSavepointParams = new CreateSavepointParams();
        createSavepointParams.setDeploymentId(deploymentId);
        createSavepointRequest.setParamsJson(JsonUtil.toJson(createSavepointParams));
        createSavepointRequest.setHttpContentType(FormatType.JSON);
        dataStr = SdkUtil.getHttpContentString(client, createSavepointRequest);
        System.out.println(dataStr);
        ResultModel<CreateSavepointResp> savepoint = JsonUtil.toBean(dataStr, new TypeReference<ResultModel
<CreateSavepointResp>>() {
        });
        System.out.println(JsonUtil.toJson(savepoint));


        // 获取Savepoints列表。
        ListSavepointsRequest listSavepointsRequest = new ListSavepointsRequest();
        listSavepointsRequest.setWorkspace(workspaceId);
        listSavepointsRequest.setNamespace(namespace);
        listSavepointsRequest.setDeploymentId(deploymentId);
        ResultModel<ListSavepointsResp> savepoints = JsonUtil.toBean(SdkUtil.getHttpContentString(client, l
istSavepointsRequest), new TypeReference<ResultModel<ListSavepointsResp>>() {
        });
        System.out.println(JsonUtil.toJson(savepoints.getData().getSavepoints()));
        System.out.println(JsonUtil.toJson(savepoints.getData().getSavepoints().size()));


        // 获取Jobs列表。
        ListJobsRequest listJobsRequest = new ListJobsRequest();
        listJobsRequest.setWorkspace(workspaceId);
        listJobsRequest.setNamespace(namespace);
        listJobsRequest.setDeploymentId(deploymentId);
        ResultModel<ListJobsResp> jobs = JsonUtil.toBean(SdkUtil.getHttpContentString(client, listJobsReque
st), new TypeReference<ResultModel<ListJobsResp>>() {
        });
```

```
        System.out.println(JsonUtil.toJson(jobs.getData().getJobs()));


        // 删除Artifact。
        DeleteArtifactRequest deleteArtifactRequest = new DeleteArtifactRequest();
        deleteArtifactRequest.setWorkspace(workspaceId);
        deleteArtifactRequest.setNamespace(namespace);
        deleteArtifactRequest.setFilename(artifactFilename);
        DeleteArtifactResponse deleteArtifactResponse = client.getAcsResponse(deleteArtifactRequest);
        System.out.println(JsonUtil.toJson(deleteArtifactResponse));


        // 获取Deployments列表。
        ListDeploymentsRequest listDeploymentsRequest = new ListDeploymentsRequest();
        listDeploymentsRequest.setWorkspace(workspaceId);
        listDeploymentsRequest.setNamespace(namespace);
        ResultModel<ListDeploymentsResp> deployments = JsonUtil.toBean(SdkUtil.getHttpContentString(client,
listDeploymentsRequest), new TypeReference<ResultModel<ListDeploymentsResp>>() {
        });
        System.out.println(JsonUtil.toJson(deployments));


        // 删除Deployment指定已存在的且状态为CANCELLED的DeploymentId。
        DeleteDeploymentRequest deleteDeploymentRequest = new DeleteDeploymentRequest();
        deleteDeploymentRequest.setWorkspace(workspaceId);
        deleteDeploymentRequest.setNamespace(namespace);
        deleteDeploymentRequest.setDeploymentId(deploymentId);
        DeleteDeploymentResponse deleteDeploymentResponse = client.getAcsResponse(deleteDeploymentRequest);
        System.out.println(JsonUtil.toJson(deleteDeploymentResponse));

        // 获取作业模板。
        GetDeploymentDefaultsRequest getDeploymentDefaultsRequest = new GetDeploymentDefaultsRequest();
        getDeploymentDefaultsRequest.setWorkspace(workspaceId);
        getDeploymentDefaultsRequest.setNamespace(namespace);
        ResultModel<GetDeploymentDefaultsResp> deploymentDefaults = JsonUtil.toBean(client.doAction(getDepl
oymentDefaultsRequest).getHttpContentString(), new TypeReference<ResultModel<GetDeploymentDefaultsResp>>()
{
        });
        System.out.println(JsonUtil.toJson(deploymentDefaults.getData()));

        // 更新作业信息。
        UpdateDeploymentParams deployment = new UpdateDeploymentParams();
        Deployment.DeploymentMetadata deploymentMetadata = new Deployment.DeploymentMetadata();
        deploymentMetadata.setNamespace(namespace);
        deploymentMetadata.setName(jobName);
        deployment.setMetadata(deploymentMetadata);

        UpdateDeploymentRequest updateDeploymentRequest = new UpdateDeploymentRequest();
        updateDeploymentRequest.setWorkspace(workspaceId);
        updateDeploymentRequest.setNamespace(namespace);
        updateDeploymentRequest.setHttpContentType(FormatType.JSON);
        updateDeploymentRequest.setDeploymentId(depid);
        String deploymentStr = JsonUtil.toJson(deployment);
        updateDeploymentRequest.setParamsJson(deploymentStr);
        System.out.println("updatedeployment request:"+JsonUtil.toJson(updateDeploymentRequest));
        ResultModel<UpdateDeploymentResp> resultModel = JsonUtil.toBean(SdkUtil.getHttpContentString(client
, updateDeploymentRequest), new TypeReference<ResultModel<UpdateDeploymentResp>>() {
        });
        System.out.println("updatedeployment response:"+JsonUtil.toJson(resultModel));

        // 替换作业信息。
        ReplaceDeploymentRequest replaceDeploymentRequest = new ReplaceDeploymentRequest();
```

```
        ReplaceDeploymentParams replaceDeploymentParams = new ReplaceDeploymentParams();

        // 您可以按照业务需求替换Metadata和Spec两个参数的值，但不可修改Metadata参数里的namespace和Id。
        replaceDeploymentParams.setMetadata(deployment.getMetadata());
        replaceDeploymentParams.setSpec(deployment.getSpec());

        replaceDeploymentRequest.setWorkspace(workspaceId);
        replaceDeploymentRequest.setNamespace(namespace);
        replaceDeploymentRequest.setDeploymentId(deploymentId);
        replaceDeploymentRequest.setHttpContentType(FormatType.JSON);
        replaceDeploymentRequest.setParamsJson(JsonUtil.toJson(replaceDeploymentParams));
        ResultModel<ReplaceDeploymentResp> resultModel = JsonUtil.toBean(SdkUtil.getHttpContentString(clien
t, replaceDeploymentRequest), new TypeReference<ResultModel<ReplaceDeploymentResp>>() {
        });
        System.out.println("replace deployment response:"+JsonUtil.toJson(resultModel));


        // 上传自定义函数资源，目前SDK仅支持通过外部URL方式上传自定义函数资源。
        // 自定义资源的名称。
        String udfName= "asi";
        // 自定义函数JAR包。上传的JAR包需保证存放的资源地址和当前环境是网络互通的，前缀支持 "oss://","https://"开头的
格式。
        String jarurl= "oss://ververica-prod/artifacts/namespaces/vvp-team/ASI_UDX-1.0-SNAPSHOT.jar";
        CreateUdfArtifactParams createUdfArtifactParams=new CreateUdfArtifactParams();
        createUdfArtifactParams.setJarUrl(jarurl);
        createUdfArtifactParams.setName(udfName);
        CreateUdfArtifactRequest createUdfArtifactRequest=new CreateUdfArtifactRequest();
        createUdfArtifactRequest.setNamespace(namespace);
        createUdfArtifactRequest.setWorkspace(workspaceName);
        createUdfArtifactRequest.setParamsJson(JsonUtil.toJson(createUdfArtifactParams));

        // 内容参数类型。如果HTTP的Method为POST、PUT和PATCH，则需要设置sethttpcontenttype参数，否则调不通。
        createUdfArtifactRequest.setHttpContentType(FormatType.JSON);
        ResultModel<CreateUdfArtifactResp> createUdfArtifactRespResultModel=JsonUtil.toBean(SdkUtil.getHttp
ContentString(client, createUdfArtifactRequest),  new TypeReference<ResultModel<CreateUdfArtifactResp>>() {
        });
        System.out.println(JsonUtil.toJson(createUdfArtifactRespResultModel));

        // 获得已经上传的自定义函数资源信息。
        // 文件名称，可以在资源列表中获取。
        String udfName1="udf-10";
        GetUdfArtifactRequest getUdfArtifactRequest=new GetUdfArtifactRequest();
        getUdfArtifactRequest.setNamespace(namespace);
        getUdfArtifactRequest.setWorkspace(workspaceName);
        getUdfArtifactRequest.setUdfArtifactName(udfName1);
        getUdfArtifactRequest.setRequireFunctionNames(true);
        ResultModel<GetUdfArtifactResp> getUdfArtifactRespResultModel=JsonUtil.toBean(SdkUtil.getHttpConten
tString(client, getUdfArtifactRequest),  new TypeReference<ResultModel<GetUdfArtifactResp>>() {
        });
        System.out.println(JsonUtil.toJson(getUdfArtifactRespResultModel));

        // 执行创建自定义资源的SQL语句来注册自定义资源。
        List<UdfClass> udfClasses = getUdfArtifactRespResultModel.getData().getUdfArtifact().getUdfClasses(
);
        List<String>functions=new ArrayList<>();
        for(UdfClass udf:udfClasses) {
        // 创建function样式：CREATE FUNCTION `MyAggFunc` AS 'com.test.MyAggFunc'。
       //  删除function样式：DROP FUNCTION `MyAggFunc`。
       String fun = "CREATE FUNCTION `" + udf.getFunctionNames().get(0) + "` AS '" + udf.getClassName() + "
'";
       functions.add(fun);
       }
       ExecuteSqlScriptsStatementsParams executeSqlScriptsStatementsParams=new ExecuteSqlScriptsStatementsP
```

```
        ExecuteSqlScriptsStatementsParams executeSqlScriptsStatementsParams=new ExecuteSqlScriptsStatementsP
arams();
        executeSqlScriptsStatementsParams.setExecuteFunctionStr(functions);
        ExecuteSqlscriptsStatementsRequest executeSqlscriptsStatementsRequest=new ExecuteSqlscriptsStatement
sRequest();
        executeSqlscriptsStatementsRequest.setNamespace(namespace);
        executeSqlscriptsStatementsRequest.setWorkspace(workspaceName);
        executeSqlscriptsStatementsRequest.setParamsJson(JsonUtil.toJson(executeSqlScriptsStatementsParams))
;
        executeSqlscriptsStatementsRequest.setHttpContentType(FormatType.JSON);
        ResultModel<ExecuteSqlScriptsStatementsResp> executeSqlScriptsStatementsRespResultModel=JsonUtil.toB
ean(SdkUtil.getHttpContentString(client, executeSqlscriptsStatementsRequest),  new TypeReference<ResultMode
l<ExecuteSqlScriptsStatementsResp>>() {
        });
        System.out.println(JsonUtil.toJson(executeSqlScriptsStatementsRespResultModel));


        // 更新自定义函数资源。
        UpdateUdfArtifactParams updateUdfArtifactParams=new UpdateUdfArtifactParams();
        updateUdfArtifactParams.setJarUrl(jarurl);
        updateUdfArtifactParams.setName(udfName);
        UpdateUdfArtifactRequest updateUdfArtifactRequest=new UpdateUdfArtifactRequest();
        updateUdfArtifactRequest.setNamespace(namespace);
        updateUdfArtifactRequest.setWorkspace(workspaceName);
        updateUdfArtifactRequest.setUdfArtifactName(udfName);
        updateUdfArtifactRequest.setParamsJson(JsonUtil.toJson(updateUdfArtifactParams));

        // 内容参数类型。如果HTTP的Method为POST、PUT和PATCH，则需要设置sethttpcontenttype参数，否则调不通。
        updateUdfArtifactRequest.setHttpContentType(FormatType.JSON);
        ResultModel<UpdateUdfArtifactResp> updateUdfArtifactRespResultModel=JsonUtil.toBean(SdkUtil.getHttpC
ontentString(client, updateUdfArtifactRequest),  new TypeReference<ResultModel<UpdateUdfArtifactResp>>() {
        });
        System.out.println(JsonUtil.toJson(updateUdfArtifactRespResultModel));

      // 删除udfArtifact。
      DeleteUdfArtifactRequest deleteUdfArtifactRequest=new DeleteUdfArtifactRequest();
      deleteUdfArtifactRequest.setNamespace(namespace);
      deleteUdfArtifactRequest.setWorkspace(workspaceName);
      deleteUdfArtifactRequest.setUdfArtifactName(udfName);
      DeleteUdfArtifactResponse deleteUdfArtifactResponse=client.getAcsResponse(deleteUdfArtifactRequest);
      System.out.println(JsonUtil.toJson(deleteUdfArtifactResponse));

      // 创建变量配置。
      String keyName = "keyName";
      String value = "value";
      CreateSecretValueParams createSecretValueParams=new CreateSecretValueParams();
      CreateSecretValueParams.SecretValueMetadata secretValueMetadata=new CreateSecretValueParams.SecretVal
ueMetadata();
      secretValueMetadata.setName(keyName);
      secretValueMetadata.setNamespace(namespace);
      SecretValueSpec.PlainSecretValue secretValueSpec=new SecretValueSpec.PlainSecretValue();
      secretValueSpec.setValue(value);
      createSecretValueParams.setMetadata(secretValueMetadata);
      createSecretValueParams.setSpec(secretValueSpec);
      CreateSecretValueRequest createSecretValueRequest = new CreateSecretValueRequest();
      createSecretValueRequest.setNamespace(namespace);
      createSecretValueRequest.setWorkspace(workspaceId);
      createSecretValueRequest.setParamsJson(JsonUtil.toJson(createSecretValueParams));
      createSecretValueRequest.setHttpContentType(FormatType.JSON);
      ResultModel<CreateSecretValueResp> createSecretValueRespResultModel = JsonUtil.toBean(SdkUtil.getHttp
ContentString(client, createSecretValueRequest), new TypeReference<ResultModel<CreateSecretValueResp>>() {
      });
      System.out.println("create secretValue response:"+JsonUtil.toJson(createSecretValueRespResultModel));
```

```
        // 获取变量配置列表。
        ListSecretValuesRequest listSecretValuesRequest = new ListSecretValuesRequest();
        listSecretValuesRequest.setNamespace(namespace);
        listSecretValuesRequest.setWorkspace(workspaceId);
        ResultModel<ListSecretValueResp> listSecretValueRespResultModel = JsonUtil.toBean(SdkUtil.getHttpCont
entString(client, listSecretValuesRequest), new TypeReference<ResultModel<ListSecretValueResp>>() {
        });
        System.out.println(JsonUtil.toJson(listSecretValueRespResultModel.getData().getSecretValues()));

        // 删除变量配置列表。
        DeleteSecretValueRequest deleteSecretValueRequest = new DeleteSecretValueRequest();
        deleteSecretValueRequest.setName(keyName);
        deleteSecretValueRequest.setNamespace(namespace);
        deleteSecretValueRequest.setWorkspace(workspaceId);
        DeleteSecretValueResponse response = client.getAcsResponse(deleteSecretValueRequest);
        System.out.println(JsonUtil.toJson(response));

    }
}
```