天津大学

《虚拟化技术与云计算》课程报告

实验题目 基于 Serverless 的数据预处理服务

专　　业 计算机科学与技术

组　　长 李润泽

学　　号 3019244266

组　　员 李佳兴

学　　号 3019244276

组　　员 赖坤丰

学　　号 3019244261

2022 年 4 月 27 日

摘 要

本地大作业题目为基于 Serverless 的数据预处理服务，目的是利用华为函数工作流，参照 matlab 或 numpy 等多个数据处理函数，提供不少于 20 个数据预处理功能，例如统计最大值、统计最小值或去除冗余值等。并在本地进行代码编写对提供的函数工作流进行本地测试。

在实验过程中，小组成员通过华为云上的文档以及相关博客学习了 Serverless 工作原理与函数工作流 FunctionGraph 的使用方法，并利用华为函数工作流 FunctionGraph 提供 22 个数据预处理功能，从而实现基于 Serverless 的数据预处理服务。

此外，小组成员使用 Python 语言并利用华为函数工作流的相关库函数进行测试代码的编写，从而可以对上传的云函数进行本地测试。对于测试程序，我们提供了异步执行函数、删除函数、显示函数/函数流以及显示函数代码等功能。测试程序能够运行并实现了理想的结果，验证了函数的正确性。

关键词：

华为云，函数工作流 FunctionGraph，Serverless，数据预处理

1 大作业主要目标

学习 Serverless 的工作原理，并利用华为函数工作流 FunctionGraph 提供不少于 20 个数据预处理功能，从而实现基于 Serverless 的数据预处理服务。

本次实验的具体目标主要有以下：

1) 在华为云上搭建华为函数工作流 FunctionGraph
2) 参照 matlab、numpy 和 pandas 等数据处理函数，理解函数中的实现逻辑，并使用 Java 或 Python 等语言在华为云上创建相关函数
3) 对每个函数添加相应测试用例，验证函数的正确性

2 环境及相关配置

弹性云服务器 ECS 基础配置	
计费模式	按需计费
区域	华北-北京四
可用区	随机分配
CPU 架构	鲲鹏计算
规格	鲲鹏通用计算增强型 | kc1.2xlarge.2 | 8vCPUs | 16GB
镜像	公共镜像 openEuler 20.03 64bit with ARM(40GB)
系统盘	高 IO | 40G
网络	按流量计费
其他配置	默认

表 2-2 弹性云服务器 ECS 基础配置

网络配置	
网络	按需计费
弹性公网 IP	华北-北京四
CPU 架构	鲲鹏计算
规格	全动态 BGP
计费方式	按流量计费
带宽	100Mbit/s

表 2-3 网络配置

3 整体步骤

3.1 学习了解 Serverless 工作流

3.1.1 Serverless 概述

许多 Serverless 应用程序是由系列函数组成，这些函数可能会根据不同的事件触发器依次执行、并行执行或在分支中执行，我们称之为函数工作流。为了使 Serverless 平台正确执行 Serverless 应用程序的函数工作流，应用程序开发人员需要对系列函数进行编排，如何支持应用开发人员简单快速进行函数编排，需要定义一套完整、易用的工作流规范。

目前，工作流语言分为四个类型，CNCF Serverless Workflow 规范是一种基于通用的、声明性的思路提炼出的工作流规范，且支持通过 YAML 或 JSON 的格式描述和定义工作流，更容易表达像 functions, events, retries 这些无服务器技术领域的元素。

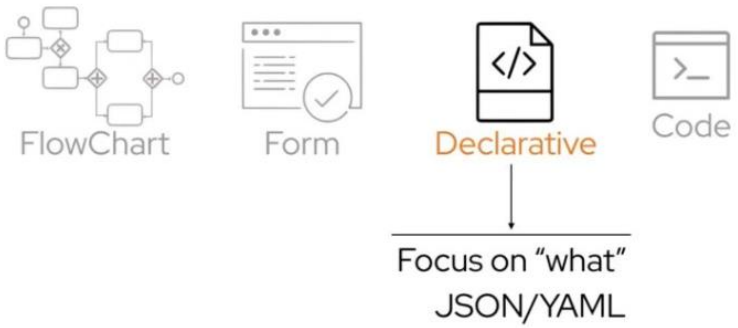

图 3-1 工作流语言

3.1.2 Serverless 的主要特点

1）事件驱动：函数在 FaaS 平台中，需要通过一系列的事件来驱动函数执行。

2）无状态：因为每次函数执行，可能使用的都是不同的容器，无法进行内存或数据共享。如果要共享数据，则只能通过第三方服务，比如 ```Redis`` 等。

3）无运维：使用 serverless 我们不需要关心服务器，也不需要关心运维，这也是 serverles 思想的核心。

4）低成本：使用 Serverless 成本很低，因为我们只需要为每次函数的运行付费。函数不运行，则不花钱，也不会浪费服务器资源过度。

3.1.3 Serverless 的价值

1）降低运营复杂度：Serverless 架构使软件应用和服务器实现了解耦，服务器不再是用户开发和运营应用的焦点。在应用上线前，用户无须再提前规划服务器的数量和规格。在运维过程中，用户无须再持续监控和维护具体服务器的状态，只需要关心应用的整体状态。应用运营的整体复杂度下降，用户的关注点可以更多地放在软件应用的体验和改进以及其他能带来更高业务价值的地方。

2）降低运营成本：服务器不再是用户关注的一个受管资源，运营的复杂度下降，应用运营所需要投入的时间和人力将大大降低。在最好的情况下，可以做到少数几个应用管理员即可管理一个处理海量请求的应用系统。

3）缩短产品的上市时间：在 Serverless 架构下，应用的功能被解构成若干个细颗粒度的无状态函数，功能与功能之间的边界变得更加清晰，功能模块之间的耦合度大大减小。这使得软件应用的开发效率更高，应用开发的迭代周期更短。

使用 Serverless，我们不需要再过多关注服务端的运维，不需要关心我们不熟悉的领域，我们只需要专注于业务的开发、专注于产品的实现。我们需要关心的事情变少了，但我们能做的事情更多了。

3.2 基于华为云函数工作流 FunctionGraph 实验准备

3.2.1 使用思路及准备

1）云函数的思路是有一个事件（如向对象存储存了一张照片），然后触发云函数对这张照片进行某种处理（比如照片美化），然后存放到另外一个地方去（如对象存储另一个位置或 mysql 数据库中）。这样的好处是，无需启用一个服务器专门等有照片上传事件，因为互联网场景下，我们不知道客户什么时候会上传照片，无需专门服务器，就是 severless 的通俗理解了。有照片上传了再自动处理，这是一种微服务的理念。

2）登录华为云官网，在“产品”-“基础服务”-“计算”-“函数工作流 Function Graph”，进入控制台。

3.3 创建函数并进行测试

1）在工作台创建一个函数

图 3-2、图 3-3 创建函数

2）设置函数的参数。“委托名称”这个概念容易理解错误。当函数需要访问云产品时，比如将照片存放到对象存储 OBS 其他路径，那么需要获得 OBS 授权，获取授权的操作，就是这里的“委托名称”。

举个生活的例子，比如我在外地工作，要委托老家的朋友去售楼部帮我买一个车位，那我需要给朋友先写好一个委托书，当车位开始售卖时，朋友拿着委托书去买车位。车位售卖==>事件触发器，朋友去买车位==>云函数，我==>要另存为的对象存储，委托书==>截图中的“委托名称”。

3）编写测试用例，云函数中的用户参数格式是自定义的，符合 JSON 格式即可。因此和测试用例需要匹配起来。

图 3-4 编写测试用例

4）编写云函数，并使用上一步的测试用例，执行测试。

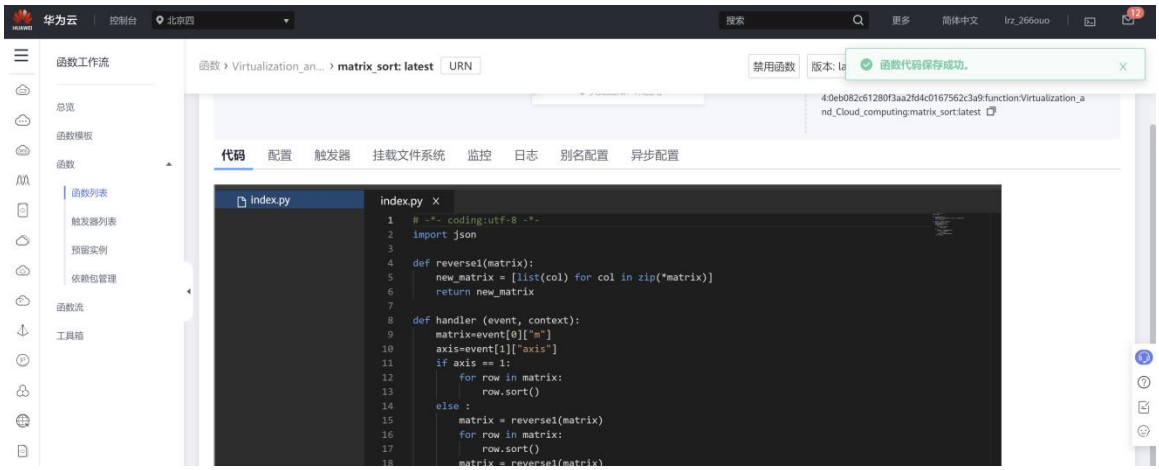

图 3-5 编写云函数

5）查看日志中结果，符合预期。

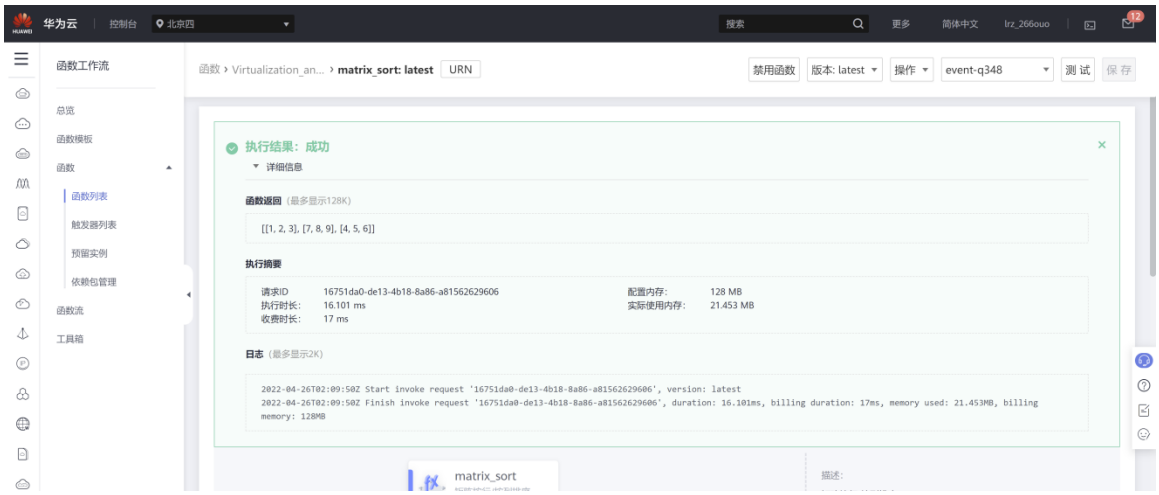

图 3-6 在华为云上进行测试

6）以此类推，创建其他函数

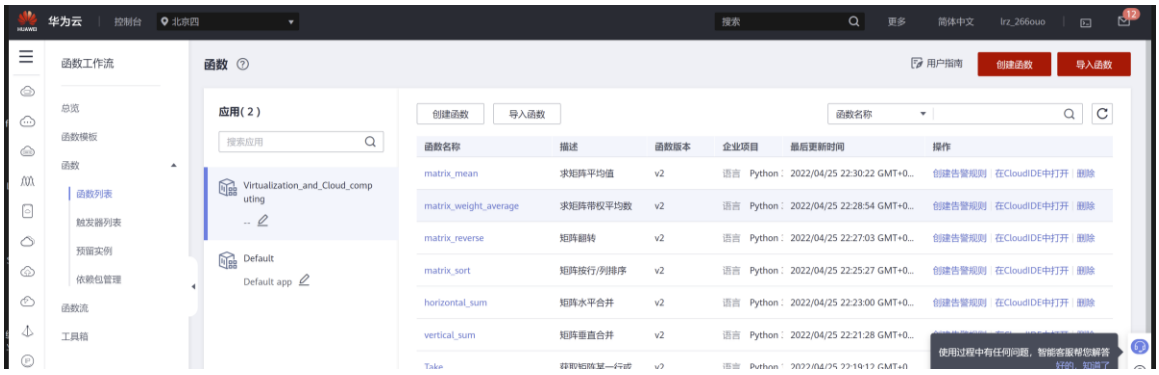

图 3-7 创建其他函数

3.4 创建函数工作流

1）进入函数工作流，并创建标准函数流

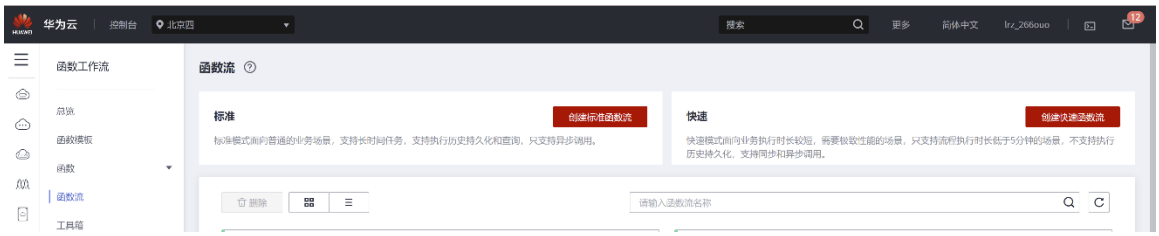

图 3-8 创建标准函数流

2）编排函数流任务，在函数流界面，通过拖拽进行流程编排，并对编辑框中的每个节点进行编辑，配置函数参数如下表所示

参数	说明
应用	函数所属应用，用户创建函数时可以进行分组，每个函数应用下面可以创建多个函数，在函数创建时可以指定其归属于某个函数应用
函数	FunctionGraph 中对应的函数
版本	FunctionGraph 中函数对应的版本
函数参数	流程中以 json 格式作为 body 参数在执行时传入函数 Key：填写参数 Value:填写参数值 DefaultValue：设置默认值，参数未获取到值时，默认获取默认值 操作：编辑或删除设置的参数
输入过滤表达式 (JSONPath)	基于上一个流程的 json 输出参数，可以使用 JSONPath 格式来选择性的过滤出当前流程的输入参数
输出过滤表达式 (JSONPath)	基于当前流程的 json 输出参数，可以使用 JSONPath 格式来选择性的过滤出下一流程的输出参数

表 3-1 函数工作流中函数配置参数

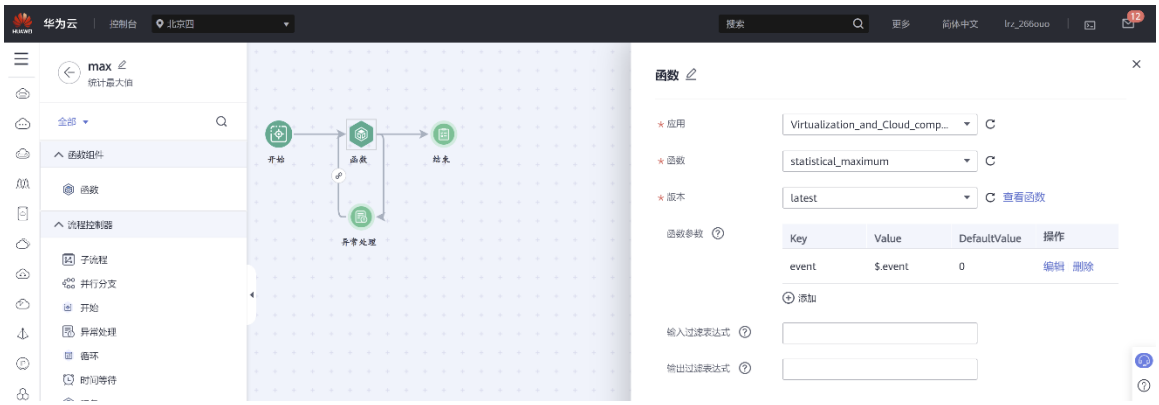

图 3-9 函数工作流配置函数参数

图 3-10 函数工作流配置异常处理参数

3）在流程中的所有节点参数配置完成后，进行保存

4）点击“启动”，在启动执行页面，输入定义值或直接启动

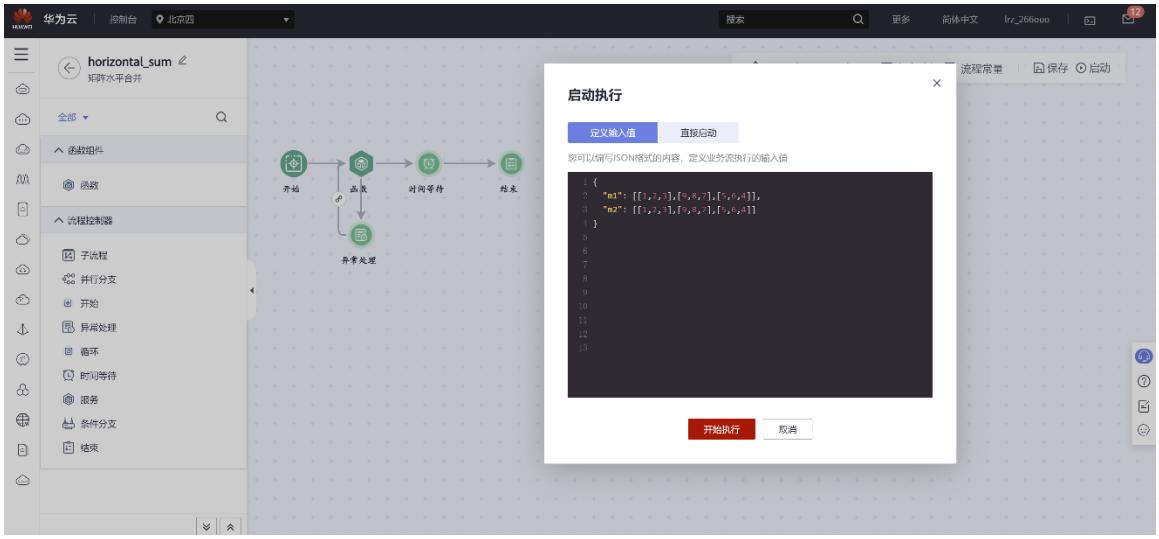

图 3-11 输入定义值并启动函数工作流

5）开始执行，页面右上角提示“启动函数流 xxx 成功”

图 3-12 页面右上角提示函数工作流启动状态

3.5 本地测试

1）获取 AK/SK

点击华为云页面右上角“用户名”，下拉选择“我的凭证”，点击“访问密钥”，点击新增访问密钥，得到文件 credentials.csv，打开即可得到 AK/SK

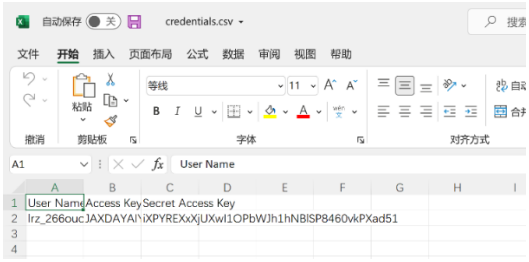

图 3-13 获取 AK/SK

2）添加库函数

首先需要在本地添加库函数，在 cmd 上输入以下命令：

```
pip install huaweicloudsdkcore==3.0.85
pip install huaweicloudsdkfunctiongraph==3.0.85
```

3）本地测试

进入华为云 API Explorer，点击“所有产品”->“函数工作流”，即可得到可以进行本地测试的代码框架。

以 ListWorkflows 为例，点击“代码示例”->“Python”，即可得到代码

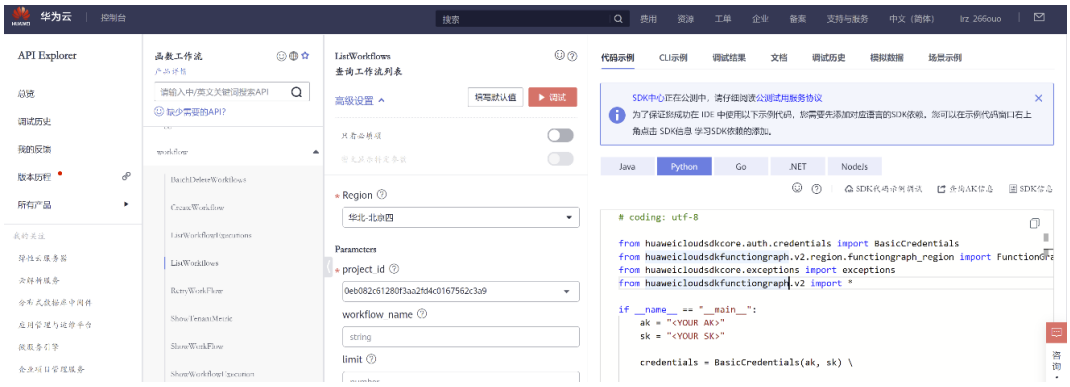

图 3-14 获取本地测试代码

4）复制代码，并输入 AK/SK，即可在本地测试

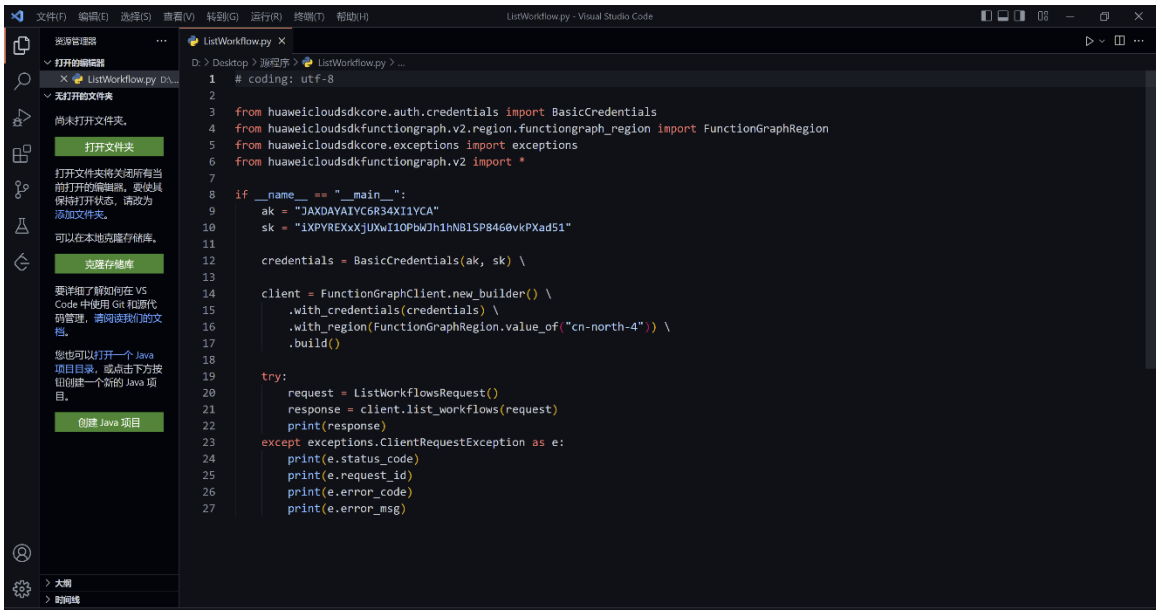

图 3-15 本地测试

4 结果分析

我们在华为云上提供了 22 个数据预处理功能，如表 4-1 所示

函数名	功能
statistical_minimum	统计最小值
statistical_maximum	统计最大值
removeDuplicates	去除冗余值
replace_outliers	检测离群值并去除
matrix_transpose	矩阵转置
dot_product	计算两个矩阵的点积
variance	计算方差
fillna	将空值填充为某一指
reshape	视图变维（一维变二维）
sum	二维矩阵求和
PRINT_INT	输出数字
matrix_sub	矩阵相减
matrix_add	矩阵相加

matrix_std	求矩阵标准差
Take	获取矩阵某一行或者某一列数据
vertical_sum	矩阵垂直合并
horizontal_sum	矩阵水平合并
matrix_reverse	矩阵翻转
matrix_weight_average	求矩阵带权平均数
matrix_mean	求矩阵平均值
matrix_sort	矩阵按行/按列排序
det	求矩阵行列式

表 4-1 华为云函数的名称及功能

通过上述步骤，我们在华为云上实现了 22 个函数以及 22 个函数工作流，如图 4-1，图 4-2 所示

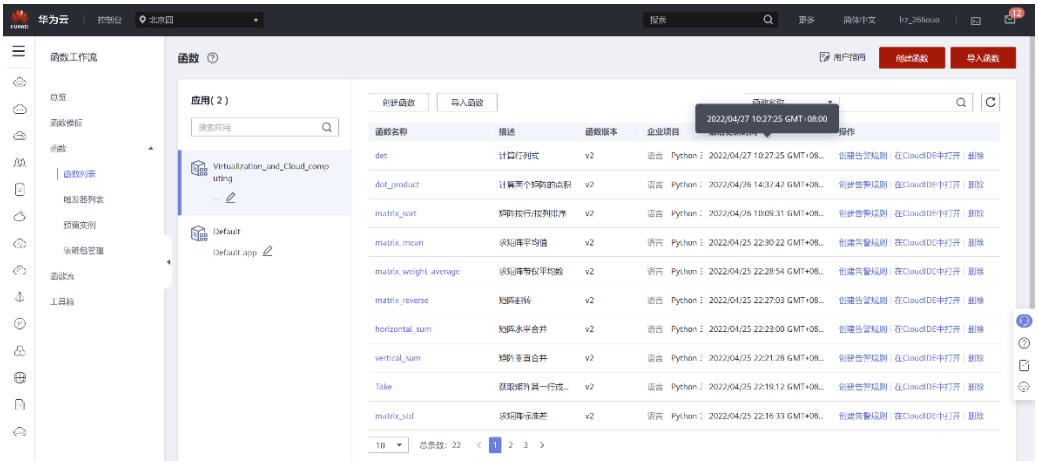

图 4-1 实现华为云函数

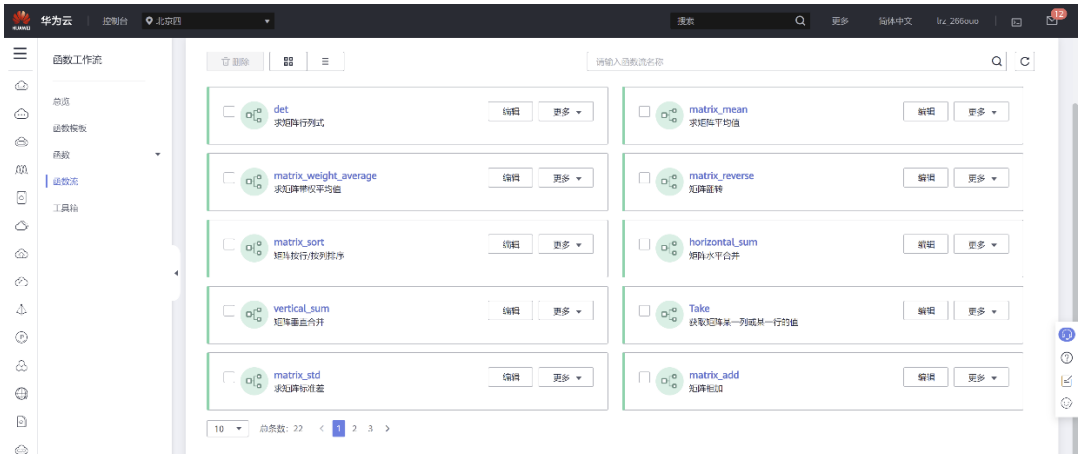

图 4-2 实现函数工作流

通过查询调试历史得到结果，如图 4-3 所示

图 4-3 查询 API Explorer 调试历史

小组成员在本地编写 Python 代码进行测试，代码功能如表 4-2 所示。

代码名称	功能
AsyncInvokeFunction	异步执行函数
DeleteFunction	删除云函数
ListFunctions	列举所有云函数
ListWorkflow	列举所有函数工作流
ShowFunctionCode	显示函数及其代码

表 4-2 本地测试代码的名称及功能

最终得到结果，如图 4-4 所示。

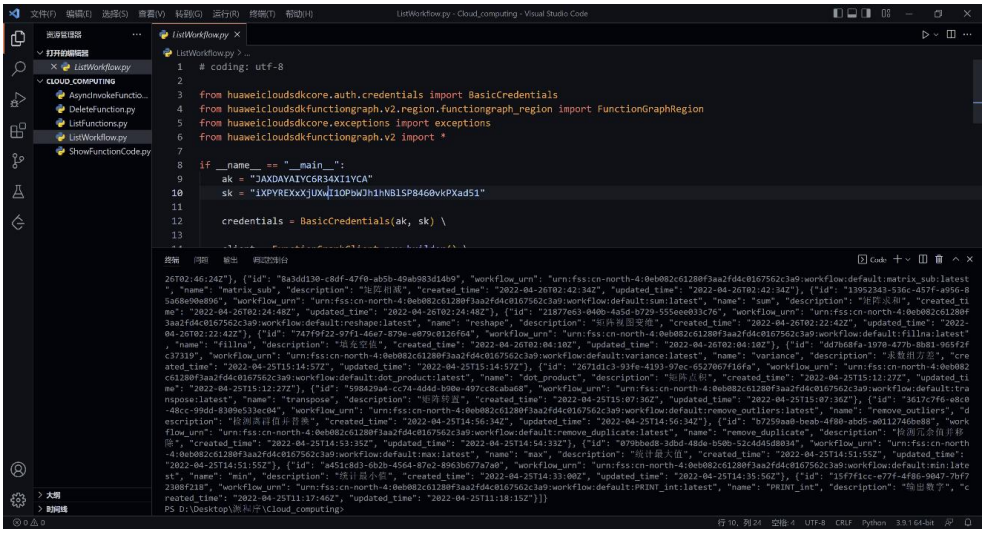

图 4-4 本地测试结果

实验最终顺利完成，小组成功利用华为函数工作流在华为云上提供超过 20 个数据预处理函数并通过本地测试。

5 知识点分析

5.1 Serverless

5.1.1 Serverless 概述

Serverless 的全称是 Serverless computing 无服务器运算，是云计算的一种模型。以平台即服务（PaaS）为基础，无服务器运算提供一个微型的架构，终端客户不需要部署、配置或管理服务器服务，代码运行所需要的服务器服务皆由云端平台来提供。 国内外比较出名的产品有 Aliyun Serverless、 Tencent Serverless、AWS Lambda、Microsoft Azure Functions 等。

Serverless 虽然被称为微服务运算，但不代表它真的不需要服务，而是说开发者再也不用过多考虑服务器的问题，计算资源作为服务而不是服务器的概念出现。它是一种构建和管理基于微服务架构的技术，允许开发者在服务部署级别而不是服务器部署级别来管理应用部署，你甚至可以管理某个具体功能或端口的部署，以便让开发者快速迭代，更快速地开发软件。

Serverless 强调的是一种架构思想和服务模型，让开发者无需关心基础设施（服务器等），而是专注到应用程序业务逻辑上。Serverless 简化了云计算的编程，其代表了程序员生产力的又一次的变革，如编程语言从汇编时代演变为高级语言时代。

对 Serverless 有以下几种理解：

- Serverless 代表的是一种服务理念或模式。这种服务理念希望用户无需关注除了业务逻辑本身之外的主机管理、服务运维、配置等事务，不需要关注运营维护问题。也即是说，只要有了 Serverless，几乎可以不需要再考虑 Devops 工作流。
- Serverless 是一种软件系统架构方法，并不代表某种技术。通常我们称 Serverless 为一种架构而不是某种技术框架。
- Serverless 是一种云服务产品形态。通常我们称 Serverless 为一种产品，比如各大厂商推出的各种 Serverless 服务或能力、封装的 API 网关等产品。

5.1.2 Serverless 的应用场景

虽然 Serverless 的应用很广泛，但是也有使用上的局限性，Serverless 比较适合的场景有以下一些：

- 异步的并发，组件可独立部署和扩展
- 应对突发或服务使用量不可预测（主要是为了节约成本，因为 Serverless 应用在不运行时不收费）

- 短暂、无状态的应用，对冷启动时间不敏感
- 需要快速开发迭代的业务

基于此，我们发现，Serverless 非常适合做实时文件处理、周期性的数据处理和移动及 Web 应用后端等。下面，我们通过一个游戏应用为例来说明什么是 Serverless 应用。

通常，一款移动端游戏至少包含如下几个特性：

- 移动端友好的用户体验
- 用户管理和权限认证
- 关卡、升级等游戏逻辑，游戏排行，玩家的等级、任务等信息

那么，应用的架构应该是下面这样的：

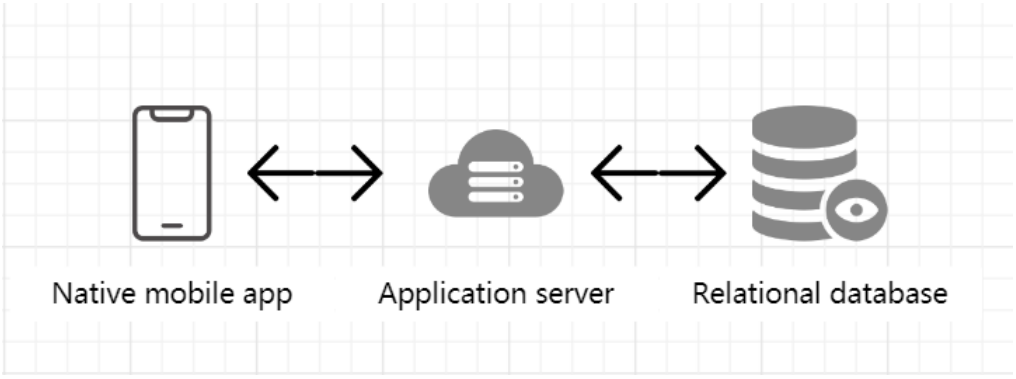

图 5-1 普通应用架构

所以，技术可能是下面这样的：

一个 app 前端，iOS 或者安卓，用 Java 写的后端，使用 JBoss 或者 Tomcat 做 server 运行 以及一个使用关系型数据库存储用户数据，如 MySQL。

不过，这样的架构虽然开发起来比较容易，但是维护起来确十分复杂，前端开发、后端的开发都需要十分专业的人员、环境的配置，还要有人专门维护数据库、应用的更新和升级。

而如果使用 Serverless 架构，那么看起来可能是下面这样的：

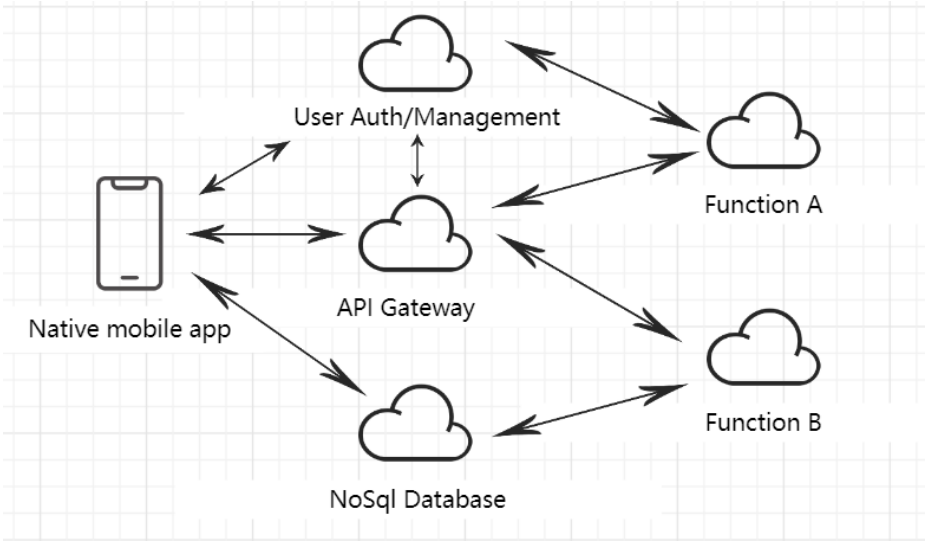

图 5-2 基于 Serverless 的应用架构

5.1.3 Serverless 的缺点

Serverless 作为一种全新的技术架构，具有很多的优点，如降低运营成本、降低运维需求、降低人力成本和减少资源开销等。不过，Serverless 也并不是一点缺点也没有，以下是 Serverless 架构的缺点：

- 状态管理方面：要想实现自由的缩放，无状态是必须的，而对于有状态的服务，使用 Serverless 这就丧失了灵活性，有状态服务需要与存储交互就不可避免的增加了延迟和复杂性。
- 延迟方面：应用程序中不同组件的访问延迟是一个大问题，我们可以通过使用专有的网络协议、RPC 调用、数据格式来优化，或者是将实例放在同一个机架内或同一个主机实例上来优化以减少延迟。而 Serverless 应用程序是高度分布式、低耦合的，这就意味着延迟将始终是一个问题，单纯使用 Serverless 的应用程序是不太现实的。
- 本地测试方面：Serverless 应用的本地测试困难是一个很棘手的问题。虽然可以在测试环境下使用各种数据库和消息队列来模拟生产环境，但是对于无服务应用的集成或者端到端测试尤其困难，很难在本地模拟应用程序的各种连接，并与性能和缩放的特性结合起来测试，并且 Serverless 应用本身也是分布式的，简单的将无数的 FaaS 和 BaaS 组件粘合起来也是有挑战性的。

5.2 数据预处理

在数据分析相关的行业中常说：“GIGO - Garbage In, Garbage Out.”，说明了如果将错误的、无意义的数据输入计算机系统，计算机自然也一定会输出错误、无意义的结果，所以数据预处理是十分必要的，本次实验的主要目的也是为了实现基于 Serverless 的预处理服务，下面将结合讲解数据预处理技术的相关内容。

5.2.1 数据预处理的概念

数据预处理是指在主要的处理以前对数据进行的一些处理。现实世界中数据大体上都是不完整，不一致的脏数据，无法直接进行数据挖掘，或挖掘结果差强人意；为了提高数据挖掘的质量产生了数据预处理技术。数据预处理有多种方法：数据清理，数据集成，数据变换，数据归约等。这些数据处理技术在数据挖掘之前使用，大大提高了数据挖掘模式的质量，降低实际挖掘所需要的时间。

5.2.2 数据预处理的主要内容

1）数据审核

对于原始数据应主要从完整性和准确性两个方面去审核。完整性审核主要是检查应调查的单位或个体是否有遗漏，所有的调查项目或指标是否填写齐全。准确性审核主要是包括两个方面：一是检查数据资料是否真实地反映了客观实际情况，内容是否符合实际；二是检查数据是否有错误，计算是否正确等。审核数据准确性的方法主要有逻辑检查和计算检查。逻辑检查主要是审核数据是否符合逻辑，内容是否合理，各项目或数字之间有无相互矛盾的现象，此方法主要适合对定性（品质）数据的审核。计算检查是检查调查表中的各项数据在计算结果和计算方法上有无错误，主要用于对定量（数值型）数据的审核。

数据审核的内容主要包括以下四个方面：

- 准确性审核。主要是从数据的真实性与精确性角度检查资料，其审核的重点是检查调查过程中所发生的误差。
- 适用性审核。主要是根据数据的用途，检查数据解释说明问题的程度。具体包括数据与调查主题、与目标总体的界定、与调查项目的解释等是否匹配。
- 及时性审核。主要是检查数据是否按照规定时间报送，如未按规定时间报送，就需要检查未及时报送的原因。
- 一致性审核。主要是检查数据在不同地区或国家、在不同的时间段是否具有可比性。

2）数据筛选

对审核过程中发现的错误应尽可能予以纠正。调查结束后，当数据发现的错误不能予以纠正，或者有些数据不符合调查的要求而又无法弥补时，就需要对数据进

行筛选。数据筛选包括两方面的内容：一是将某些不符合要求的数据或有明显错误地数据予以剔除；二是将符合某种特定条件的数据筛选出来，对不符合特定条件的数据予以剔除。数据的筛选在市场调查、经济分析、管理决策中是十分重要的。

3）数据排序

数据排序是按照一定顺序将数据排列，以便于研究者通过浏览数据发现一些明显的特征或趋势，找到解决问题的线索。除此之外，排序还有助于对数据检查纠错，为重新归类或分组等提供依据。在某些场合，排序本身就是分析的目的之一。排序可借助于计算机很容易的完成。

5.2.3 数据预处理的方法

1）数据清洗

通过填写缺失的值、光滑噪声数据、识别或删除离群点并解决不一致性来“清理”数据。主要是达到如下目标：格式标准化，异常数据清除，错误纠正，重复数据的清除。

2）数据集成

数据集成是将多个数据源中的数据结合起来并统一存储，建立数据仓库的过程实际上就是数据集成。

3）数据变换

通过平滑聚集，数据概化，规范化等方式将数据转换成适用于数据挖掘的形式。

4）数据归约

数据挖掘时往往数据量非常大，在少量数据上进行挖掘分析需要很长的时间，数据归约技术可以用来得到数据集的归约表示，它小得多，但仍然接近于保持原数据的完整性，结果与归约前结果相同或几乎相同。

5.3 函数/函数流

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。函数能提高应用的模块性，和代码的重复利用率。

Python 提供了许多内建函数，比如 print()等等。但我们也可以自己创建函数，称之为用户自定义函数。

Python 中定义函数需要满足以下几个最基本的规则：

- 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号()。
- 任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数。
- 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。

- 函数内容以冒号起始，并且缩进。
- return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的 return 相当于返回 None。

本实验的重点在于 FunctionGraph，所以在简略介绍函数的普遍概念后我们还是要将目光转移到华为的函数工作流上来。

函数流是用来编排 FunctionGraph 函数的工具，可以将多个函数编排成一个协调多个分布式函数任务执行的工作流。

用户通过在可视化的编排页面，将事件触发器、函数和流程控制器通过连线关联在一个流程图中，每个节点的输出作为连线下一个节点的输入。编排好的流程会按照流程图中设定好的顺序依次执行，执行成功后支持查看工作流的运行记录，方便诊断和调试。

函数流提供了多种类型的组件，用户可以通过拖拽组件、配置组件和连接组件进行可视化编排，实现函数任务流的编排。函数流功能如表 5-1 所示：

类型	名称	说明
函数组件	函数	FunctionGraph 函数，如何创建函数请参见创建函数。
流程控制器	子流程	把已创建的“函数流”任务作为“子流程”组合成一个新的函数流任务。
	并行分支	用于创建多个并行分支的控制器，以便同时执行多个分支任务，并可根据分支执行结束后控制下一步流程。
	开始节点	只能加入触发器，用于标识流程的开始，一个流程只能有一个开始节点。
	异常处理	用于控制函数执行失败后的下一步流程。
	循环节点	用于对数组中每个元素进行循环处理。每次循环会执行一次循环内部的子流程。
	时间等待	用于控制当前流程在指定时间延迟后再调用下一个流程。

	服务节点	用于对多个函数构成的复杂操作进行抽象，可以将多个函数操作合并成一个原子节点进行管理。
	条件分支	用于根据条件判断是否执行下一分支。
	结束节点	用于标识流程的结束。

表 5-1 函数流组件说明

此外，设计的函数流必须是一个有向无环图，从开始节点出发，开始节点后续必须且只能连接一个节点（除了异常处理和结束节点）；流程必须在某一个节点结束，结束流程有两种形式：

- 流程中存在的节点没有任何后继节点，且后续节点非条件分支，并行分支或开始节点。
- 流程中存在结束节点，且结束节点后续无其他节点。

5.4 华为函数工作流 FunctionGraph

FunctionGraph 是一项基于事件驱动的函数托管计算服务。使用 FunctionGraph 函数，只需编写业务函数代码并设置运行的条件，无需配置和管理服务器等基础设施，函数以弹性、免运维、高可靠的方式运行。此外，按函数实际执行资源计费，不执行不产生费用。

5.4.1 函数使用流程

华为云函数使用流程如图 5-3 所示。

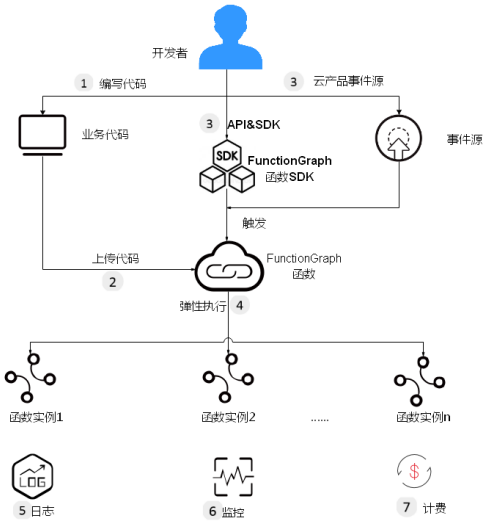

图 5-3 华为云函数使用流程

- 编写代码：用户编写业务代码，目前支持 Node.js、Python、Java、Go、C#、PHP 等语言。
- 上传代码：目前支持在线编辑、上传 ZIP 或 JAR 包，从 OBS 引用 ZIP 包等。
- API 和云产品事件源触发函数执行：通过 RESTful API 或者云产品事件源触发函数执行，生成函数实例，实现业务功能。
- 弹性执行：函数在执行过程中，会根据请求量弹性扩容，支持请求峰值的执行，此过程用户无需配置，由 FunctionGraph 完成。
- 查看日志：FunctionGraph 函数实现了与云日志服务的对接，您无需配置，即可查看函数运行日志信息。
- 查看监控：FunctionGraph 函数实现了与云监控服务的对接，您无需配置，即可查看图形化监控信息。
- 计费方式：函数执行结束后，根据函数请求执行次数和执行时间计费。（v1 版本的计量粒度精确到 100ms，v2 版本的计量粒度精确到 1ms）

5.4.2 FunctionGraph 应用场景分析

1）实时文件处理

客户端上传文件到 OBS，触发 FunctionGraph 函数，可以在上传数据后立即进行处理。例如：可以使用 FunctionGraph 实时创建图像缩略图、转换视频编码、进行数据文件汇聚、筛选等。函数工作流的优势就是业务爆发时可以自动调度资源运行更多函数实例以满足处理需求，而且只有对函数处理文件数据的时间进行计费，无需购买冗余的资源用于非峰值处理。

实时文件处理的架构，如图 5-4 所示

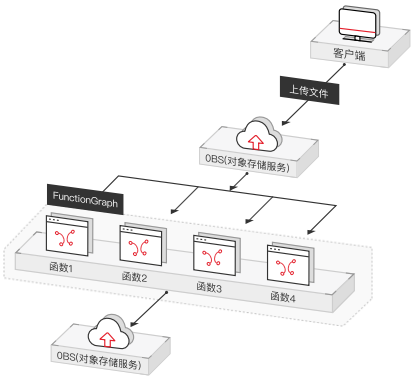

图 5-4 华为云函数流实时文件处理架构

2）实时数据流处理

使用 FunctionGraph 和 DIS 处理实时流数据，跟踪应用程序活动、处理事务处理顺序、分析数据流、整理数据、生成指标、筛选日志、建立索引、分析社交媒体以及遥测和计量 IoT 设备数据。

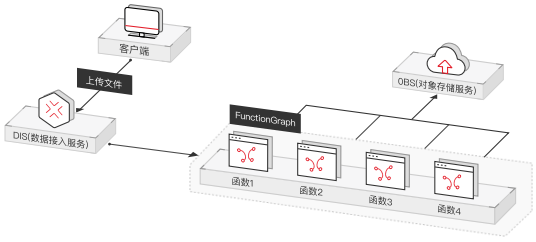

图 5-5 华为云函数流实时数据流处理架构

3） Web/移动应用后端

使用 FunctionGraph 和华为其他云服务或租户 VM 结合，用户可以快速构建高可用，自动伸缩的 Web/移动应用后端。利用 OBS，Cloud Table 的高可用性实现网站数据的高可靠性，利用 API Gateway 和 FunctionGraph 的高可用性实现网站逻辑的高可用。

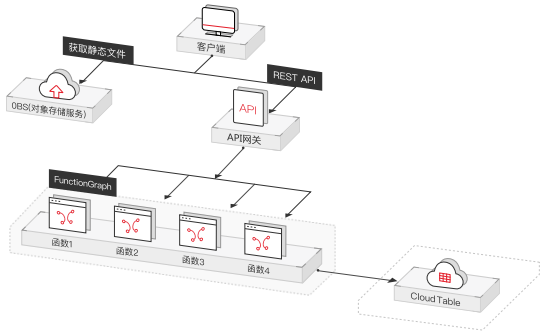

图 5-6 华为云函数流 Web/移动应用后端架构

4）人工智能场景

使用 FunctionGraph 和华为云 EI 企业智能服务结合，用户可以快速构建文字识别、图像内容审核等应用。用户上传图像后触发函数工作流执行调用文字识别/内容检测服务针对图像进程处理，并将结果以 JSON 结构化数据返回。按需使用函数

与多个智能服务集成，形成丰富的应用处理场景。并随时根据业务改变对函数处理过程做调整，实现业务灵活变更。用户只需开通相关云服务并在函数服务中编写业务逻辑，无需配置或管理服务器，专注于业务创新。业务爆发时可以自动调度资源运行更多函数实例以满足处理需求。

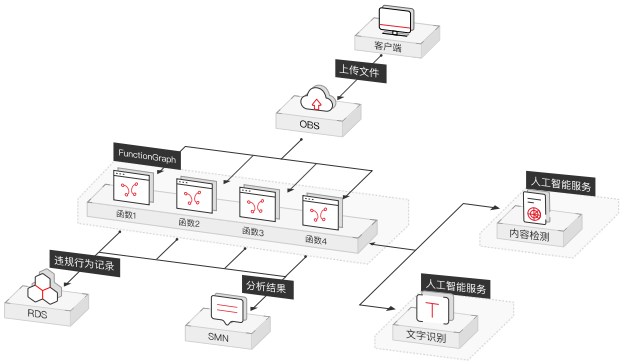

图 5-7 华为云函数流人工智能场景使用架构

5.5 FunctionGraph API 调用

FunctionGraph API 为开发者、合作伙伴提供开发、部署、托管、运维的开放接口，帮助用户快速、低成本地实现业务创新，缩短应用上线周期。

FunctionGraph API 提供的接口有如下几种类型：

类型	说明
函数数据域接口	函数的数据域接口，包括同步执行函数和异步执行函数。
函数管理域接口	函数的管理域接口，包括函数获取、创建、修改、发布等。
函数流接口	函数流接口，包括函数流的执行、工作流列表的删除、创建等。

表 5-2 API 接口简介

在函数工作流中调用 API 的方法如下：

首先，需要构造请求。请求 URI 由如下部分组成，

```
{URI-scheme} :// {Endpoint} / {resource-path} ? {query-string}
```

尽管请求 URI 包含在请求消息头中，但大多数语言或框架都要求我们从请求消息中单独传递它，所以在此单独强调。其中，

- URI-scheme：表示用于传输请求的协议，当前所有 API 均采用 HTTPS 协议。
- Endpoint：指定承载 REST 服务端点的服务器域名或 IP，不同服务不同区域的 Endpoint 不同，我们可以从地区和终端节点中获取，也可以从具体 API 的 URI 模块获取。

- query-string：查询参数，是可选部分，并不是每个 API 都有查询参数。查询参数前面需要带一个“？”，形式为“参数名=参数取值”，例如“limit=10”，表示查询不超过 10 条数据。

一般我们使用的请求方法是 HTTP 请求方法（也称为操作或动词），它告诉服务你正在请求什么类型的操作，有如下一些请求命令：

- GET：请求服务器返回指定资源。
- PUT：请求服务器更新指定资源。
- POST：请求服务器新增资源或执行特殊操作。
- DELETE：请求服务器删除指定资源，如删除对象等。
- HEAD：请求服务器资源头部。
- PATCH：请求服务器更新资源的部分内容。当资源不存在的时候，PATCH 可能会去创建一个新的资源。

接下来需要进行的操作是认证鉴权。调用接口有两种认证方式，可以选择其中一种进行认证鉴权：

- Token 认证：通过 Token 认证通用请求；
- AK/SK 认证：通过 AK（Access Key ID）/SK（Secret Access Key）加密调用请求。

最后可以收到返回结果。

5.6 调用函数

5.6.1 同步调用

同步调用指的是客户端请求需要明确等到响应结果，也就是说这样的请求必须得调用到用户的函数，并且等到调用完成才返回。

5.6.2 异步调用

异步调用是指客户端不关注请求调用的结果，服务端收到请求后将请求排队，排队成功后请求就返回，服务端在空闲的情况下会逐个处理排队的请求。

如果函数被异步调用并受到限制，会自动重试受限制的事件，并在重试之间有一定的时间间隔。在使用异步事件来调用函数之前，会将它们排队。

华为云函数工作流支持的事件源与调用的方式如下表 5-3 所示：

事件源	调用方式
消息通知服务 SMN	异步调用
分布式消息服务 DMS	异步调用

API 网关 APIG	同步调用
对象存储服务 OBS	异步调用
数据接入服务 DIS	异步调用
定时器 TIMER	异步调用
云日志服务 LTS	异步调用
云审计服务 CTS	异步调用
文档数据库服务 DDS	异步调用
分布式消息服务 Kafka 版	异步调用
云数据库 GaussDB(for Mongo)	异步调用

表 5-3 函数调用方式

5.6.3 重试机制

同步调用执行失败，需要自行尝试重试。

异步调用的重试机制和错误类型有关，具体见表 5-3。

错误类型	重试机制
用户代码异常失败	不支持重试
函数服务内部失败	FunctionGraph 支持对异步函数调用进行重试。异步调用可在界面配置最大重试次数和消息最大有限期。函数平台会根据配置的最大重试次数和消息最大有限期（最大有限期为 24 小时），进行重试。

表 5-3 不同错误类型的重试机制

5.6.4 测试管理

事件数据作为 event 参数传入入口函数，配置后保存可以持久化，以便下次测试使用。每个函数最多可配置 10 个测试事件。

5.7 FunctionGraph 单实例多并发

单实例多并发是指单个实例可以同时处理的请求数量，合理配置并发数具备如下优势：提高函数执行效率，减少执行时长；减少冷启动概率；单实例多并发可以降低总的实例数，节省用户实例资源。函数调用请求时，若有较长时间需等待下游服务的响应，则可以配置“单实例多并发”，让请求并发处理。

单实例单并发与单实例多并发的对比见表 5-4。

对比项	单实例单并发	单实例多并发
计费说明	函数实例在同一时间只能处理 1 个请求，1 个请求处理完了再去处理下一个请求。计费时长为每次函数调用的执行时长。	只有在函数配置项“单实例并发数 > 1”时，才开始统计“资源占用时长”。按资源占用时长进行计费，以实例的实际占用时间作为计费的执行时长，即从连续请求的第一个请求开始，到最后一个请求结束期间的时长。
日志打印		针对 Node.js Runtime，原来的日志方式是使用 console.info()函数，该方式会把当前请求的 Request ID 包含在日志内容中。当多请求在同一个实例并发处理时，当前请求可能有很多个，继续使用 console.info()打印日志会导致 Request ID 错乱。
共享变量	不涉及	单实例多并发处理时，修改共享变量会导致错误。这要求您在编写函数时，对于非线程安全的变量修改要进行互斥保护。
监控指标	按实际情况进行监控。	相同负载下，函数的实例数明显减少。
流控错误	不涉及	太多请求时，body 中的 errorcode 为“FSS.0429”，响应头中的 status 为 429，错误信息提示：Your request has been controlled by overload sdk, please retry later。

表 5-4 单并发与多并发对比

注意：当函数实例的并发度大于 1，需要考虑函数作用域外资源的竞争和并发安全。单实例并发数设置的大于 1，不能及时处理的请求会等待一定的时间。比如：设置了单实例并发数为 10，超过处理能力的请求不会立刻返回错误，部分请求等待后成功处理。

5.8 FunctionGraph 预留实例管理

函数工作流提供了按量和预留两种类型的实例。

按量实例是由函数工作流根据用户使用函数的实际情况来创建和释放，当函数工作流收到函数的调用请求时，自动为此请求分配执行环境。

预留实例是将函数实例的创建和释放交由用户管理，当我们为某一函数创建了预留实例，函数工作流收到此函数的调用请求时，会优先将请求转发给我们的预留实例，当请求的峰值超过预留实例处理能力时，剩余部分的请求将会转发给按量实例，由函数工作流自动为我们分配执行环境。预留实例在创建完成后，会自动加载该函数的代码、依赖包以及执行初始化入口函数，且预留实例会常驻环境，消除冷启动对业务的影响。

用户在不同的时间段，业务使用的实例数可能不一样，可以通过定时触发器调用函数，为各个时间段设置不同的预留实例数，避免在业务繁忙时未设置预留实例，导致函数被冷启动影响业务或者在业务空闲时设置多个预留实例，导致资源闲置。

可以直接创建或者通过函数创建预留实例，两者的区别如表 5-5 所示：

创建方式	优点	缺点
直接创建	创建步骤简单，易操作	只能创建固定个数的预留实例，可能导致繁忙时预留实例不够用，或者空闲时，预留实例资源浪费
通过函数创建	支持创建不同时间段不同数量的预留实例，避免繁忙时预留实例不够用，或者空闲时，预留实例资源浪费	创建步骤繁杂

表 5-5 两种方式创建预留实例的区别

5.9 华为云 Cloud IDE

在本次实验中我们编写代码使用的是华为云 Cloud IDE。

华为云 Cloud IDE 是一种按需获取的开发工具，开发者通过智能设备访问华为云 Cloud IDE 即可获取云端的开发环境，里面的编码、调试、运行、预览功能都是可以开箱即用的，不需要经过复杂的环境配置。它对各类高级语言具有良好的开发体验，对于与主流的编程语言如 C/C++、Java、Python、NodeJS、GO、PHP 等都原生支持，更多语言也可以通过安装插件方式来支持。对于当前新兴的各类编程框架，Cloud IDE 通过与华为与开发者工具 Dev Star 对接，实现了应用框架和示例工程的复用，避免开发者从零开始。华为云可以轻松利用云端资源，IDE 中使用的计算资源可以横跨鲲鹏和 X86 技术架构，可以轻松访问云端基础设施，可以使用内置能力将开发的应用一键式部署云端，快速验证能力。它还可以极大提升开发者创

新力。通过开放的插件集成机制、开源的插件开发框架、开放的 API 集成接口，配合华为开发者工具的各类代码工程模板，行业开发者可以快速实现经验和技术传承，应用开发者可以专注于自身业务逻辑和业务创新。第五，它深度整合人工智能和云技术，通过智能化编码辅助来提升开发者在编码阶段的生产效率，尽可能的通过最少的键盘输入打成编码目标，通过云调试技术在多微服务的复杂环境下实现如单体软件调试一样的良好体验，快速定位问题。

华为云 Cloud IDE 优势有以下几点：

- 本地零资源消耗：依托线上容器技术，开发对本地资源的要求趋近于零，开发资源的获取从预支转变为按需
- 跨平台接入：跨越开发平台的差异，开发者通过 Web 方式进入 IDE，代码、检查、构建、部署、调试能力尽享
- 多语言支持：对多种开发语言的支持，Java、PHP、NodeJS 等语法高亮、在线检查、随时入库

Cloud IDE 的应用场景，如图 5-8 所示：

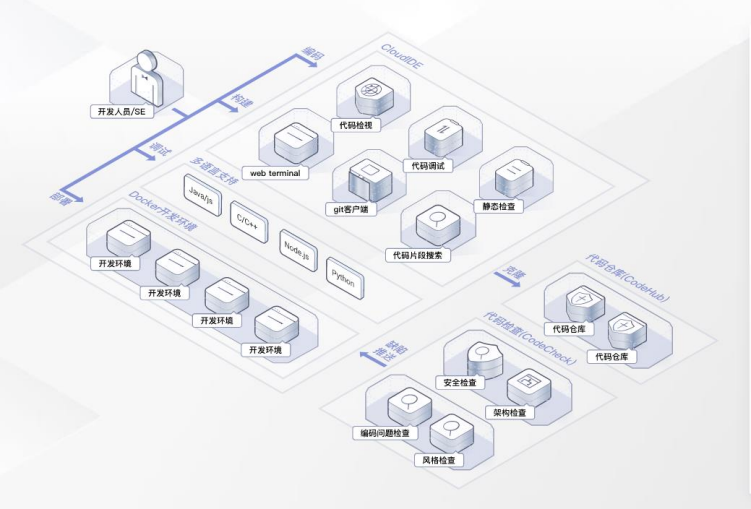

图 5-8 Cloud IDE 应用场景

5.10 触发器

触发器是触发函数执行的方式。触发器提供了一种集中、统一的方式来管理不同的事件源。在事件源中，当事件发生时，如果满足触发器定义的规则，事件源会自动调用触发器关联的函数。

函数计算提供了一种事件驱动的计算模型。函数的执行是由事件驱动的。函数的执行可以通过函数计算控制台或 SDK 触发，也可以由其他一些事件源来触发。

我们可以在指定函数中创建触发器，该触发器描述了一组规则，当某个事件满足这些规则，事件源就会触发关联的函数。

在华为云函数工作流中，已经创建的触发器，通过设置停用/启用，控制触发器的状态。SMN 触发器、OBS 触发器、APIG 触发器创建以后，不能停用，只能删除。使用时需要注意的点有：登录 FunctionGraph 控制台，在左侧导航栏选择“函数 > 函数列表”，进入函数列表界面。单击函数名称，进入函数详情界面。单击“触发器”，进入“触发器”页签。在需要停用/启用的触发器所在行，单击“停用”/“启用”，停用/启用触发器。

由于可以编写 FunctionGraph 函数来处理 OBS 存储桶事件，例如对象创建事件或对象删除事件。当用户将一张照片上传到存储桶时，OBS 存储桶调用 FunctionGraph 函数，实现读取图像和创建照片缩略图。

6 小组分工

6.1 李润泽：

- 在华为云平台上搭建环境，并部署华为云函数工作流
- 添加 10 条 numpy 或 pandas 相关的数据预处理函数
- 将 22 条华为云函数添加到函数工作流中并将其进行创建
- 撰写实验文档并进行排版
- 组织小组成员按时完成各项任务

6.2 李佳兴：

- 在华为云平台上搭建环境，并部署华为云函数工作流
- 添加 2 条 numpy 相关的数据预处理函数
- 编写测试数据并对数据预处理函数进行测试
- 撰写实验文档

6.3 赖坤丰：

- 在华为云平台上搭建环境，并部署华为云函数工作流
- 查阅数据预处理函数相关文档
- 添加 10 条 matlab 相关的数据预处理函数
- 撰写实验文档

附录1 数据预处理函数代码

matrix_sub 矩阵相减

```
# -*- coding:utf-8 -*-
import json
def handler (event, context):
    m1=event[0]["m1"]
    m2=event[1]["m2"]
    p=len(m1)
    q=len(m1[0])
    for i in range(p):
        for j in range(q):
            m1[i][j]=m1[i][j]-m2[i][j]
    return m1
```

matrix_add 矩阵相加

```
# -*- coding:utf-8 -*-
import json
def handler (event, context):
    m1=event[0]["m1"]
    m2=event[1]["m2"]
    p=len(m1)
    q=len(m1[0])
    for i in range(p):
        for j in range(q):
            m1[i][j]=m1[i][j]+m2[i][j]
    return m1
```

matrix_std 求矩阵标准差

```
# -*- coding:utf-8 -*-
import json

def mean(rep):
    if type(rep[0])==type([]):#二维
        return sum(sum(row) for row in rep) / (len(rep) * len(rep[0]))
    else:#一维
        return sum(rep)/len(rep)


def handler (event, context):
    m1=event
    avg = mean(m1)
    total=[]
```

```
    if type(m1[0])==type([]):
        for i in range(len(m1)):
            for j in range(len(m1[0])):
                tmp = pow(m1[i][j]-avg,2)
                total.append(tmp)
        return pow(sum(total)/(len(m1)*len(m1[0])),0.5)
    else:
        for i in m1:
            tmp=pow(int(i)-avg,2)
            total.append(tmp)
        return pow( sum(total)/len(m1) ,0.5)
```

take函数 获取矩阵某一行或者某一列数据

```
# -*- coding:utf-8 -*-

import json

def reverse1(matrix):
    new_matrix = [list(col) for col in zip(*matrix)]
    return new_matrix

def handler (event, context):
    m1=event[0]["m1"]
    num=event[1]
    rc=event[2]
    if rc == 0 :
        return m1[num]
    else:
        new_m1=reverse1(m1)
        return new_m1[num]
```

vstack矩阵垂直合并

```
# -*- coding:utf-8 -*-

import json
def reverse1(matrix):
    new_matrix = [list(col) for col in zip(*matrix)]
    return new_matrix

def hstack(m1,m2):
    if type(m1[0])==type([]) and type(m2[0])==type([]):
        l1 = len(m1)
        m3 = [[0]*l1*2]*l1
        for row in range(l1):
            m3[row]=m1[row]+m2[row]
```

```
    else:
        m3=m1+m2
    return m3

def handler (event, context):
    m1=event[0]["m1"]
    m2=event[1]["m2"]
    new_m1,new_m2=reverse1(m1),reverse1(m2)
    new_m3 = hstack(new_m1,new_m2)
    m3=reverse1(new_m3)
    return m3
```

hstack矩阵水平合并 测试数据跟上面的vstack一样

```
# -*- coding:utf-8 -*-

import json
def handler (event, context):
    m1=event[0]["m1"]
    m2=event[1]["m2"]
    if type(m1[0])==type([]) and type(m2[0])==type([]):
        l1 = len(m1)
        m3 = [[0]*l1*2]*l1
        for row in range(l1):
            m3[row]=m1[row]+m2[row]

    else:
        m3=m1+m2
    return m3
```

numpy.sort 函数

```
# -*- coding:utf-8 -*-

import json
def reverse1(matrix):
    new_matrix = [list(col) for col in zip(*matrix)]
    return new_matrix

def handler (event, context):
    matrix=event[0]["m"]
    axis=event[1]["axis"]
    if axis == 1:
        for row in matrix:
            row.sort()
    else :
        matrix = reverse1(matrix)
        for row in matrix:
```

```
            row.sort()
        matrix = reverse1(matrix)
    return matrix
```

reverse矩阵翻转

```
# -*- coding:utf-8 -*-

import json
def handler (event, context):
    matrix = event
    print("原矩阵为:{}".format(matrix))
    new_matrix = [list(col) for col in zip(*matrix)]
    print("矩阵反转后为:{}".format(new_matrix))
    return new_matrix
```

matrix_weight_average 求矩阵平均数，带权

```
# -*- coding:utf-8 -*-

import json
def handler (event, context):
    m=event[0]['m']
    w=event[1]['w']
    if  type(m[0])==type([]):
        for i in range(len(m)):
            for j in range(len(m[0])):
                m[i][j] = m[i][j]*w[i][j]
        return sum(sum(row)for row in m )/ (len(m)*len(m[0]))
    else:
        for i in range(len(m)):
            m[i]=m[i]*w[i]
        return sum(m)/len(m)
```

mean 求矩阵平均值

```
# -*- coding:utf-8 -*-

import json
def handler (rep, context):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
```

```
            "Content-Type": "application/json"
        }
    }
    '''
    if type(rep[0])==type([]):#二维
        print( sum(sum(row) for row in rep) / (len(rep) * len(rep[0])) )
    else:#一维
        print( sum(rep)/len(rep) )
```

statistical_minimum 统计最小值

```
# -*- coding:utf-8 -*-

import json
def handler (event, context):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
    '''
    Min = min(event)
    print(Min)
```

statistical_maximum 统计最大值

```
# -*- coding:utf-8 -*-

import json
def handler (event, context):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
    '''
    Max = max(event)
    print(Max)
```

removeDuplicates 去除冗余值

```
# -*- coding:utf-8 -*-

import json
def handler (event, context):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
    '''
    res = []
    for each in event:
        if each not in res:
            res.append(each)
    return res
```

replace_outliers 检测离群值并替换

```
# -*- coding:utf-8 -*-
import json
import math
def handler (event, context):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
    '''
    limit = 5
    average = 0.0
    for each in event:
        average += each
    average = average/len(event)
    for each in event:
        if abs(each-average) > limit:
            event.remove(each)
    # print(average)
    return event
```

matrix_transpose 矩阵转置

```
# -*- coding:utf-8 -*-

import json
def handler (event, context):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
    '''
    row = len(event)
    col = len(event[0])
    matrix = []
    for i in range(col):
        temp = []
        for j in range(row):
            temp.append(event[j][i])
        matrix.append(temp)
    return matrix
```

dot_product 求矩阵点积

```
# -*- coding:utf-8 -*-

import json
def handler (a, b):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
    '''
    arow = len(a)
    acol = len(a[0])
    brow = len(b)
    bcol = len(b[0])
    if acol != brow:
        print("error")
    else:
        matrix = []
```

```
        for i in range(arow):
            temp = []
            for j in range(bcol):
                x = 0.0
                for k in range(acol):
                    x += a[i][k]*b[k][j]
                temp.append(x)
            matrix.append(temp)
        print(matrix)
```

variance 计算方差

```
# -*- coding:utf-8 -*-

import json
def handler (event, context):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
    '''
    ave = 0.0
    for each in event:
        ave += each
    ave = ave/len(event)
    var = 0.0
    for each in event:
        var += (each-ave)*(each-ave)
    var = var/len(event)
    return var
```

fillna 将空值填充为某一指定值

```
# -*- coding:utf-8 -*-

import json
def handler (event, context=100.0):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
```

```
            "Content-Type": "application/json"
        }
    }
    '''
    for i in range(len(event)):
        for j in range(len(event[0])):
            if event[i][j] == 0:
                event[i][j] = context
    return event
```

reshape 视图变维（一维变二维）

```
# -*- coding:utf-8 -*-

import json
def handler (event, a, b):
    '''
    return {
        "statusCode": 200,
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
    '''
    if len(event) != a*b:
        print("error")
    else:
        count = 0
        matrix = []
        for i in range(a):
            temp = []
            for j in range(b):
                temp.append(event[count])
                count = count+1
            matrix.append(temp)
        return matrix
```

sum 矩阵求和

```
# -*- coding:utf-8 -*-

import json
def handler (event, context):
    '''
    return {
        "statusCode": 200,
```

```
        "isBase64Encoded": False,
        "body": json.dumps(event),
        "headers": {
            "Content-Type": "application/json"
        }
    }
    '''
    res = 0
    for i in range(len(event)):
        for j in range(len(event[0])):
            res += event[i][j]
    return res
```

det 求矩阵行列式

```
# -*- coding:utf-8 -*-
import json

def det(mat):
    A = 0
    n=len(mat)
    if n == 1:
        A = mat[0][0]
        return A
    if n == 2:
        A = mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0]
        return A

    b = [0 for x in range(n)]
    t1 = [[0 for x in range(n)] for y in range(n)]

    for i in range(n):
        b[i] = mat[i][0] * (-1) ** (i + 2)
    for i in range(n):
        for j in range(n - 1):
            t1[i][j] = mat[i][j + 1]
    for i in range(n):
        t2 = [[0 for x in range(n - 1)] for y in range(n - 1)]
        for j in range(n - 1):
            for k in range(n - 1):
                if j < i:
                    t2[j][k] = t1[j][k]
                else:
                    t2[j][k] = t1[j + 1][k]
        A += b[i] * det(t2)
    return A

def handler (event, context):
    return det(event)
```

附录2 添加华为云API Explorer库

```
pip install huaweicloudsdkcore==3.0.85
pip install huaweicloudsdkfunctiongraph==3.0.85
```

附录3 本地测试代码

AsyncInvokeFunction 异步执行函数

```
# coding: utf-8

from huaweicloudsdkcore.auth.credentials import BasicCredentials
from huaweicloudsdkfunctiongraph.v2.region.functiongraph_region import
FunctionGraphRegion
from huaweicloudsdkcore.exceptions import exceptions
from huaweicloudsdkfunctiongraph.v2 import *

if __name__ == "__main__":
    ak = "JAXDAYAIYC6R34XI1YCA"
    sk = "iXPYREXxXjUXwI1OPbWJh1hNBlSP8460vkPXad51"

    credentials = BasicCredentials(ak, sk) \

    client = FunctionGraphClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(FunctionGraphRegion.value_of("cn-north-4")) \
        .build()

    try:
        request = AsyncInvokeFunctionRequest()
        request.function_urn = "urn:fss:cn-north-
4:0eb082c61280f3aa2fd4c0167562c3a9:function:Virtualization_and_Cloud_computing:mat
rix_mean:latest"
        response = client.async_invoke_function(request)
        print(response)
    except exceptions.ClientRequestException as e:
        print(e.status_code)
        print(e.request_id)
        print(e.error_code)
        print(e.error_msg)
```

DeleteFunction 删除函数

```
# coding: utf-8

from huaweicloudsdkcore.auth.credentials import BasicCredentials
```

```
from huaweicloudsdkfunctiongraph.v2.region.functiongraph_region import
FunctionGraphRegion
from huaweicloudsdkcore.exceptions import exceptions
from huaweicloudsdkfunctiongraph.v2 import *

if __name__ == "__main__":
    ak = "JAXDAYAIYC6R34XI1YCA"
    sk = "iXPYREXxXjUXwI1OPbWJh1hNBlSP8460vkPXad51"

    credentials = BasicCredentials(ak, sk) \

    client = FunctionGraphClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(FunctionGraphRegion.value_of("cn-north-4")) \
        .build()

    try:
        request = AsyncInvokeFunctionRequest()
        request.function_urn = "urn:fss:cn-north-
4:0eb082c61280f3aa2fd4c0167562c3a9:function:Virtualization_and_Cloud_computing:mat
rix_mean:latest"
        response = client.async_invoke_function(request)
        print(response)
    except exceptions.ClientRequestException as e:
        print(e.status_code)
        print(e.request_id)
        print(e.error_code)
        print(e.error_msg)
```

ListFunctions 查询函数列表

```
# coding: utf-8

from huaweicloudsdkcore.auth.credentials import BasicCredentials
from huaweicloudsdkfunctiongraph.v2.region.functiongraph_region import
FunctionGraphRegion
from huaweicloudsdkcore.exceptions import exceptions
from huaweicloudsdkfunctiongraph.v2 import *

if __name__ == "__main__":
    ak = "JAXDAYAIYC6R34XI1YCA"
    sk = "iXPYREXxXjUXwI1OPbWJh1hNBlSP8460vkPXad51"

    credentials = BasicCredentials(ak, sk) \

    client = FunctionGraphClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(FunctionGraphRegion.value_of("cn-north-4")) \
        .build()

    try:
```

```
        request = ListFunctionsRequest()
        response = client.list_functions(request)
        print(response)
    except exceptions.ClientRequestException as e:
        print(e.status_code)
        print(e.request_id)
        print(e.error_code)
        print(e.error_msg)
```

ListWorkflow 查询函数工作流列表

```
# coding: utf-8

from huaweicloudsdkcore.auth.credentials import BasicCredentials
from huaweicloudsdkfunctiongraph.v2.region.functiongraph_region import
FunctionGraphRegion
from huaweicloudsdkcore.exceptions import exceptions
from huaweicloudsdkfunctiongraph.v2 import *

if __name__ == "__main__":
    ak = "JAXDAYAIYC6R34XI1YCA"
    sk = "iXPYREXxXjUXwI1OPbWJh1hNBlSP8460vkPXad51"

    credentials = BasicCredentials(ak, sk) \

    client = FunctionGraphClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(FunctionGraphRegion.value_of("cn-north-4")) \
        .build()

    try:
        request = ListWorkflowsRequest()
        response = client.list_workflows(request)
        print(response)
    except exceptions.ClientRequestException as e:
        print(e.status_code)
        print(e.request_id)
        print(e.error_code)
        print(e.error_msg)
```

ShowFunctionCode 显示函数代码

```
# coding: utf-8

from huaweicloudsdkcore.auth.credentials import BasicCredentials
from huaweicloudsdkfunctiongraph.v2.region.functiongraph_region import
FunctionGraphRegion
from huaweicloudsdkcore.exceptions import exceptions
```

```
from huaweicloudsdkfunctiongraph.v2 import *

if __name__ == "__main__":
    ak = "JAXDAYAIYC6R34XI1YCA"
    sk = "iXPYREXxXjUXwI1OPbWJh1hNBlSP8460vkPXad51"

    credentials = BasicCredentials(ak, sk) \

    client = FunctionGraphClient.new_builder() \
        .with_credentials(credentials) \
        .with_region(FunctionGraphRegion.value_of("cn-north-4")) \
        .build()

    try:
        request = ShowFunctionCodeRequest()
        request.function_urn = "urn:fss:cn-north-
4:0eb082c61280f3aa2fd4c0167562c3a9:function:Virtualization_and_Cloud_computing:mat
rix_mean:latest"
        response = client.show_function_code(request)
        print(response)
    except exceptions.ClientRequestException as e:
        print(e.status_code)
        print(e.request_id)
        print(e.error_code)
        print(e.error_msg)
```