可信计算服务

开发指南

产品文档

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

开发指南

最近更新时间：2023-02-01 17:06:23

简介

腾讯云区块链可信计算服务（TBCC）支持用户自定义的算法模型。用户可以根据业务场景需求开发契合自身情况的算法模型，上传到 TBCC 平台的可信计算节点中，参与密文数据的处理。TBCC 支持类似 Spark 的分布式计算框架，而 TEE 版 Spark 的应用开发流程和标准版 Spark 有所不同，需要感知到可信计算平台定义的任务。本文以 示例项目 讲解应用开发过程。
本说明中模型开发语言为 Scala。

环境

docker 镜像：sparkee-builder.tar，其中软件具体版本如下：

- Oracle JDK：1.8.0_192
- Scala：2.12.10

使用以下命令加载镜像：

```
docker load < sparkee-builder.tar
```

开发

根据业务需求开发业务逻辑，样例代码参见已打包的 scala 项目 。

模型开发样例及数据结构说明

模型样例

本章节提供一个模型样例代码。该样例模型的功能是将数据源提供的数据转存入使用者的数据库中，过程中对数据没有任何其他处理。该样例模型提供了从任务信息中提取数据源的数据地址、使用者的数据库信息的能力，并包含读取数据、写入数据的逻辑。开发者可以尝试在读取、写入数据这两个步骤之间添加计算逻辑。
样例代码如下：

```
/**
 * 此应用实现将任务第一笔输入指定的 MySQL 表转储到第一笔输出指定的 MySQL 表。
 */
package com.woa.git.xiangminli.helloworld

import java.util.Properties

import org.apache.spark.sql.{SaveMode, SparkSession}
```

```
object HelloWorldMySQL extends App {

  val ss = SparkSession
    .builder()
    .appName("Hello World with MySQL by XML")
    .getOrCreate()

  // 获取任务输入/输出参数组成的 TaskParams 结构
  // TaskParams 的详情参见《3.1. 核心数据结构》
  val params = ss.sparkContext.getConf.teeConf.get.taskParams()

  // 打印输入配置信息用于调试
  for (v <- params.inputs) {
    println(
      s"- uri: ${v.uri.getUri()}, name: ${v.uri.getName()}, token: '${v.authToken}'",
    )

    // 强制要求指定数据源 CA 证书
    val certPem = v.caCertChainPem.getOrElse(
      throw new IllegalStateException("miss input trust cert key store"),
    )
    println(s"  CA:\n${certPem}")
  }

  assert(!params.inputs.isEmpty, "miss inputs")

  // 解析数据库地址和表名
  val (dsn, tableName) = {
    val uri = params.inputs(0).uri
    (s"jdbc:${uri.getUri()}", uri.getName())
  }
  println(s"dsn: $dsn, table: $tableName")

  // 从第一笔输入的配置信息解析出账号、密码和 MySQL 服务端的 CA 证书
  // CA 证书用于建立 TLS 链接过程验证 MySQL 服务证书的合法性
  val (user, password, inTrustCertKeyStore) = {
    val input = params.inputs(0)
    val token = input.authToken.split(":") // 假设 authToken 的形式为 "用户名：密码"
    if (token.length != 2) {
      throw new IllegalStateException(s"${input.authToken} is malformed")
    }

    (token(0), token(1), input.trustCertKeyStore.get)
  }
```

```
// 打印信息用于调试。发布的应用应删除打印这些敏感信息的打印语句。
println(s"user: '$user', password: '$password'")
println(
  s"trustCertificateKeyStoreUrl: '${inTrustCertKeyStore.url}'," +
    s" trustCertificateKeyStorePassword: '${inTrustCertKeyStore.password}'",
)

val props = new Properties
props.put("user", user)
props.put("password", password)
// 添加验证服务端 TLS 通信配置
// @warn 此步骤对保护数据源安全至关重要，能够有效抵御中间人攻击
props.put("sslMode", "VERIFY_CA")
// 设置服务端 JKS 格式 CA 证书路径
// @warn 此步骤对保护数据源安全至关重要，能够有效抵御中间人攻击
props.put("trustCertificateKeyStoreUrl", inTrustCertKeyStore.url)
// 设置解密服务端 CA 证书的口令
props.put("trustCertificateKeyStorePassword", inTrustCertKeyStore.password)

// 从数据库读取输入
val inDF = ss.read.jdbc(dsn, tableName, props)

// 按需对 inDF 执行具体计算
// 例如，打印 inDF 对应的表结构信息
inDF.printSchema()
// 例如，打印 inDF 对应表的前 3 行
inDF.show(3)

// 还可以调用 DataFrame 支持的所有 API 对 inDF 进行其他转换
// 具体 API 参见这里
//   https://spark.apache.org/docs/3.1.3/api/scala/org/apache/spark/sql/Dataset.html
// 在此仅简单地将 inDF 复制给 outDF
val outDF = inDF

// 获取第一笔输出的配置信息 Output
val out = params.outputs(0)

// 构造结果输出地址
val outUrl = s"jdbc:${out.uri}"
println(s"save output to $outUrl in table ${out.name}")

// 将结果写出至具体数据库
// @note：此处假设结果和输入采用相同的账号和密码。正常情况下，应执行和解析输入数据库账号
// 密码用于读写数据库。
outDF.write.mode(SaveMode.Overwrite).jdbc(outUrl, out.name, props)
```

```
  println("done :)")
}
```

模型开发大体上是与原生 Spark 兼容的，可以查看 Spark 的开发文档 。与原生 Spark 不同的地方在于模型的使用的数据、参数，以及模型输出的结果的信息被统一到了样例代码中的 `SparkSession.sparkContext.getConf.teeConf.get.taskParams()` 这个 Spark 接口中。模型的使用者从前端调用模型时，数据、参数、结果输出信息都会在经过各参与方审核、签名后，传入 Spark。模型开发者可以使用这个接口去获取这些数据。

TaskParams 数据结构

```
case class TaskParams(inputs: Array[Input], outputs: Array[Output])
```

`taskParams()` 接口返回的数据结构主要有两个字段是开发者需要关注的：

- Inputs：Input 结构的数组，模型的输入数据、参数的相关信息，记录数据源的数据库信息、数据结构等。
- Outputs：Output 结构的数组，模型输出结果的目标数据库信息。

TaskParams 结构中，Inputs、Outputs 分别是数组，Inputs 数组中每一个 Input 类型成员都代表一份输入数据信息，Outputs 数组中每一个 Output 类型成员都代表一份输出结果的信息。同一个模型是可能需要整合多份输入数据后进行计算，也可能需要把结果拆分并分发给多个结果使用方的。

Input 数据结构

```
case class Input(
    uri: ExtendedURI,
    authToken: String,
    seq: Array[Byte],
    caCertChainPem: Option[String],
    ...
) {
  /**
   * JKS 编码 caCertChainPem 所得可信证书链
   */
  val trustCertKeyStore: TrustCertKeyStore = ...
}
```

- uri：包含数据库地址和表名，数据库地址可通过其成员函数 getUri() 获取，表名可通过其成员函数 getName() 获取。
- authToken：数据库的用户名口令，传输时是密文，模型内接口获取时，会在可信环境内被解密。该解密过程不需要模型开发者感知。这个字段的数据格式为 “用户名:密码” 形式的字符串，需要模型开发者拆分。

- seq：该字段依业务逻辑而定，用于指定数据的用途。模型可以固定一套自己的数据用途标识。例如一个计算除法 A/B 的模型，那么数据 A 的 seq 字段可以规定为 “被除数”，B 的 seq 字段可以规定为 “除数”，模型逻辑可以通过 seq 字段确定数据该用于替换模型公式中的哪一个变量。
- caCertChainPem：与数据源数据库做 TLS 通信的服务端 CA 证书链。

Output 数据结构

```
case class Output(
    uri: String,
    authToken: String,
    name: String,
    seq: Array[Byte],
    caCertChainPem: Option[String],
    ...
) {
  /**
   * JKS 编码 caCertChainPem 所得可信证书链
   */
  val trustCertKeyStore: TrustCertKeyStore = ...
}
```

输出的目标数据存储信息结构与 Input 相似。

- uri：包含数据库地址和表名，数据库地址可通过其成员函数 getUri() 获取，表名可通过其成员函数 getName() 获取。
- authToken：数据库的用户名口令，传输时是密文，模型内接口获取时，会在可信环境内被解密。该解密过程不需要模型开发者感知。这个字段的数据格式为 “用户名:密码” 形式的字符串，需要模型开发者拆分。
- seq：该字段依业务逻辑而定，如果计算结果的各个不同部分需要发送到不同的结果使用者，则可以用这个字段标识。
- caCertChainPem：与数据源数据库做 TLS 通信的服务端 CA 证书链。

与输入、输出数据库建立安全信道所需的证书链相关数据结构

```
case class TrustCertKeyStore(certChainPem: String, password: String, mType: String = "JK
  val url: String = ...
}
```

- certChainPem：指定 PEM 格式的 CA 证书。
- password：加密 key store 的口令
- mType：指定 key store 类型

根据 mType 指定类型编码 certChainPem 和 password 生成并加密的 key store 的文件路径。详情见 文档。

> ⚠ 注意：
>
> - **模型依赖处理**
>
> 当模型逻辑需要依赖外部 jar 包时，在编译模型 jar 包时需要使用 sbt assembly 命令将模型依赖包打进模型 jar 包中。
> - **连接数据库**
>
> TEE 版 spark 目前支持连接 mysql 和达梦数据库，在连接数据库时，不需要在模型代码中显式添加 jdbc 驱动。

模型构建

1. 模型构建

   用以下命令启动容器作为模型构建环境。其中 PWD 当前目录需为模型 Scala 项目的根目录。

```
repo_tag=tbcchub.tencentcloudcr.com/prod/teepot/sparkee-builder:c7277f4503-3.1.3-

docker run -it --rm -v $PWD:/workspace -w /workspace $repo_tag \
  bash -c "ln -sf /opt/spark /workspace/lib && bash"
```

2. 编译 JAR 包

   在启动的容器中执行以下命令：

```
sbt package
```

3. 打包模型应用包

   执行以下脚本：

```
set -e

cd $(dirname ${BASH_SOURCE[0]})
workdir=$PWD

# 此脚本用于打包编译后的项目以发布至模型仓库。

name=$(grep '^name' build.sbt | head -1 | awk -F'"' '{print $2}')
version=$(grep '^version' build.sbt | head -1 | awk -F'"' '{print $2}')

app_jar_path=$workdir/target/scala-2.12/${name}_2.12-$version.jar
```

```
# 检查应用是否已被编译生成
if [ ! -f $app_jar_path ]; then
  echo "请先编译项目生成应用 JAR 包 $app_jar_path :("
  exit 1
fi

outdir=$workdir/target/output

rm -rf $outdir
mkdir -p $outdir

# 拷贝应用 JAR 包
cp $app_jar_path $outdir/app.jar

# 拷贝源代码
mkdir $outdir/codes

cp -r project src build.sbt $outdir/codes/

# 清理 sbt 生成的垃圾文件
cd $outdir/codes/project
rm -rf $(ls . | grep -vE '(plugins.sbt|build.properties)') .bloop
cd -

# 拷贝静态文件，目前没有需要拷贝的静态文件
mkdir $outdir/static

cd $outdir

out=$workdir/target/app.tar.gz
tar -zcf $out *
```

打包好的应用包路径为当前项目根目录的 target/app.tar.gz。

4. 可以从产品页面上传打包好的模型到系统仓库中，构建计算任务时即可选取这个模型。