ALIBABA CLOUD

Alibaba Cloud

ApsaraDB for HBase

HBase Ganos Spatio Temporal Engine

Document Version: 20200902

Alibaba Cloud

Legal disclaimer

Document conventions

Style	Description	Example
⚠ Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	⚠ Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
? Note	A note indicates supplemental instructions, best practices, tips, and other content.	? Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings> Network> Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

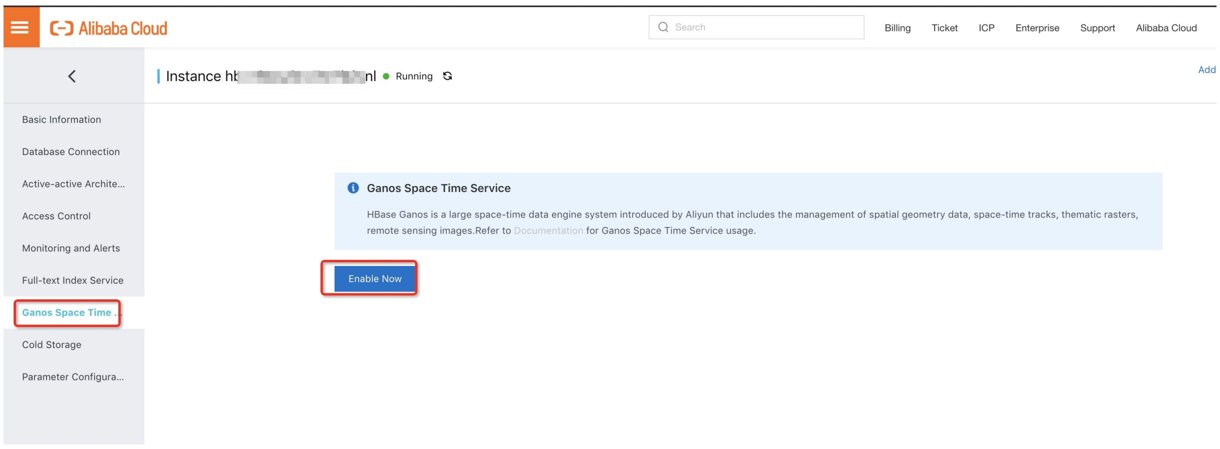
Table of Contents

1.HBase Ganos introduction

2.Activate HBase Ganos

HBase Ganos is a component of ApsaraDB for HBase. It is free of charge. Only ApsaraDB for HBase is charged. After you purchase ApsaraDB for HBase, you can activate HBase Ganos in the ApsaraDB for HBase console.

How to activate HBase Ganos



Ganos Space Time Service

HBase Ganos is a large space-time data engine system introduced by Aliyun that includes the management of spatial geometry data, space-time tracks, thematic rasters, remote sensing images.Refer to Documentation for Ganos Space Time Service usage.

3.FAQ

Can I use HBase Ganos to store and retrieve geo-fencing data?

Geo-fencing data is "Plane" data in GeoJSON standard data format. Geo-fencing data in HBase Ganos is stored and retrieved in the same way as "Point" and "Line" data.

```
{
"features": [
{
"geometry": {
"coordinates": [
[0,0],
[0,1],
[1,1],
[1,0],
[0,0]
],
"type": "Polygon"
},
"id": "polygon_feature",
"properties": {
"name": "shanghai"
},
"type": "Feature"
}
],
"type": "FeatureCollection"
}
```

How can I create an ID?

If you use IDs as indexes, you must ensure that all the IDs are unique. We recommend that you create IDs in the format of biz_id + timestamp.

What date format do I choose to create temporal indexes?

Temporal data in HBase Ganos is stored in the properties attribute. Temporal data is saved in timestamp format. A timestamp accurate to milliseconds is a 13-digit integer, for example, 1542628013000. A string type date is stored in the following format: 2018/11/19 19:46:53. If the date contains time zone information, it is stored in the following format: 2018/11/19T19:46:53Z. As shown in the following example, the dtg field in the properties attribute is a date type field. The name of the field can be customized. You can specify the date=properties.dtg parameter to create indexes on dates. date is the key of the parameter. properties.dtg is a date field in GeoJSON.

```
{
"type": "Feature",
"geometry": {
"type": "Point",
"coordinates": [125.6, 10.1]
},
"properties": {
"name": "Dinagat Islands",
"dtg": 1536041936000,
"id": "1"
}
}
```

What are the possible reasons that cause slow temporal data queries?

- Verify that indexes are created on the temporal data. GeoMesa is required if you want to create indexes on temporal data.
- Verify that the date format of the query conditions is the same as that of the temporal data to be queried. Date type fields are stored in timestamp format (integer type, for example, 1542628013000). If the query conditions are of the string type, for example, 2018/11/19 19:46:53, the temporal indexes become invalid.

How can I update data? In HBase, when the same {row, column} is queried, only data of the latest version is returned. Therefore, data updating is equivalent to data overwriting. Example: insert rows 1:a,b,c and 1:a1,b1 in sequence. Only 1:a1,b1 is stored. Insert rows 1:a,b,c and 1:a1,b1,c1,d,e,f in sequence. Only 1:a1,b1,c1,d,e,f is stored.

How can I perform elevation queries?

Spatio-temporal indexes are two-dimensional, including spatial coordinates and temporal coordinates. Therefore, you cannot use spatio-temporal indexes to perform elevation queries. You can store elevation information in attributes and create indexes on the elevation information.

What are the considerations when I call the RESTful API to create a DataStore?

When you call the RESTful API to create a DataStore, you must specify the HBase catalog endpoint and ZooKeeper endpoint. These two parameters determine the location where the data is stored. Unlike data sources in other databases, a DataStore in this topic is the name of a configuration. You can use this name to find the HBase catalog and ZooKeeper service. If the HBase catalog and ZooKeeper service use different endpoints, we recommend that you create different DataStores for them. Otherwise, we recommend that you use the DataStore for both of them.

Why does the system return the same result when I query an index with the same name in two different DataStores? A DataStore is an alias. The endpoints of the HBase catalog and ZooKeeper service determine the DataStore that you query. If two different DataStores are using the same HBase catalog and ZooKeeper endpoints, the indexes with the same name in the two DataStores refer to the same index. As a result, the system returns the same query result.

4.Spatio-temporal geometries

4.1. Terms

This topic introduces terms about HBase Ganos.

Spatio-temporal geometries

Spatio-temporal geometries in HBase Ganos include the following:

- Objects of spatio-temporal geometries
  - Vector data, such as the Point, Line, and Plane features.
  - Spatial data (spatial trajectory data) consists of vector data and temporal attributes.
- Operations related to objects of spatio-temporal geometries, such as spatial relation verification.

Spatio-temporal indexes

HBase Ganos supports high-performance data queries, which is based on spatio-temporal indexes. Spatio-temporal indexes are stored as rowkeys in HBase Ganos. For more information, see Create an index table.

Spatio-temporal relations

A spatio-temporal relation describes the locations of two geometries in time and space in relation to each other. Typical spatio-temporal relations include INTERSECT, DISJOINT, OVERLAP, and CONTAIN. A geofence usually refers to the relations of the geofences of an area to an object (Point, Line, or Plane). If the object lies inside the geofences, the area contains the object. If the object lies outside the geofences, the area and object are disjoint. HBase Ganos uses Common Query Language (CQL) to define spatio-temporal relations. For more information, see Create an index table.

OGC

OGC is short for Open Geospatial Consortium. It is an international voluntary consensus standards organization. It has created a set of standards for data models and related operations. Geographic information system (GIS) vendors develop geographic information systems based on these standards to ensure that their systems can interact with spatial databases.

GeoTools

The SDK of HBase Ganos is based on GeoTools. GeoTools is a library that provides tools for processing geospatial data. It complies with OGC standards, and can interact with OGC standard data models and interfaces. Many geographic information systems are developed based on GeoTools. For more information, visit https://geotools.org/.

Geometries

A geometry in OGC is defined as a spatial object, such as a point, a line, or a plane. A geometry only contains location information of a spatial object. It does not contain any attribute information. GeoTools provides tools for you to build geometries when you use HBase Ganos.

SimpleFeatures

Simple features. A SimpleFeature contains geometries and other attributes. A trajectory point is a SimpleFeature. The SimpleFeature contains the location information of the trajectory point, temporal information, and other attributes. The temporal information is also a part of the attribute information.

CQL & ECQL

Common Query Language (CQL) is a language defined by OGC to support geospatial data queries. Extended Common Query Language (ECQL) is an extension of CQL and is more powerful than CQL. In most cases, ECQL is used to define filters by using SQL-like where clauses based on the well-known text representation. For more information, see Query spatio-temporal objects. CQL described in this user guide refers to ECQL.

WKT

Well-known text (WKT) is a text markup language defined by OGC to describe spatial objects. For example, a point can be represented by using POINT(0,0). WKT is commonly used in query statements and is much easier to read. CQL and ECQL also use WKT to describe spatial objects. For more information about WKT, visit http://www.opengeospatial.org/standards/wkt-crs.

WKB

Well-known binary (WKB) is a language defined by OGC to mark up geometries by using byte arrays. WKB data is smaller than WKT data. You can use WKT for data transmission. GeoTools provides a tool for you to transform data between WKB and WKT.

4.2. Quick start

After you activate HBase Ganos, you can call the Ganos API to start your development work. For more information about how to activate HBase Ganos, see .

Sample code

1. Download the sample code

ganos-sample

After you download the sample code, modify the dependencies in the pom.xml file as needed. For more information, see the Dependencies section.

2. Sample code descriptions

2.1. trajectory-sample module

This module uses automatic identification system (AIS) data as the sample data to demonstrate how to import and query trajectory data.

- Main: the input of the demo.
- Writer: writes data into HBase Ganos. The entire procedure includes these steps: connect to HBase Ganos, create a schema (spatio-temporal index table), read sample data, and write sample data to the spatio-temporal index table.
- Reader: a common query interface. Examples are provided for attribute queries (equal-value queries, prefix queries, and suffix queries), spatial data queries, and spatio-temporal data queries.

2.2. spark-sample module

This module uses automatic identification system (AIS) data as the sample data to demonstrate how to import and query trajectory data based on Spark.

- Writer: writes data into HBase Ganos. The entire procedure includes these steps: connect to HBase Ganos, create a schema (spatio-temporal index table), read sample data, and write sample data to the spatio-temporal index table.
- SparkDemo: demonstrates how to read data from HBase Ganos, transform the data to Spark DataFrames, and query spatio-temporal data based on SparkSQL.

Dependencies

You can choose between the GeoMesa and Ganos clients. The Ganos client supports more features.

GeoMesa client

HBase Ganos is compatible with the open source GeoMesa client. You can directly integrate the GeoMesa client without using Ganos features.

Notice The GeoMesa client currently only supports Ganos based on HBase 1.x.

The configuration is as follows:

```
<properties>
<! --Note: Enter the version of the HBase client -->
<alihbase.version>1.3.1</alihbase.version>
</properties>

<dependency>
<groupId>org.locationtech.geomesa</groupId>
<artifactId>geomesa-hbase-datastore_2.11</artifactId>
<version>2.2.1</version>
<exclusions>
<exclusion>
<groupId>com.google.protobuf</groupId>
<artifactId>protobuf-java</artifactId>
</exclusion>
</exclusions>
</dependency>
<! -- alihbase as a dependency -->
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-client</artifactId>
<version>${alihbase.version}</version>
<exclusions>
<exclusion>
```

```
<artifactId>com.google.guava</artifactId>
<groupId>guava</groupId>
</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-server</artifactId>
<version>${alihbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-common</artifactId>
<version>${alihbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-protocol</artifactId>
<version>${alihbase.version}</version>
</dependency>
```

Ganos client

- Ganos client based on HBase 1.x

Notice The dependency of the current Ganos client version has not been released to the maven library. You must manually download and integrate the dependency. Click ganos-hbase-distributed-runtime_2.11-2.2.1-2.1.0.jar to download the dependency, and configure the settings as follows:

```
<properties>
<! -- Note: Enter the version of the HBase client -->
<alihbase.version>1.3.1</alihbase.version>
</properties>

<dependency>
<groupId>com.aliyun.tst.ganos</groupId>
<artifactId>ganos-geomesa-hbase15</artifactId>
<version>2.0.0</version>
<scope>system</scope>
<systemPath>/ganos-jar-path</systemPath>
</dependency>
<! -- alihbase as a dependency -->
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-client</artifactId>
<version>${alihbase.version}</version>
<exclusions>
<exclusion>
<artifactId>com.google.guava</artifactId>
<groupId>guava</groupId>
</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-server</artifactId>
<version>${alihbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-common</artifactId>
<version>${alihbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-protocol</artifactId>
<version>${alihbase.version}</version>
</dependency>
```

- Ganos client based on HBase 2.x

Notice The dependency of the current Ganos client version has not been released to the maven library. You must manually download and integrate the dependency. Click ganos-hbase-distributed-runtime_2.11-2.2.1-2.5.0.jar to download the dependency, and configure the settings as follows:

```
<properties>
<! -- Note: Enter the version of the HBase client -->
<alihbase.version>2.0.0</alihbase.version>
</properties>

<dependency>
<groupId>com.aliyun.tst.ganos</groupId>
<artifactId>ganos-geomesa-hbase20</artifactId>
<version>2.5.0</version>
<scope>system</scope>
<systemPath>/ganos-jar-path</systemPath>
</dependency>
<! -- alihbase as a dependency-->
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-client</artifactId>
<version>${alihbase.version}</version>
<exclusions>
<exclusion>
<artifactId>com.google.guava</artifactId>
<groupId>guava</groupId>
</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-server</artifactId>
<version>${alihbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-common</artifactId>
<version>${alihbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-protocol</artifactId>
<version>${alihbase.version}</version>
</dependency>
```

4.3. Create an index table

Ganos is developed with multiple built-in spatial indexes. Users only need to input spatial data at the frontend, and then specify the index to be created. They do not need to worry about how HBase key-value pairs are designed and built.

Before you import data, you must define the schema of the index table.

Index types

Ganos supports four types of indexes to meet the requirements of different scenarios. **You can choose to create all of the supported types of indexes or only some of them.** In a scenario that only involves surrounding object queries, you only need to create Z2 indexes.

ID indexes

ID indexes are applicable to scenarios where feature IDs (FIDs) of spatial objects are used to query data. FIDs must be unique.

Z2/XZ2 indexes

Z2 and XZ2 indexes are suitable for spatial data queries, such as geofence queries and surrounding object queries. Z2 indexes are created on "Point" objects and XZ2 indexes are created on "Line" and "Plane" objects.

Z3/XZ3 indexes

Z3 and XZ3 indexes are suitable for spatial-temporal data queries, for example, historical trajectory queries within a specified spatial range or temporal range. Z3 indexes are created on "point" objects and XZ3 indexes are created on "line" and "plane" objects.

XYZ indexes

XYZ indexes are three-dimensional indexes including longitude, latitude, and elevation information. Currently, XYZ indexes are created on "Point" objects.

Attribute indexes

Attribute indexes are suitable for attribute queries.

Create an index

Follow these steps to create an index:

1. Use SimpleFeatureType to define a spatio-temporal schema.
    i. Specify the name of the schema. You can consider it the name of an index table.
    ii. Specify the content of the schema, which includes the definitions of attributes, geometries, and dates.
    iii. You can also customize the following information:
        a. Specify the compression algorithm.
        b. Specify whether to use the tiny well-known binary (TWKB) format.
        c. Create an index on a specified column.
2. Create geometries based on the defined spatio-temporal schema.

Note: You can reference the Write class in the sample code of the **Quick start** topic. The preceding settings are all wrapped in this class.

Example

As described in the preceding section, **you can use different types of indexes in Ganos or use only a specific type of indexes.** In a scenario that only involves surrounding object queries, you only need to create Z2 indexes. We recommend that you use all types of indexes to improve the overall performance of data queries. However, this consumes storage space. You can also choose index types based on the actual scenario to save storage space. Ganos is developed with built-in Hints. Hints are used to specify indexes when you define a SimpleFeatureType. The syntax is as follows:

```
//sft specifies a SimpleFeatureType instance.
sft.getUserData().put("geomesa.indices.enabled", "{index_name}:{col1}:{col2}:...,{index_name}:{col}");
```

- Assign one of the following constants to index_name: id, attr, z2, z3, xz2, xz3, and xyz.
- col specifies the columns defined in SimpleFeatureType.
- You can create one or more indexes, and separate them with commas (,).
- If you want to add multiple columns to an index, separate the index name and column names with colons (:).

In the following example, two types of indexes are created: z3 and attr. A z3 index is created on the start and dtg columns, and another z3 index is created on the end and dtg columns. An attr index is created on the name and dtg columns.

```
sft.getUserData().put("geomesa.indices.enabled", "z3:start:dtg,z3:end:dtg,attr:name:dtg");
```

4.4. Create a spatio-temporal object

In the trajectory scenario, a spatio-temporal object is considered as a trajectory point, which contains the spatial location, time, and other attributes. In the HBase Ganos SDK, a spatio-temporal object is mapped to a SimpleFeature object. After you define the schema of an index table, you can create a SimpleFeature object.

Create a SimpleFeature object

Each SimpleFeature contains an ID, Geometry, time information and other attributes. In the **Quick start** topic, the GanosClient class is encapsulated in the sample code to help you create indexes and trajectory points. We recommend that you use the sample code to simplify coding. If you use the native GeoTools API, you can create a SimpleFeature object by using the SimpleFeatureBuilder class, as shown in the following sample code:

```
SimpleFeatureType sft = .... ;
SimpleFeatureBuilder sfBuilder = new SimpleFeatureBuilder(sft);
builder.set("attribute name", attribute value);
...
builder.set("geom", Geometry); // Create a spatial object. Keep the default format of the geom attribu
te name.
SimpleFeature feature = builder.buildFeature(object_id + "_" + date.getTime());
```

Create a Geometry object

When you create a SimpleFeature object, you need to create a geometry object, as shown in the preceding sample code. The Geometry is contained in the SimpleFeature object. The Geometry stores the spatial information of features. These features include Point, Line, and Plane features, which can be considered as trajectory points in the trajectory scenario. The GeoTools API provides the GeometryFactory class to help you create Geometry objects. You can create Geometry objects in the following ways:

- Create a Geometry object by using a Coordinate object.

This object represents a coordinate point. We recommend that you set the Geometry object in this way.

- Create a Geometry object by using Well-known text (WKT).

WKT is a text markup language that is used to specify spatial objects. For example, the string "POINT (1 1)" specifies a Point feature with coordinates 1, 1. The string "LINESTRING(0 2, 2 0, 8 6)" specifies a Line feature that is composed of three coordinates. The string "POLYGON((20 10, 30 0, 40 10, 30 20, 20 10))" specifies a Plane feature, and the first coordinate and last coordinate of which are the same and form a circle. For more information, visit https://en.wikipedia.org/wiki/Well-known_text.

- Point feature

You can create Point features in the following ways:

- Create a Point feature by using a Coordinate object.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
Coordinate coord = new Coordinate(1, 1);
Point point = geometryFactory.createPoint(coord);
```

- Create a Point feature by using WKT.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
WKTReader reader = new WKTReader(geometryFactory);
Point point = (Point) reader.read("POINT (1 1)");
```

- Line feature

You can create Line features in the following ways:

- Create a Line feature by using a Coordinate object.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
Coordinate[] coords =
new Coordinate[] {new Coordinate(0, 2), new Coordinate(2, 0), new Coordinate(8, 6) };
LineString line = geometryFactory.createLineString(coordinates);
```

- Create a Line feature by using WKT.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
WKTReader reader = new WKTReader( geometryFactory );
LineString line = (LineString) reader.read("LINESTRING(0 2, 2 0, 8 6)");
```

- Plane feature

You can create Plane features in the following ways:

- Create a Plane feature by using a Coordinate object.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
Coordinate[] coords =
new Coordinate[] {new Coordinate(4, 0), new Coordinate(2, 2),
new Coordinate(4, 4), new Coordinate(6, 2), new Coordinate(4, 0) };
LinearRing ring = geometryFactory.createLinearRing( coords );
LinearRing holes[] = null; // use LinearRing[] to represent holes
Polygon polygon = geometryFactory.createPolygon(ring, holes );
```

- Create a Plane feature by using WKT.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory( null );
WKTReader reader = new WKTReader( geometryFactory );
Polygon polygon = (Polygon) reader.read("POLYGON((20 10, 30 0, 40 10, 30 20, 20 10))");
```

Configure other attributes

You can set other custom information by using the UserData column of the SimpleFeature. For example, HBase Ganos provides many built-in hints, as shown in the following section:

- Disable an index. For more information, see the Create an index and Disable an index topics.
- Use a custom FeatureID that saves the storage space.

```
SimpleFeature feature =...
feature.getUserData().put(Hints.USE_PROVIDED_FID, java.lang.Boolean.TRUE);
```

4.5. Insert spatio-temporal objects

Spatio-temporal objects are created after an index table is created. You can then call the writer to insert data to HBase Ganos.

Insert a single SimpleFeature

HBase Ganos uses the SimpleFeatureWriter in the GeoTools API to insert a single SimpleFeature. The SimpleFeatureWriter supports inserting transactions. You can use the getFeatureWriterAppend method of DataStore to get a FeatureWriter.

```
SimpleFeatureType sft = .... ;
SimpleFeatureWriter writer=(SimpleFeatureWriter)ds.getFeatureWriterAppend(sft.getTypeName(), Tr
ansaction.AUTO_COMMIT);
SimpleFeature toWrite=writer.next();
toWrite.setAttributes(feature.getAttributes());
toWrite.getUserData().putAll(feature.getUserData());
writer.write();
writer.close();
```

Insert multiple SimpleFeatures

HBase Ganos allows you to insert multiple SimpleFeatures at a time by using the SimpleFeatureStore class in the GeoTools API.

```
List<SimpleFeature> features=...
SimpleFeatureStore featureStore = (SimpleFeatureStore) ds.getFeatureSource(sft.getTypeName());
List<FeatureId> featureIds = featureStore.addFeatures(new ListFeatureCollection(sft,features));
```

For more information, see the sample code in the Quick start topic.

4.6. Query spatio-temporal objects

Each Ganos query is encapsulated in a Query object of GeoTools. Query conditions in a Query are constructed by using Common Query Language (CQL) statements. The Query object also supports other features. For example, you can use it to return a specified attribute column or sort the order of a specified attribute. Data is returned as a SimpleFeatureCollection. You can use an iterator to scan all SimpleFeatures and then parse the requested data. The following sections describe how to query spatio-temporal objects. For more information, see the GanosClient class of the sample code in the **Quick start** topic.

Construct query conditions

Query conditions are the Common Query Language (CQL) representation of filters. CQL is similar to the WHERE clause in SQL. You can use query conditions to filter data.

Spatial queries

- Spatial queries support the following predicates. "Expression" in the following table represents a geometry in the WKT representation, which can be a point, line, or plane. | Syntax | Description | | --- | --- | | INTERSECTS(Expression , Expression) | Indicates whether two geometries are **intersected**.For example, it can indicate whether two roads are intersected. | | DISJOINT(Expression , Expression) | Indicates whether two geometries are **disjoint**. | | CONTAINS(Expression , Expression) | Indicates whether the first geometry **contains** the second geometry. | | WITHIN(Expression , Expression) | It is the contrary of the CONTAINS predicate. It indicates whether the first geometry is within the second geometry. | | TOUCHES(Expression , Expression) | Indicates whether two geometries are **touched**. If they are touched, they have at least one touch point. | | CROSSES(Expression , Expression) | Indicates whether two geometries cross each other. The two geometries cross each other if only a part of them are in common. | |

BBOX (Expression , Xmin , Ymin , Xmax , Ymax [, CRS]) | Indicates whether the geometry [Expression] is intersected by the quadrilateral plane [Xmin,Ymin,Xmax,Ymax].

A coordinate reference system (CRS) is a string that contains spatial reference system (SRS) code. For example, 'EPSG:1234'. By default, the CRS of the queried graph is used. |

- Example 1: Query all geometries, such as restaurants, vessels, and vehicles within the spatial range (120E, 30N, 130E, 40N):

```
DataStore ds = DataStoreFinder.getDataStore(params);


SimpleFeatureType schema=...


String stFilter = "bbox(geom, 120,30,130,40)"


Query query = new Query(schema, ECQL.toFilter(stFilter));


SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

- Example 2: Query all vehicles within a radius of 5 kilometers.
  - First construct a geometry (plane) within a radius of 5 kilometers, for example, (46.9 48.9, 47.1 48.9, 47.1 49.1, 46.9 49.1, 46.9 48.9).
  - Use the CONTAINS predicate to retrieve all objects.

```
String stFilter = "contains('POLYGON ((46.9 48.9, 47.1 48.9, 47.1 49.1, 46.9 49.1, 46.9 48.9))', geom)


Query query = new Query(schema, ECQL.toFilter(stFilter));
```

Spatio-temporal queries

Spatio-temporal queries are also widely used in different scenarios. For example, you may want to query the trajectories of all vehicles within the range of (20E, 30N, 130E, 40N) in the last two hours. **Temporal queries** in HBase Ganos support the following predicates. "Expression" in the following table specifies the name of the date column. "Time" specifies the date string in UTC format.

Syntax	Description
Expression BEFORE Time	Indicates any date earlier than "Time".
Expression BEFORE OR DURING Time Period	Indicates any date earlier than "Time Period" or within "Time Period".
Expression DURING Time Period	Indicates any date within "Time Period".
Expression DURING OR AFTER Time Period	Indicates any date within "Time Period" or later than "Time Period".

Syntax	Description
Expression AFTER Time	Indicates any date later than "Time".

HBase Ganos allows you to use the following syntax to specify a time period:

Syntax	Description
Time / Time	A time period defined by the beginning time and end time.
Duration / Time	A time period earlier than "Time".
Time / Duration	A time period later than "Time".

- Example: If you want to query all vehicle trajectories located within (120E, 30N, 130E, 40N) from 2014-01-01T11:45:00 to 2014-01-01T12:15:00, you must use a combination of spatial queries and temporal queries:

```
String stFilter = "bbox(geom, 120,30,130,40) AND dtg DURING 2014-01-01T11:45:00.000Z/2014-01-01T12:
15:00.000Z";

Query query = new Query(schema, ECQL.toFilter(stFilter));

SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

Attribute queries

- Supported comparison operators are =, <>, >, >=, <, and <=. If you want to select a city with a population greater than 15 million, set the query condition to PERSONS>15000000. PERSONS represents the population field.
- BETWEEN is used to specify a specific range, for example, PERSONS BETWEEN 1000000 AND 3000000.
- The comparison operators support string values. You can specify a string value on the right side of the equal sign (=). For example, CITY_NAME='Beijing' specifies the Beijing city. You can also use the LIKE operator. For example, CITY_NAME LIKE 'N%' specifies cities whose names start with the letter "N".
- You can compare two attributes. For example, MALE > FEMALE specifies cities where the population of males is greater than females.
- Supported arithmetic operators are +, -, *, and /. For example, UNEMPLOY/(EMPLOYED + UNEMPLOY) > 0.07 specifies all cities with an unemployment rate greater than 7%.
- You can use the IN operator to specify attributes whose values are within the specified range. For example, use ID IN ('cities.1', 'cities.12') or CITY_NAME IN ('Beijing', 'Shanghai', 'Guangzhou') to query cities whose names contain the specified values.
- You can use any filter functions in GeoServer. For example, you can use strToLowerCase(CITY_NAME) like '%m%' to query cities whose names contain the letter "M" or "m".
- You can use the geometric filter bounding box (BBOX). For example, use BBOX(the_geom, -90,

40, -60, 45) to query cities whose geographic coordinates are within the spatial rage of (-90, 40, -60, 45).

```
String filter = " name = 'bob'"

val q = new Query(sft.getTypeName, ECQL.toFilter(filter))

SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

In the preceding example, the values in the name column are filtered.

Specify the names of the returned columns

You can configure the parameters of the Query object to specify the columns to be returned. Example:

```
String[] returnFields=... //The names of the returned columns.

Query query = new Query(schema, ECQL.toFilter(ecqlPredicate));

query.setPropertyNames(returnFields);

SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

Specify a sorting method

You can configure the parameters of the SortBy object to specify the columns to be sorted. Example:

```
String sortField=... //Specify the names of the columns to be sorted.

FilterFactory2 ff = CommonFactoryFinder.getFilterFactory2();

SortBy[] sort = new SortBy[]{ff.sort(sortField, order)};

query.setSortBy(sort);

SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

4.7. Delete spatio-temporal objects

In most cases, SimpleFeatures are continuously "appended" to a database. For example, vehicle trajectory data is written into a database every few seconds. You may want to delete a SimpleFeature in some scenarios. Example:

- In an Extract-Transform-Load (ETL) processing system, the system may need to mask and aggregate data by deleting sensitive or unused data before it imports the data into databases.
- The system may also need to delete the data that contains errors.

HBase Ganos allows you to delete a SimpleFeature by specifying the FID of the SimpleFeature. Note: An FID is a unique identifier of a SimpleFeature.

Example

The following example shows how to use the GeoTools interface to delete a SimpleFeature with a specified FID. For more information, see the sample code in the Quick start topic.

- The input parameter simpleFeatureId specifies the ID of the SimpleFeature to which the data is written. We recommend that you use a user-readable and unique ID, for example, objectId+timeStamp. This makes it easy for you to construct an FID when you want to delete a SimpleFeature.
- Build a FeatureWriter, and specify the FID of the target SimpleFeature. You can set the transaction type to AUTO_COMMIT.
- Use the remove function of the FeatureWriter to delete the SimpleFeature.
- Close the FeatureWriter

```
public void removeById(String schema,String simpleFeatureId) throws Exception{
FilterFactory2 ff = CommonFactoryFinder.getFilterFactory2();
FeatureWriter<SimpleFeatureType, SimpleFeature> writer =
ds.getFeatureWriter(schema, ff.id(ff.featureId(simpleFeatureId)), Transaction.AUTO_COMMIT);
while (writer.hasNext()) {
writer.next();
writer.remove();
}
writer.close();
}
```

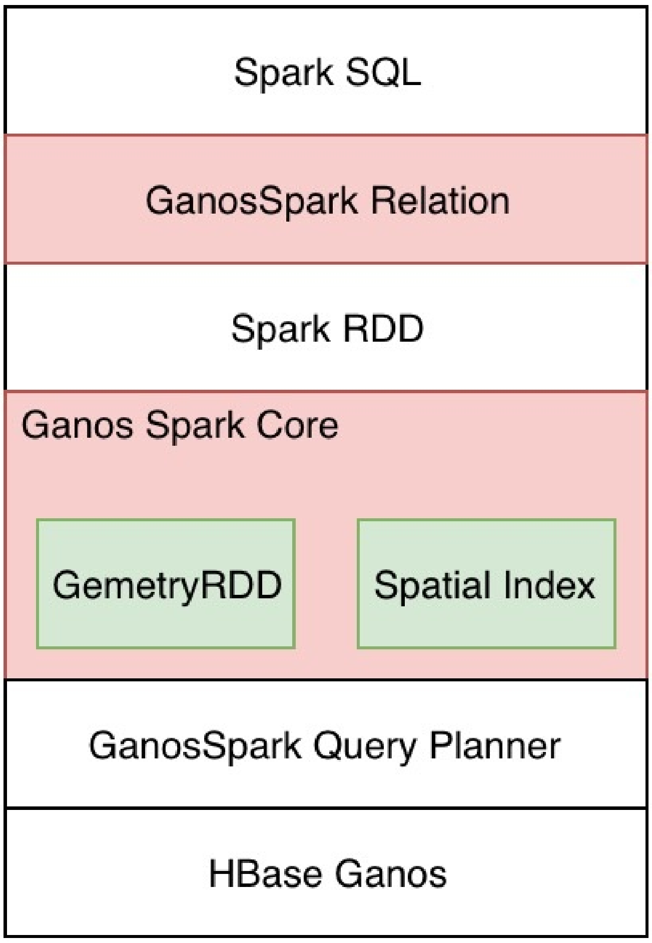
Considerations

- Currently, you can only delete SimpleFeatures by FID.
- You must enable the ID index before you perform this operation. If you fail to delete the SimpleFeature, check whether the ID index is disabled. For more information, see Create an index table.
  - If the code snippet contains the following line, delete it: > sft.getUserData().put("geomesa.disable.id.index", true);
  - You can also choose to create the specified ID index as follows:

sft.getUserData().put("geomesa.indices.enabled", "id:object_id");

4.8. Data analytics integrated with Spark

This topic describes how to use Ganos Spark to manage and analyze large-scale geographic data based on the Apache Spark distributed system. Ganos Spark provides a variety of Spark-based API operations for data loading, analytics and storage. Ganos Spark provides different levels of data analytics models. The most basic model is GeometryRDD, which is used to convert between SimpleFeatures in Ganos data and resilient distributed dataset (RDD) models in Spark. Based on Spark SQL, Ganos Spark also provides a set of user-defined types (UDTs), user-defined functions (UDFs), and user-defined aggregation functions (UDAFs) to manage spatial data. These functions allow you to use a structured query language that is similar to SQL for data query and analytics. The following figure shows the architecture of Ganos Spark:



1. Download the Ganos Spark toolkit

Click Ganos Spark driver to download the Ganos Spark toolkit.

Add dependencies to the pom file in the project directory:

```
<! -- Ganos Spark -->
<dependency>
<groupId>com.aliyun.ganos</groupId>
<artifactId>ganos-spark-runtime</artifactId>
<version>1.0-SNAPSHOT</version>
<scope>system</scope>
<systemPath>../ganos-spark-runtime-1.0-SNAPSHOT.jar</systemPath>
```

```
</dependency>

<! -- Spark -->
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-core_${scala.binary.version}</artifactId>
<version>${spark.version}</version>
</dependency>
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-catalyst_${scala.binary.version}</artifactId>
<version>${spark.version}</version>
<exclusions>
<exclusion>
<groupId>org.scala-lang</groupId>
<artifactId>scala-reflect</artifactId>
</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-sql_${scala.binary.version}</artifactId>
<version>${spark.version}</version>
</dependency>
<dependency>
<groupId>org.apache.spark</groupId>
<artifactId>spark-yarn_${scala.binary.version}</artifactId>
<version>${spark.version}</version>
</dependency>
<dependency>
<groupId>io.netty</groupId>
<artifactId>netty-all</artifactId>
<version>4.1.18.Final</version>
</dependency>
```

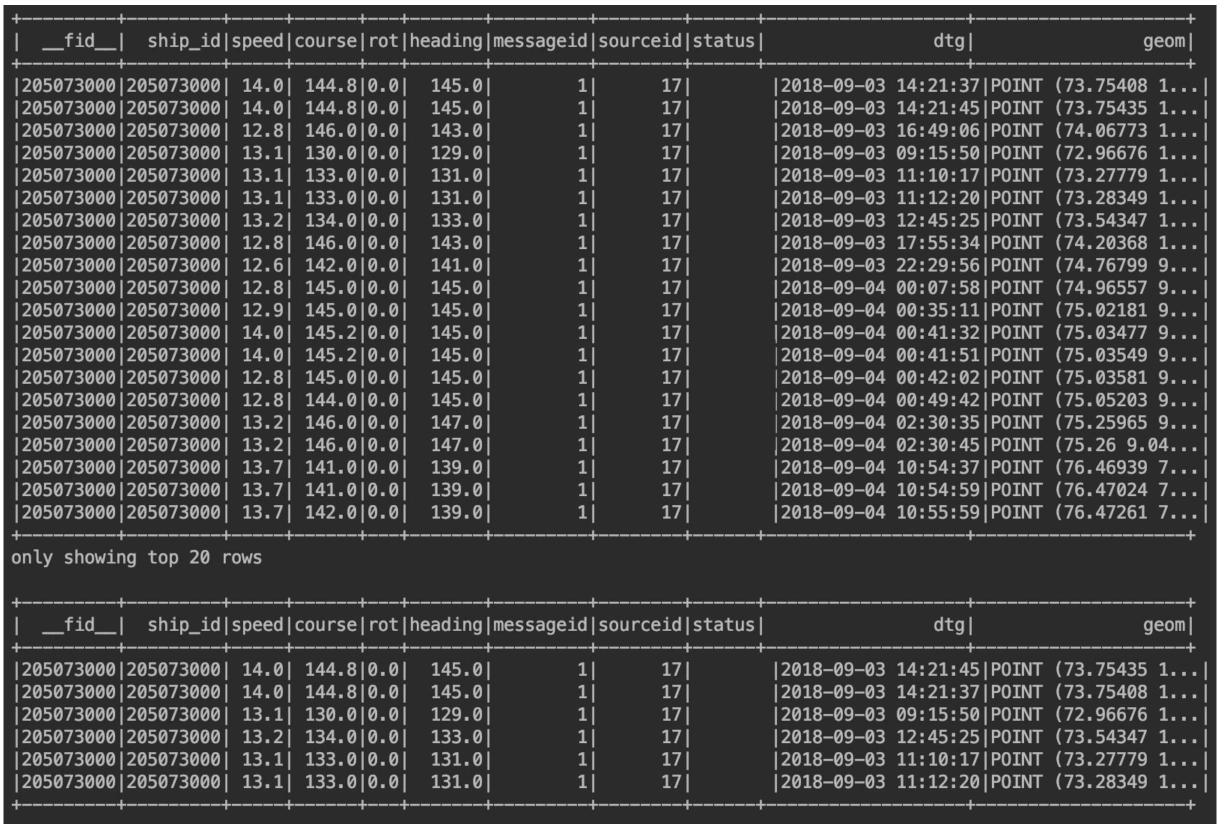
2. Query data in HBase Ganos by using Ganos Spark

After you configure the runtime environment, you can connect to HBase Ganos to query data by using the Ganos Spark service, as shown in the following example:

```
package com.aliyun.ganos
```

```
import com.aliyun.ganos.spark.GanosSparkKryoRegistrator
import org.apache.log4j.{ Level, Logger}
import org.apache.spark.sql.SparkSession

object GanosSparkDemo {

def main(args: Array[String]): Unit = {
Logger.getLogger("org").setLevel(Level.ERROR)
Logger.getLogger("com").setLevel(Level.ERROR)

// Specify connection parameters of ApsaraDB for HBase. The catalog name is set to POINT.
val params = Map(
"hbase.catalog" -> "POINT",
"hbase.zookeepers" -> "The ZooKeeper address used to connect to ApsaraDB for HBase",
"geotools" -> "true")

// Initialize a SparkSession object.
val sparkSession = SparkSession.builder
.appName("Simple Application")
.config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
.config("spark.sql.crossJoin.enabled", "true")
.config("spark.kryo.registrator", classOf[GanosSparkKryoRegistrator].getName)
.master("local[*]")
.getOrCreate()

// Load the data from automatic identification system (AIS).
val dataFrame = sparkSession.read
.format("ganos")
.options(params)
.option("ganos.feature", "AIS")
.load()

// Query all data.
dataFrame.createOrReplaceTempView("ais")
val r=sparkSession.sql("SELECT * FROM ais")
r.show()

// Query spatio-temporal data.
val r1=sparkSession.sql("SELECT * FROM ais WHERE st_contains(st_makeBBOX(70.00000,11.00000,75.000
00,14.00000), geom)")
r1.show()

// Write query results to HBase Ganos.
r1.write.format("ganos").options(params).option("ganos.feature", "result").save()
}
}
```

The following figure shows the output:



```
+---------+---------+-----+------+---+-------+---------+--------+------+-------------------+--------------------+
|  __fid__|  ship_id|speed|course|rot|heading|messageid|sourceid|status|                dtg|                geom|
+---------+---------+-----+------+---+-------+---------+--------+------+-------------------+--------------------+
|205073000|205073000| 14.0| 144.8|0.0|  145.0|        1|      17|      |2018-09-03 14:21:37|POINT (73.75408 1...|
|205073000|205073000| 14.0| 144.8|0.0|  145.0|        1|      17|      |2018-09-03 14:21:45|POINT (73.75435 1...|
|205073000|205073000| 12.8| 146.0|0.0|  143.0|        1|      17|      |2018-09-03 16:49:06|POINT (74.06773 1...|
|205073000|205073000| 13.1| 130.0|0.0|  129.0|        1|      17|      |2018-09-03 09:15:50|POINT (72.96676 1...|
|205073000|205073000| 13.1| 133.0|0.0|  131.0|        1|      17|      |2018-09-03 11:10:17|POINT (73.27779 1...|
|205073000|205073000| 13.1| 133.0|0.0|  131.0|        1|      17|      |2018-09-03 11:12:20|POINT (73.28349 1...|
|205073000|205073000| 13.2| 134.0|0.0|  133.0|        1|      17|      |2018-09-03 12:45:25|POINT (73.54347 1...|
|205073000|205073000| 12.8| 146.0|0.0|  143.0|        1|      17|      |2018-09-03 17:55:34|POINT (74.20368 1...|
|205073000|205073000| 12.6| 142.0|0.0|  141.0|        1|      17|      |2018-09-03 22:29:56|POINT (74.76799 9...|
|205073000|205073000| 12.8| 145.0|0.0|  145.0|        1|      17|      |2018-09-04 00:07:58|POINT (74.96557 9...|
|205073000|205073000| 12.9| 145.0|0.0|  145.0|        1|      17|      |2018-09-04 00:35:11|POINT (75.02181 9...|
|205073000|205073000| 14.0| 145.2|0.0|  145.0|        1|      17|      |2018-09-04 00:41:32|POINT (75.03477 9...|
|205073000|205073000| 14.0| 145.2|0.0|  145.0|        1|      17|      |2018-09-04 00:41:51|POINT (75.03549 9...|
|205073000|205073000| 12.8| 145.0|0.0|  145.0|        1|      17|      |2018-09-04 00:42:02|POINT (75.03581 9...|
|205073000|205073000| 12.8| 144.0|0.0|  145.0|        1|      17|      |2018-09-04 00:49:42|POINT (75.05203 9...|
|205073000|205073000| 13.2| 146.0|0.0|  147.0|        1|      17|      |2018-09-04 02:30:35|POINT (75.25965 9...|
|205073000|205073000| 13.2| 146.0|0.0|  147.0|        1|      17|      |2018-09-04 02:30:45|POINT (75.26 9.04...|
|205073000|205073000| 13.7| 141.0|0.0|  139.0|        1|      17|      |2018-09-04 10:54:37|POINT (76.46939 7...|
|205073000|205073000| 13.7| 141.0|0.0|  139.0|        1|      17|      |2018-09-04 10:54:59|POINT (76.47024 7...|
|205073000|205073000| 13.7| 142.0|0.0|  139.0|        1|      17|      |2018-09-04 10:55:59|POINT (76.47261 7...|
+---------+---------+-----+------+---+-------+---------+--------+------+-------------------+--------------------+
only showing top 20 rows

+---------+---------+-----+------+---+-------+---------+--------+------+-------------------+--------------------+
|  __fid__|  ship_id|speed|course|rot|heading|messageid|sourceid|status|                dtg|                geom|
+---------+---------+-----+------+---+-------+---------+--------+------+-------------------+--------------------+
|205073000|205073000| 14.0| 144.8|0.0|  145.0|        1|      17|      |2018-09-03 14:21:45|POINT (73.75435 1...|
|205073000|205073000| 14.0| 144.8|0.0|  145.0|        1|      17|      |2018-09-03 14:21:37|POINT (73.75408 1...|
|205073000|205073000| 13.1| 130.0|0.0|  129.0|        1|      17|      |2018-09-03 09:15:50|POINT (72.96676 1...|
|205073000|205073000| 13.2| 134.0|0.0|  133.0|        1|      17|      |2018-09-03 12:45:25|POINT (73.54347 1...|
|205073000|205073000| 13.1| 133.0|0.0|  131.0|        1|      17|      |2018-09-03 11:10:17|POINT (73.27779 1...|
|205073000|205073000| 13.1| 133.0|0.0|  131.0|        1|      17|      |2018-09-03 11:12:20|POINT (73.28349 1...|
+---------+---------+-----+------+---+-------+---------+--------+------+-------------------+--------------------+
```

For more information about spatial functions supported by Ganos Spark, see Ganos Spark functions.

3. Use Ganos Spark in Jupyter

Ganos Spark provides the toolkit that allows you to query data and display the result in Jupyter.

Click Leaflet tool to download the Ganos Spark Leaflet toolkit.

Log on to the console and perform the following tasks:

3.1. Install Jupyter.

```
$ pip install --upgrade jupyter
```

or

```
$ pip3 install --upgrade jupyter
```

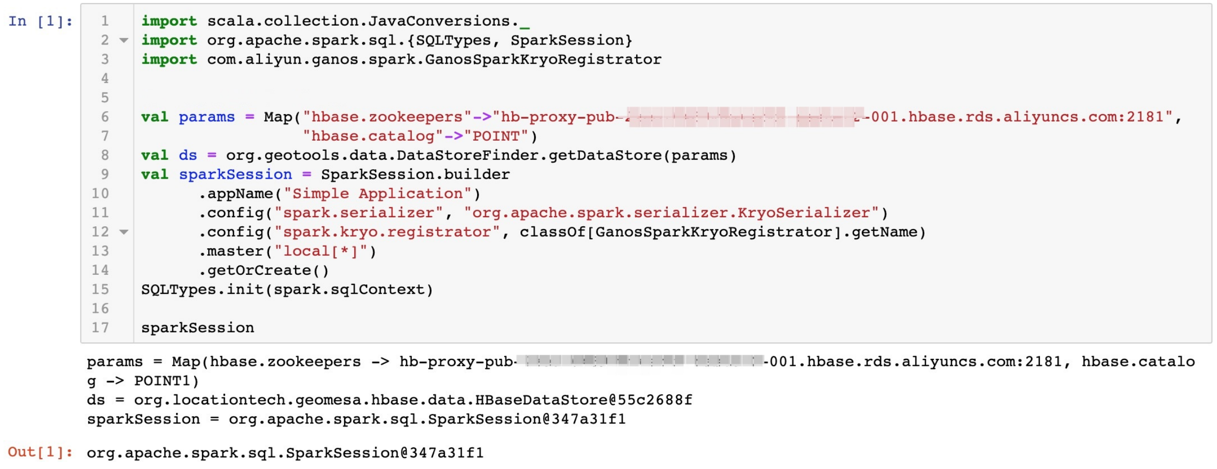
3.2. Configure the SPARK_HOME environment variable, add a kernel named Ganos Spark Test by using toree, and then launch Jupyter:

```
$ jars="ganos-spark-runtime-1.0-SNAPSHOT.jar,ganos-spark-jupyter-leaflet-1.0-SNAPSHOT.jar"
$ jupyter toree install --replace --user --kernel_name "Ganos Spark Test" --spark_home=${SPARK_HOM
E} --spark_opts="--master localhost[*] --jars $jars"
$ jupyter notebook
```

After the server is started, you can visit http://localhost:8888 and create a Ganos Spark Test session in the Jupyter console.

3.3. Load HBase Ganos data.

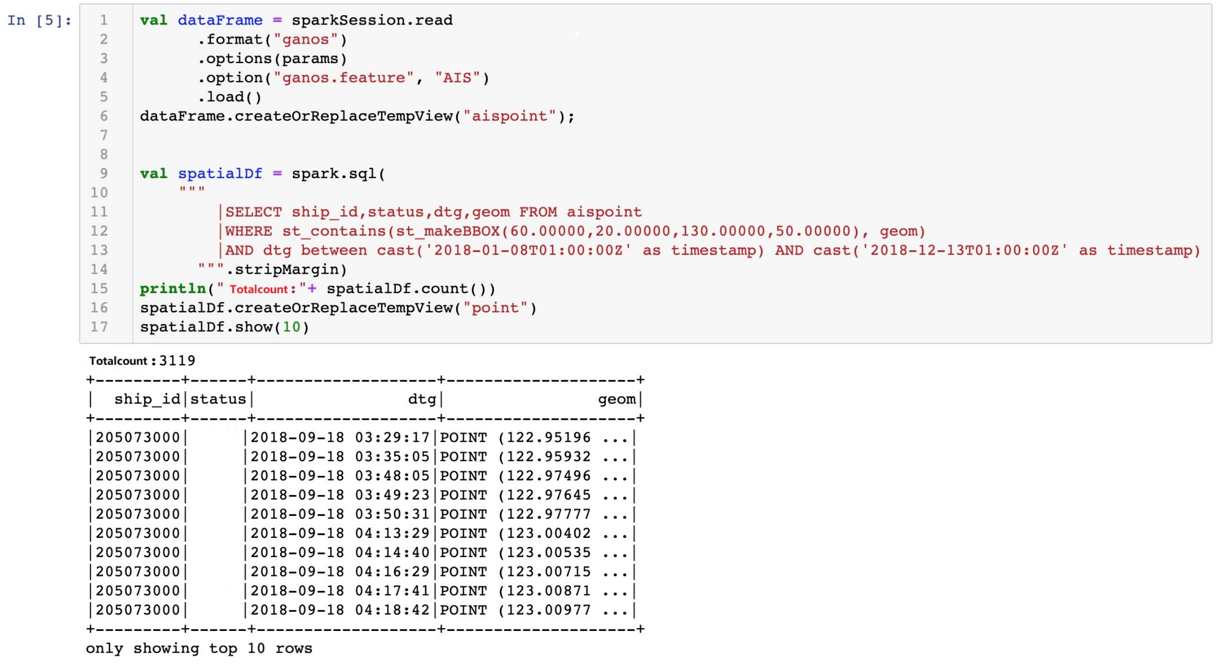
3.3.1. Create a Spark session.



```
In [1]:
import scala.collection.JavaConversions._
import org.apache.spark.sql.{SQLTypes, SparkSession}
import com.aliyun.ganos.spark.GanosSparkKryoRegistrator


val params = Map("hbase.zookeepers"->"hb-proxy-pub-[redacted]-001.hbase.rds.aliyuncs.com:2181",
                 "hbase.catalog"->"POINT")
val ds = org.geotools.data.DataStoreFinder.getDataStore(params)
val sparkSession = SparkSession.builder
      .appName("Simple Application")
      .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
      .config("spark.kryo.registrator", classOf[GanosSparkKryoRegistrator].getName)
      .master("local[*]")
      .getOrCreate()
SQLTypes.init(spark.sqlContext)

sparkSession
```

```
params = Map(hbase.zookeepers -> hb-proxy-pub-[redacted]-001.hbase.rds.aliyuncs.com:2181, hbase.catalo
g -> POINT1)
ds = org.locationtech.geomesa.hbase.data.HBaseDataStore@55c2688f
sparkSession = org.apache.spark.sql.SparkSession@347a31f1

Out[1]: org.apache.spark.sql.SparkSession@347a31f1
```

3.3.2. Query data in HBase Ganos by using Spark SQL.

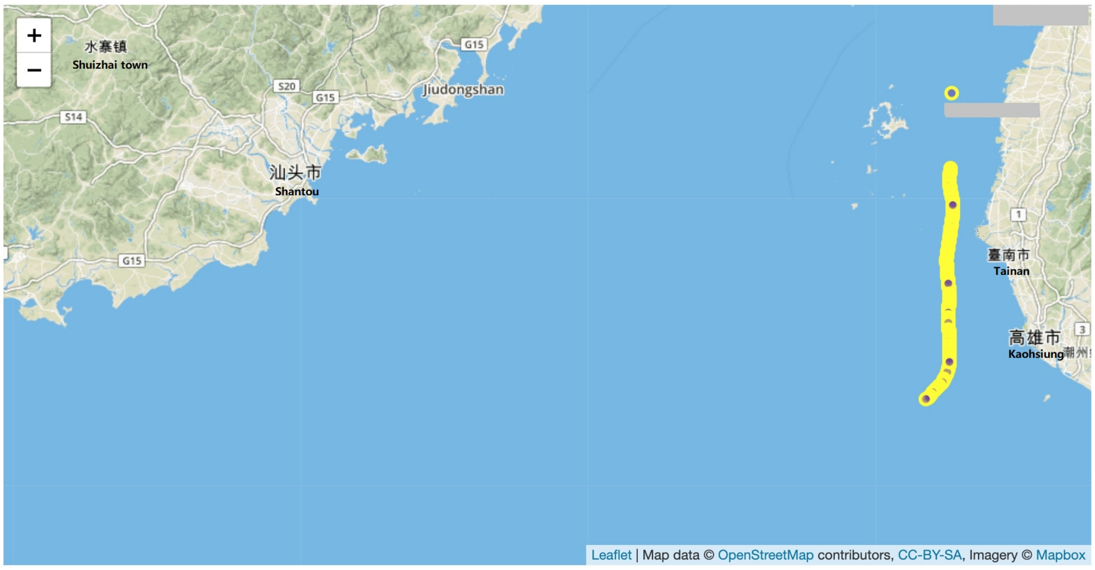


```
In [5]:
val dataFrame = sparkSession.read
      .format("ganos")
      .options(params)
      .option("ganos.feature", "AIS")
      .load()
dataFrame.createOrReplaceTempView("aispoint");


val spatialDf = spark.sql(
    """
        |SELECT ship_id,status,dtg,geom FROM aispoint
        |WHERE st_contains(st_makeBBOX(60.00000,20.00000,130.00000,50.00000), geom)
        |AND dtg between cast('2018-01-08T01:00:00Z' as timestamp) AND cast('2018-12-13T01:00:00Z' as timestamp)
      """.stripMargin)
println(" Totalcount:"+ spatialDf.count())
spatialDf.createOrReplaceTempView("point")
spatialDf.show(10)
```

```
Totalcount:3119
+---------+------+-------------------+--------------------+
|  ship_id|status|                dtg|                geom|
+---------+------+-------------------+--------------------+
|205073000|      |2018-09-18 03:29:17|POINT (122.95196 ...|
|205073000|      |2018-09-18 03:35:05|POINT (122.95932 ...|
|205073000|      |2018-09-18 03:48:05|POINT (122.97496 ...|
|205073000|      |2018-09-18 03:49:23|POINT (122.97645 ...|
|205073000|      |2018-09-18 03:50:31|POINT (122.97777 ...|
|205073000|      |2018-09-18 04:13:29|POINT (123.00402 ...|
|205073000|      |2018-09-18 04:14:40|POINT (123.00535 ...|
|205073000|      |2018-09-18 04:16:29|POINT (123.00715 ...|
|205073000|      |2018-09-18 04:17:41|POINT (123.00871 ...|
|205073000|      |2018-09-18 04:18:42|POINT (123.00977 ...|
+---------+------+-------------------+--------------------+
only showing top 10 rows
```

3.3.3. Display data in the Leaflet:

```
In [6]: implicit val displayer: String => Unit = { s => kernel.display.content("text/html", s) }
        import com.aliyun.ganos.spark.jupyter._
        displayer(L.render(Seq(L.DataFrameLayerPoint(spatialDf,"ship_id",L.StyleOptions(color="yellow",fillColor="#63A",f
```



You can click here to download the complete notebook document for test.

4.9. Ganos Spark functions

After you construct a SparkSession object, you can call the read method to load DataFrames. Ganos Spark provides a set of spatio-temporal operators for you to query spatio-temporal data. Spatio-temporal operators are user-defined functions (UDFs) and can be classified into the following:

1. Spatial Constructors functions

ST_GeomFromGeoHash

```
Geometry st_geomFromGeoHash(String geohash, Int prec)
```

Returns a GeoHash value from a given GeoHash code. The prec argument specifies the precision of GeoHash.

ST_GeomFromWKT

```
Geometry st_geomFromWKT(String wkt)
```

Constructs a Geometry object from the well-known text (WKT) representation.

ST_GeomFromWKB

```
Geometry st_geomFromWKB(Array[Byte] wkb)
```

Constructs a Geometry object from the well-known binary (WKB) representation.

ST_LineFromText

```
LineString st_lineFromText(String wkt)
```

Constructs a LineString object from the WKB representation.

ST_MakeBox2D

```
Geometry st_makeBox2D(Point lowerLeft, Point upperRight)
```

Constructs a Box2d object defined by lowerLeft and upperRight.

ST_MakeBBOX

```
Geometry st_makeBBOX(Double lowerX, Double lowerY, Double upperX, Double upperY)
```

Constructs a bounding box (BBOX) object defined by the given boundary coordinates.

ST_MakePolygon

```
Polygon st_makePolygon(LineString shell)
```

Constructs a Polygon object defined by a given shell. The input shell must be a closed LineString.

ST_MakePoint

```
Point st_makePoint(Double x, Double y)
```

Constructs a Point object defined by the given X-coordinate and Y-coordinate.

ST_MakeLine

```
LineString st_makeLine(Seq[Point] points)
```

Constructs a LineString object defined by the given Point objects.

ST_MakePointM

```
Point st_makePointM(Double x, Double y, Double m)
```

Constructs a Point object defined by the given X-coordinate, Y-coordinate, and M-coordinate.

ST_MLineFromText

```
MultiLineString st_mLineFromText(String wkt)
```

Constructs a MultiLineString object defined by the given WKT representation.

ST_MPointFromText

```
MultiPoint st_mPointFromText(String wkt)
```

Constructs a MultiPoint object defined by the given WKT representation.

ST_MPolyFromText

```
MultiPolygon st_mPolyFromText(String wkt)
```

Constructs a MultiPolygon object defined by the given WKT representation.

ST_Point

```
Point st_point(Double x, Double y)
```

Constructs a Point object defined by the given X-coordinate and Y-coordinate. This function is similar to ST_MakePoint.

ST_PointFromGeoHash

```
Point st_pointFromGeoHash(String geohash, Int prec)
```

Return the Point at the geometric center of the bounding box defined by the GeoHash (base-32 encoded). The prec argument specifies the precision of GeoHash.

ST_PointFromText

```
Point st_pointFromText(String wkt)
```

ST_PointFromWKB

```
Point st_pointFromWKB(Array[Byte] wkb)
```

Constructs a Point object from the given WKB representation.

ST_Polygon

```
Polygon st_polygon(LineString shell)
```

Constructs a Polygon object from a given LineString.

ST_PolygonFromText

```
Polygon st_polygonFromText(String wkt)
```

Constructs a Polygon object from the given WKT representation.

2. Geometry Accessors functions

ST_Boundary

```
Geometry st_boundary(Geometry geom)
```

Returns the boundary of a ST_Geometry.

ST_CoordDim

```
Int st_coordDim(Geometry geom)
```

Returns the number of dimensions of the coordinates of a Geometry object. To query the dimension of a geometry, use the ST_Dimension function.

ST_Dimension

```
Int st_dimension(Geometry geom)
```

Returns the inherent number of dimensions of a Geometry object.

ST_Envelope

```
Geometry st_envelope(Geometry geom)
```

Returns the BBOX of a ST_Geometry value.

ST_ExteriorRing

```
LineString st_exteriorRing(Geometry geom)
```

Returns a LineString representing the exterior ring of a given Polygon geometry.

ST_GeometryN

```
Int st_geometryN(Geometry geom, Int n)
```

Returns the Nth geometry within a Geometry object. N is a given value. For example, if you want to return the second point within a MultiPoint object, set the value of N to 2.

ST_InteriorRingN

```
Int st_interiorRingN(Geometry geom, Int n)
```

Returns a LineString representing the *N*th interior ring of a given Polygon geometry.

ST_IsClosed

```
Boolean st_isClosed(Geometry geom)
```

Returns a value that indicates whether a LineString is closed. TRUE is returned if a LineString is closed.

ST_IsCollection

```
Boolean st_isCollection(Geometry geom)
```

Returns a value that indicates whether a geometry is a collection.

ST_IsEmpty

```
Boolean st_isEmpty(Geometry geom)
```

Returns a value that indicates whether a geometry is empty. This function is used to verify a ST_Geometry. Value 1 (TRUE) is returned if the geometry is empty and value 0 (FALSE) is returned if it is not empty.

ST_IsRing

```
Boolean st_isRing(Geometry geom)
```

Returns a value that indicates whether a given ST_LineString is a ring. Value 1 is returned if the ST_LineString is a ring and value 0 is returned if it is not a ring.

ST_IsSimple

```
Boolean st_isSimple(Geometry geom)
```

Returns a value that indicates whether a given ST_LineString, ST_MultiPoint, or ST_MultiLineString is simple.

ST_IsValid

```
Boolean st_isValid(Geometry geom)
```

Returns a value that indicates whether a given geometry collection is valid.

ST_NumGeometries

```
Int st_numGeometries(Geometry geom)
```

Returns the number of geometries of a given type in a geometry collection, such as the number of ST_MultiPoint, ST_MultiLineString, or ST_MultiPolygon geometries.

ST_NumPoints

```
Int st_numPoints(Geometry geom)
```

Returns the number of points in a given ST_LineString.

ST_PointN

```
Point st_pointN(Geometry geom, Int n)
```

Returns the *N*th point in a given ST_LineString.

ST_X

```
Float st_X(Geometry geom)
```

Returns a double-precision digit that indicates the X-coordinate of a point.

ST_Y

```
Float st_y(Geometry geom)
```

Returns a double-precision digit that indicates the Y-coordinate of a point.

3. Geometry Cast functions

ST_CastToPoint

```
Point st_castToPoint(Geometry g)
```

Casts a Geometry to a Point.

ST_CastToPolygon

```
Polygon st_castToPolygon(Geometry g)
```

Casts a Geometry to a Polygon.

ST_CastToLineString

```
LineString st_castToLineString(Geometry g)
```

Casts a Geometry to a LineString.

ST_ByteArray

```
Array[Byte] st_byteArray(String s)
```

Casts a String to a Byte Array by using UTF-8 encoding.

4. Geometry Editors functions

ST_Translate

```
Geometry st_translate(Geometry geom, Double deltaX, Double deltaY)
```

Returns a new geometry whose coordinates are translated from deltaX and deltaY.

5. Geometry Outputs functions

ST_AsBinary

```
Array[Byte] st_asBinary(Geometry geom)
```

Returns the Byte Array representation of a Geometry.

ST_AsGeoJSON

```
String st_asGeoJSON(Geometry geom)
```

Returns the GeoJSON representation of a Geometry.

ST_AsLatLonText

```
String st_asLatLonText(Point p)
```

Returns the Degrees, Minutes, and Seconds representation of a Point. It assumes that the point is in a latitude/longitude projection.

ST_AsText

```
String st_asText(Geometry geom)
```

Returns the well-known text (WKT) representation of a Geometry.

ST_GeoHash

```
String st_geoHash(Geometry geom, Int prec)
```

Returns the GeoHash representation of a Geometry. The prec argument specifies the precision.

6. Spatial Relationships functions

ST_Contains

```
Boolean st_contains(Geometry a, Geometry b)
```

Returns true if and only if no points of Geometry B lie in the exterior of Geometry A, and at least one point of the interior of Geometry B lies in the interior of Geometry A.

ST_Covers

```
Boolean st_covers(Geometry a, Geometry b)
```

Returns true if no point in Geometry B is outside Geometry A.

ST_Crosses

```
Boolean st_crosses(Geometry a, Geometry b)
```

Returns true if Geometry A and Geometry B are partially intersected. This means that the geometries have some, but not all interior points in common.

ST_Disjoint

```
Boolean st_disjoint(Geometry a, Geometry b)
```

Returns true if Geometry A and Geometry B are disjoint. This function is equivalent to NOT ST_Intersects.

ST_Equals

```
Boolean st_equals(Geometry a, Geometry b)
```

Returns true if Geometry A and Geometry B are spatially equal.

ST_Intersects

```
Boolean st_intersects(Geometry a, Geometry b)
```

Returns true if Geometry A and Geometry B share any portion of the space (intersected).

ST_Overlaps

```
Boolean st_overlaps(Geometry a, Geometry b)
```

Returns true if Geometry A and Geometry B spatially overlap. This means that they are equal in size, and the intersection point and the geometries are equal in size.

ST_Touches

```
Boolean st_touches(Geometry a, Geometry b)
```

Returns true if Geometry A and Geometry B have at lease one point in common, but they do not have any interior intersection.

ST_Within

```
Boolean st_within(Geometry a, Geometry b)
```

Returns true if Geometry A completely lies in the interior of Geometry B.

ST_Relate

```
String st_relate(Geometry a, Geometry b)
```

Returns the DE-9IM matrix pattern that shows the dimension of intersections between the Interior, Boundary, and Exterior of two geometries.

ST_RelateBool

```
Boolean st_relateBool(Geometry a, Geometry b, String mask)
```

Returns true if the mask of the DE-9IM matrix pattern matches that of the DE-9IM matrix pattern returned by st_relate(a, b).

ST_Area

```
Double st_area(Geometry g)
```

Returns the area of a Geometry.

ST_Centroid

```
Point st_centroid(Geometry g)
```

Returns the geometric center of a Geometry.

ST_ClosestPoint

```
Point st_closestPoint(Geometry a, Geometry b)
```

Returns the point on Geometry A that is closest to Geometry B.

ST_Distance

```
Double st_distance(Geometry a, Geometry b)
```

Returns the 2-dimensional Cartesian distance between two geometries in projected units, such as EPSG:4236.

ST_DistanceSphere

```
Double st_distanceSphere(Geometry a, Geometry b)
```

Returns the minimum distance between two longitude/latitude geometries. It assumes a spherical earth.

ST_Length

```
Double st_length(Geometry geom)
```

Returns the 2-dimensional length of a Line geometry or the perimeter of a Plane geometry, such as EPSG:4236. The units are determined by the spatial reference system of the geometry. It returns 0.0 for other types of geometries, such as Point.

ST_LengthSphere

```
Double st_lengthSphere(LineString line)
```

Returns the approximate 2-dimensional length of a LineString. It assumes a spherical earth. The returned length is measured in meters. The approximation is within 0.3% of st_lengthSpheroid and this function is more efficient in computation.

7. Geometry Processing functions

ST_antimeridianSafeGeom

```
Geometry st_antimeridianSafeGeom(Geometry geom)
```

Attempts to convert the geometry into an equivalent form of "antimeridian-safe" if the geometry spans the antimeridian. For example, the output geometry is covered by BOX (-180 -90, 180 90). In some cases, this method may fail, and the input geometry will be returned and then an error will be logged.

ST_BufferPoint

```
Geometry st_bufferPoint(Point p, Double buffer)
```

Returns a geometry covering all points within a given radius of Point p. The radius is measured in meters.

ST_ConvexHull

```
Geometry st_convexHull(Geometry geom)
```

An aggregate function. The convex hull of a geometry represents the minimum convex geometry that encloses all geometries in the aggregated rows.

5.SDK for Java

5.1. Build a development environment

The HBase Ganos API interacts with databases based on Ali-Hbase Client and GeoTools. You must configure the pom.xml file in Maven to import the corresponding packages as follows:

```
<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<geomesa.version>2.1.0</geomesa.version>
<scala.abi.version>2.11</scala.abi.version>
<gt.version>18.0</gt.version>
<hbase.version>1.1.2</hbase.version>
<zookeeper.version>3.4.9</zookeeper.version>
</properties>

<dependencies>
<dependency>
<groupId>org.locationtech.geomesa</groupId>
<artifactId>geomesa-hbase-datastore_2.11</artifactId>
<version>2.0.2</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-client</artifactId>
<version>${hbase.version}</version>
<exclusions>
<exclusion>
<artifactId>com.google.guava</artifactId>
<groupId>guava</groupId>
</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-server</artifactId>
<version>${hbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
```

```
<artifactId>alihbase-common</artifactId>
<version>${hbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-protocol</artifactId>
<version>${hbase.version}</version>
</dependency>
</dependencies>
```

Launch the command-line tool, switch to the root of the Maven project, and run the following command to compile the project:

```
mvn clean install
```

If you can find the compiled class files and JAR packages in the target directory, the environment has been successfully built.

5.2. Quick start

This topic is intended to help users quickly build a Ganos development environment for test purposes by coding. After you read this topic, you will learn how to manage Ganos spatio-temporal data, including how to connect to a database, create an index, import data, and query data.

Dependencies:

```
<dependencies>
<dependency>
<groupId>org.locationtech.geomesa</groupId>
<artifactId>geomesa-hbase-datastore_2.11</artifactId>
<version>${geomesa.version}</version>
<exclusions>
<exclusion>
<groupId>org.apache.hbase</groupId>
<artifactId>hbase-client</artifactId>
</exclusion>
<exclusion>
<groupId>org.apache.hbase</groupId>
<artifactId>hbase-server</artifactId>
</exclusion>
<exclusion>
<groupId>org.apache.hbase</groupId>
<artifactId>hbase-common</artifactId>
```

```
</exclusion>
<exclusion>
<groupId>org.apache.hbase</groupId>
<artifactId>hbase-protocol</artifactId>
</exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-client</artifactId>
<version>${hbase.version}</version>
<exclusions>
<exclusion>
<artifactId>com.google.guava</artifactId>
<groupId>guava</groupId>
</exclusion>
</exclusions>
</dependency>


<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-server</artifactId>
<version>${hbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-common</artifactId>
<version>${hbase.version}</version>
</dependency>
<dependency>
<groupId>com.aliyun.hbase</groupId>
<artifactId>alihbase-protocol</artifactId>
<version>${hbase.version}</version>
</dependency>
<dependency>
<groupId>com.alibaba</groupId>
<artifactId>fastjson</artifactId>
<version>1.2.49</version>
</dependency>
</dependencies>
```

```
<build>
<plugins>
<plugin>
<inherited>true</inherited>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
```

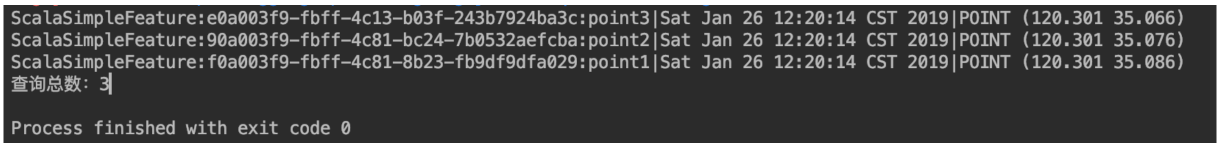
Sample code and descriptions

```
package com.aliyun.tst;

import com.vividsolutions.jts.geom.Coordinate;
import com.vividsolutions.jts.geom.GeometryFactory;
import com.vividsolutions.jts.geom.Point;
import org.geotools.data.DataStore;
import org.geotools.data.DataStoreFinder;
import org.geotools.data.Query;
import org.geotools.data.Transaction;
import org.geotools.data.simple.SimpleFeatureCollection;
import org.geotools.data.simple.SimpleFeatureIterator;
import org.geotools.data.simple.SimpleFeatureWriter;
import org.geotools.factory.CommonFactoryFinder;
import org.geotools.factory.Hints;
import org.geotools.feature.simple.SimpleFeatureBuilder;
import org.geotools.filter.text.ecql.ECQL;
import org.geotools.geometry.jts.JTSFactoryFinder;
import org.locationtech.geomesa.utils.geotools.SimpleFeatureTypes;
import org.opengis.feature.Feature;
import org.opengis.feature.simple.SimpleFeature;
import org.opengis.feature.simple.SimpleFeatureType;
import org.opengis.filter.FilterFactory2;
import org.opengis.filter.sort.SortBy;
import org.opengis.filter.sort.SortOrder;

import java.io.BufferedReader;
import java.io.FileReader;
import java.text.SimpleDateFormat;
import java.util.*;

public class Demo {
public static final String ZK = "localhost"; //The ZooKeeper endpoint.
public static void main(String args[]){
try{
DataStore ds=null;
SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd' 'HH:mm:ss");

//Configure connection parameters.
Map<String, String> params= new HashMap<>();
params.put("hbase.zookeepers",ZK);
params.put("hbase.catalog","test_catalog");

//Initialize the DataStore.
ds= DataStoreFinder.getDataStore(params);

//Use SimpleFeatureType to define the table schema.
String sft_name="point";
SimpleFeatureType sft=
SimpleFeatureTypes.createType(sft_name, "name:String,dtg:Date,*geom:Point:srid=4326");

//Specify Gzip as the compression algorithm.
sft.getUserData().put("geomesa.table.compression.enabled", "true");
sft.getUserData().put("geomesa.table.compression.type", "gz");

//Create a table.
ds.createSchema(sft);

/*
* Create a GeometryFactory object.
*/
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
SimpleFeatureBuilder builder = new SimpleFeatureBuilder(sft);

//Construct spatial data (points)
```

```
//Construct spatial data (points).
Point point1 = geometryFactory.createPoint(new Coordinate(120.301,35.086));
Point point2 = geometryFactory.createPoint(new Coordinate(120.301,35.076));
Point point3 = geometryFactory.createPoint(new Coordinate(120.301,35.066));

//Construct SimpleFeatures of points.
List<SimpleFeature> features=new ArrayList<>();
features.add(builder.buildFeature("1", new Object[]{"point1",new Date(),point1}));
features.add(builder.buildFeature("2", new Object[]{"point2",new Date(),point2}));
features.add(builder.buildFeature("3", new Object[]{"point3",new Date(),point3}));

//Import SimpleFeature content.
SimpleFeatureWriter writer=(SimpleFeatureWriter)ds.getFeatureWriterAppend(sft_name, Transaction.
AUTO_COMMIT);
for(SimpleFeature feature:features){
SimpleFeature toWrite=writer.next();
toWrite.setAttributes(feature.getAttributes());
toWrite.getUserData().putAll(feature.getUserData());
writer.write();
}
writer.close();



//Construct spatio-temporal query conditions.
long t1=format.parse("2019-01-19 11:45:00").getTime();
long t2=format.parse("2019-02-21 12:15:00").getTime();

String sortField="dtg";//Specify the sorting field. In this example, the data is sorted by date.
FilterFactory2 ff = CommonFactoryFinder.getFilterFactory2();
SortBy[] sort = new SortBy[]{ff.sort(sortField, SortOrder.DESCENDING)};

//Construct a query object.
Query query = new Query(sft_name, ECQL.toFilter( "bbox(geom,120,20,130,40) AND dtg >= "+t1+" AND d
tg <= "+t2));
query.setSortBy(sort);
SimpleFeatureCollection result=ds.getFeatureSource(sft_name).getFeatures(query);
SimpleFeatureIterator iterator=result.features();

//Print the query result.
long sum = 0;
while (iterator.hasNext()) {
```

```
System.out.println(iterator.next());
sum++;
}
System.out.println("Total number of queries:" + sum);


}
catch (Exception e){
e.printStackTrace();
}
}
}
```

After you successfully run the preceding script, the three "Point" SimpleFeatures are printed. The query result is as follows:



```
ScalaSimpleFeature:e0a003f9-fbff-4c13-b03f-243b7924ba3c:point3|Sat Jan 26 12:20:14 CST 2019|POINT (120.301 35.066)
ScalaSimpleFeature:90a003f9-fbff-4c81-bc24-7b0532aefcba:point2|Sat Jan 26 12:20:14 CST 2019|POINT (120.301 35.076)
ScalaSimpleFeature:f0a003f9-fbff-4c81-8b23-fb9df9dfa029:point1|Sat Jan 26 12:20:14 CST 2019|POINT (120.301 35.086)
查询总数: 3

Process finished with exit code 0
```

result

5.3. Connect to HBase Ganos

The HBase Ganos API interacts with databases based on GeoTools interfaces. In GeoTools, DataStores are used to access and store data. A DataStore class is defined in the format of DataAccess<SimpleFeatureType,SimpleFeature>. SimpleFeature specifies a vector feature model managed by GeoTools. Each point, line, or polygon is considered a SimpleFeature. The schema of a SimpleFeature is defined by SimpleFeatureType.

For more information about the definition of the DataStore interface and how to use DataStores, see the official GeoTools documentation: http://docs.geotools.org/latest/userguide/library/api/datastore.html.

The HBase DataStore parameters are as follows. Parameters marked with asterisks (*) are required parameters.

Parameter	Data type	Description
hbase.catalog *	String	The name of the GeoMesa catalog.
hbase.zookeepers*	String	The ZooKeeper endpoint. Multiple ZooKeeper endpoints are separated with commas (,).
geomesa.query.timeout	String	The validity period of each query.

Parameter	Data type	Description
geomesa.query.threads	Integer	The number of threads used to process each query.

You can call the getDataStore method of DataStoreFinder to access a DataStore. Example:

```
//Configure connection parameters.
Map<String, String> params= new HashMap<>();
params.put("hbase.zookeepers","localhost");
params.put("hbase.catalog","catalog_name");

//Create a DataStore.
DataStore ds=DataStoreFactory.getDataStore(params);
```

You can then use the created HBase DataStore instance to interact with HBase Ganos. HBase DataStores are a subclass of DataStores.

5.4. Create an index table

The HBase Ganos API uses index tables to store and query data. You can call the createSchema (SimpleFeatureType) method of DataStore to create an index. The SimpleFeatureType class is used to define the schema of the SimpleFeature structure in Ganos, and it consists of a set of common properties. HBase Ganos supports all standard GeoTools property types and also extends some other property types. You can use the SimpleFeatureTypes class provided by Geomesa to create a SimpleFeatureTypes object in HBase Ganos:

```
import org.locationtech.geomesa.utils.interop.SimpleFeatureTypes;

SimpleFeatureTypes.createType("sft_name", "dtg:Date,*geom:Point:srid=4326,name1:type1,name2,ty
pe2,...,nameN,typeN");
```

In the preceding example, a SimpleFeatureType class named sft_name has been created. This SimpleFeatureType class contains a Data column named dtg, a spatial column of the Point type named geom (coordinate reference system 4326), and several property fields including name1 and name2.

The following table lists the data types supported by HBase Ganos.

HBase Ganos type	Java data type	Support creating indexes
String	java.lang.String	Yes
Integer	java.lang.Integer	Yes
Double	java.lang.Double	Yes
Long	java.lang.Long	Yes

HBase Ganos type	Java data type	Support creating indexes
Float	java.lang.Float	Yes
Boolean	java.lang.Boolean	Yes
UUID	java.util.UUID	Yes
Date	java.util.Date	Yes
Timestamp	java.sql.Timestamp	Yes
Point	org.locationtech.jts.geom.Point	Yes
LineString	org.locationtech.jts.geom.LineString	Yes
Polygon	org.locationtech.jts.geom.Polygon	Yes
MultiPoint	org.locationtech.jts.geom.MultiPoint	Yes
MultiLineString	org.locationtech.jts.geom.MultiLineString	Yes
MultiPolygon	org.locationtech.jts.geom.MultiPolygon	Yes
GeometryCollection	org.locationtech.jts.geom.GeometryCollection	Yes
Geometry	org.locationtech.jts.geom.Geometry	Yes
List[A]	java.util.List	Yes
Map[A,B]	java.util.Map<A, B>	No
Bytes	byte[]	No

Notes: 1. The geom property is required. You can create only one index on each spatial column at a time. The indexed column is applied to primary spatio-temporal index systems such as Z2, Z3, XZ2, and XZ3. 2. A column of the Date type can be used as a primary spatio-temporal index or a common property index.

2. Specify index parameters

When you create an index, you can set static properties of SimpleFeatureType to specify a set of required parameters.

- Create indexes on specified property columns. HBase Ganos allows you to create indexes on common property columns, which can improve the query efficiency without specified spatio-temporal parameters. For example, you can run the following command to create indexes on the name column:

```
SimpleFeatureType sft = ...
sft.getDescriptor("name").getUserData().put("index", "true");
```

- You can specify the compression option in UserData options of SimpleFeatureType class. The supported compression types are snappy, lzo, gz, bzip2, lz4, and zstd.

```
SimpleFeatureType sft = ....;
sft.getUserData().put("geomesa.table.compression.enabled", "true");
sft.getUserData().put("geomesa.table.compression.type", "snappy");
```

5.5. Import data

HBase Ganos uses the SimpleFeature class to specify spatial features. Each SimpleFeature class contains an ID, a Geometry object, and other properties. The GeoTools API provides the SimpleFeatureBuilder class to help users create SimpleFeature objects.

Build a SimpleFeature class

```
SimpleFeatureType sft = .... ;
SimpleFeatureBuilder sfBuilder = new SimpleFeatureBuilder(sft);
builder.set("property name", property value);
...
builder.set("geom", Geometry);
SimpleFeature feature = builder.buildFeature(object_id + "_" + date.getTime());
```

Note: When you create a SimpleFeature class, HBase Ganos generates a 128-bit UUID as the default Feature ID. To save storage space, you can run the following command to define your own Feature ID:

```
SimpleFeature feature =...
feature.getUserData().put(Hints.USE_PROVIDED_FID, java.lang.Boolean.TRUE);
```

Create a Geometry object

Each SimpleFeature contains a Geometry object, which indicates the spatial object of a feature. The following figure shows the Entity Relationship Diagram (ER) of various Geometry spatial objects. For more information, visit http://docs.geotools.org/stable/userguide/library/jts/geometry.html.

The GeoTools API provides the GeometryFactory class to help you create Geometry objects. You can create Geometry objects in the following ways:

- The Point feature. Method 1: Create a Geometry object by using a Coordinate object.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
Coordinate coord = new Coordinate(1, 1);
Point point = geometryFactory.createPoint(coord);
```

Method 2: Create a Geometry object by using the Well-known text (WKT). The WKT is a text markup language that is used to specify vector spatial objects, spatial referencing systems, and the transformation between spatial reference systems. For more information, visit https://en.wikipedia.org/wiki/Well-known_text.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
WKTReader reader = new WKTReader(geometryFactory);
Point point = (Point) reader.read("POINT (1 1)");
```

- The LineString feature. Method 1: Create a Geometry object by using a Coordinate object.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
Coordinate[] coords =
new Coordinate[] {new Coordinate(0, 2), new Coordinate(2, 0), new Coordinate(8, 6) };
LineString line = geometryFactory.createLineString(coordinates);
```

Method 2: Create a Geometry object by using the WKT.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
WKTReader reader = new WKTReader( geometryFactory );
LineString line = (LineString) reader.read("LINESTRING(0 2, 2 0, 8 6)");
```

- The Polygon feature. Method 1: Create a Geometry object by using a Coordinate object.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory();
Coordinate[] coords =
new Coordinate[] {new Coordinate(4, 0), new Coordinate(2, 2),
new Coordinate(4, 4), new Coordinate(6, 2), new Coordinate(4, 0) };
LinearRing ring = geometryFactory.createLinearRing( coords );
LinearRing holes[] = null; // use LinearRing[] to represent holes
Polygon polygon = geometryFactory.createPolygon(ring, holes );
```

Method 2: Create a Geometry object by using the WKT.

```
GeometryFactory geometryFactory = JTSFactoryFinder.getGeometryFactory( null );
WKTReader reader = new WKTReader( geometryFactory );
Polygon polygon = (Polygon) reader.read("POLYGON((20 10, 30 0, 40 10, 30 20, 20 10))");
```

2. Write data in the database

HBase Ganos writes data by using the SimpleFeatureWriter class in the GeoTools API. SimpleFeatureWriter supports transactions and can be built by calling the getFeatureWriterAppend method of DataStore.

- Insert a single SimpleFeature structure:

```
SimpleFeatureType sft = .... ;
SimpleFeatureWriter writer=(SimpleFeatureWriter)ds.getFeatureWriterAppend(sft.getTypeName(), Tr
ansaction.AUTO_COMMIT);
SimpleFeature toWrite=writer.next();
toWrite.setAttributes(feature.getAttributes());
toWrite.getUserData().putAll(feature.getUserData());
writer.write();
writer.close();
```

- Insert multiple SimpleFeature structures:

HBase Ganos allows you to insert multiple SimpleFeature structures at a time. You can use the SimpleFeatureStore class of the GeoTools API to implement this operation:

```
List<SimpleFeature> features=...
SimpleFeatureStore featureStore = (SimpleFeatureStore) ds.getFeatureSource(sft.getTypeName());
List<FeatureId> featureIds = featureStore.addFeatures(new ListFeatureCollection(sft,features));
```

5.6. Query data

To use the HBase Ganos API to perform spatio-temporal queries, you must first create an org.geotools.data.Query object. Then, specify the filter conditions, returned columns, and sorting parameters, and use DataStore to submit queries to your HBase Ganos cluster. The result is returned as a SimpleFeature collection.

You can use Common Query Language (CQL) statements to specify conditions in the Query object. CQL is a query language provided by Open Geospatial Consortium (OGC) for the Catalogue Web Services specification. Compared with XML-based coding languages, CQL is encoded in text syntax that we are more familiar with and provides better readability and adaptability.

1.Common Query Language

You can use Common Query Language (CQL) statements to specify conditions in the Query object. CQL is a query language provided by Open Geospatial Consortium (OGC) for the Catalogue Web Services specification. Compared with XML-based coding languages, CQL is encoded in text syntax that we are more familiar with and provides better readability and adaptability.

- The comparison operators include: equals (=), not-equal-to (<>), greater-than (>), greater-than-or-equal-to (≥), less-than (<), and less-than-or-equal-to (≤). To query a city with a population greater than 15 million, you can use the PERSONS > 15000000 condition. The

PERSONS column specifies the population size.

- The BETWEEN operator is used to specify a range. For example, PERSONS BETWEEN 1000000 AND 3000000.
- The comparison operators support values of the STRING type. You can specify a STRING value on the right side of the equal sign (=) operator. For example, use the CITY_NAME = 'Beijing' statement to query the city whose name is Beijing. You can also use the LIKE operator, which is used in the same way as the LIKE operator in SQL. For example, use the CITY_NAME LIKE 'N%' statement to query all cities whose names start with the letter N.
- Compare two properties. For example, use MALE > FEMALE to query a city where the number of male residents is greater than female residents.
- Arithmetic operators include: plus (+), minus (-), multiply (*), and divide (/). For example, you can use the UNEMPLOY / (EMPLOYED + UNEMPLOY) > 0.07 condition to query all cities whose unemployment rate is greater than 7%.
- You can use the IN operator to specify properties whose values are within the specified range. The IN operator is used in the same way as the IN operator in SQL. For example, specify the ID IN ('cities.1', 'cities.12') condition, or use the CITY_NAME IN ('Beijing', 'Shanghai', 'Guangzhou') statement to query cities whose names are within the specified values.
- You can use all filter functions in Geoserver. For example, use the strToLowerCase (CITY_NAME) like '%m%' statement to convert the city names to all lowercase and then query all cities whose names contain the letter m. In this example, the letter M is not case-sensitive.
- Use the bounding box (BBOX) for the geometric filter. For example, execute the BBOX (the_geom, -90, 40, -60, 45) statement to select cities whose geographic coordinates are within the spatial range of (-90, 40, -60, 45). For more information about CQL, visit ECQL Reference.

2. Query spatial relationships

The following table lists the predicates defined in CQL that are used to query spatial relationships:

Syntax	Description
INTERSECTS(Expression , Expression)	Evaluates whether two spatial features intersect.
DISJOINT(Expression , Expression)	Evaluates whether two spatial features are disjoint.
CONTAINS(Expression , Expression)	Evaluates whether the first feature topologically contains the second feature.
WITHIN(Expression , Expression)	Evaluates whether the first feature is topologically contained in the second feature.
TOUCHES(Expression , Expression)	Evaluates whether two features touch. Features touch if they have at least one point in common.
CROSSES(Expression , Expression)	Evaluates whether two spatial features cross. Features cross if they have some but not all interior points in common.

Syntax	Description
EQUALS(Expression , Expression)	Evaluates whether two spatial features are topologically equal.
BBOX (Expression , Number , Number , Number , Number [, CRS])	Tests whether a spatial feature intersects a bounding box specified by its minimum and maximum X and Y values. The optional CRS is a string that contains an SRS code. Example: 'EPSG:1234'. The default value is the CRS of the queried layer.

For example, to obtain all features located in the spatial range (120E, 30N, 130E, 40N), run the following command:

```
DataStore ds = DataStoreFinder.getDataStore(params);
SimpleFeatureType schema=...
String stFilter = "bbox(geom, 120,30,130,40)"
Query query = new Query(schema, ECQL.toFilter(stFilter));
SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

Or to obtain all elements in the polygon built by (46.9 48.9, 47.1 48.9, 47.1 49.1, 46.9 49.1, 46.9 48.9), run the following command:

```
String stFilter = "contains('POLYGON ((46.9 48.9, 47.1 48.9, 47.1 49.1, 46.9 49.1, 46.9 48.9))', geom)
Query query = new Query(schema, ECQL.toFilter(stFilter));
```

3. Spatio-temporal queries

The following table lists the temporal predicates supported by HBase Ganos.

Syntax	Description
Expression BEFORE Time	Evaluates whether a time value is before a point in time.
Expression BEFORE OR DURING Time Period	Evaluates whether a time value is before or within a time period.
Expression DURING Time Period	Evaluates whether a time value is within a time period.
Expression DURING OR AFTER Time Period	Evaluates whether a time value is within or after a time period.
Expression AFTER Time	Evaluates whether a time value is after a point in time.

The following table lists the formats of time period supported by HBase Ganos.

Syntax	Description
Time / Time	Specifies the period defined by a beginning time and end time.
Duration / Time	Specifies the period before a given time.
Time / Duration	Specifies the period after a given time.

Note: HBase Ganos does not support queries that contain only time conditions. You must perform spatio-temporal queries.

For example, if you want to query all features located in (120E, 30N, 130E, 40N), with the time between 2014-01-01T11:45:00 and 2014-01-01T12:15:00, run the following command:

```
String stFilter = "bbox(geom, 120,30,130,40) AND dtg DURING 2014-01-01T11:45:00.000Z/2014-01-01T12:
15:00.000Z";
Query query = new Query(schema, ECQL.toFilter(stFilter));
SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

4. Property queries

After you create an index on a property column, you can run the following command to perform a property query on this column.

```
String filter = " name = 'bob'"
val q = new Query(sft.getTypeName, ECQL.toFilter(filter))
SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

In the preceding example, the value of the name column is limited.

5. Specify returned columns

You can specify parameters of the Query object to specify columns to be returned. For example:

```
String[] returnFields=... // Specify columns to be returned.
Query query = new Query(schema, ECQL.toFilter(ecqlPredicate));
query.setPropertyNames(returnFields);
SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

6. Specify the sorting order

You can create a SortBy object to specify columns to be returned. For example:

```
String sortField=... // Specify columns for sorting.
FilterFactory2 ff = CommonFactoryFinder.getFilterFactory2();
SortBy[] sort = new SortBy[]{ff.sort(sortField, order)};
query.setSortBy(sort);
SimpleFeatureCollection features=ds.getFeatureSource(schema).getFeatures(query);
```

5.7. Instructions to SDK for Java

SDK for Java allows you to access HBase Ganos based on the GeoTools API. For more information, contact Alibaba Cloud for technical support.

6.RESTful API

6.1. GeoJson format description

The REST API of HBase Ganos uses the GeoJson format to describe spatio-temporal data. For more information, see GEOJSON RFC.

The following sample code shows the spatio-temporal Point data, which can be considered as a track point:

```
{
"type": "Feature",
"geometry": {
"type": "Point",
"coordinates": [125.6, 10.1]
},
"properties": {
"name": "Dinagat Islands",
"dtg": 1536041936000,
"id": "1"
}
}
```

The data consists of three parts:

- Time data: The data is stored as properties information in properties.dtg. In this example, the time data is a timestamp in milliseconds, which is a 13-digit integer.
- Spatial data: The geometry.type parameter indicates that the spatial data is a Point data, and the geometry.coordinates indicates the latitude and longitude coordinates of the Point.
- Other properties: Other properties are stored in properties, such as properties.name and properties.id, which indicate the property information of the Point.

GeoJson libraries

You can use common GeoJson libraries to generate GeoJson strings.

- The GeoJson library geojson-jackson for Java. For more information, visit https://github.com/opendatalab-de/geojson-jackson. Run the following command to use the library.

```
FeatureCollection featureCollection = new FeatureCollection();
featureCollection.add(new Feature());

String json= new ObjectMapper().writeValueAsString(featureCollection);
```

- The GeoJson library geojson-jackson for Python.

Run the following command to include the geojson package.

```
pip install geojson
```

Run the following command to use the library.

```
point_feature = Feature(id="1",geometry=Point((1.6432, -19.123)),properties={"id":"1","dtg":1536041936
000,"name": "Dinagat Islands"})

polygon_feature = Feature(id="my_feature2",geometry=Polygon([(0,0),(0,1),(1,1),(1,0),(0,0)]),propertie
s={"id":"2","dtg":1536041936000,"name": "Dinagat Islands"})

feature_collection = FeatureCollection([point_feature,polygon_feature])
```

6.2. Register a DataStore

1. Register a DataStore

The DataStore (DS) class in the HBase Ganos RESTful API records the parameters required to connect to the backend storage. These parameters includes the Catalog Name mapped to ApsaraDB for HBase, and the ZooKeeper address used to connect to ApsaraDB for HBase. DS is used as aliases of the configurations. The following section shows how to register a DS in HBase Ganos:

URL	/ds/:alias
Method	POST
URL parameters	alias=[alphanumeric] specifies the alias of the DS, which is used as the unique identifier of a DS object.
Data parameters	The "hbase.catalog: HBase Catalog Name" parameter specifies the catalog name, which you can also customize. The "hbase.zookeepers: zookeeper" parameter specifies the ZooKeeper address that is used to connect to ApsaraDB for HBase.
Success response	Code: 200; Content: empty.
Error response	Code: 400; Content: empty.

Example: Register the data source named my_ds in the catalog_name directory of ApsaraDB for HBase.

```
curl \
'localhost:8080/geoserver/geomesa/geojson/ds/my_ds' \
-d hbase.catalog=catalog_name
-d hbase.zookeepers=localhost
```

After you register the DS, run the following command to view all registered DSs:

```
curl 'localhost:8080/geoserver/geomesa/geojson/ds'
```

6.3. Create/delete indexes

After DS is registered, you can create index tables. Index tables correspond to HBase tables. HBase Ganos automatically creates different index tables based on data types in the GeoJson format. These index tables include property index tables and spatio-temporal index tables. The following table lists the syntax to create indexes:

URL	/index/:alias/:index
Method	POST
URL parameters	alias=[alphanumeric] specifies the name of DS. index=[alphanumeric] specifies the name of the index table, which is used as the unique identifier of the index.
Data parameters	points=[Boolean] specifies whether it is a point layer. HBase Ganos optimizes point data storage. date=[alpha numeric] specifies the JSONPath of the time attribute and can be null (no time index is created); id=[alpha numeric] to specify the JSONPath of the feature id; attr=[alpha numeric] to specify the JSONPath of the secondary index attribute, which can be null (no attribute indexes are created); compression=[alpha numeric] specifies the compression option. Valid values: gz, lzo, snappy. Empty (data is not compressed)
Success response	Code: 201. Content: empty.
Error response	Code: 400, which indicates that some required parameters are not specified. Content: empty.

Example 1: Create an index named my_index in the DS named my_ds and specify the properties.id column as the id index.

```
curl \ 'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index'\
-d id=properties.id
```

Note that this statement does not create indexes for date, attr. When there are the time or other properties in the query statement, the query will trigger a full table scan.

Example 2: Create an index named my_index in my_ds, set the storage type to Point, set the JSONPath of ID to properties.id, set the date to properties.dtg, create a secondary index on the name column, and compress the data.

```
Curl \
'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index
-d id=properties.id
-d points=true
-d date=properties.dtg
-d attr=properties.name
-d compression=gz
```

6.4. PUT data

After you create an index, you can import data into the index table.

URL	/index/:ds/:index/features
Method	POST
URL parameters	alias=[alphanumeric] specifies the name of the newly created DS. index=[alphanumeric] specifies the name of the newly created index.
Data parameters	The Feature collection in the GeoJSON format.
Success response	Code: 200. Content: Add a list of Feature IDs.
Error response	Code: 400, which indicates that some required parameters are not specified. Content: empty.

Example: Import data into the index table in my_ds. Write the Feature information in the GeoJson format to the features.json file, and import the file to HBase Ganos.

```
echo '{"type":"FeatureCollection","features":[' \
'{"type":"Feature","geometry":{"type":"Point",' \
'"coordinates":[32,10]},"properties":{"id":"1","name":"n1"}},' \
'{"type":"Feature","geometry":{"type":"Point",' \
'"coordinates":[34,10]},"properties":{"id":"2","name":"n2"}}]}' \
> features.json

curl \
'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index/features'\
-H 'Content-type: application/json' \
-d @features.json
```

If the operation is successful, the system returns the ID list of the newly input data: [1,2].

6.5. Delete data

You can delete features by specifying feature IDs in the following way:

URL	/index/:alias/:index/features/:ids
Method	DELETE
URL parameters	alias=[alphanumeric] specifies the name of the newly created DS. index=[alphanumeric] specifies the name of the newly created index. id=[alphanumeric] specifies IDs of features that are to be deleted. Separate multiple feature IDs with commas (,).
Success response	Code: 200. Content: empty.
Error response	Code: 400, which indicates that some required parameters are not specified. Content: empty.

Example: Delete the features whose IDs are 1 and 2 from the my_index index on my_ds.

```
curl \
'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index/features/1,2' -X DELETE
```

6.6. Queries in HBase Ganos

HBase Ganos supports various queries such as the attribute query, ID query, and spatio-temporal range query. You can query data in the following ways:

URL	/index/:ds/:index/features

Method	GET
URL parameters	Required: alias=[alpha numeric] ds name index=[alpha numeric] index name optional: query condition indicated by q=[alpha numeric] JSON
Success response	Code: 200. Content: The Feature collection in the GeoJSON format.
Error response Code: 400, which indicates that some required parameters are not specified. Content: empty.	

```
curl\
'localhost:8080/geoserver/geomesa/geojson/index/:alias/:index/features' \ --get --data-urlencode 'q
=query conditions in JSON format'
```

The following table lists the predicates supported by HBase Ganos for attribute queries.

$lt	Less than
$lte	Less than or equal to
$gt	Greater than
$gte	Greater than or equal to

The following sections list several ways to query data:

1. Attribute queries. HBase Ganos attribute queries are performed by using predicates:

Example 1: Query features by using the id=0 condition.

```
curl \
'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index/features'\
--get --data-urlencode
'q={
"properties.id":"0"
}'
```

Example 2: Query features by using the name=n1 condition.

```
curl \
'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index/features'\
--get --data-urlencode
'q={"properties.name":"n1"}'
```

Example 3: Query features by using the age<30 condition.

```
curl \
'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index/features'\
--get --data-urlencode
'q={"properties.age":{"$lt":30}}'
```

2. Spatial queries

Example 1: Query features by using the bounding box (BBOX).

```
q={
"geometry": {
{
"$bbox" : [-180, -90, 180, 90]
}
}
```

Example 2: Query features by using the INTERSECTS predicate.

```
q={
"geometry": {
"$intersects" : {
"geometry": {
"type": "Point",
"coordinates" : [30, 10]
}
}
}
}
```

Example 3: Query features by using the WITHIN predicate.

```
q={
"geometry": {
"$within" : { "$geometry" : {
"type": "Polygon"
"coordinates": [ [ [0,0], [3,6], [6,1], [0,0] ] ]
}}
}
}
```

Example 4: Query features by using the CONTAINS predicate.

```
q={
"geometry": {
"$contains" : {
"geometry": {
"type": "Point",
"coordinates" : [30, 10]
}
}
}
}
```

3. Time queries

Example 1: Use the $during predicates to specify a time period.

```
curl \
'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index/features'\
--get --data-urlencode
'q={"dtg":{"$during":"2018-01-02T08:00:00Z/2018-03-02T10:00:00Z"}}'
```

Example 2: Use the $lt(), $gt, $lte, and $gte predicates to specify a time period.

```
curl \ 'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index'\
--get --data-urlencode
'q={"dtg":{"$lt" : "2013-01-02 00:00:00"}}'
```

4. Combination queries

AND: specify the a = 5 and b = 6 condition in the statement.

```
{ "a" : 5, "b" : 6 }
```

OR: specify the a = 5 or a = 6 condition in the statement.

```
{ "$or" : [ { "a" : 5 }, { "b" : 6 } ] }
```

Example 1: Perform a spatio-temporal query together with an attribute query:

```
curl /
'localhost:8080/geoserver/geomesa/geojson/index/my_ds/my_index/features'\
--get --data-urlencode
'q={
"geometry":{"$bbox":[116.3383,39.8291,116.3384,39.8292]},
"properties.taxi_num":"1131",
"properties.dtg":{"$gt" : "2008-02-08T08:00:00.000+0000"},
"properties.dtg":{"$lt" : "2008-02-08T12:21:16.000+0000"}
}'
```

6.7. RESTful API sample code for Python

Click here to download the RESTful API sample code for Python.

7.Spatio-temporal raster

7.1. Terms

Spatio-temporal raster data

A raster is a grid that is formed by a matrix of cells (or pixels) that are organized into rows and columns. Each cell contains attribute values that represent the information for the area within the cell. These values are also known as raster data. Raster data can be classified into two types: the thematic data and image data.

- Thematic data: The value of each cell represents a measurement or a classification to describe information, such as the pollutant concentration, rainfall, land ownership types, or vegetation types.
- Image data: Image data is also called remote sensing image. It refers to a film or photo taken by using ground remote sensing, aerial remote sensing, or aerospace remote sensing technologies to record the electromagnetic wave size of various ground objects. For example, aerial images and remote sensing satellite images.

A raster is also called a spatio-temporal raster because raster data contains both spatial and temporal attributes. The spatio-temporal raster also emphasizes the temporal characteristics of raster data, such as the raster data that manages time series.

Ganos Raster

HBase Ganos provides Ganos Raster, which provides a spatio-temporal data engine and toolset to manage and process raster data. It allows users to use ApsaraDB for HBase to store, index, query, analyze, and transmit raster data and related metadata. Raster data is stored in ApsaraDB for HBase as tiles or blocks. A primary key is assigned to each tile. You can perform spatial and temporal queries based on the primary key. Ganos Raster also supports the integration and analytics of raster data from multiple sources, such as remote sensing, photogrammetry, and thematic maps, and supports data service release features such as the Tile Map Service (TMS) and Web Map Tile Service (WMTS). Ganos Raster can be used in the fields such as location-based services, geographic image archiving, environmental monitoring and assessment, geological engineering and exploration, natural resource management, national defense, emergency response, telecommunications, transportation, urban planning, and homeland security.

Ganos Raster data model

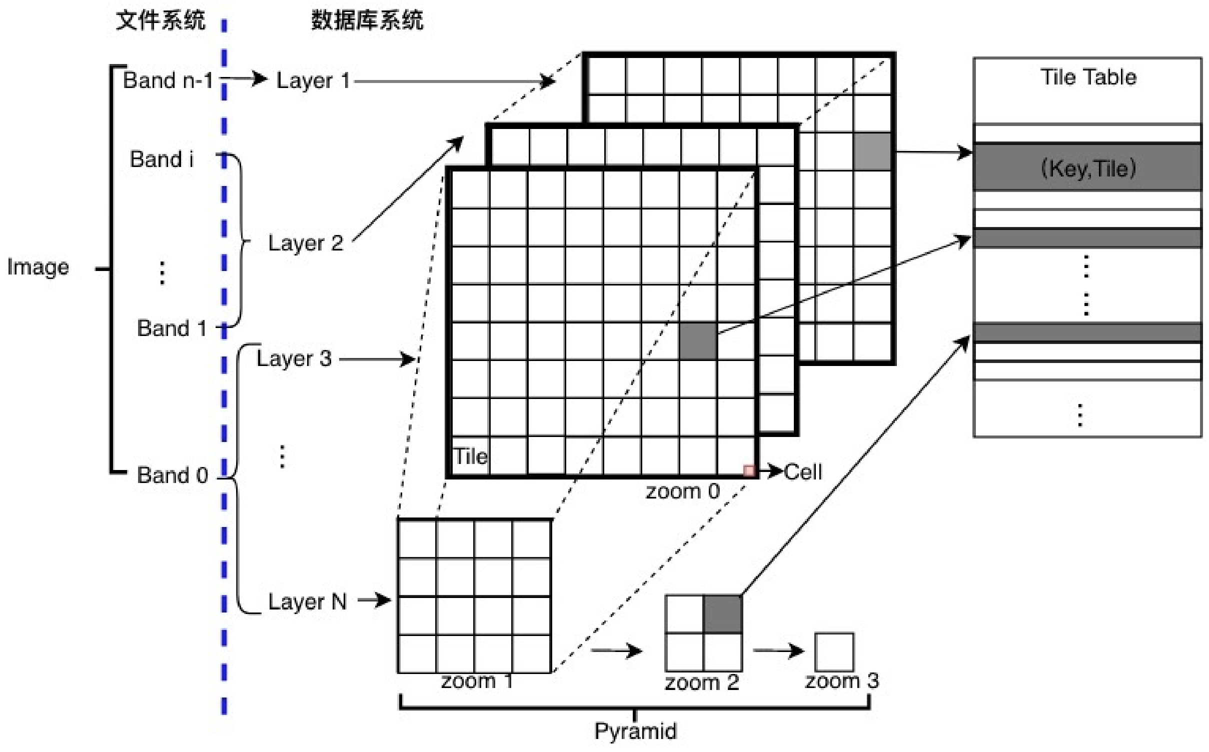
Terms

The Ganos Raster data model consists of the following features:

- Image: Specifies a remote sensing image, for example, a Tagged Image File Format (TIFF) file.
- Catalog: Specifies a data catalog, which is similar to a database. The Catalog is a logical definition. It consists of all layers and a metadata table in an ApsaraDB for HBase database. Each layer is stored in a table. The metadata of each layer is stored in a row of the metadata table.
- Cover or Coverage: A dataset that consists of multiple rasters, which is the same as a mosaic dataset.

- Layer: A 2D raster data layer that consists of multiple tiles. Each tile has a row number and a column number.
- Tile or Block: The data blocks. Each data block is a collection of pixels. A tile is the basic unit for storing raster data in a database. Each tile contains several cells. Each tile can be 256 × 256 pixels or 512 × 512 pixels.
- Cell or Pixel: A pixel in the tile. It supports various data types, such as byte, short, integer, and double.
- Key: The key value, which is the unique identifier of a tile. Valid values: SpatialKey, SpaceTimeKey, and TimeKey.
- Pyramid: The raster pyramid. Raster pyramids are used to speed up the display of raster data. Each pyramid has different levels of raster datasets. Each level corresponds to a layer. Level 0 refers to the raw raster dataset.
- Metadata: The metadata of a raster, such as the spatial range, projection types, and pixel types. The metadata of the remote sensing platform is excluded.
- Layout Definition or Layout: Defines chunking mode for tiles in a layer, the geographical range represented by each pixel, and the mapping relationship between a key and an actual coordinate system.
- Layout Scheme: A layout scheme consists of zoom numbers of all layers in a pyramid and the corresponding layout definitions.

The file representation of raster data and the logical model stored in the database are shown in the following figure:



Band and layer

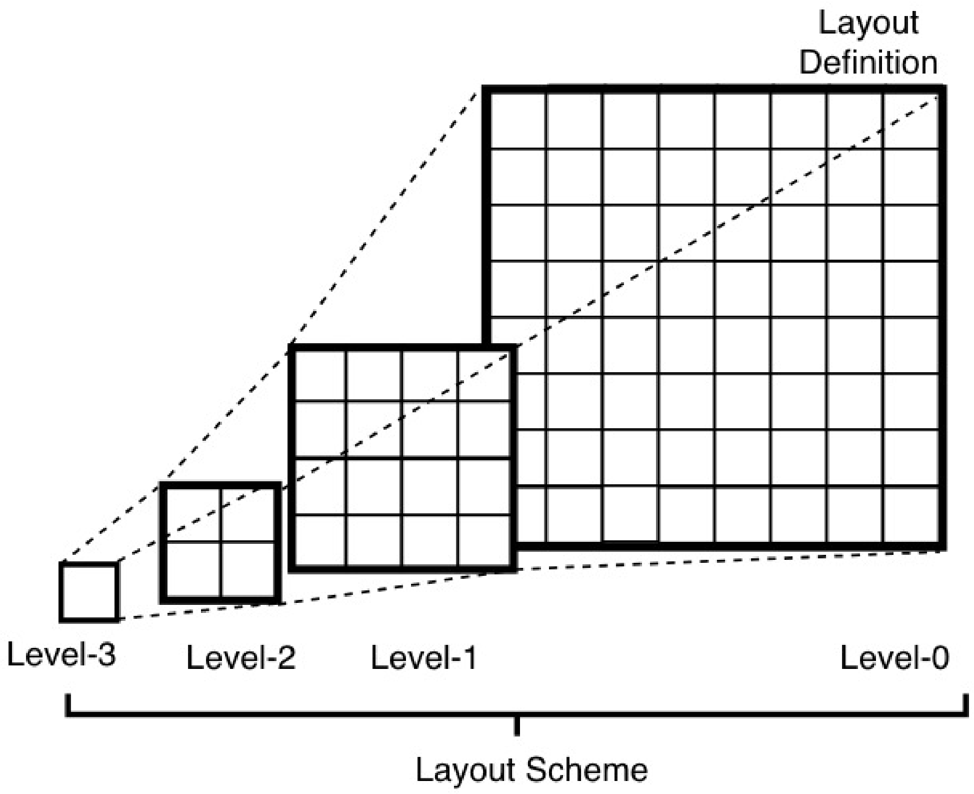
Ganos Raster uses a simple and efficient raster data model to manage the thematic data and remote sensing image data. An image consists of several bands that can be represented as a 2D raster layer. Each pixel of a band is represented as a cell. Note: To simplify the storage and management of data, make sure that all cells in each band are of the same data type and have the same projection parameters. However, different bands of an image can be heterogeneous. Each image has corresponding metadata, such as the extent, data types, projection information, and row/column numbers. Raster data is stored as layers in databases. Each layer is stored and managed as a tile in Ganos Raster. Tiles can be classified into two types: single-band tiles and multi-band tiles. Each multi-band tile contains multiple tiles.

As shown in the preceding figure, the band and layer have three types of relationships:

- One band corresponds to one layer: For single-band raster data such as the output of the model and remote sensing image analytics results, each pixel includes only one value. If a pyramid model is not built, each band corresponds to a layer.
- Multiple bands form a layer: If a layer of the remote sensing image is composed of RGB, the layer can be represented by the multi-band tile. In this case, the R, G, and B bands form a layer.
- A band includes multiple layers: If you have created a pyramid model for the raster data, each band contains multiple magnification levels. Each magnification level corresponds to a layer.

Raster pyramids

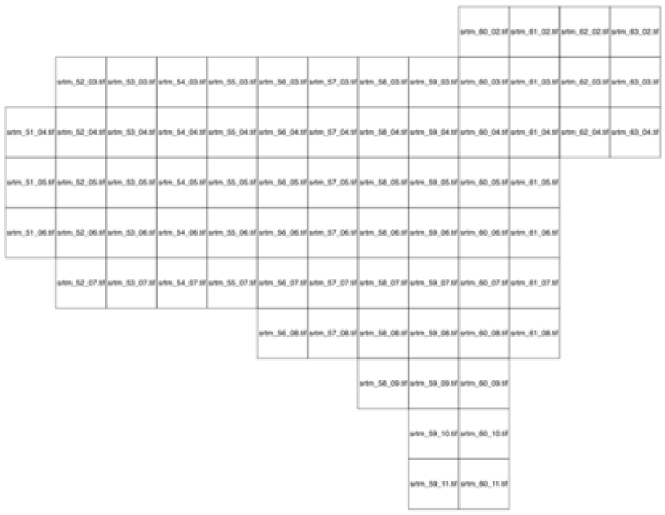
You can build pyramids for raster data to improve the efficiency of data consumption. Pyramids are a downsampled version of the original raster dataset. Each pyramid can contain many downsampled layers. Each successive layer of the pyramid is downsampled at a scale of 2:1. The following figure shows an example of four levels of pyramids created for a raster dataset:

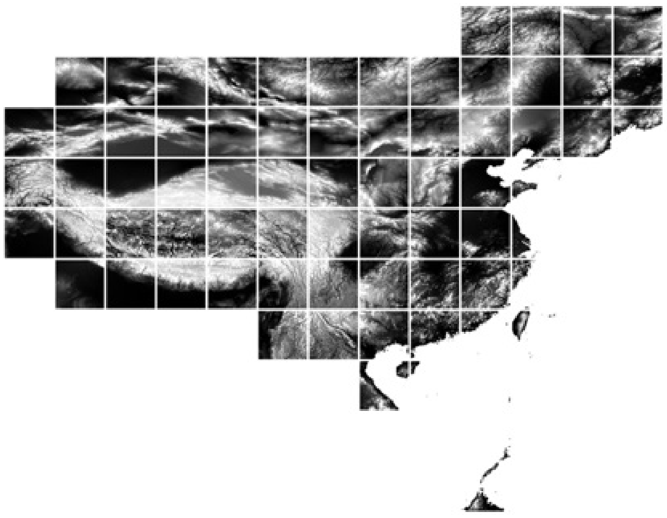


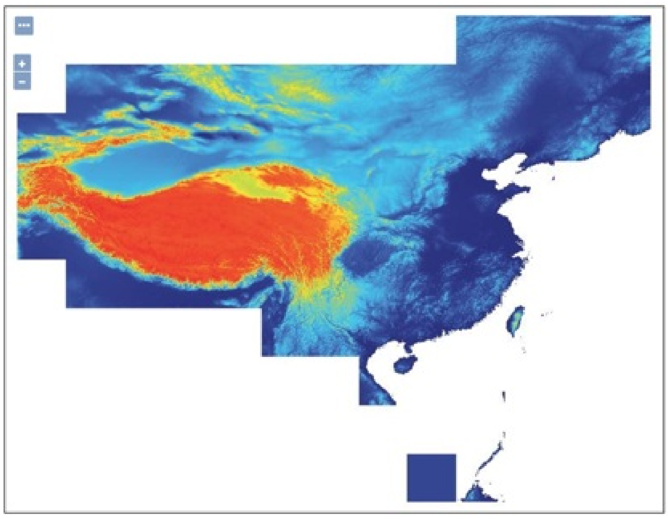
Pyramids can speed up the display of the raster data by retrieving only the data at a specified resolution. The resolution depends on the display requirement. When you draw an entire dataset, you can use pyramids to quickly display tiles of lower resolutions. As you zoom in, levels with higher resolutions are drawn. However, the performance remains unchanged. The database automatically chooses the most appropriate pyramid level based on your display scale. You only need to build pyramids once for each raster dataset. The pyramids are accessed each time you view the raster dataset. It takes more time to build a set of pyramids on a larger raster dataset. However, this saves you more time in the future.

Cover dataset

All images generated by the same sensor, such as Landsat ETM+, form a spatial coverage (or cover). The coverage provides efficient data filtering and sorting capabilities, supports spatial and temporal queries, and allows you to retrieve petabytes of raster data through the interface. The cover dataset is used to manage a set of raster datasets (images) that are stored in the form of catalogs and displayed as mosaic images. The cover dataset splices multiple images together to form a virtual dataset, which is called ImageMosaic. The cover dataset supports advanced raster query features and processing functions, and can be used as a data source to provide image services. The following figures show the extent, raw data, and mosaicked dataset of a cover dataset.





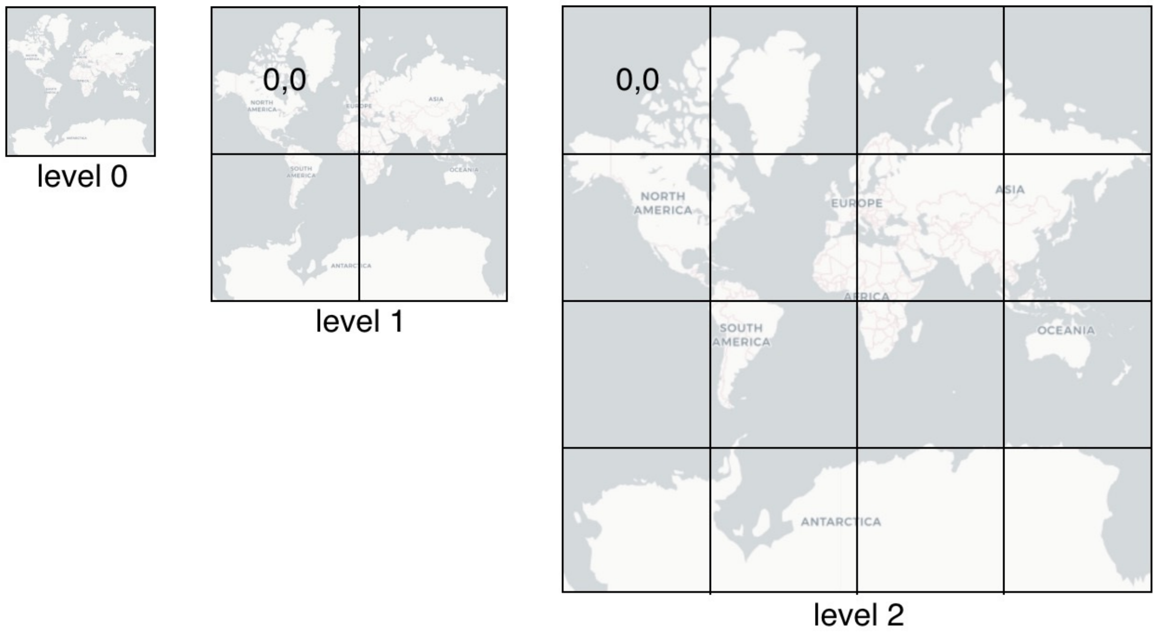


The cover dataset in Ganos Raster consists of the following parts:

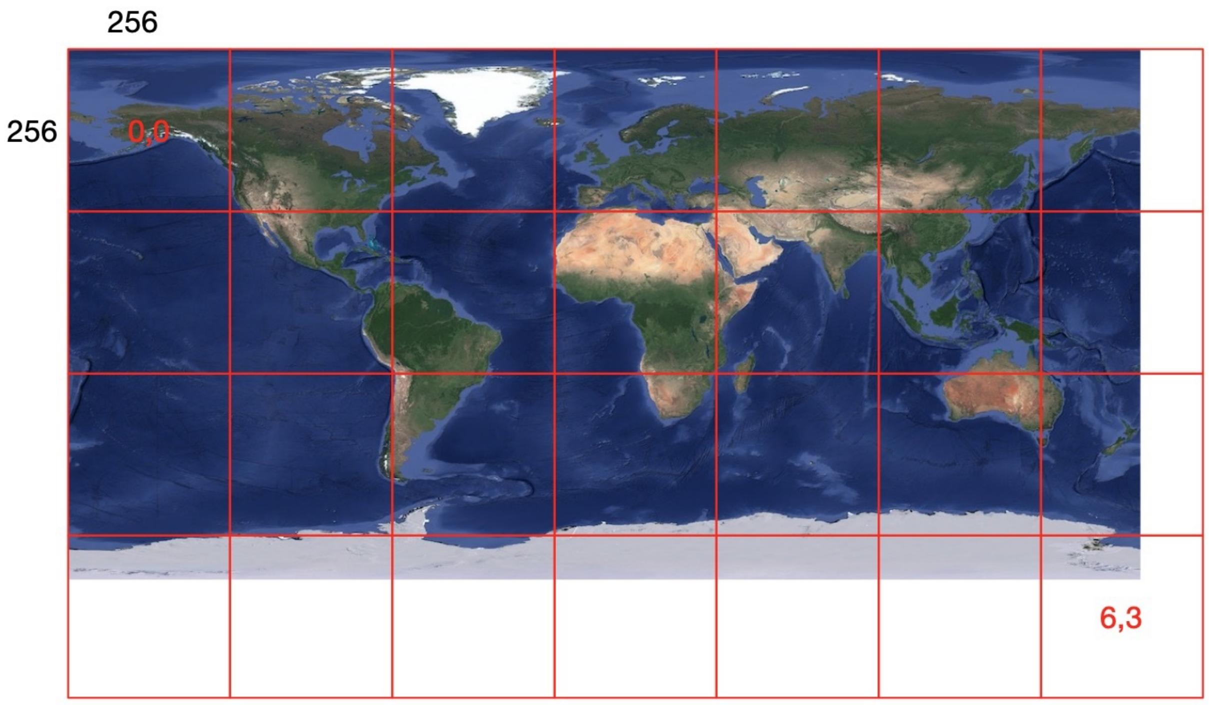
- The footprint of the raster data extent.
- The generation time of the raster data.
- The storage location of a tile of the raster data.

Layout scheme of Ganos Raster

The layout definition and layout scheme are used to define the data chunking mode of each layer. If you specify the extent and cell size (the actual spatial range defined by a cell), the layout scheme can provide a zoom and the layout definition of each level. Ganos Raster supports two modes for data chunking: zoom and local. In zoom mode, data within the globe is chunked and encoded according to the TMS standard. The tile in the upper-left corner is defined as the starting point (0,0). The coordinates increase from left to right and top to bottom. All types of raster data are chunked into tiles based on spatio-temporal grids, which makes it easy for the accumulable analytics and multivariate data fusion. By following the TMS standard, tiles chunked by this mode can be directly published to TMS for display. For example, you can use the OpenLayers component. However, the disadvantage of this mode is that the speed of data chunking is slow. The following figure shows tiles in the zoom mode:



In addition to the zoom mode, Ganos Raster also provides a mode based on the local coordinate system of the image: local. In local mode, the upper-left corner of the extent is defined as the starting point (0,0). Then the data is chunked into tiles of 256 × 256 pixels until these tiles completely cover the image area. The remaining pixels are filled with NoData values. The benefit of this mode is that Layer 0 uses resolution of the raw data, which retains the original metadata. The local mode speeds up the data chunking, and helps you update the image easily and perform efficient queries. However, the disadvantage is that different raster data do not have a unified mode for data chunking, which makes it difficult for the accumulable analytics. By default, Ganos Raster uses the local mode to chunk data and create pyramids.



Coordinate system

Ganos Raster supports coordinate systems defined by the Open Geospatial Consortium (OGC) Coordinate Reference System (CRS) standard. Users can define coordinate systems used by raster data based on European Petroleum Survey Group (EPSG) parameters. The following EPSG coordinate reference systems are often used in Ganos Raster: (1) EPSG:4326: The WGS84 projection coordinate system, which displays the latitude and longitude dimension. It is one of the most popular coordinate systems. (2) EPSG:3857:

The Pseudo-Mercator projection coordinate system, which is also called Web Mercator coordinate system. Mainstream Web mapping applications such as Google Maps all use this coordinate system.

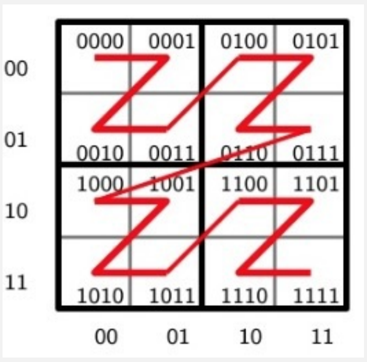
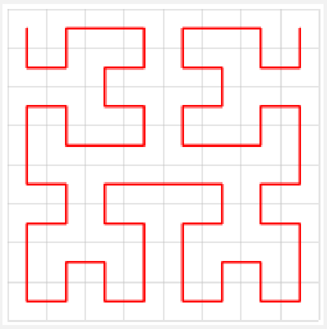
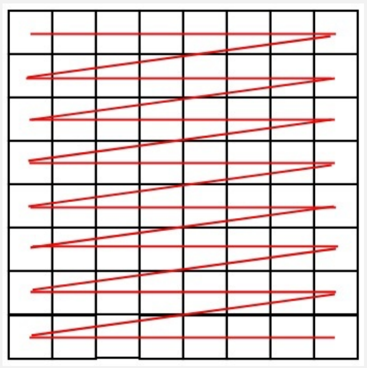
For more information about EPSG parameters, visit https://epsg.io/.

Primary key and indexes

When the raster data is chunked into tiles, you need to define how they are organized so that you can create indexes on them. Tiles are stored in ApsaraDB for HBase in the form of key-value pairs. The key of each tile consists of attributes such as the layer name, level, SpaceTimeKey, row number, and column number. HBase Ganos Raster provides two key models:

SpatialKey: The spatial primary key.

SpatialKey uses the Space Filling Curve (SFC) to encode and index tiles. Ganos Raster supports the following three types of spatial indexes:

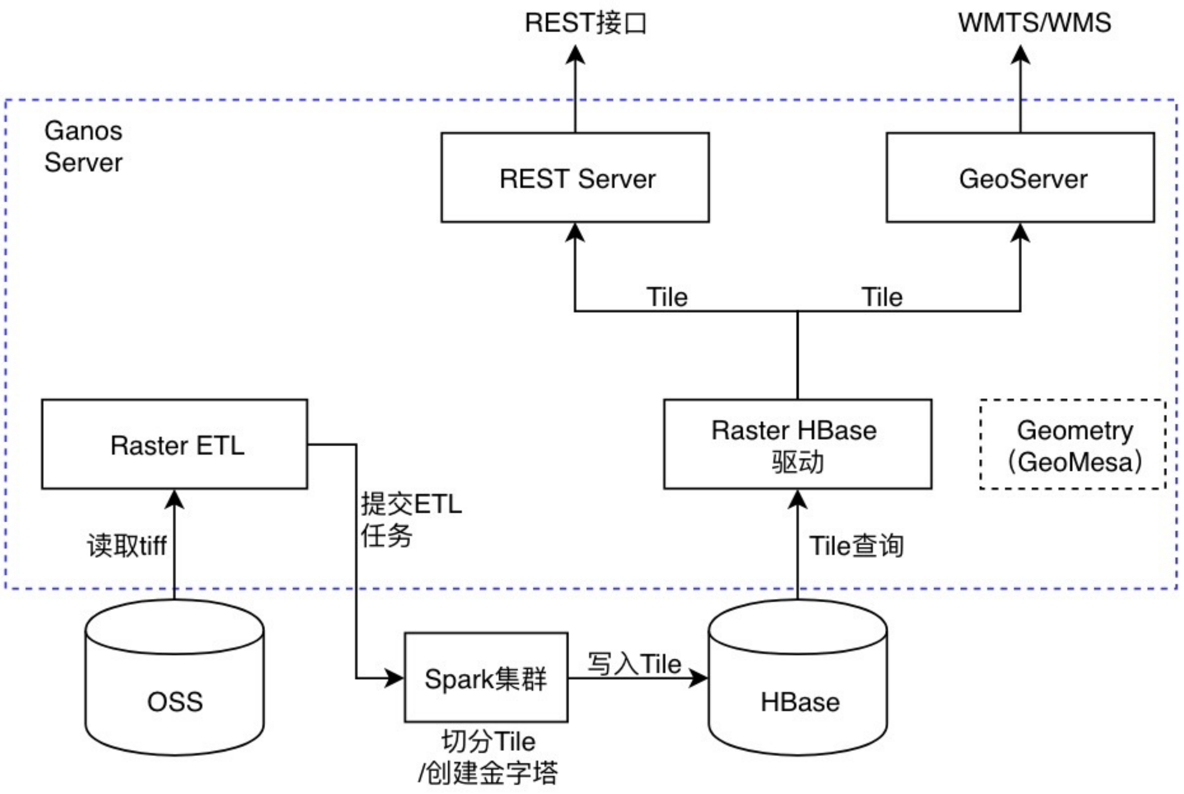
![image.png]  00 01 10 11 0000 0001 0100 0101 0010 0011 0110 0111 1000 1001 1100 1101 1010 1011 1110 1111 00 01 10 11	![image.png] 	![image.png] 
Z-Order curve	Hilbert curve	Master-Row curve

SpaceTimeKey: The spatio-temporal primary key

SpaceTimeKey is a three-dimensional SFC, which adds a time dimension based on SpatialKey.

Use an ETL tool to import raster data

Ganos Raster provides an Extract Transformation Load (ETL) tool to help you migrate data. The ETL tool allows you to migrate raster data in Object Storage Service (OSS) or Hadoop Distributed File System (HDFS) to ApsaraDB for HBase. Before you run an ETL task, you must modify the configuration file to specify parameters used during the ETL process. For more information, see: Import raster data to ApsaraDB for HBase. When the ETL task is running, the raster data in OSS or HDFS is loaded into Spark for image splicing, chunking, and re-projecting to generate a set of tiles. These tiles generate key-value pairs in the format of <Key,Value> based on the defined LayoutScheme and write them into ApsaraDB for HBase. The following figure shows the detailed process.



7.2. Import raster data to ApsaraDB for HBase

1. Preparation

Before you use the Ganos Raster Extract Transformation Load (ETL) tool to import data to the database, you must complete the following tasks:

1.1 Configure a Spark cluster

ETL uses a Spark cluster as the running environment to perform loading, re-projecting, splicing, generating tiles, and importing of the raster data. Step 1: Create a Spark cluster. For more information, visit https://help.aliyun.com/document_detail/93900.html?spm=a2c4g.11186623.6.585.70482e22jbAKMJ.

Step 2: Click here to download the ETL toolkit. Create the lib and job directories on the Cluster page in the Spark console. Then, save the downloaded ganos-raster-etl-1.0-SNAPSHOT.jar file to the lib directory.

1.2 Upload the raw raster data to OSS

ETL loads data based on the Object Storage Service (OSS) file path specified by users.

2. Import raster data to the database

2.1 Create a cover dataset

Before you import data to the database, you can select whether to create a cover dataset to support the image mosaic. Run the following commands to create a cover dataset:

```
curl '[address]:[port]/geoserver/geomesa/geojson/raster/cover/:cid/?
```

The cid parameter specifies the name of the cover dataset to be created. After you create the cover dataset, run the following command to retrieve a list of all covers:

```
curl '[address]:[port]/geoserver/geomesa/geojson/raster/cover
```

Use the DELETE method to specify the cover object to be deleted:

```
curl '[address]:[port]/geoserver/geomesa/geojson/raster/cover/:cid -X DELETE
```

You can also use the granules method to retrieve the information about all layers in the cover:

```
curl '[address]:[port]/geoserver/geomesa/geojson/raster/cover/granules/:cid
```

2.2 Write a script to import data

Before you import raster data to the database, you must create an ETL configuration file. The configuration file consists of four parts: basic configurations, input parameters, output parameters, and backend parameters. The following sample code shows a simple configuration file:

```
{
"spark":{
"spark_id":"[your spark_id]",
"spark_oss":"[the path of OSS files]",
"spark_jar":[the path of the Ganos Raster JAR file]",
"spark_httpfs":"http://[your spark_id]-master2-001.spark.rds.aliyuncs.com:14000",
"livy_server":"http://[your spark_id]-master1-001.spark.rds.aliyuncs.com:8998",
"driver-memory":"8G",
"driver-cores":4,
"num-executors":4,
"executor-memory":"4G",
"executor-cores":4,
"cover":"[the name of the cover dataset]",
},
"input":{
"format": "multiband-geotiff",
"name": "[the name of the layer]",
"cache": "NONE",
"backend": {
"type": "oss",
"path":"[the path of OSS files]"
}
},
"output":{
"backend": {
"type": "hbase",
"profile":"hbase",
"path": "[the name of the table that is used to store tiles of the layer]"
},
"reprojectMethod": "buffered",
"pyramid": true,
"tileSize": 256,
"keyIndexMethod": {
"type": "zorder"
},
"resampleMethod": "nearest-neighbor",
"layoutScheme": "zoomed",
"crs": "EPSG:3857"
"attributeTable":"[the name of the metadata table of Ganos Raster tiles]"
},
"backend":{
"backend-profiles": [{
"name":"hbase",
"type":"hbase",
"master":"master",
"hbase-name":"hbase",
"zookeepers":"[the ZooKeeper address used to connect to HBase]"
}
]
}
}
```

The following sections show parameters in the configuration file:

Spark basic configuration attributes

The basic configuration attributes define some system environment variables that are used by Spark ETL jobs, as shown in the following table:

Attribute	Description
livy_server	The Livy server address provided by the Spark cluster.
spark_httpfs	The path of auxiliary files to be uploaded.
spark_id	The ID of the Spark instance.
spark_jar	The path of the Ganos Raster JAR file.
spark_oss	The OSS path provided by Spark, where store various resources.
driver-memory	The amount of memory allocated to each driver node. This parameter is one of the Spark runtime parameters.
driver-cores	The number of cores allocated to each driver node. This parameter is one the of Spark runtime parameters.
num-executors	The number of executors. This parameter is one of the Spark runtime parameters.
executor-memory	The amount of memory allocated to each executor node. This parameter is one of the Spark runtime parameters.
executor-cores	The number of cores allocated to each executor node. This parameter is one of the Spark runtime parameters.
cover	The name of a cover dataset.

Note: The preceding Spark runtime parameters must be set based on the size and number of specified files.

Input parameters

Input parameters are used to define attributes of the data source, as shown in the following table:

Attribute	Description
format	Specifies the type of the remote sensing raster data. Valid values: geotiff and multiband-geotiff.

Attribute	Description
name	The name of a layer.
path	The path of the raw file in OSS. It can be the path of a single file or a directory. If you set this parameter to a directory, all files in the directory are mosaicked and then stored in the database as a layer.

format: The data conversion format.

geotiff	The single-band raster data, which is read in Tile Resilient Distributed Dataset (RDD) mode.
multiband-geotiff	The multi-band raster data, which is read in MultibandTile RDD mode.

Output parameters

Output parameters are used to define specific attributes of the output data, as shown in the following table:

Attribute	Description
path	The path of the stored tile.
reprojectMethod	The re-projection method. Valid values: per-tile and buffered.
resampleMethod	The resampling method. Valid values: nearest-neighbor, bilinear, cubic-convolution, cubic-spline, and average.
layoutScheme	The organized mode of the tile. Valid values: zoomed and local.
crs	The Coordinate Reference System (CRS) information about the projection.
attributeTable	The name of the metadata table of Ganos Raster tiles.

Parameters description: reprojectMethod: The re-projection method

per-tile	Indicates that values of adjacent tiles are not considered when ETL re-projects a single tile. The re-projection is fast, but it cannot reach the same effect as the buffered method.

per-tile	Indicates that values of adjacent tiles are not considered when ETL re-projects a single tile. The re-projection is fast, but it cannot reach the same effect as the buffered method.
buffered	Indicates that values of adjacent tiles are considered when ETL re-projects a single tile. Compared with the per-tile method, the re-projection is slow because a large amount of data needs to be exchanged. However, this method can help you achieve the best result.

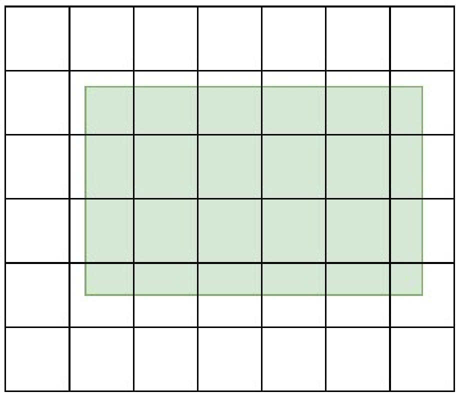
resampleMethod: The resampling method

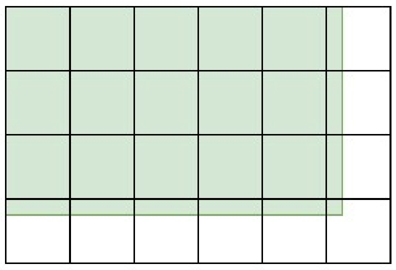
nearest-neighbor	Indicates the nearest resampling method.
bilinear	Indicates the bilinear resampling method.
cubic-convolution	Indicates the cubic convolution resampling method.
cubic-spline	Indicates the cubic spline resampling method.
average	Indicates the average value resampling method.

layoutScheme: The organized modes of tiles

zoomed	In zoomed mode, you can perform data chunking within the globe based on the TMS standard, create pyramids, and use the Web Mercator projection method.
local	In local mode, you can perform data chunking according to the data extent and coordinate system. The upper-left corner of the extent is defined as the starting point (0,0).

The following figures show the comparison of these two modes:


zoomed


local

Note: In zoomed mode, the world scope from CRS is required to build TMS pyramids. The system may need to input resampling of rasters to match the resolution of TMS levels.

Backend parameters

Backend parameters define how tiles are stored in the database, as shown in the following table:

Attribute	Description
zookeepers	The ZooKeeper address used to connect to HBase.

2.3 Submit a job

After you set runtime parameters in the JSON format, you can use the RESTful API to submit the job to the server. The following command shows the request URL:

```
curl '[address]:[port]/geoserver/geomesa/geojson/raster/etl
```

The parameters are listed in the following table:

URL	raster/etl
Request body	A set of parameters in the GeoJSON format.
Success response	Code: 200. Content: the ID of the started Spark job.
Error response	Code: 400, which indicates that some required parameters are not specified. Content: empty.

Example 1: Use the JSON file created in the preceding step to start the Spark ETL job.

```
echo '{
"livy_server":"http://[your spark_id]-master1-001.spark.rds.aliyuncs.com:8998",
"httpfs":"http://[your spark_id]-master2-001.spark.rds.aliyuncs.com:14000",
"spark_id":"[your spark_id]",
"spark_oss":"[the path of OSS files]",
"input":{
"format": "temporal-geotiff",
}
.....
}'
> params.json

curl localhost:20180/geoserver/geomesa/geojson/raster/etl
-H 'Content-type: application/json' \
-d @params.json
```

If the job is started successfully, the server returns the ID of the ETL job started by Spark, as shown in the following sample code:

```
{
"job_id": 131,
"resource_id": "ccaf0607-7009-4ed2-a9c9-a1b8f5b2b03b",
"params": {
"output": {
"path": "[the table name of the tile]",
"resampleMethod": "average",
"layoutScheme": "zoomed",
"temporalResolution": 3600000,
"crs": "ESRI:4326",
"reprojectMethod": "per-tile"
},
"input": {
"path": "oss://ganos-test/MTSAT",
"timeTag": "TIFFTAG_DATETIME",
"timeFormat": "yyyy-MM-dd HH:mm:ss",
"format": "temporal-geotiff",
"name": "[the name of the layer]"
},
"spark_id": "[your spark_id]",
"livy_server": "http://[your spark_id]-master1-001.spark.rds.aliyuncs.com:8998",
"httpfs": "http://[your spark_id]-master2-001.spark.rds.aliyuncs.com:14000",
"driver-memory": "8G",
"num-executors": 4,
"spark_oss": "[the path of OSS files]",
"backend": {
"zookeepers": "[the ZooKeeper address used to connect to HBase]"
},
"driver-cores": 4,
"executor-cores": 4,
"executor-memory": "6G"
},
"status": "STARTING"
}
```

2.4 Query jobs

After you start a job, you can run the following command to query the running ETL jobs:

```
curl '[address]:[port]/geoserver/geomesa/geojson/raster/etl/jobs
```

The parameters are listed in the following table:

URL	raster/etl /jobs
Success response	Code: 200. Content: The IDs of all jobs started by Spark.
Error response	Code: 400.

Example 1:

```
curl localhost:20180/geoserver/geomesa/geojson/raster/etl/jobs
```

Returned content:

```
{
"jobs": {
"from": 0,
"total": 1,
"sessions": [
{
"id": 131,
"state": "success",
"appId": "application_1556160980419_0137",
"appInfo": {
"driverLogUrl": "https://[your spark_id]-nginx-master1-001.spark.rds.aliyuncs.com/spark-master3-1/n
ode/containerlogs/container_1556160980419_0137_01_000001/livy",
"sparkUiUrl": "https://[your spark_id]-nginx-master1-001.spark.rds.aliyuncs.com/proxy/application_15
56160980419_0137/"
},
"log": [
"\t tracking URL: http://[your spark_id]-master1-1:9088/proxy/application_1556160980419_0137/",
"\t user: livy",
"19/06/02 15:53:25 INFO ShutdownHookManager: Shutdown hook called",
......
]
}
]
}
}
```

2.5 Query the state of a job

```
curl '[address]:[port]/geoserver/geomesa/geojson/raster/etl/status/:jobid
```

The parameters are listed in the following table:

URL	raster/etl /jobs
URL parameters	jobid: The ID of the ETL job.
Success response	Code: 200. Content: The state of the job with the specified ID.
Error response	Code: 400.

Example 1:

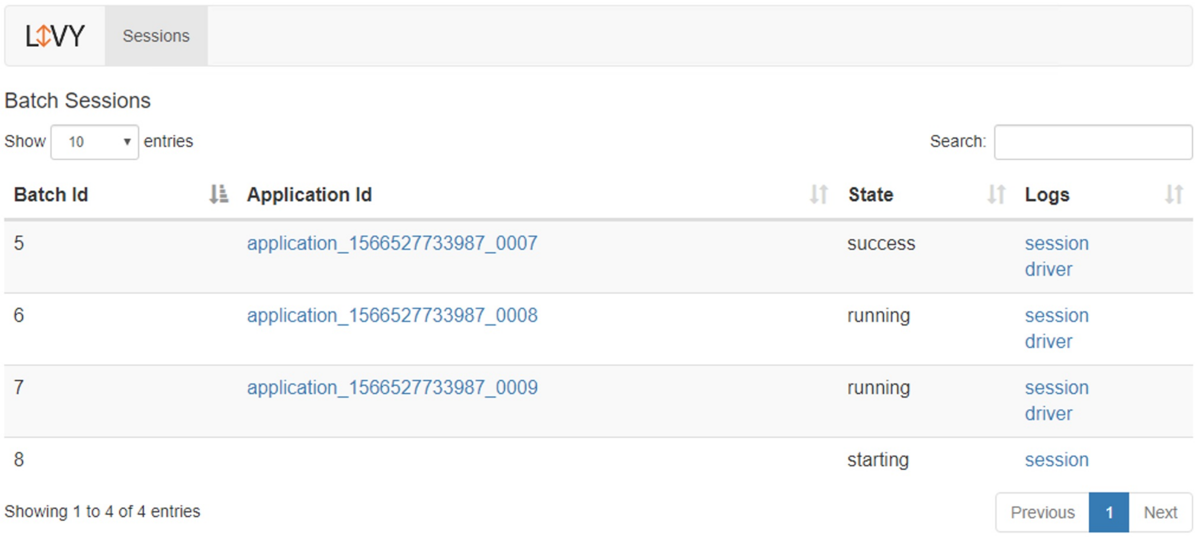
```
curl localhost:20180/geoserver/geomesa/geojson/raster/etl/status/123
```

Returned content:

```
{
"job_id": 131,
"job_info": {
"id": 131,
"state": "success",
"appId": "application_1556160980419_0137",
"appInfo": {
"driverLogUrl": "https://[your spark_id]-nginx-master1-001.spark.rds.aliyuncs.com/spark-master3-1/n
ode/containerlogs/container_1556160980419_0137_01_000001/livy",
"sparkUiUrl": "https://[your spark_id]-nginx-master1-001.spark.rds.aliyuncs.com/proxy/application_15
56160980419_0137/"
},
"log": [
"\t tracking URL: http://spark-master1-1:9088/proxy/application_1556160980419_0137/",
"\t user: livy",
"19/06/02 15:53:25 INFO ShutdownHookManager: Shutdown hook called",
.......
]
},
"status": "SUCCESS"
}
```

2.6 Access the Livy server to view jobs

You can access the Spark Livy service to view completed and running ETL jobs, as shown in the following figure:



Batch Sessions

Show 10 entries Search:

Batch Id	Application Id	State	Logs
5	application_1566527733987_0007	success	session driver
6	application_1566527733987_0008	running	session driver
7	application_1566527733987_0009	running	session driver
8		starting	session

Showing 1 to 4 of 4 entries

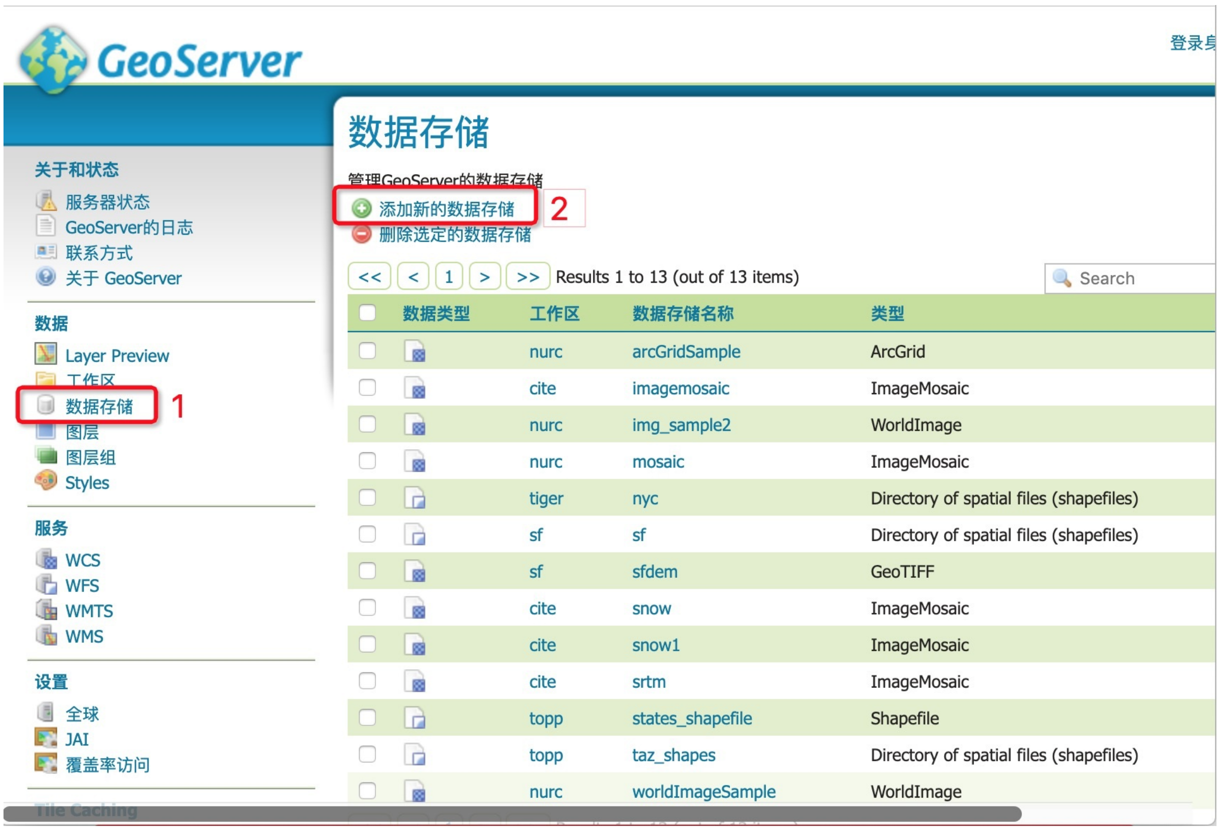
7.3. Publish raster data as a service

Overview

Ganos Raster provides the GeoServer plug-in. The plug-in allows Ganos Raster to publish raster data stored in ApsaraDB for HBase as services such as Web Map Service (WMS) and Web Map Tile Service (WMTS), which comply with the Open Geospatial Consortium (OGC) standard. This published service consists of two plug-ins: ganos-raster-image and ganos-raster-cover. The ganos-raster-image plug-in is used to publish a layer as a WMS service. The ganos-raster-cover plug-in supports mosaic datasets and is used to publish multiple layers as WMS services. You can select one of them as needed.

How to publish a service:

The following example shows how to use the Shuttle Radar Topography Mission (SRTM) data to publish a Ganos Raster layer as a WMS service. Assume that you have created a layer named srtm_china in HBase Ganos. Click here to download the ganos-raster-image-21.1.jar file, and store it in the %GEOSERVER_HOME%/WEB-INF/lib directory of the HBase Ganos instance. Then, log on to the GeoServer console, and choose Stores > Add new Store:



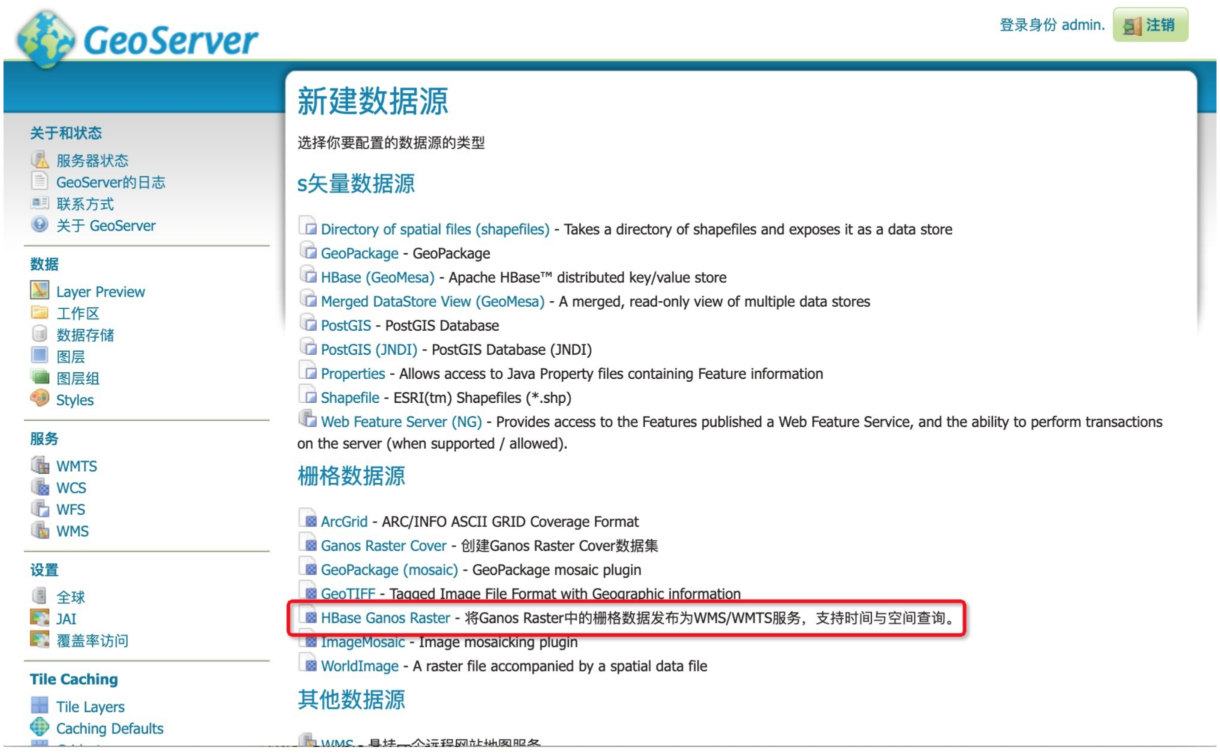
数据存储

管理GeoServer的数据存储

Results 1 to 13 (out of 13 items)

	数据类型	工作区	数据存储名称	类型
☐		nurc	arcGridSample	ArcGrid
☐		cite	imagemosaic	ImageMosaic
☐		nurc	img_sample2	WorldImage
☐		nurc	mosaic	ImageMosaic
☐		tiger	nyc	Directory of spatial files (shapefiles)
☐		sf	sf	Directory of spatial files (shapefiles)
☐		sf	sfdem	GeoTIFF
☐		cite	snow	ImageMosaic
☐		cite	snow1	ImageMosaic
☐		cite	srtm	ImageMosaic
☐		topp	states_shapefile	Shapefile
☐		topp	taz_shapes	Directory of spatial files (shapefiles)
☐		nurc	worldImageSample	WorldImage

On the New data source page, you can see the HBase Ganos Raster dataset in the Raster Data Sources section.



登录身份 admin. 注销

新建数据源

选择你要配置的数据源的类型

s矢量数据源

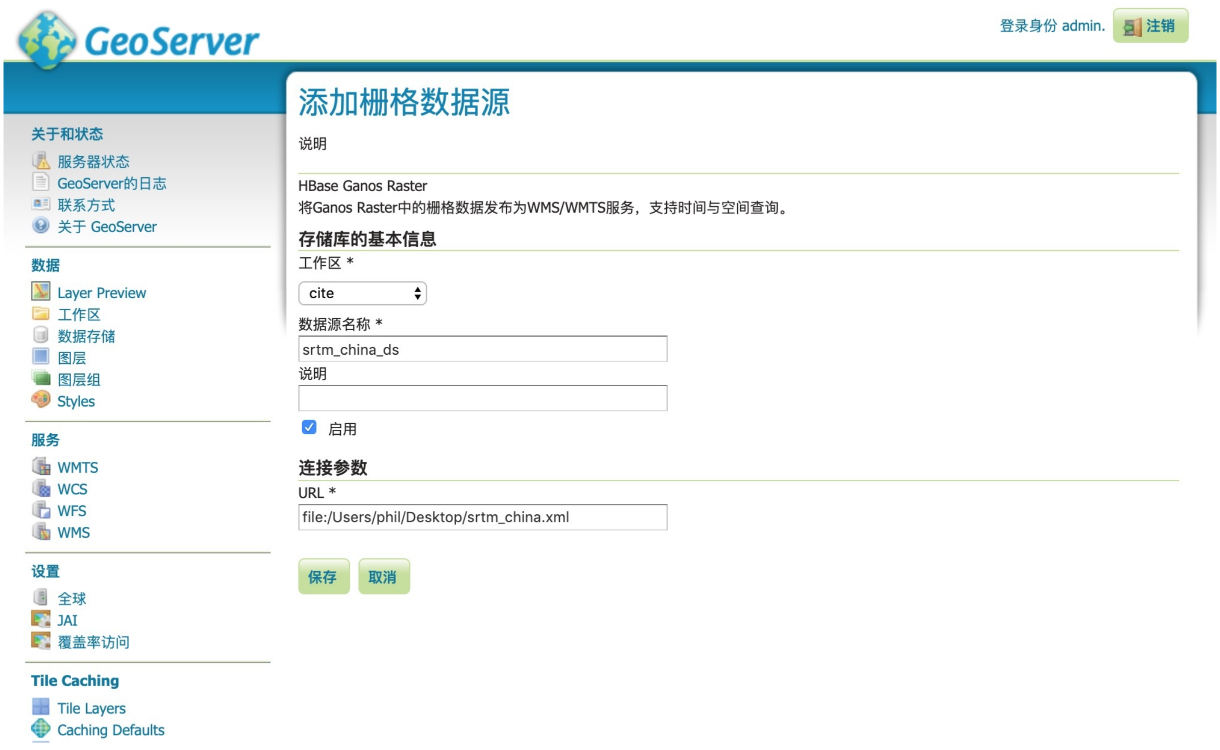
- Directory of spatial files (shapefiles) - Takes a directory of shapefiles and exposes it as a data store
- GeoPackage - GeoPackage
- HBase (GeoMesa) - Apache HBase™ distributed key/value store
- Merged DataStore View (GeoMesa) - A merged, read-only view of multiple data stores
- PostGIS - PostGIS Database
- PostGIS (JNDI) - PostGIS Database (JNDI)
- Properties - Allows access to Java Property files containing Feature information
- Shapefile - ESRI(tm) Shapefiles (*.shp)
- Web Feature Server (NG) - Provides access to the Features published a Web Feature Service, and the ability to perform transactions on the server (when supported / allowed).

栅格数据源

- ArcGrid - ARC/INFO ASCII GRID Coverage Format
- Ganos Raster Cover - 创建Ganos Raster Cover数据集
- GeoPackage (mosaic) - GeoPackage mosaic plugin
- GeoTIFF - Tagged Image File Format with Geographic information
- HBase Ganos Raster - 将Ganos Raster中的栅格数据发布为WMS/WMTS服务，支持时间与空间查询。
- ImageMosaic - Image mosaicking plugin
- WorldImage - A raster file accompanied by a spatial data file

其他数据源

Click the HBase Ganos Raster hyperlink to configure parameters.



登录身份 admin. 注销

添加栅格数据源

说明

HBase Ganos Raster
将Ganos Raster中的栅格数据发布为WMS/WMTS服务，支持时间与空间查询。

存储库的基本信息

工作区 *: cite

数据源名称 *: srtm_china_ds

说明:

[✓] 启用

连接参数

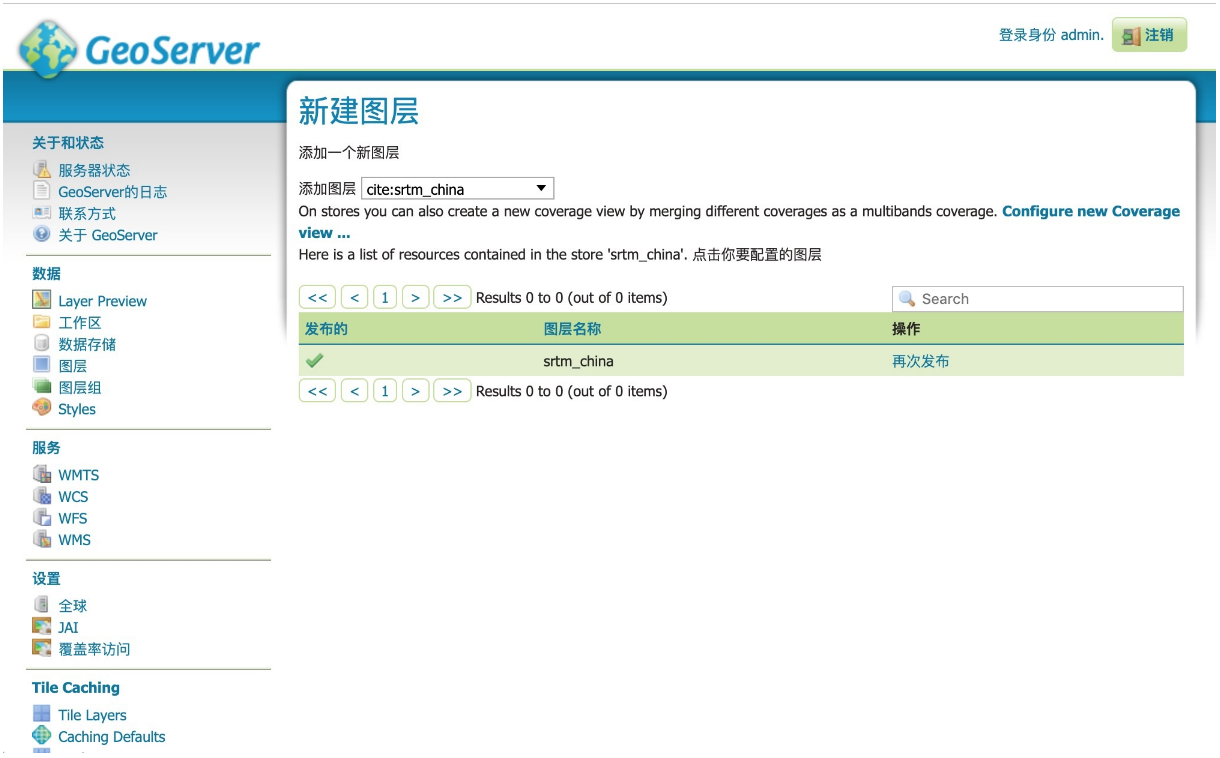
URL *: file:/Users/phil/Desktop/srtm_china.xml

In the configuration file, you must specify the ZooKeeper (ZK) data source and layer name of HBase Ganos, as shown in the following example:

```
<? xml version="1.0" encoding="UTF-8"? >
<ImageMosaicJDBCConfig>
<config version="1.0">
<coverageName name="srtm_china" />
<coordsys name="EPSG:4326" />
<zkAddress value="[the ZooKeeper address used to connect to your HBase instance]" />
<! -- interpolation 1 = nearest neighbour, 2 = bipolar, 3 = bicubic -->
<scaleop interpolation="1" />
</config>
</ImageMosaicJDBCConfig>
```

The coverageName parameter specifies the layer name in Ganos Raster to be published. The coordsys parameter specifies the projection information. The zkAddress parameter specifies the ZooKeeper address required to connect to the HBase Ganos database.

Click Save and you can find a list of layers on the New Layer page of the data source. Then, you can click Publish to publish the layer as a WMS service:



新建图层

添加一个新图层

添加图层 cite:srtm_china

On stores you can also create a new coverage view by merging different coverages as a multibands coverage. **Configure new Coverage view ...**

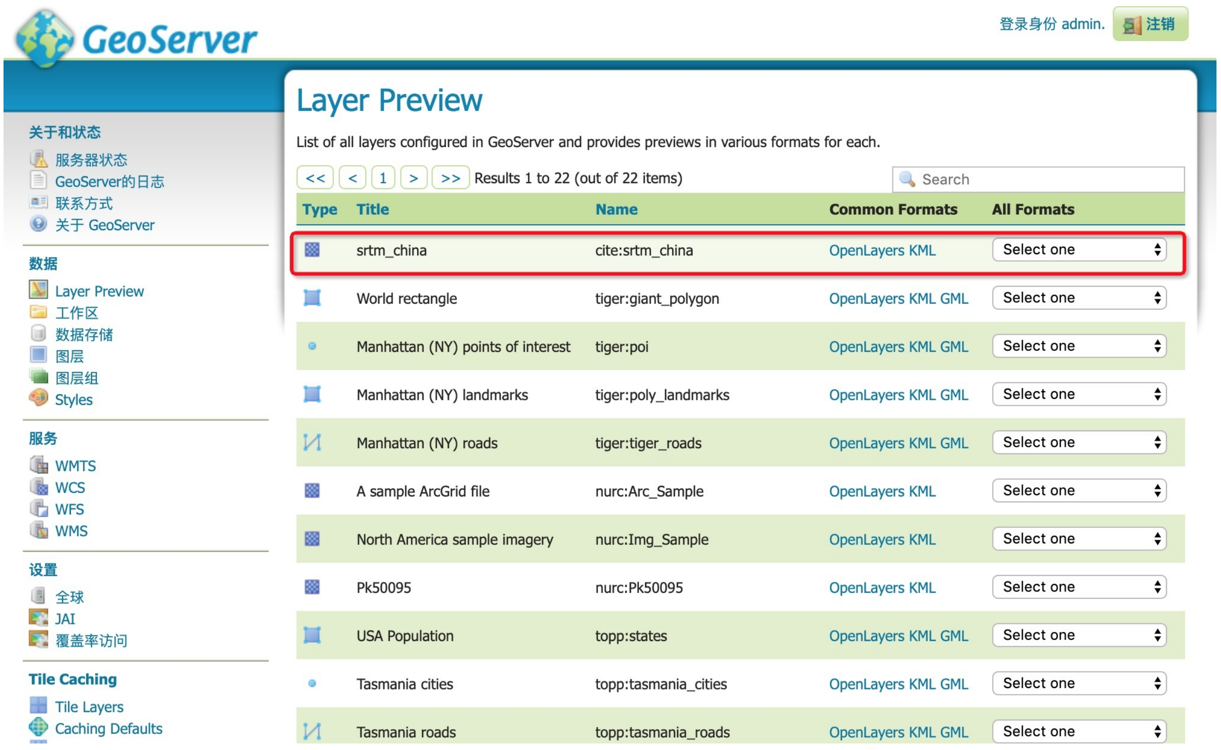
Here is a list of resources contained in the store 'srtm_china'. 点击你要配置的图层

<< < 1 > >> Results 0 to 0 (out of 0 items) Search

发布的	图层名称	操作
✓	srtm_china	再次发布

<< < 1 > >> Results 0 to 0 (out of 0 items)

After you publish the layer, click Layer Preview in the left-side navigation pane, find the newly published layer, and click OpenLayers link in the Common Formats column:



GeoServer

登录身份 admin. 注销

关于和状态
- 服务器状态
- GeoServer的日志
- 联系方式
- 关于 GeoServer

数据
- Layer Preview
- 工作区
- 数据存储
- 图层
- 图层组
- Styles

服务
- WMTS
- WCS
- WFS
- WMS

设置
- 全球
- JAI
- 覆盖率访问

Tile Caching
- Tile Layers
- Caching Defaults

Layer Preview

List of all layers configured in GeoServer and provides previews in various formats for each.

<< < 1 > >> Results 1 to 22 (out of 22 items)

Search

Type	Title	Name	Common Formats	All Formats
▨	srtm_china	cite:srtm_china	OpenLayers KML	Select one
■	World rectangle	tiger:giant_polygon	OpenLayers KML GML	Select one
●	Manhattan (NY) points of interest	tiger:poi	OpenLayers KML GML	Select one
■	Manhattan (NY) landmarks	tiger:poly_landmarks	OpenLayers KML GML	Select one
⟋	Manhattan (NY) roads	tiger:tiger_roads	OpenLayers KML GML	Select one
▨	A sample ArcGrid file	nurc:Arc_Sample	OpenLayers KML	Select one
▨	North America sample imagery	nurc:Img_Sample	OpenLayers KML	Select one
▨	Pk50095	nurc:Pk50095	OpenLayers KML	Select one
■	USA Population	topp:states	OpenLayers KML GML	Select one
●	Tasmania cities	topp:tasmania_cities	OpenLayers KML GML	Select one
⟋	Tasmania roads	topp:tasmania_roads	OpenLayers KML GML	Select one

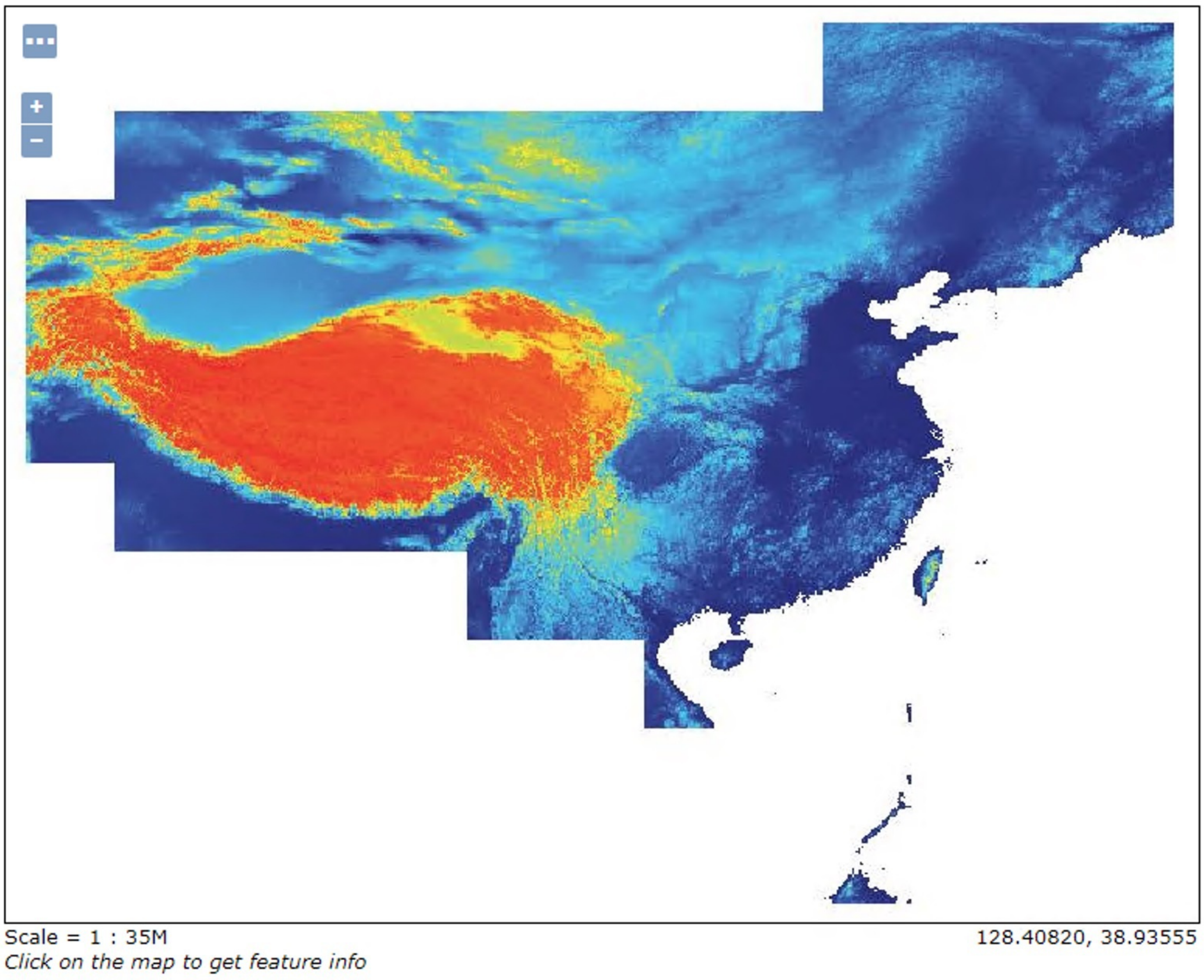
You can see a rendering of the WMS request for publishing raster data. By default, the render is displayed in a grayscale because you have not configured the Styled Layer Descriptor (SLD) file.

Scale = 1 : 35M
Click on the map to get feature info

To achieve a better display, you need to configure a SLD in GeoServer, and set colors in layers based on the elevation data:

```
<? xml version="1.0" encoding="UTF-8"? >
<StyledLayerDescriptor version="1.0.0" xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.
opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
e"
xsi:schemaLocation="http://www.opengis.net/sld http://schemas.opengis.net/sld/1.0.0/StyledLayerD
escriptor.xsd">
<NamedLayer>
<Name>gtopo</Name>
<UserStyle>
<Name>SRTM</Name>
<Title>Simple SRTM style</Title>
<Abstract>Classic elevation color progression</Abstract>
<FeatureTypeStyle>
<Rule>
<RasterSymbolizer>
<Opacity>1.0</Opacity>
<ColorMap>
<ColorMapEntry color="#2a2e7f" quantity="0" label="values" />
<ColorMapEntry color="#3d5aa9" quantity="500"/>
<ColorMapEntry color="#4698d3" quantity="1000" label="values" />
<ColorMapEntry color="#39c6f0" quantity="1500" label="values" />
<ColorMapEntry color="#76c9b3" quantity="2000" label="values" />
<ColorMapEntry color="#a8d050" quantity="2500" label="values" />
<ColorMapEntry color="#f6eb14" quantity="3000" label="values" />
<ColorMapEntry color="#fcb017" quantity="3500" label="values" />
<ColorMapEntry color="#f16022" quantity="4000" label="values" />
<ColorMapEntry color="#ee2c24" quantity="6000" label="values" />
<ColorMapEntry color="#7d1416" quantity="9000" label="values" />
</ColorMap>
</RasterSymbolizer>
</Rule>
</FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>
```

The new image is shown in the following figure:


Scale = 1 : 35M
Click on the map to get feature info
128.40820, 38.93555