第 10 节 GEE 的参数类型（Filter，Join)

筛选和连接本质上属于“属性操作”，在 GEE 中筛选和连接以“筛选器”和“连接器”的方式存在。通过本节的学习，我们可以掌握通过筛选获得目标数据，以及利用连接来结合多源数据。

10.1 Filter

筛选本质上是对冗余数据的去除。在进行数据分析之前，首先要进行的就是冗余数据的去除以及目标数据的提取，将数据量尽可能的缩减是提高运算效率的重要前提。

下边介绍比较筛选，代码及执行效果如下：

```
var China_Provinces = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");
var Name_CQ_Filter = ee.Filter.eq('NAME','Chong_Qing')
var CQ = China_Provinces.filter(Name_CQ_Filter).first()

var CQ_Area = CQ.get('Shape_Area')
var Smaller_Than_CQ_Filter = ee.Filter.lt('Shape_Area',CQ_Area)
var Smaller_Than_CQ_Provinces = China_Provinces.filter(Smaller_Than_CQ_Filter)

Map.centerObject(China_Provinces,4)
Map.addLayer(Smaller_Than_CQ_Provinces)
print(Smaller_Than_CQ_Provinces)
```

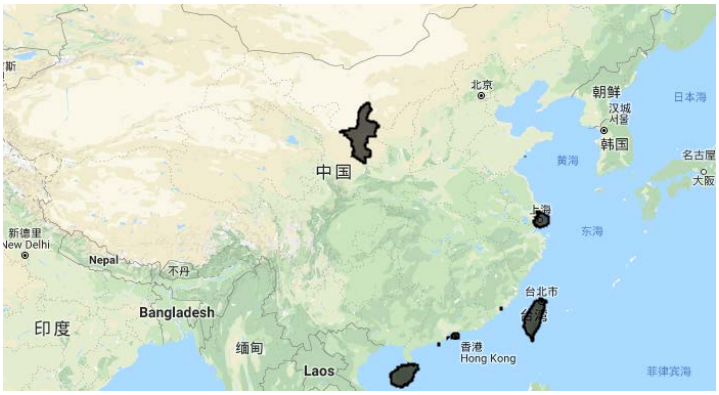

图 10.1 比较筛选

应该注意的是，比较筛选的命令一般用简写形式，下边给出其完整形式，可以作为参考：

eq	/neq	/gt	/gte	/lt	/lte
equales	/notEquals	/greaterThan	/greaterThanOrEquals	/lessThan	/lessThanOrEquals

下边介绍最大差值筛选，代码及执行效果如下：

```
var China_Provinces = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");
var Name_CQ_Filter = ee.Filter.eq('NAME','Chong_Qing')
var CQ = China_Provinces.filter(Name_CQ_Filter).first()

var CQ_Area = CQ.get('Shape_Area')
var Area_Within_10_CQ_Filter = ee.Filter.maxDifference(10,'Shape_Area',CQ_Area)
var Area_Within_10_CQ_Provinces = China_Provinces.filter(Area_Within_10_CQ_Filter)

Map.centerObject(China_Provinces,4)
Map.addLayer(Area_Within_10_CQ_Provinces)
print(Area_Within_10_CQ_Provinces)
```

图 10.2 最大差值筛选

下边介绍属性条件筛选，代码及执行效果如下：

```
var China_Provinces = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");

var Start_Shan = ee.Filter.stringStartsWith('NAME','Shan')
var End_Nan    = ee.Filter.stringEndsWith('NAME','Nan')
var Congtain_Bei = ee.Filter.stringContains('NAME','Bei')

var Provinces_Start_Shan = China_Provinces.filter(Start_Shan)
var Provinces_End_Nan = China_Provinces.filter(End_Nan)
var Provinces_Congtain_Bei = China_Provinces.filter(Congtain_Bei)

Map.centerObject(China_Provinces,4)
Map.addLayer(Provinces_Start_Shan,{color:'00ffff'})
```

```
Map.addLayer(Provinces_End_Nan,{color:'ff0000'})
Map.addLayer(Provinces_Congtain_Bei)
```

图 10.3 属性条件筛选

下边介绍属性包含筛选，代码及执行效果如下：

```
var China_Provinces = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");

var Range_Contain = ee.Filter.rangeContains( 'NAME', 'X','Z')
var Range_Provinces = China_Provinces.filter(Range_Contain)

Map.centerObject(China_Provinces,4)
Map.addLayer(Range_Provinces,{color:'00ffff'})
```

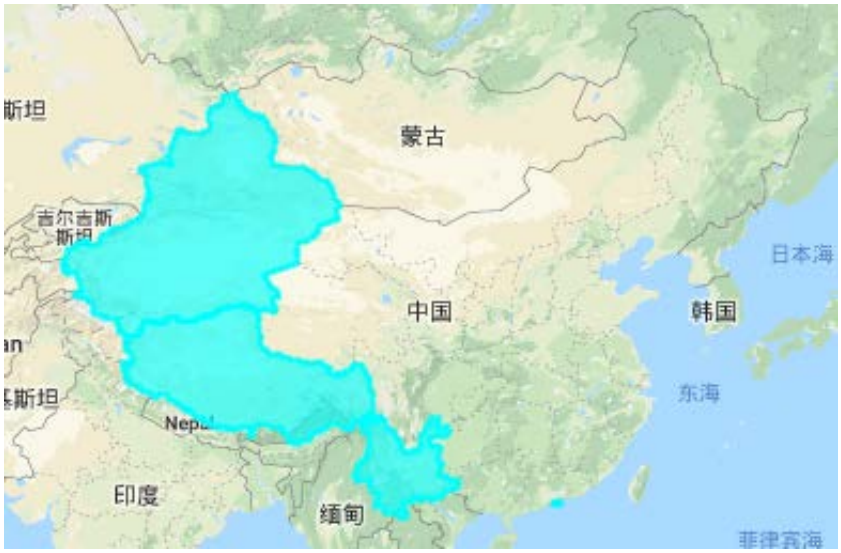

图 10.4 属性包含筛选

下边介绍字符包含筛选，代码及执行效果如下：

```
var China_Provinces = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");
var Name_CQ_Filter = ee.Filter.eq('NAME','Chong_Qing')
var Name_SC_Filter = ee.Filter.eq('NAME','Si_Chuan')
var Name_HN_Filter = ee.Filter.eq('NAME','He_Nan')

var Chongqing = China_Provinces.filter(Name_CQ_Filter).first()
                               .set('Letter',['c','h','o','n','g','q','i','n','g'])
var Sichuan     = China_Provinces.filter(Name_SC_Filter).first()
                               .set('Letter',['s','i','c','h','a','n'])
var Henan       = China_Provinces.filter(Name_CQ_Filter).first()
                               .set('Letter',['h','e','n','a','n'])
var Collection = ee.FeatureCollection([Chongqing,Sichuan,Henan],'geometry')

var List_Contain_i_Filter = ee.Filter.listContains('Letter','i')
var List_Contain_i = Collection.filter(List_Contain_i_Filter)

print(List_Contain_i)
```

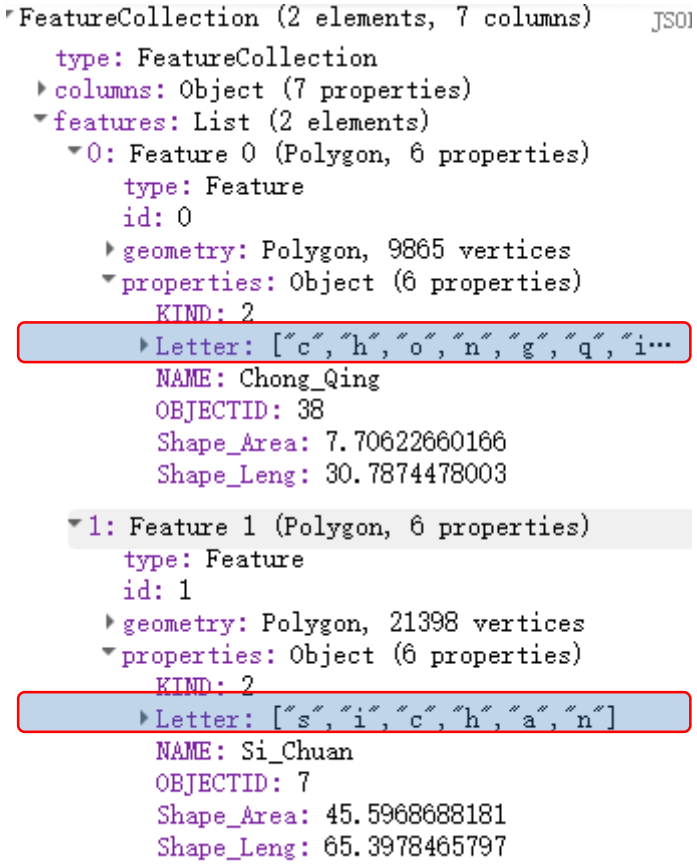

图 10.5 字符包含筛选

下边介绍字符内容筛选，代码及执行效果如下：

```
var China_Provinces = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");
var Name_List            = ee.List( ['Chong_Qing','Si_Chuan','Yun_Nan','Gui_Zhou'] );
var Inlist_Filter      = ee.Filter.inList( 'NAME', Name_List);

var List_Features      = China_Provinces.filter( Inlist_Filter );

Map.centerObject( List_Features,4);
Map.addLayer ( China_Provinces );
Map.addLayer ( List_Features, {color:'00ffff'});
```

图 10.6 字符内容筛选

下边介绍字日历筛选，代码及执行效果如下：

```
var Chongqing_Point = ee.Geometry.Point(106.3069, 29.7818)
var L8_2018 = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                  .filterDate('2018-01-01','2018-12-31')
                  .filterBounds(Chongqing_Point)
var L8_100_200_Filter = ee.Filter.calendarRange(100,200,'day_of_year')
var L8_100_200_Images = L8_2018.filter(L8_100_200_Filter)

print(L8_2018,L8_100_200_Images)
```

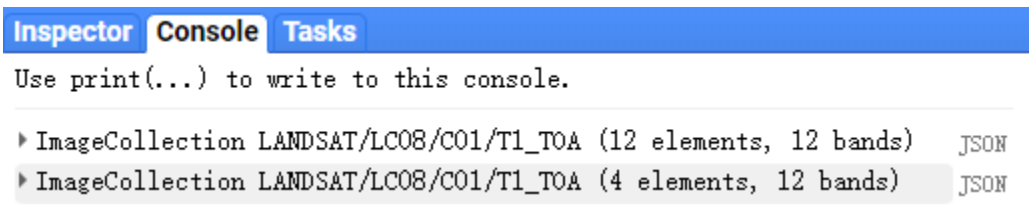

图 10.7 日历筛选

下边介绍字时间范围包含筛选，代码及执行效果如下：

```
var Chongqing_Point = ee.Geometry.Point(106.3069, 29.7818)
var L8_2018 = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                  .filterDate('2018-01-01','2018-12-31')
                  .filterBounds(Chongqing_Point)

var Date_Range      = ee.DateRange( '2018-01-01', '2018-05-01' );
var Rang_Filter    = ee.Filter.dateRangeContains( null, null, 'DATE_ACQUIRED', Date_Range );
var L8_Filted_Images = L8_2018.filter(Rang_Filter)

print(L8_2018,L8_Filted_Images)
```

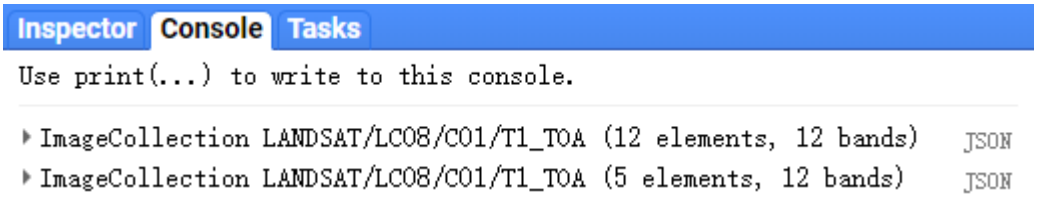

图 10.8 时间范围包含筛选

下边介绍年中日筛选，代码及执行效果如下：

```
var Chongqing_Point = ee.Geometry.Point(106.3069, 29.7818)
var L8_2018 = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                  .filterDate('2018-01-01','2018-12-31')
                  .filterBounds(Chongqing_Point)
var Day_Of_Year_Filter    = ee.Filter.dayOfYear( 50, 100 );
var L8_Day_Of_Year_Images = L8_2018.filter(Day_Of_Year_Filter)

print(L8_2018,L8_Day_Of_Year_Images)
```

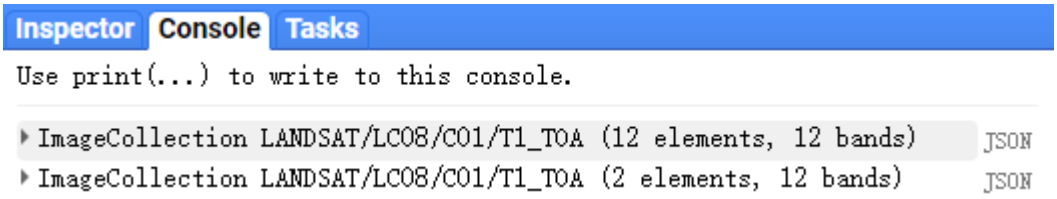

图 10.9 年中日筛选

下边介绍与或非筛选，代码及执行效果如下：

```
var China_Provinces = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");
var Start_H = ee.Filter.stringStartsWith('NAME','H')
var End_n    = ee.Filter.stringEndsWith('NAME','n')

var Filter_And = ee.Filter.and(Start_H,End_n)
var Filter_Or = ee.Filter.or(Start_H,End_n)
var Filter_Not = Filter_And.not()

var Filted__And_Provinces = China_Provinces.filter(Filter_And)
var Filted_Or_Provinces      = China_Provinces.filter(Filter_Or)
var Filted_Not_Provinces = China_Provinces.filter(Filter_Not)

Map.centerObject(Filted__And_Provinces,4)
Map.addLayer(Filted_Or_Provinces,{color:'00ffff'})
Map.addLayer(Filted__And_Provinces,{color:'0000ff'})
Map.addLayer(Filted_Not_Provinces)
```

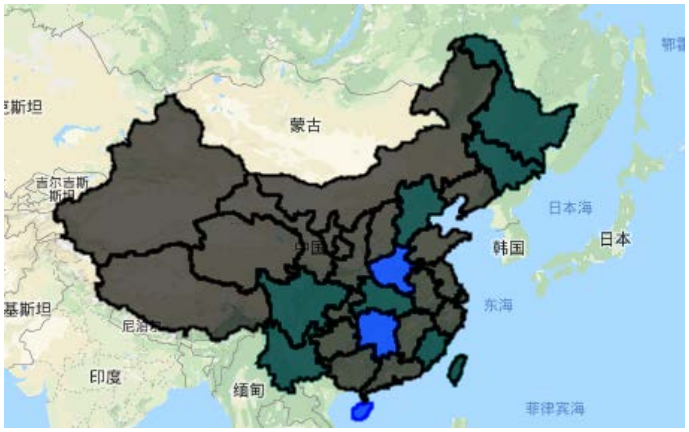

图 10.10 与或非筛选

下边是本节介绍的有关筛选的常用命令，尝试回忆其语法与功能：

```
ee.Filter.eq()    ee.Filter.neq() ee.Filter.ge()    ee.Filter.gte() ee.Filter.le()
ee.Filter.lte() ee.Filter.maxDifference()ee.Filter.stringContains()
ee.Filter.StarsWith()      ee.Filter.EndWith()          ee.Filter.Rangecontains()
ee.Filter.listContains()       ee.Filter.inList()    ee.Filter.calendarRange()
ee.FilterDateRangeContains()       ee.Filter.dayOfYear()
ee.Filter.and() ee.Filter.or()    ee.Filter.not() ee.Filter()
```

10.2 Join

连接是将两个数据集结合到一起的操作，这种操作可以分为两个部分，第一个是解决“用什么字段连接”的问题，第二个是解决“连接之后怎么办”的问题。由于 GEE 的连接涉及到筛选，因此代码较为复杂，本节代码主要来源于 GEE 的 Guide 网页。

下边介绍简单连接，目的是获得与 Sentinel 拍摄时间相差一天以内的 Landsat 8 图像，代码及执行效果如下：

```
// 引入 Landsat8 和 Sentinal 数据，利用重庆市点进行地理筛选
var L8 = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
            .filterBounds(ee.Geometry.Point(106.3737, 29.9262));
var Sentinel = ee.ImageCollection("COPERNICUS/S2")
            .filterBounds(ee.Geometry.Point(106.3737, 29.9262));
// 筛选出 2018 年的图像
var L8_2018= L8.filterDate('2018-01-01','2018-12-31')
var ST_2018= Sentinel.filterDate('2018-01-01','2018-12-31')
// 定义 1 天的变量
var One_Day_Millis = 1*24*60*60*1000
// 通过 maxDifference 定义 Sentinel 图像获取前后一天内的图像
var L8_Within_Sentinel = ee.Filter.maxDifference({
      difference: One_Day_Millis,
      leftField: 'system:time_start',
      rightField: 'system:time_start'
   })
// 定义一个 simpleJoin
var simpleJoin = ee.Join.simple();
// 应用 SimpleJoin.
var simpleJoined = simpleJoin.apply(L8_2018, ST_2018, L8_Within_Sentinel);
// 打印结果.
print('Simple join: ', simpleJoined);
```

可以看出，.join 命令的基本格式是 Join.apply()，该命令一共有三个参数，分别是“左侧数据集”，“右侧数据集”和“连接条件”，其中连接条件是一个筛选器(ee.Filter())。对于筛选器来说，需要制定的参数包括“左侧数据集连接词”，“右侧数据集连接词”和“筛选条件”。

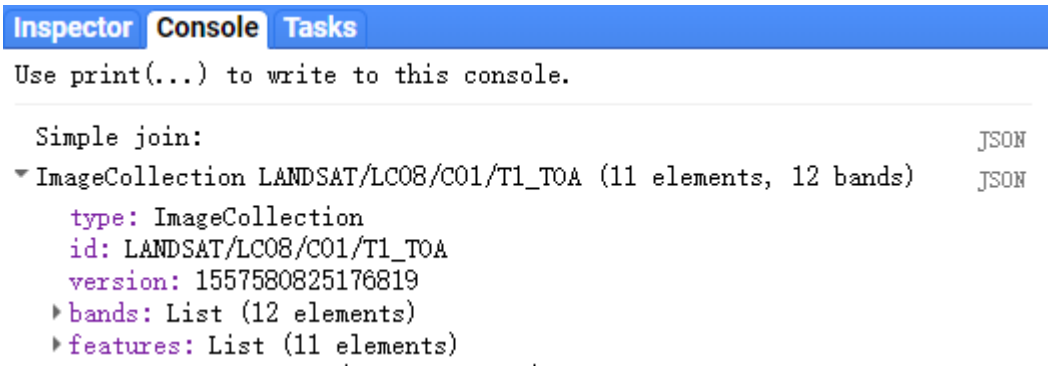

图 10.11 简单连接

下边介绍反向连接，目的是获得与 Sentinel 拍摄时间超过一天的 Landsat 8 图像，代码及执行效果如下：

```
// 引入 Landsat8 和 Sentinal 数据，利用重庆市点进行地理筛选
var L8 = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
      .filterBounds(ee.Geometry.Point(106.3737, 29.9262));
var Sentinel = ee.ImageCollection("COPERNICUS/S2")
      .filterBounds(ee.Geometry.Point(106.3737, 29.9262));

// 筛选出 2018 年的图像
var L8_2018 = L8.filterDate('2018-01-01','2018-12-31')
var ST_2018 = Sentinel.filterDate('2018-01-01','2018-12-31')

// 定义 1 天的变量
var One_Day_Millis = 1*24*60*60*1000

// 通过 maxDifference 定义 Sentinel 图像获取前后一天内的图像
var L8_Within_Sentinel = ee.Filter.maxDifference({
      difference: One_Day_Millis,
      leftField: 'system:time_start',
      rightField: 'system:time_start'
   })

// 定义一个 invertJoin
var Invet_Join = ee.Join.inverted();

// 应用 invertJoin

var Invet_Join_Results = Invet_Join.apply(L8_2018, ST_2018, L8_Within_Sentinel);
// 打印结果.
print('Invet_Join: ', Invet_Join_Results);
```

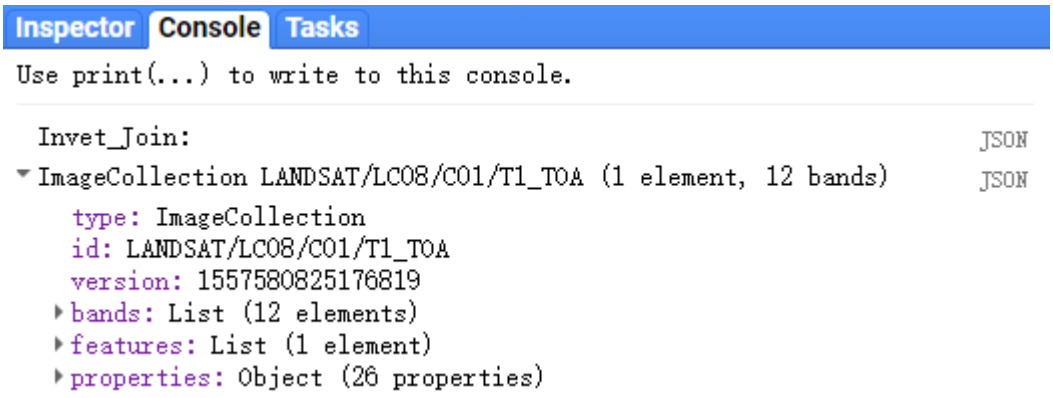

图 10.12 反向连接

下边介绍内部连接，目的是获得一个数据集，并且这个数据集中同时保留着符合条件的来自两个连接数据集的数据，代码及执行效果如下：

```
// 创建 FeatureCollection_1.
var primaryFeatures = ee.FeatureCollection([
  ee.Feature(null, {foo: 0, label: 'a'}),
  ee.Feature(null, {foo: 1, label: 'b'}),
  ee.Feature(null, {foo: 1, label: 'c'}),
  ee.Feature(null, {foo: 2, label: 'd'}),      ]);
//创建 FeatureCollection_2.
var secondaryFeatures = ee.FeatureCollection([
  ee.Feature(null, {bar: 1, label: 'e'}),
  ee.Feature(null, {bar: 1, label: 'f'}),
  ee.Feature(null, {bar: 2, label: 'g'}),
  ee.Feature(null, {bar: 3, label: 'h'}),      ]);
// 定义一个 ee.Filter.equals，要求 foo=bar.
var toyFilter = ee.Filter.equals({
  leftField: 'foo',
  rightField: 'bar'                  });
// 定义一个 innerJoin.
var innerJoin = ee.Join.inner('primary', 'secondary');
// 运用 innerJoin.
// 打印结果.
var toyJoin = innerJoin.apply(primaryFeatures, secondaryFeatures, toyFilter);
print('Inner join toy example:', toyJoin);
```

代码来源：https://developers.google.com/earth-engine/joins_inner

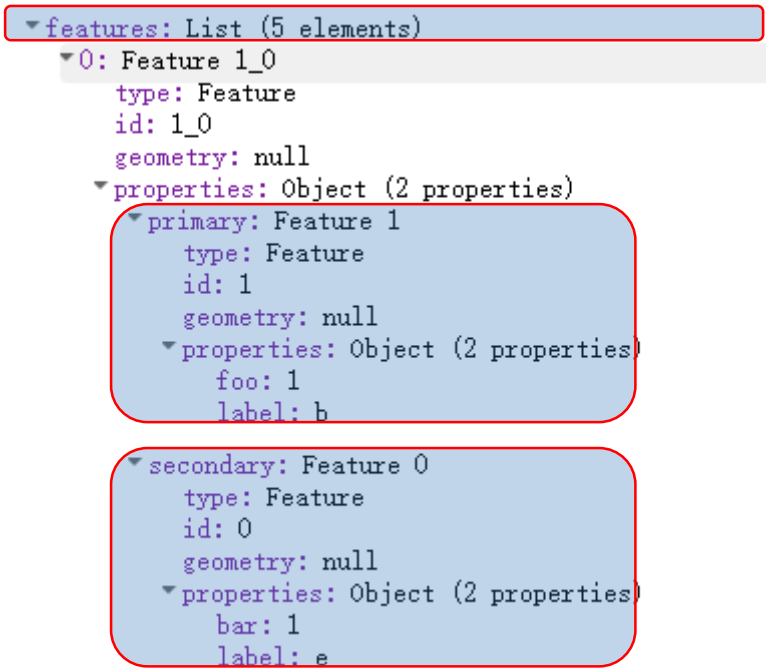

图 10.13 内部连接 1

下边介绍内部连接的一个实例，目的是将 MODIS 的 EVI 和 Quality 数据集连接并整合成为一个数据集，代码及执行效果如下：

```
// 定义时间 filter.
var dateFilter = ee.Filter.date('2014-01-01', '2014-02-01');
// 引入 MODIS 图像（EVI 产品）.
var mcd43a4 = ee.ImageCollection('MODIS/MCD43A4_006_EVI').filter(dateFilter);
// 引入 MODIS 图像（Quality 产品）.
var mcd43a2 = ee.ImageCollection('MODIS/006/MCD43A2').filter(dateFilter);
// 定义 Inner Join.
var innerJoin = ee.Join.inner();
// 定义 ee.Filter.equals，通过“系统时间”联系两种产品.
var filterTimeEq = ee.Filter.equals({
  leftField: 'system:time_start',
  rightField: 'system:time_start'
});
// 运用 Inner Join.
var innerJoinedMODIS = innerJoin.apply(mcd43a4, mcd43a2, filterTimeEq);
// 显示结果，结果为 FeatureCollection.
print('Inner join output:', innerJoinedMODIS);
// 利用 map 命令将两种产品整合.
var joinedMODIS = innerJoinedMODIS.map(function(feature) {
  return ee.Image.cat(feature.get('primary'), feature.get('secondary'));
});
// 打印结果.
print('Inner join, merged bands:', joinedMODIS);
```

代码来源：https://developers.google.com/earth-engine/joins_inner

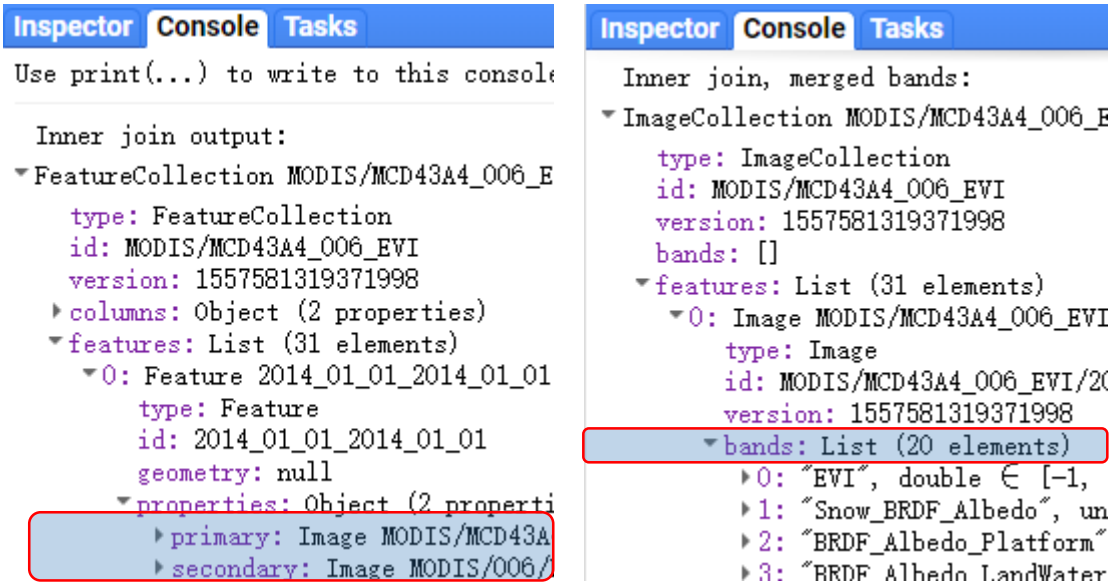

图 10.14 内部连接 2

下边介绍 SaveAll，目的是把与 Landsat 8 图像拍摄相差时间一天以内的 Sentinel 图像放到符合条件的 Landsat 8 图像的属性里，代码及执行效果如下，代码及执行效果如下：

```
// 引入 Landsat8 和 Sentinal 数据，利用重庆市点进行地理筛选
var L8 = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
      .filterBounds(ee.Geometry.Point(106.3737, 29.9262));
var Sentinel = ee.ImageCollection("COPERNICUS/S2")
      .filterBounds(ee.Geometry.Point(106.3737, 29.9262));
// 筛选出 2018 年的图像
var L8_2018= L8.filterDate('2018-01-01','2018-12-31')
var ST_2018= Sentinel.filterDate('2018-01-01','2018-12-31')
// 定义 1 天的变量
var One_Day_Millis = 1*24*60*60*1000
// 通过 maxDifference 定义 Sentinel 图像获取前后一天内的图像
var L8_Within_Sentinel = ee.Filter.maxDifference({
      difference: One_Day_Millis,
      leftField: 'system:time_start',
      rightField: 'system:time_start'                    })
// 定义一个 simpleJoin
var SaveALL_Join = ee.Join.saveAll({
   matchesKey: 'Sentinel_Match',
   ordering: 'system:time_start',
   ascending: true                    });
// 应用 SimpleJoin.
var SaveAll_Join_Images = SaveALL_Join.apply(L8_2018, ST_2018, L8_Within_Sentinel);
// 打印结果.
print('SaveALL_Join: ', SaveAll_Join_Images);
```

代码来源：https://developers.google.com/earth-engine/joins_save_all

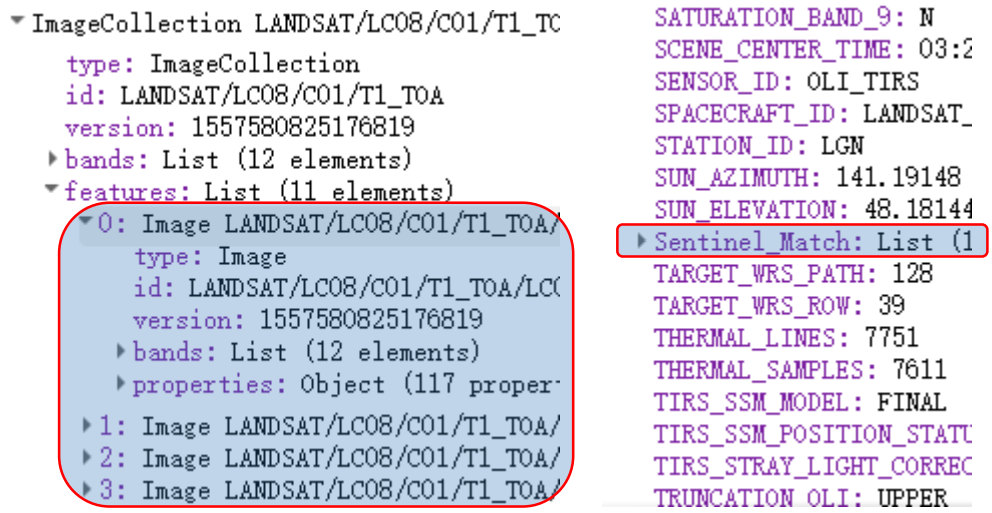

图 10.15 SaveAll 连接

下边介绍 SaveBest，目的是把与 Landsat 8 图像拍摄相差最小的 GRIDMET 气象图像数据放到符合条件的 Landsat 8 图像的属性里，代码及执行效果如下，代码及执行效果如下：

```
// 加载 Landsat8 图像
var primary = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
      .filterDate('2014-04-01', '2014-06-01')
      .filterBounds(ee.Geometry.Point(-122.092, 37.42));
// 加载 GRIDMET 气象图像数据
var gridmet = ee.ImageCollection('IDAHO_EPSCOR/GRIDMET');
// 利用 maxDifference 定义两种产品的最大时间差为 2 天.
var maxDiffFilter = ee.Filter.maxDifference({
   difference: 2 * 24 * 60 * 60 * 1000,
   leftField: 'system:time_start',
   rightField: 'system:time_start'
});
//定义 SaveBest join.
var saveBestJoin = ee.Join.saveBest({
   matchKey: 'bestImage',
   measureKey: 'timeDiff'
});
// 应用 SaveBest join.
var landsatMet = saveBestJoin.apply(primary, gridmet, maxDiffFilter);
// 打印结果.
print(landsatMet);
```

代码来源：https://developers.google.com/earth-engine/joins_save_best

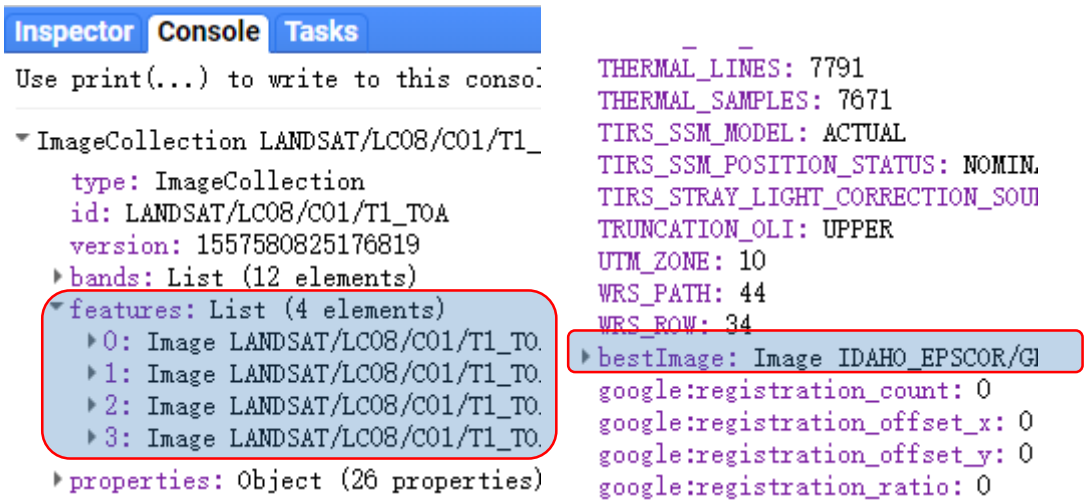

图 10.16 最优连接

SaveAll 和 SaveBest 命令都是将符合条件的图像放到“左数据集”的属性中，而内部连接 InnerJoin 则是根据筛选条件形成一一对应的数据集和。因此，SaveAll 和 SaveBest 命令适合于分析属性，而内部连接 InnerJoin 则更适合于提取数据。

下边介绍空间连接 SpatialJoin，本例的目的是把与 Landsat 8 图像的距离小于 100km 以内的入 FLUXNET 站点数据信息加入到 Landsat 8 图像中，代码及执行效果如下：

```
// 引入 Landsat-8 图像并进行时间和地点筛选.
var primary = ee.ImageCollection('LANDSAT/LC08/C01/T1_TOA')
      .filterDate('2014-04-01', '2014-06-01')
      .filterBounds(ee.Geometry.Point(-122.09, 37.42));
// 引入 FLUXNET 站点数据.
var fluxnet = ee.FeatureCollection('ft:1f85fvccyKSlaZJiAta8ojlXGhgf-LPPNmICG9kQ');
// 定义空间筛选：筛选 100km 以内数据.
var distFilter = ee.Filter.withinDistance({
  distance: 100000,
  leftField: '.geo',
  rightField: '.geo',
  maxError: 10
});
// 定义 SaveAll-Join.
var distSaveAll = ee.Join.saveAll({
  matchesKey: 'points',
  measureKey: 'distance'
});
// 应用 SaveAll-Join.
var spatialJoined = distSaveAll.apply(primary, fluxnet, distFilter);

// 打印结果.
print(spatialJoined);
```

代码来源：https://developers.google.com/earth-engine/joins_spatial

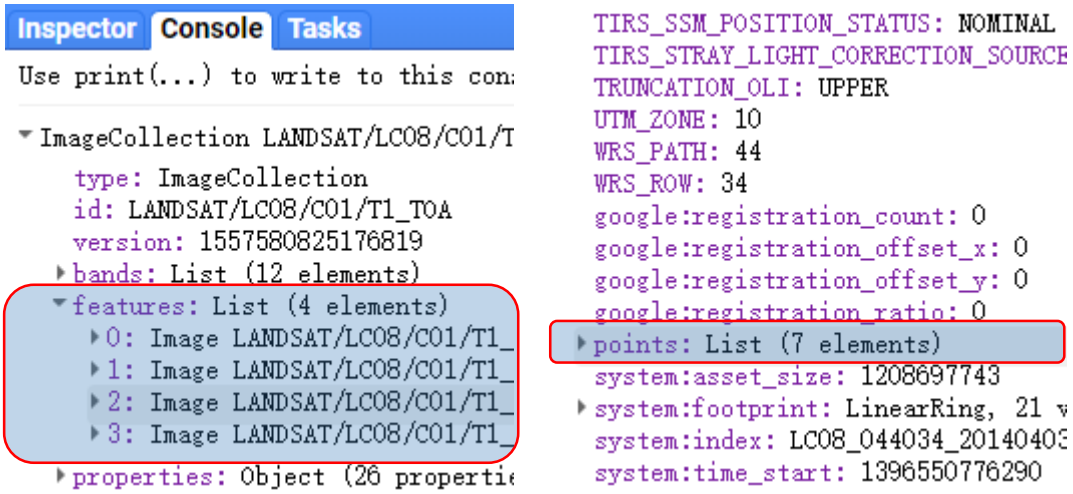

图 10.17 空间连接 1

这里需要指出的是，空间筛选器 ee.Filter.withinDistance()的参数中.geo 代表的是数据的 Geometry。

下边介绍空间连接 SpatialJoin 的相交连接，本例的目的得出与加州相交的所有 Landsat 卫星轨道边界，代码及执行效果如下：

```
// 引入加州行政边界.
var cali = ee.FeatureCollection('ft:1fRY18cjsHzDgGiJiS2nnpUU3v9JPDc2HNaR7Xk8')
  .filter(ee.Filter.eq('Name', 'California'));
// 引入 Landsat 卫星轨道数据.
var wrs = ee.FeatureCollection('ft:1_RZgjlcqixp-L9hyS6NYGqLaKOlnhSC35AB5M5Ll');
// 定义空间筛选：利用边界进行叠加筛选.
var spatialFilter = ee.Filter.intersects({
  leftField: '.geo',
  rightField: '.geo',
  maxError: 10
});
// 定义 SaveAll-Join.
var saveAllJoin = ee.Join.saveAll({
  matchesKey: 'scenes',
});
//运用 SaveAll-Join.
var intersectJoined = saveAllJoin.apply(cali, wrs, spatialFilter);
// 显示结果.
var intersected = ee.FeatureCollection(ee.List(intersectJoined.first().get('scenes')));
Map.centerObject(cali);
Map.addLayer(intersected, {}, 'WRS-2 polygons');
Map.addLayer(cali, {color: 'FF0000'}, 'California polygon');
```

代码来源：https://developers.google.com/earth-engine/joins_spatial

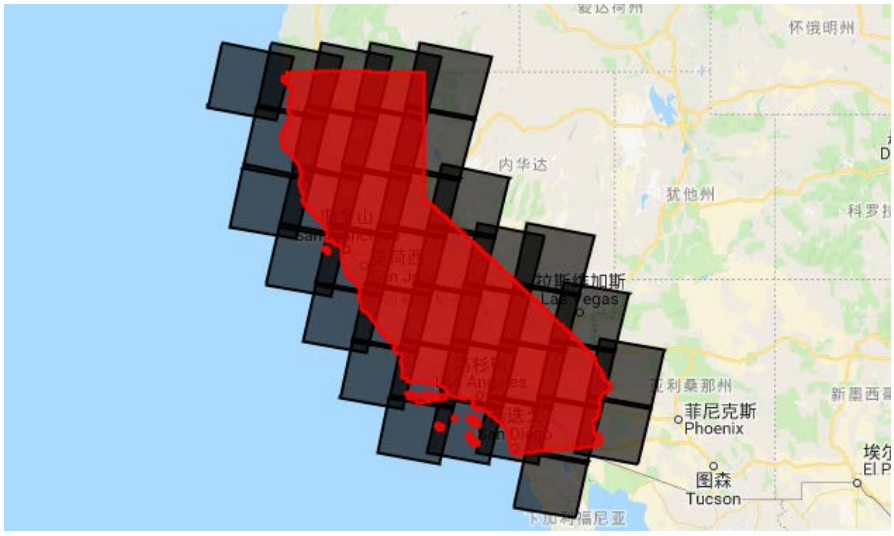

图 10.18 空间连接 2

下边是连接操作的常见命令，尝试回忆其语法与功能：

ee.Join.simple()	ee.Join.inverted()	ee.Join.inner()
ee.Join.saveAll()	ee.Join.saveBest()	ee.Join.saveFirst()
Spatial Joins (ee.Filter.withinDistance/ee.Filter.intersects)		

本材料由王金柱（西南大学&迪肯大学）创作。如有需要请与我联系。
This doc contributed by Jinzhu Wang of Southwest University & Deakin University.
Email: wangjinzhulala@gmail.com