第 7 节 GEE 的数据类型（Image，Image Collection)

GEE 作为一个遥感大数据平台，栅格图像是其核心数据。在 GEE 的代码中，栅格图像以 Image 来表示。在本节中，我们将学习 GEE 中有关栅格图像的常见命令。

7.1 Image

下边介绍 GEE 中常见的栅格图像数据。第一种是遥感图像，包括 Landsat，Sentinel 和 MODIS 系列卫星的所有图像数据。第二种是地形影像，主要是 90m 和 30m 精度的 SRTM 影像，以及局部地区的高精度地形影像。第三种则是其他影像，包括土地利用图像、气象图像等。

图 7.1 GEE 中的常见图像数据

用户还可以上传个人数据到 GEE 中，相应的步骤与矢量数据的上传类似。

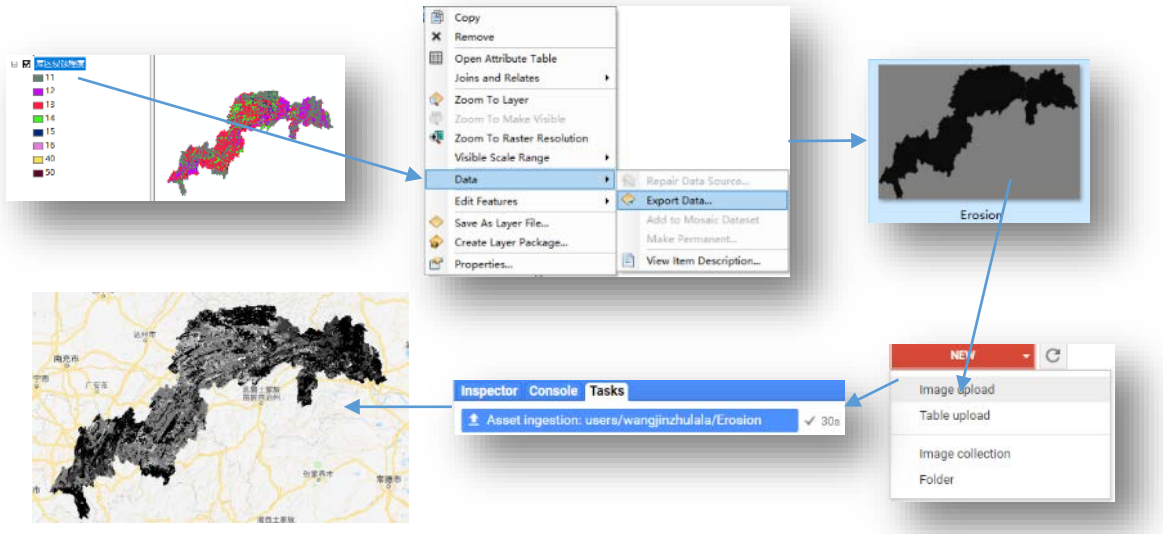

图 7.2 上传个人栅格数据

下边介绍栅格数据的创建，代码及执行效果如下：

```
var Image_1 = ee.Image( 10 );
var Image_2 = ee.Image.constant( 20 );
var Image_3 = Image_1.add(Image_2)

print(Image_1,Image_2,Image_3);
```

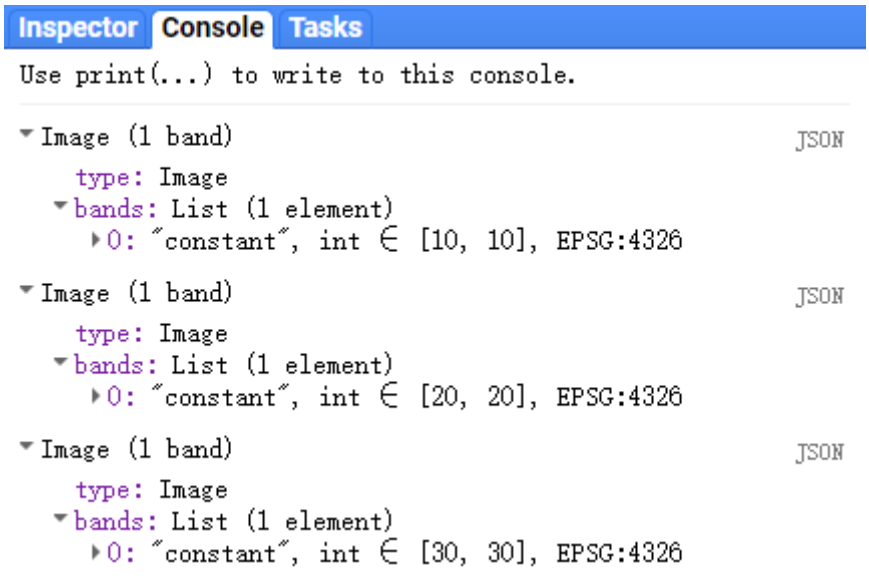

图 7.3 创建栅格数据

此时创建的栅格数据没有“分辨率”的概念，或者说此时栅格的分辨率无限小，任意位置都是指定的常数。常数栅格的创建常常见于遥感图像的时序分析，被当做方程中的常量。

下边介绍栅格的掩膜分析，代码及执行效果如下：

```
var Image_DEM    = ee.Image( 'CGIAR/SRTM90_V4' );
var Image_Cropland = ee.Image("ESA/GLOBCOVER_L4_200901_200912_V2_3")
                          .select('landcover').eq(11);
var Image_Masked    = Image_DEM.mask( Image_Cropland );

print(Image_DEM,Image_Cropland,Image_Masked);
Map.setCenter(106.4, 34.78, 4);
Map.addLayer( Image_DEM,
              {"opacity":1,"bands":["elevation"],"min":0,"max":1400,"palette":["2dff07","ff0b0b"]},
              ' Image_DEM ' );
Map.addLayer( Image_Cropland,
              {"opacity":1,"bands":["landcover"],"palette":["ffffff","fbff2d"]},
              ' Image_Cropland ' );
Map.addLayer( Image_Masked,
              {"opacity":1,"bands":["elevation"],"min":0,"max":1400,"palette":["2dff07","ff0b0b"]},
              'Image_Masked ');
```

图 7.4 栅格的掩膜分析

下边介绍栅格的裁剪，代码及执行效果如下：

```
var DEM = ee.Image("USGS/SRTMGL1_003");
var CQ = ee.FeatureCollection("users/wangjinzhulala/China_Provinces")
                .filterBounds(ee.Geometry.Point([106.8, 29.3]))
var CQ_DEM = DEM.clip(CQ)

Map.setCenter(106.8, 29.3,7)
Map.addLayer(CQ_DEM,{min:0,max:2000})
```

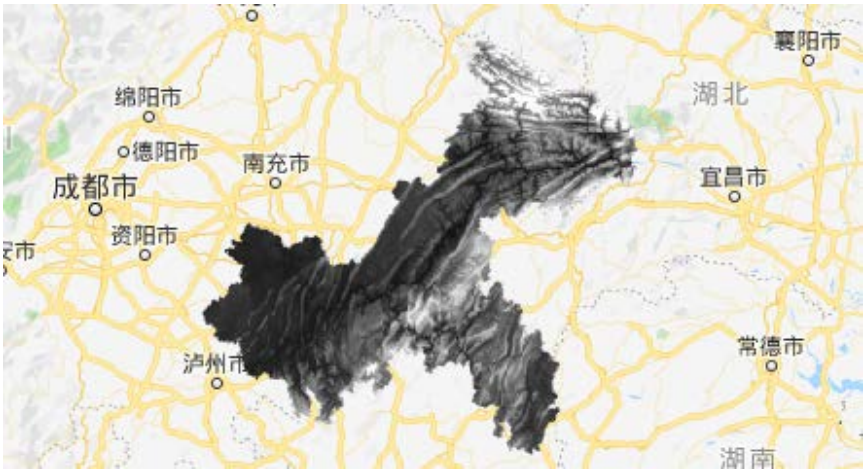

图 7.5 栅格的裁剪

需要指出的是，栅格图像的掩膜操作只是给 GEE 指出图像的哪些部分不参加运算，而栅格的裁剪则是直接将裁剪数据意外的部分删除。但在多数情况下，两者可以互用。

下边介绍栅格的选择创建，代码及执行效果如下：

```
var Glob_Cover = ee.Image("ESA/GLOBCOVER_L4_200901_200912_V2_3");
var Land_cover = Glob_Cover.select('landcover')
var Land_cover_quality = Glob_Cover.select('qa')

Map.setCenter(107.56, 34.79,4)
Map.addLayer(Land_cover)
Map.addLayer(Land_cover_quality)
```

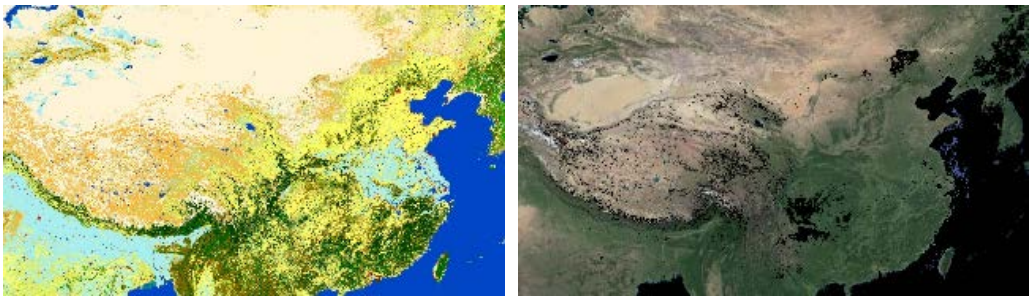

图 7.6 栅格的选择创建

这里需要说明，GLOBCOVER_L4_2009 数据包含两个波段，一个波段是“landcover”，用来说明土地利用类型。另一个波段是“qa”，用来说明某位置的数据来源(0 表示来源于 GLOBCOVER 项目，1 表示来源于其他数据集)。

下边介绍栅格的波段裁取创建，代码及执行效果如下：

```
var L8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT");
var L8_One = ee.Image(L8.filterBounds(ee.Geometry.Point(106.4958, 29.5856)).first())
var L8_Slice = L8_One.slice(2,5);

print(L8_One,L8_Slice)
```

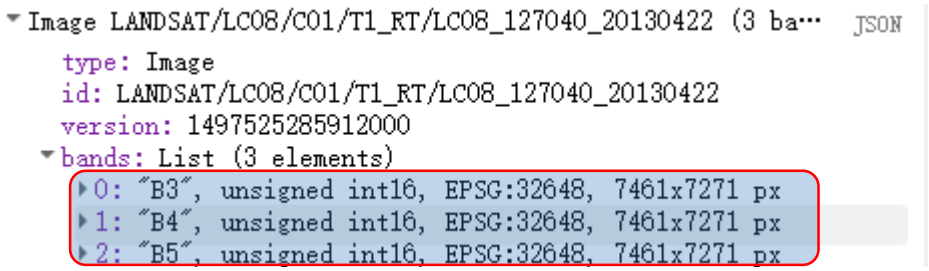

图 7.7 栅格的波段裁取

波段裁取命令的两个参数分别表示裁取的起始和结束位置，由于代码语言中是从 0 开始计数的，所以本例中的 2 和 5 分别表示第 3 和第 6 个位置。同时也可以看出，波段裁取只执行到结束位置的前一个位置。

下边介绍栅格的添加，代码及执行效果如下：

```
var L8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT");
var L8_One = ee.Image(L8.filterBounds(ee.Geometry.Point(106.4958, 29.5856)).first())

var L8_B5 = L8_One.select('B5');
var L8_B4 = L8_One.select('B4');
var L8_B5_4 = L8_B5.addBands(L8_B4)

print(L8_B5,L8_B4,L8_B5_4)
```

```
Image LANDSAT/LC08/C01/T1_RT/LC08_127040_20130422 (2 bands)  JSON
  type: Image
  id: LANDSAT/LC08/C01/T1_RT/LC08_127040_20130422
  version: 1497525285912000
  bands: List (2 elements)
    0: "B5", unsigned int16, EPSG:32648, 7461x7271 px
    1: "B4", unsigned int16, EPSG:32648, 7461x7271 px
```

图 7.8 栅格的添加

下边介绍栅格的重投影命令，代码及执行效果如下：

```
var L8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT");
var L8_One = ee.Image(L8.filterBounds(ee.Geometry.Point(106.4958, 29.5856)).first())
var L8_Reproject = L8_One.reproject('EPSG:3857',null,100)

print(L8_One.select('B1').projection(),L8_Reproject.projection())

Map.setCenter(106.58365, 29.56972,15)
Map.addLayer(L8_One,{"bands":["B5","B4","B3"],"min":10586,"max":18154},'30m')
Map.addLayer(L8_Reproject,{"bands":["B5","B4","B3"],"min":10586,"max":18154}, '100m')
```

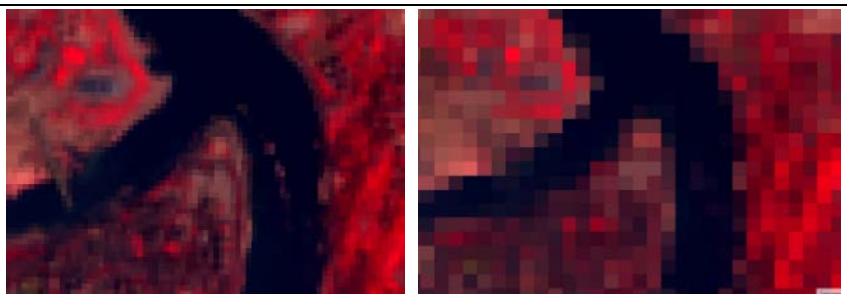

图 7.9 栅格的重投影

这里注意，矢量的重投影命令是.transpose()，而栅格的重投影命令式.Reproject，两者的操作目的是相同的，但却是不可混用的两个命令。

下边介绍栅格的色彩转换，代码及执行效果如下：

```
var L8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT");
var L8_One = ee.Image(L8.filterBounds(ee.Geometry.Point(106.4958, 29.5856))
                      .first()).slice(2,5).unitScale(0,32468)
var L8_HSV = L8_One.rgbToHsv()
var L8_RGB = L8_HSV.hsvToRgb()

print(L8_One,L8_HSV,L8_RGB)

Map.setCenter(106.5, 29.5,9)
Map.addLayer(L8_One,{},'Origin')
Map.addLayer(L8_HSV,{},'HSV')
Map.addLayer(L8_RGB,{},'RGB')
```

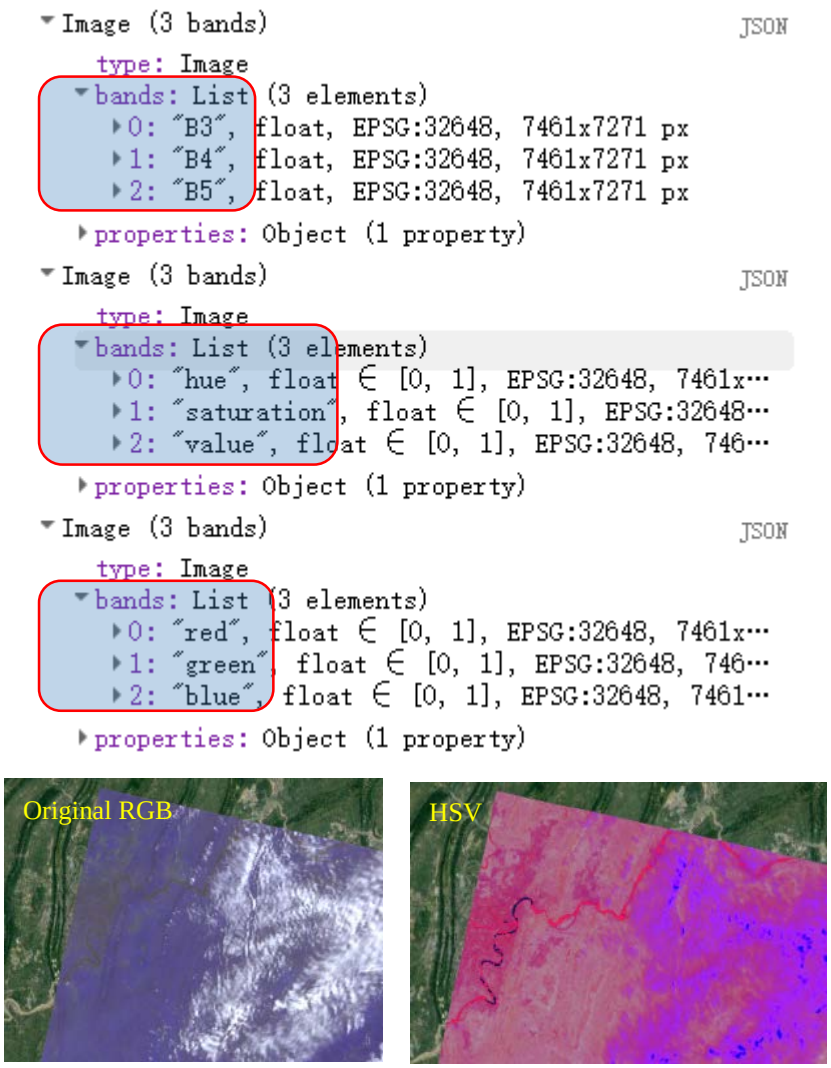

图 7.10 栅格的色彩转换

下边介绍栅格的数字格式转换，此处与前述例子相同，限于篇幅这里只给出命令代码，其中上下两行的对应命令是等效的：

.uint8	.uint16	.uint32	.uint64	.int8	
.toUnit8	toUint16	toUint32	toUint64	toInt8	
.int16	.int32	.int64	.long	.float	.double
.toInt16	.toInt32	.toInt64	.toLong	.toFloat	.toDouble

下边介绍栅格的波段数据格式定义，代码及执行效果如下：

```
var L8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT");
var L8_One = ee.Image(L8.filterBounds(ee.Geometry.Point(106.4958, 29.5856))
                .first()).slice(2,5).unitScale(0,32468)
var L8_Cast = L8_One.cast({'B5':'long','B4':'double','B3':'float'},['B5','B4','B3'])
print(L8_One,L8_Cast)
```

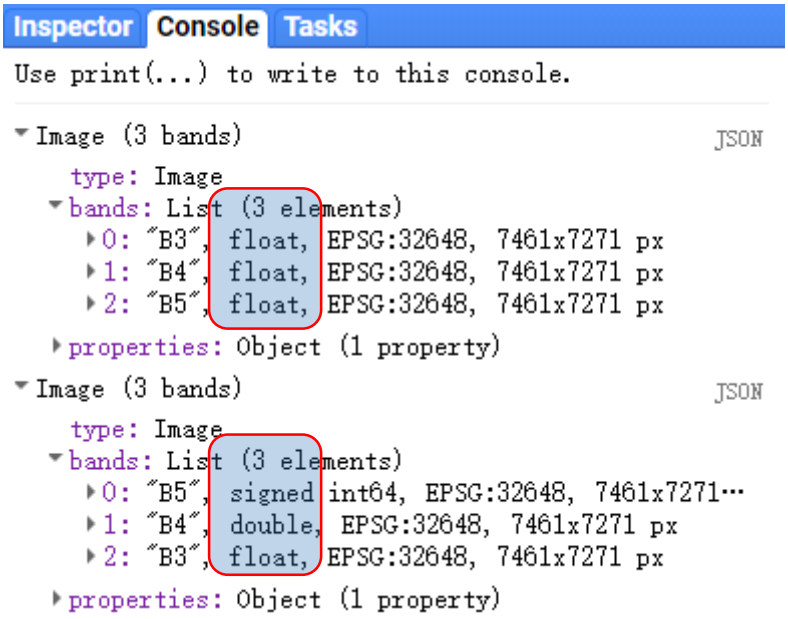

图 7.11 栅格的波段数据格式定义

下边介绍栅格的属性改写，代码及执行效果如下：

```
var L8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT");
var L8_One = L8.first().set('note',"This image was created by 'rgbToHsv()'")
                              .setMulti({'author':'Jinzhu Wang', 'mood':'Not Bad'})

print(L8_One)
```

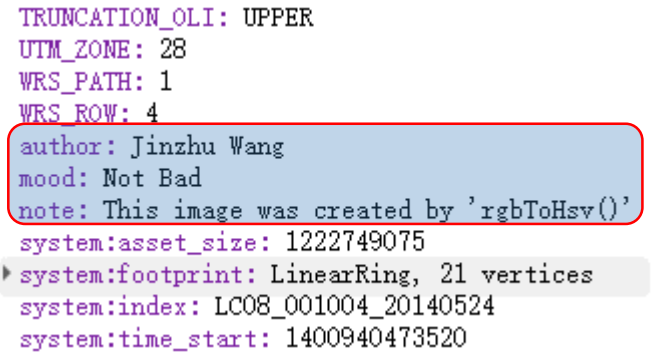

图 7.12 栅格的属性改写

下边介绍栅格的 remap 命令，代码及执行效果如下：

```
var Old_Id = ee.List([11,14,20,30,40,50,60,70,90,100,110,120,130,140,150,160,170,180,190,200,210,220,230])
var New_Id = ee.List([0, 0, 0, 0, 1, 1, 1, 1,  2,  2,  2,  3,  3,  3,  4,  4,  4,  5,  5,  6,  7,  8])
var Land_Cover = ee.Image("ESA/GLOBCOVER_L4_200901_200912_V2_3").select('landcover');
var Land_Remap = Land_Cover.remap(Old_Id,New_Id,8,'landcover')

Map.setCenter(102.919, 30.199,6)
Map.addLayer(Land_Cover)
Map.addLayer(Land_Remap,{"palette":["f1ff27","35ce48","07fff3","f017ff","e78845","250bff","ffffff","ffffff"]},
             'remaped')
```

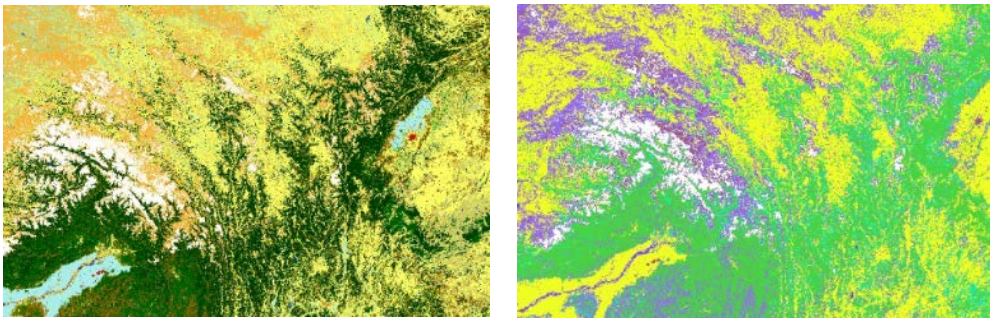

图 7.13 栅格的 remap

需要指出的是，上述 remap 命令相当于对栅格 value 值的“重分类”操作。通过 remap 操作，原始土地利用分类图像被重新归纳成了“耕、林、水、草、建等”9 中地类。具体的地类编码见下表。

表 7.1 GLOBCOVER_L4_2009 的 remap

旧代码	地类	新编码
11	Post-flooding or irrigated croplands	0
14	Rainfed croplands	0
20	Mosaic cropland (50-70%) / vegetation (grassland, shrubland, forest) (20-50%)	0
30	Mosaic vegetation (grassland, shrubland, forest) (50-70%) / cropland (20-50%)	0
40	Closed to open (>15%) broadleaved evergreen and/or semi-deciduous forest (>5m)	1
50	Closed (>40%) broadleaved deciduous forest (>5m)	1
60	Open (15-40%) broadleaved deciduous forest (>5m)	1
70	Closed (>40%) needleleaved evergreen forest (>5m)	1
90	Open (15-40%) needleleaved deciduous or evergreen forest (>5m)	1
100	Closed to open (>15%) mixed broadleaved and needleleaved forest (>5m)	2
110	Mosaic forest-shrubland (50-70%) / grassland (20-50%)	2
120	Mosaic grassland (50-70%) / forest-shrubland (20-50%)	2
130	Closed to open (>15%) shrubland (<5m)	3
140	Closed to open (>15%) grassland	3
150	Sparse (>15%) vegetation (woody vegetation, shrubs, grassland)	3
160	Closed (>40%) broadleaved forest regularly flooded - Fresh water	4
170	Closed (>40%) broadleaved semi-deciduous and/or evergreen forest regularly flooded - saline water	4
180	Closed to open (>15%) vegetation (grassland, shrubland, woody vegetation) on regularly flooded or waterlogged soil - fresh, brackish or saline water	4
190	Artificial surfaces and associated areas (urban areas >50%) GLOBCOVER 2009	5
200	Bare areas	5
210	Water bodies	6
220	Permanent snow and ice	7
230	Unclassified	8

下边介绍栅格的区间赋值，代码及执行效果如下：

```
var DEM = ee.Image("USGS/SRTMGL1_003");
var Land_Cover = ee.Image("ESA/GLOBCOVER_L4_200901_200912_V2_3").select('landcover');

var High_Land = Land_Cover.where(DEM.lt(4000),0)

Map.setCenter(88.96, 33.77,4)
Map.addLayer(DEM.lt(4000))
Map.addLayer(High_Land)
```

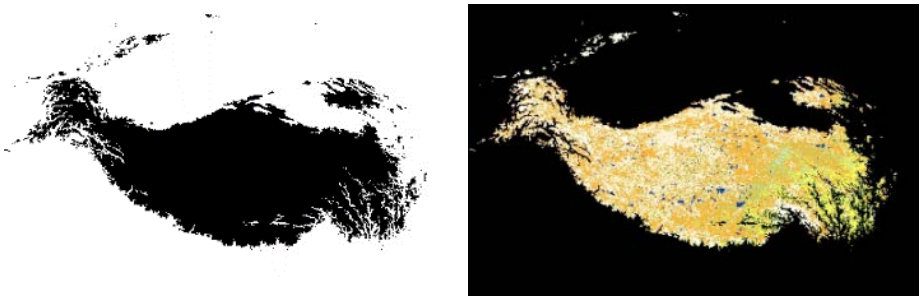

图 7.14 栅格的区间赋值

下边介绍栅格的区间截取，代码及执行效果如下：

```
var DEM = ee.Image("USGS/SRTMGL1_003");
var DEM_Clamp = DEM.clamp(450,795)

Map.setCenter(106.5662, 29.5589,10)
Map.addLayer(DEM_Clamp)
```

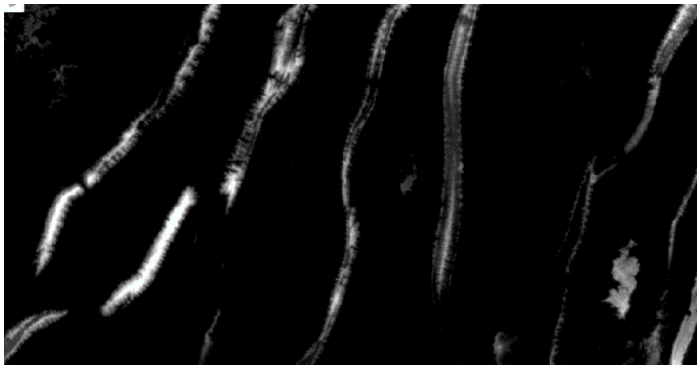

图 7.15 栅格的区间截取

这里需要注意区分区间赋值.where()和区间截取.clamp()的异同。.where(DEM.lt(4000),0)的含义就是将高程小于 4000 的地区赋值为 0；而.clapm()的功能是截取某一段区间的值，大于或小于这个区间的值都以最大或最小值处理，如 DEM.clamp(450,795)的作用就是将高程大于 795 米的地区赋值为 795，小于 450 米的地区赋值为 450.

下边介绍栅格的一值化，代码及执行效果如下：

```
var L8_One = ee.Image(ee.ImageCollection("LANDSAT/LC08/C01/T1")
                    .filterBounds(ee.Geometry.Point(106.5776, 29.5577)).first());
var L8_Unitscale = L8_One.unitScale(0,32767)

print(L8_One,L8_Unitscale)

Map.addLayer(L8_One,       {"bands":["B5","B4","B3"],"max":30000})
Map.addLayer(L8_Unitscale, {"bands":["B5","B4","B3"],"max":0.92})
```

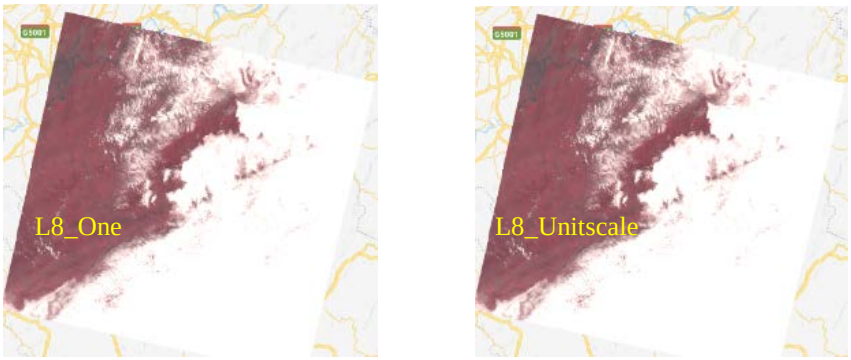

图 7.16 栅格的一值化

从本例中可以看出，一值化的功能在于将给定区间内的数值范围转换为 0-1 的区间内。当对原图像进行一值化后，新图像显示范围最大值 0.92 与原图像显示范围最大值 30000 是相同的，因为 30000/32767 = 0.92。

下边介绍栅格的插值，代码及执行效果如下：

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

function Add_Center (feature){
   return ee.Geometry(feature.centroid())
}

var Provinces_Centers = China_Provinces.map(Add_Center)
var Center_Distance = Provinces_Centers.distance(200000)
var Distance_Interpolate = Center_Distance.interpolate(
                                  [ 0, 50000, 100000, 150000, 200000 ],
                                  [ 0,      25,      50,      100,      75 ], '
                                  extrapolate'                );

Map.centerObject(Provinces_Centers,4)
Map.addLayer(Center_Distance,{min:0,max:200000},'origin')
Map.addLayer(Distance_Interpolate,{min:0,max:100},'interpolated')
Map.addLayer(Provinces_Centers, {color:'0000ff'},'centers')
```

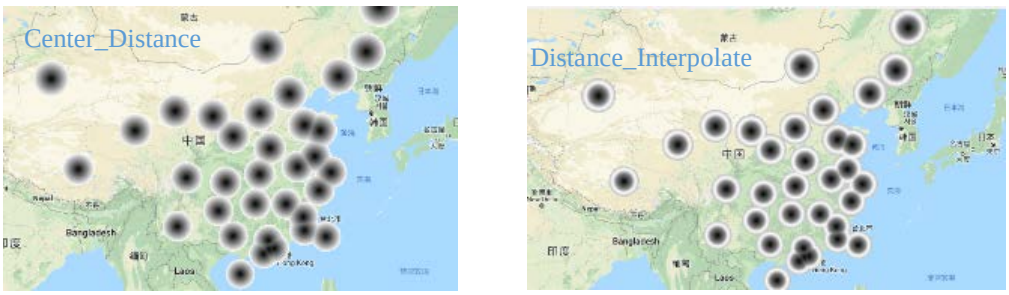

图 7.17 栅格的插值

这里注意插值方式为“extrapolate”，这种插值的效果是按照.interpolate()的第二个参数值来确定插值大小，相当于分阶段的线性运算。

下边介绍栅格的比较筛选，代码及执行效果如下：

```
var China = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");
var night_Lights = ee.Image('NOAA/DMSP-OLS/NIGHTTIME_LIGHTS/F182013')
                          .select('stable_lights').clip(China);

var urban = night_Lights.gte(20)


Map.setCenter(115.218, 33.697, 8);
Map.addLayer(urban, {"opacity":0.33,"bands":["stable_lights"],"palette":["171801","fbff0f"]},
'Nighttime Lights');
```

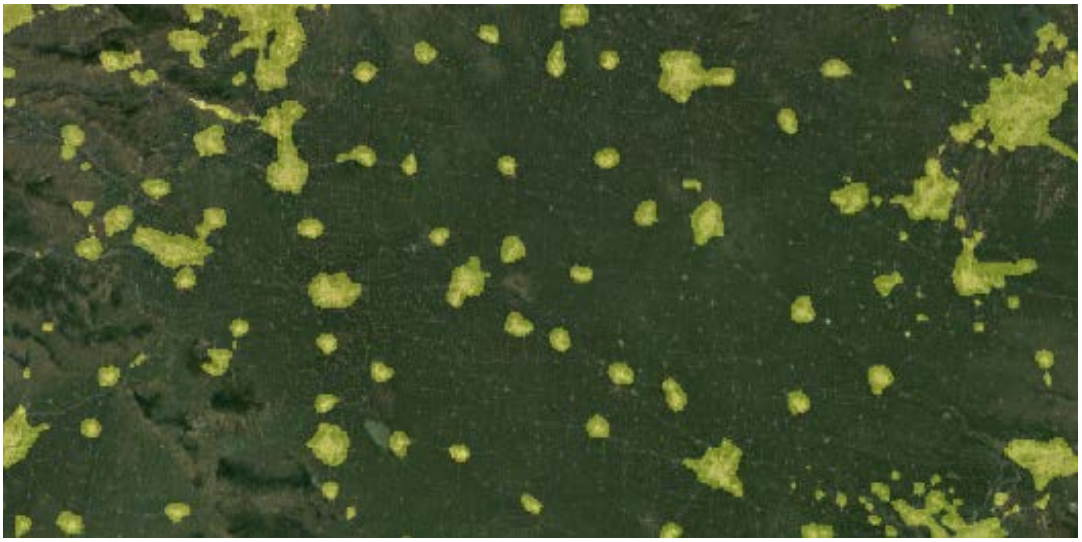

图 7.18 栅格的比较筛选

类似的比较筛选命令还有：

.eq	.neq	.gt	.gte	.lt	.lte
=	≠	>	≥	<	≤

下边介绍栅格的逻辑运算，代码及执行效果如下：

```
var China = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");
var night_Lights = ee.Image('NOAA/DMSP-OLS/NIGHTTIME_LIGHTS/F182013')
                                .select('stable_lights').clip(China);
var urban = night_Lights.gte(20)
var Low_Area = ee.Image("USGS/SRTMGL1_003").lt(200).clip(China)
var Low_Urban = urban.and(Low_Area)
print(Low_Area,urban)

Map.setCenter(115.218, 33.697, 4);
Map.addLayer(night_Lights)
Map.addLayer(Low_Area)
Map.addLayer(Low_Urban);
```

图 7.19 栅格的逻辑运算

下边介绍栅格的函数运算，代码及执行效果如下：

```
var China_Provinces = ee.FeatureCollection("users/wangjinzhulala/China_Provinces");
var CQ = China_Provinces.filterMetadata('NAME','equals','Chong_Qing')
var CQ_Distance = CQ.distance(50000)
var Distance_Mod = CQ_Distance.mod(10000)

Map.centerObject(CQ,6)
Map.addLayer(CQ_Distance,{"min":0,"max":50000,
                          "palette":["0172ff","0dff09","05ffe8","fff707","ffa605","ff5909"]},
                          'The Distance Effect')
Map.addLayer(Distance_Mod,{"min":0,"max":10000,
                          "palette":["0172ff","0dff09","05ffe8","fff707","ffa605","ff5909"]},
                          'The Distance.mod() Effect')
```

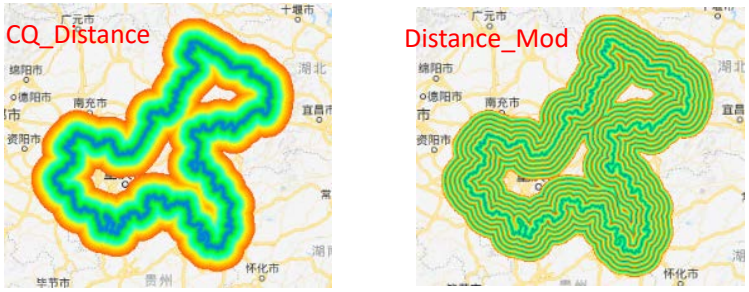

图 7.20 栅格的函数运算

下边介绍栅格的数学运算，代码及执行效果如下：

```
var L8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT")
                .filterBounds(ee.Geometry.Point(106.5749, 29.5588))
                .first()
var B5 = ee.Image(L8.select('B5'))
var B4 = ee.Image(L8.select('B4'))
var B3 = ee.Image(L8.select('B3'))

var NDVI = B5.subtract(B4).divide(B5.add(B3))

Map.centerObject(NDVI)
Map.addLayer(NDVI,
                {"min":-1,"max":1,"palette":["0f01e2","ff9d93","fff707","11ff2e","04ce14"]})
```

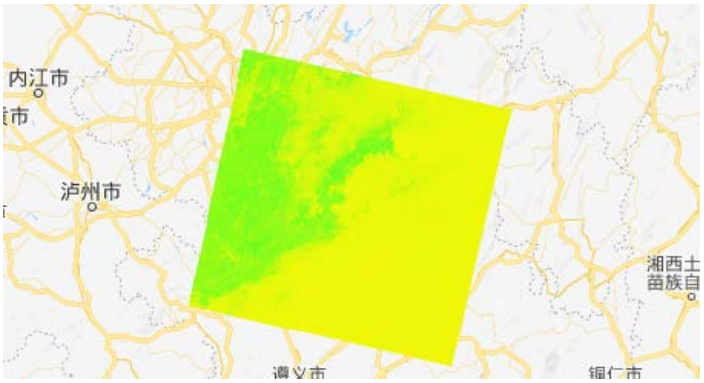

图 7.21 栅格的数学运算

其它类似的数学运算命令如下，运算的具体操作可参考前述内容。

.add	.subtract	multiply	.divide	.max	.min			
.mod	.pow	.hypot	.first	.first_nonzero				
.sin	.cos	.tan	. sinh	.cosh	.tanh	.acos	.asin	.atan ()

下边介绍栅格的表达式运算，代码及执行效果如下：

```
var L8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT")
                .filterBounds(ee.Geometry.Point(106.5749, 29.5588))
                .first()
var evi = L8.expression(
    '2.5 * ((NIR - RED) / (NIR + 6 * RED - 7.5 * BLUE + 1))', {
      'NIR': L8.select('B5'),
      'RED': L8.select('B4'),
      'BLUE': L8.select('B2')
});

Map.centerObject(evi)
Map.addLayer(evi,
                {"min":-1,"max":1,"palette":["0f01e2","ff9d93","fff707","11ff2e","04ce14"]})
```

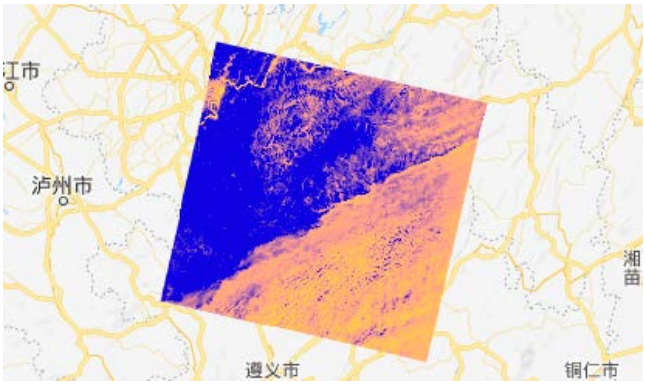

图 7.22 栅格的表达式运算

表达式运算命令有两个参数，第一个参数以文本形式描绘出运算规则，第二个参数以字典形式制定运算规则中变量制定的具体波段。

下边介绍栅格的位运算，代码及执行效果如下：

```
var Image_1 = ee.Image( 1 );
var Image_2 = ee.Image( 2 );
var Bit_And = Image_1.bitwiseAnd( Image_2 );
var Bit_Or    = Image_1.bitwiseOr( Image_2 );
print ( 'Image of bitwise 1 and 2 =', Bit_And      );
print ( 'Image of bitwise 1 or 2 =',    Bit_Or );

Map.addLayer( Image_1, null, 'Image of 1s' );
Map.addLayer( Image_2, null, 'Image of 2s' );
Map.addLayer( Bit_And, null, 'Image of Bits from Both' );
Map.addLayer( Bit_Or , null, 'Image of Bits from Either' );
```

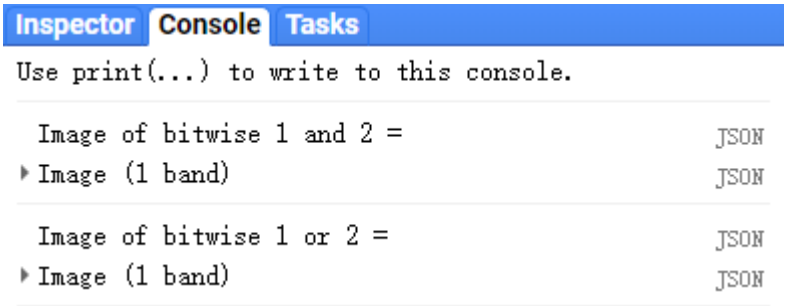

图 7.23 栅格的位运算

栅格图像的位运算在本质上与数字的位运算是相同的，即都是将十进制数字转换成二进制数字后，通过对相同位置的二进制数字进行逻辑对比得到最终结果。位运算在栅格中的主要作用是用作去云处理，其原理是(主要针对 Landsat 系列数据)遥感图像每个像素都存在类似“头文件”的十几个字节的数据，其中有若干字节的数据可以表示这个像素是否被云层覆盖(0 代表无云，1 代表有云)，对部分数据进行位运算筛选即可得到去云目的。

下属是其它类似的位运算命令，具体操作可参考前述数字的位运算介绍。

bitwiseAnd	bitwiseOr	bitwiseXor	bitwiseNot	leftShift	rightShift
bitwise_and	bitwise_or	bitwise_xor	bitwise_notl	eft_shift	right_shift

下边介绍栅格的微分操作，代码及执行效果如下：

```
var OldIMAGE = ee.Image( 'CGIAR/SRTM90_V4' );
var NewIMAGE = OldIMAGE.derivative( );

Map.setCenter(106.4448, 29.5684,10)
Map.addLayer( NewIMAGE, {bands:['elevation_x'], min:-90, max:90 }, 'Slope from Left to Right' );
Map.addLayer( NewIMAGE, {bands:['elevation_y'], min:-90, max:90 }, 'Slope from Top to Bottom' );
```

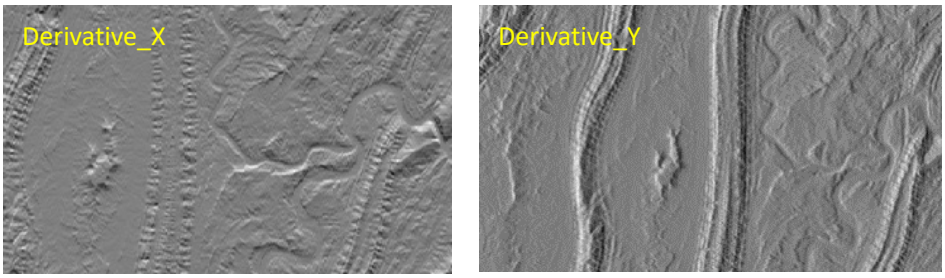

图 7.24 栅格的微分运算

该命令的功能是计算图像像素值在 X(横)和 Y(纵)方向上的数值变化率。将两者进行 atan()运算，并且求取运算后两值平方和的平方根就得到了坡度。

下边介绍栅格的地形操作，代码及执行效果如下：

```
var DEM = ee.Image( 'CGIAR/SRTM90_V4' );
var Terrain = ee.Terrain.products( DEM );

Map.setCenter( 106.371, 29.6188 );
Map.addLayer( DEM,      { min:0,max:2000,palette:'000099,dddd00'}, 'Elevation');
Map.addLayer( Terrain,{bands:['slope'], min:0,max:90,palette:'000000,ff0000' }, 'Slope');
Map.addLayer( Terrain,{bands:['aspect'], palette:'00ff00,0000ff,00ff00' },'Aspect');
Map.addLayer( Terrain,{bands:['hillshade'],min:0,max:255,},   'Hillshading');
```

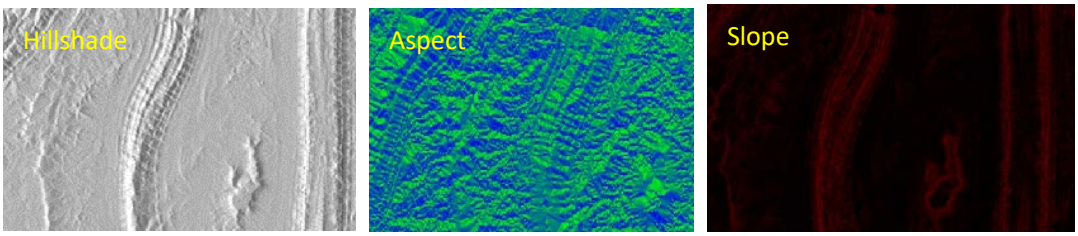

图 7.25 栅格的地形运算

下边介绍栅格的山体阴影操作，代码及执行效果如下：

```
var DEM    = ee.Image( 'CGIAR/SRTM90_V4' );
var DEM_Mercator = DEM.reproject( 'EPSG:3857', null, 30 );
var Shadow_1 = ee.Terrain.hillShadow( DEM_Mercator, 315.0, 75.0,0,true);
var Shadow_2 = ee.Terrain.hillShadow( DEM_Mercator, 315.0, 70.0,0,false);
Map.setCenter( 106.5568, 29.5527, 12 );

Map.addLayer( DEM, {min:0, max:1500, opacity:0.7, palette:'0000ff,ff0000'}, 'Elevation' );
Map.addLayer( Shadow_1, {min:0, max:1, opacity:0.3}, 'Longer Shadows' );
Map.addLayer( Shadow_2, {min:0, max:1, opacity:0.3}, 'Shorter Shadows' );
```

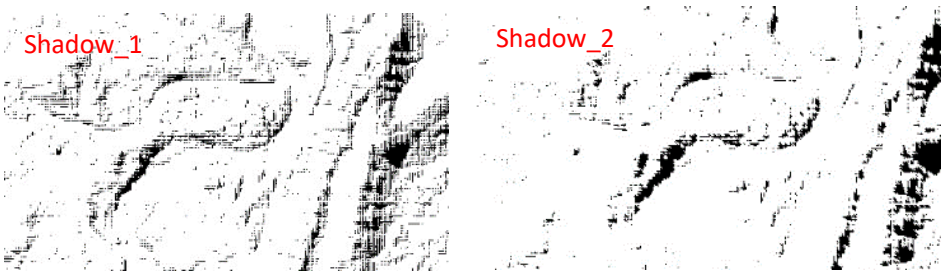

图 7.26 栅格的山体阴影运算

虽然上个例子也能求取山体阴影 Hillshade，但 ee.Terrain.hillShadow 能更加个性化的调整阴影状态，该命令一共有 5 个参数，分别是“图像”，“天顶角”，“太阳高度角”，“邻域大小”和“是否磁滞”。其中“是否磁滞”如果输入为 true，会降低阴影的准确性，但是会提高阴影的显示效果。

下边介绍栅格的填洼操作，代码及执行效果如下：

```
var DEM    = ee.Image( 'CGIAR/SRTM90_V4' );
var DEM_Fill    = ee.Terrain.fillMinima( DEM, 0, 50 );

Map.setCenter(107.8729, 29.2441, 12 );
Map.addLayer( DEM, {min:0, max:1500, palette:["2f5bff","0dff09","efff05","ff9931"]}, 'Unfilled' );
Map.addLayer( DEM_Fill, {min:0, max:1500, palette:["2f5bff","0dff09","efff05","ff9931"]}, 'Filled');
```

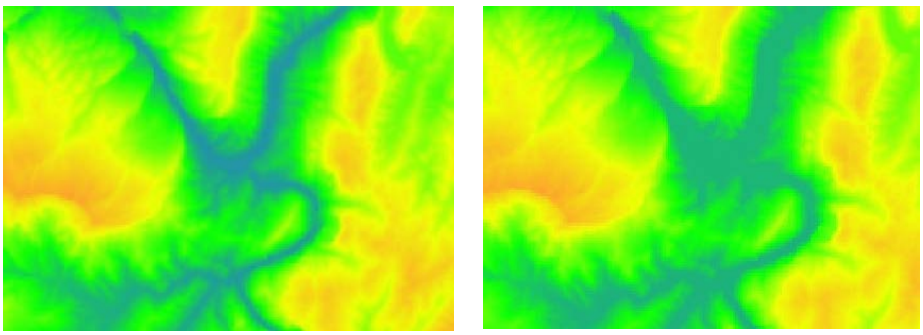

图 7.27 栅格的填洼运算

下边介绍栅格的熵值操作，代码及执行效果如下：

```
var DEM    = ee.Image( 'CGIAR/SRTM90_V4' );
var DEM_Entropy    = DEM.entropy(    ee.Kernel.circle( 5, 'pixels', true ) );
Map.setCenter( 106.1201, 29.4977, 10 );
Map.addLayer( DEM,       {min:0, max:1500,
palette:["050dff","ddd3cd","0bff34","e4ff09","ff9007","ff1c05"]}, 'Original' );
Map.addLayer( DEM_Entropy,       {min:3.31,max:4.55,
palette:["050dff","ddd3cd","0bff34","e4ff09","ff9007","ff1c05"]}, 'Entropy'    );
```

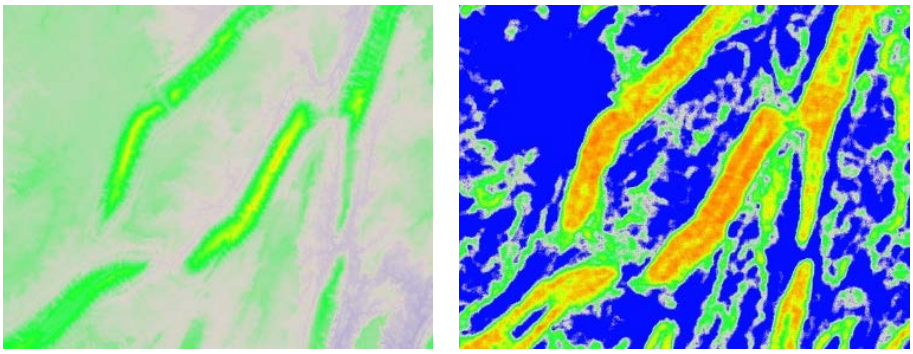

图 7.28 栅格的熵值操作

下边介绍栅格的 glcm 纹理操作，代码及执行效果如下：

```
var DEM    = ee.Image( 'CGIAR/SRTM90_V4' );
var GLCM = DEM.glcmTexture(5, ee.Kernel.circle( 5, 'pixels', true ) );
var Texture = GLCM.select( 0 );
print( GLCM, Texture );

Map.setCenter( 106.1201, 29.4977, 10 );
Map.addLayer( DEM,       {min:0, max:1500,
palette:["050dff","ddd3cd","0bff34","e4ff09","ff9007","ff1c05"]}, 'Original' );
Map.addLayer( Texture,       {min:0.0067,max:0.0115,
palette:["050dff","ddd3cd","0bff34","e4ff09","ff9007","ff1c05"]}, 'Texture'    );
```

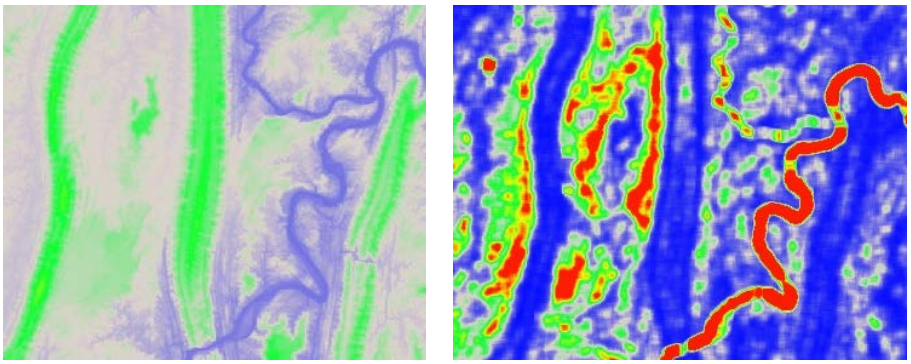

图 7.29 栅格的 glcm 纹理求取

下边介绍栅格的 zeroCrossing 操作，代码及执行效果如下：

```
var DEM = ee.Image( 'CGIAR/SRTM90_V4' ).subtract(465);

var Zero_Crossing = DEM.zeroCrossing( );

Map.setCenter( 106.1201, 29.4977, 7 );
Map.addLayer( DEM, { min:0, max:1500,
palette:["050dff","ddd3cd","0bff34","e4ff09","ff9007","ff1c05"]}, 'DEM'        );
Map.addLayer( Zero_Crossing, { palette:'ffffff,ff0000'}, 'Zero Crossing' );
```

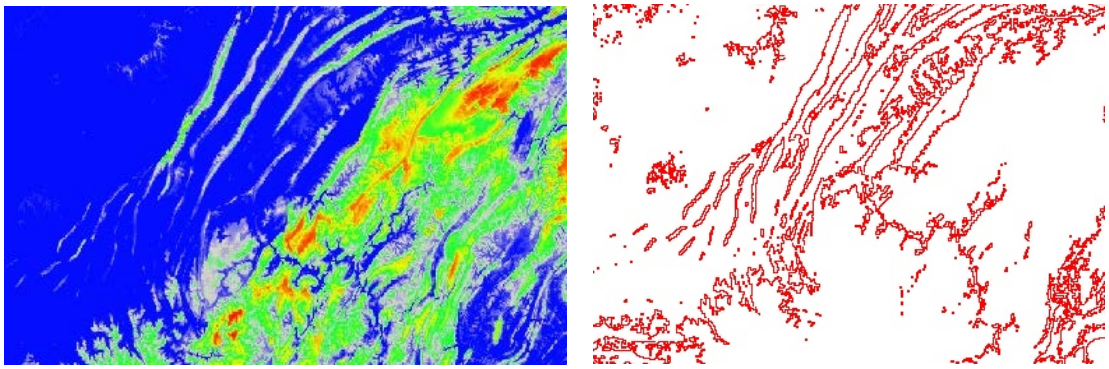

图 7.30 栅格的 zeroCrossing 纹理求取

下边介绍栅格的 CannyEdge 纹理操作，代码及执行效果如下：

```
var L8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_RT")
                .filterBounds(ee.Geometry.Point(106.5749, 29.5588))
                .first().select('B7')

var L8_Canny= ee.Algorithms.CannyEdgeDetector( L8, 4000, 0 );

Map.setCenter( 106.65616, 29.72455, 13);
Map.addLayer( L8, { min:0, max: 15000, gamma:0.7 },              'Original' );
Map.addLayer( L8_Canny,     { opacity:0.2, palette:'ffffff,0000ff' }, 'Canny' );
```

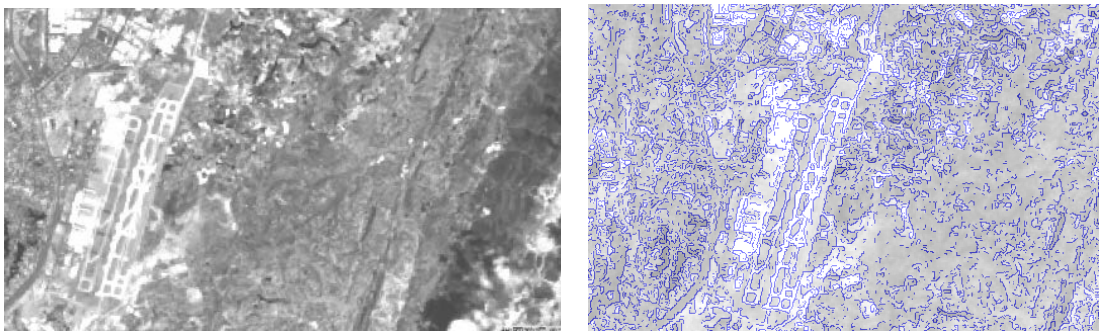

图 7.31 栅格的 CannyEdge 纹理求取

下边介绍栅格的距离操作，代码及执行效果如下：

```
var DEM = ee.Image("USGS/SRTMGL1_003");
var Land_Cover = ee.Image("ESA/GLOBCOVER_L4_200901_200912_V2_3").select('landcover');

var Zero_One = Land_Cover.where(DEM.gt(500),0)
var The_Distance = Zero_One.distance(ee.Kernel.euclidean( 1000, 'meters' ))

Map.setCenter(107.7167, 29.3865,10)
Map.addLayer(Zero_One,{max:1})
Map.addLayer(The_Distance,{max:1500})
```

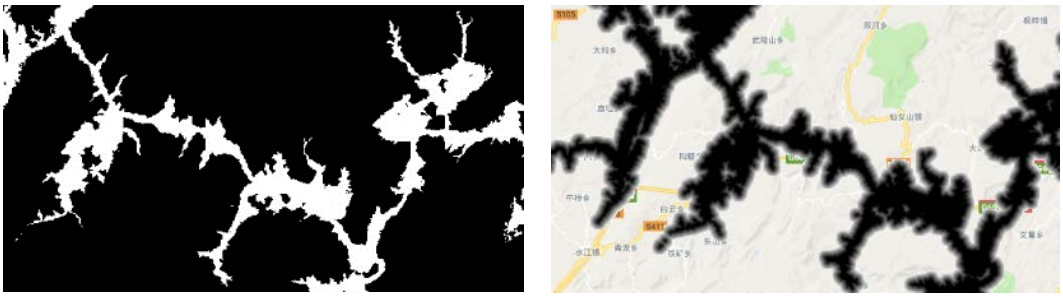

图 7.32 栅格的距离求取

这里需要注意，栅格的距离命令的功能是求取图上任一点距离最近的非零点的距离。

下边介绍栅格的焦点操作，代码及执行效果如下：

```
var OldIMAGE = ee.Image('MCD12Q1/MCD12Q1_005_2001_01_01').select('Land_Cover_Type_1');
var MaxIMAGE = OldIMAGE.focal_max(      5, 'circle', 'pixels' );
var MinIMAGE = OldIMAGE.focal_min(      5, 'circle', 'pixels' );
var MedianIMAGE = OldIMAGE.focal_median( 5, 'circle', 'pixels' );
var ModeIMAGE   = OldIMAGE.focal_mode(     5, 'circle', 'pixels' );

Map.setCenter( 106.0734, 29.4929, 10 );
Map.addLayer( OldIMAGE,      {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Original '        );
Map.addLayer( MaxIMAGE,      {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Focal Maximum' );
Map.addLayer( MinIMAGE,      {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Focal Minimum' );
Map.addLayer( MedianIMAGE, {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Focal Median'   );
Map.addLayer( ModeIMAGE,     {min:0, max:17, palette:'000000,0000dd,ffffff'}, 'Focal Mode'      );
```

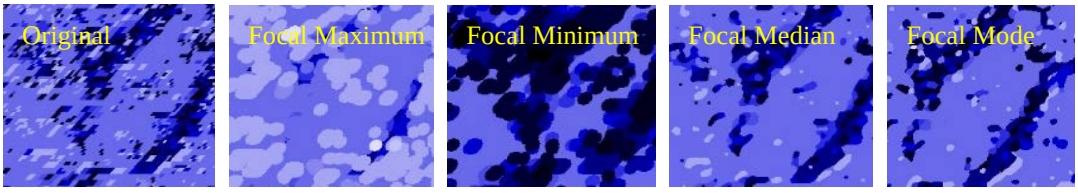

图 7.33 栅格的焦点操作

下边介绍栅格的卷积操作，代码及执行效果如下：

```
var DEM = ee.Image("USGS/SRTMGL1_003");
var KERNEL = ee.Kernel.fixed( 7,7,      [[ 4, 0, 0, 0, 0, 0, 1 ],
                                         [ 0, 0, 0, 0, 0, 0, 0 ],
                                         [ 0, 0, 0, 0, 0, 0, 0 ],
                                         [ 0, 0, 0, 0, 0, 0, 0 ],
                                         [ 0, 0, 0, 0, 0, 0, 0 ],
                                         [ 0, 0, 0, 0, 0, 0, 0 ],
                                         [ 1, 0, 0, 0, 0, 0, 3 ]] );
var Convelve_DEM    = DEM.convolve( KERNEL );

Map.setCenter(105.8125, 29.4033,9)
Map.addLayer(DEM,{min:0,max:2000},'DEM')
Map.addLayer(Convelve_DEM,{max:12000},'Convelve_DEM')
```

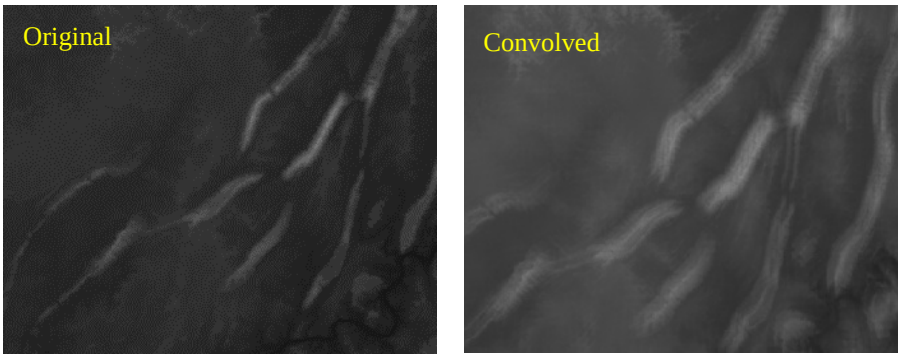

图 7.34 栅格的卷积操作

下边介绍栅格的邻域缩减操作，代码及执行效果如下：

```
var DEM = ee.Image("USGS/SRTMGL1_003");
var DEM_ReduceNeighbor = DEM.reduceNeighborhood( ee.Reducer.mean( ), ee.Kernel.circle( 4 ) );

Map.setCenter( 105.9479, 29.4165, 8 );
Map.addLayer(DEM, {min: 0, max: 2000, }, 'DEM');
Map.addLayer(DEM_ReduceNeighbor, {min: 0, max: 2000, }, 'DEM_ReduceNeighbor')
```

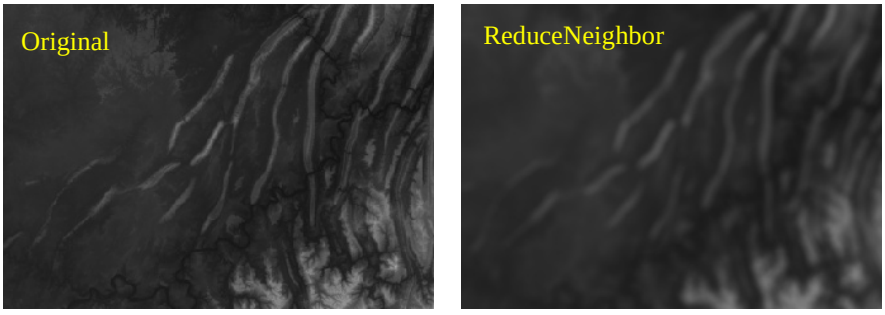

图 7.35 栅格的邻域缩减操作

下边介绍栅格的转矢量操作，代码及执行效果如下：

```
var Water = ee.Image("ESA/GLOBCOVER_L4_200901_200912_V2_3")
                    .select('landcover')
                    .clamp(209,211);
var geometry = ee.Geometry.Polygon(
          [[[106.32728668689242, 29.46329064748005],
            [106.76948639392367, 29.462692803378452],
            [106.77017303943148, 29.65740304071832],
            [106.31355377673617, 29.654419523039962]]]);

var Water_Vecter        = Water.reduceToVectors( null, geometry,   null, 'polygon',   false);
var Water_BoundBox      = Water.reduceToVectors( null, geometry,   null, 'bb',      false);
var Water_Center_4      = Water.reduceToVectors( null, geometry,   null, 'centroid', false);
var Water_Center_8      = Water.reduceToVectors( null, geometry,   null, 'centroid', true );

Map.setCenter(106.5587, 29.5631,10)
Map.addLayer(Water,{"palette":["ffffff","1707ff"]},'Water')
Map.addLayer( Water_Vecter,     {color:'dddddd'}, 'The Polygons' );
Map.addLayer( Water_BoundBox, {color:'888888'}, 'The Bounding Boxes' );
Map.addLayer( Water_Center_4, {color:'ff0000'}, '4-way Centroids' );
Map.addLayer( Water_Center_8, {color:'ffffff'}, '8-way Centroids' );
```

图 7.36 栅格转矢量

栅格转矢量共有 12 个参数，这里着重介绍第 4 个“矢量类型”和第五个“是否计算斜边”。矢量类型共有三种：“polygon”，“bb”和“centroid”，分别代表不规则面、矩形和点。是否计算斜边的功能是告诉 GEE 栅格转矢量的时候是否要把斜向相接(除了上下左右之外的其他四个方向)的栅格算作矢量的一部分。

下边介绍栅格的转一维矩阵操作，代码及执行效果如下：

```
var L8 = ee.Image('LANDSAT/LC08/C01/T1_RT/LC08_127040_20130422').select('B[1-7]')
var L8_Array = L8.toArray(0)
var L8_Flatten = L8_Array.arrayFlatten([['a','b','c','d','e','f','g']])

Map.setCenter(106.4949, 29.458,10)
Map.addLayer(L8_Array,{},'Array Image')
Map.addLayer(L8_Flatten,{"bands":["e","d","c"],"max":32000,"gamma":0.73})
```

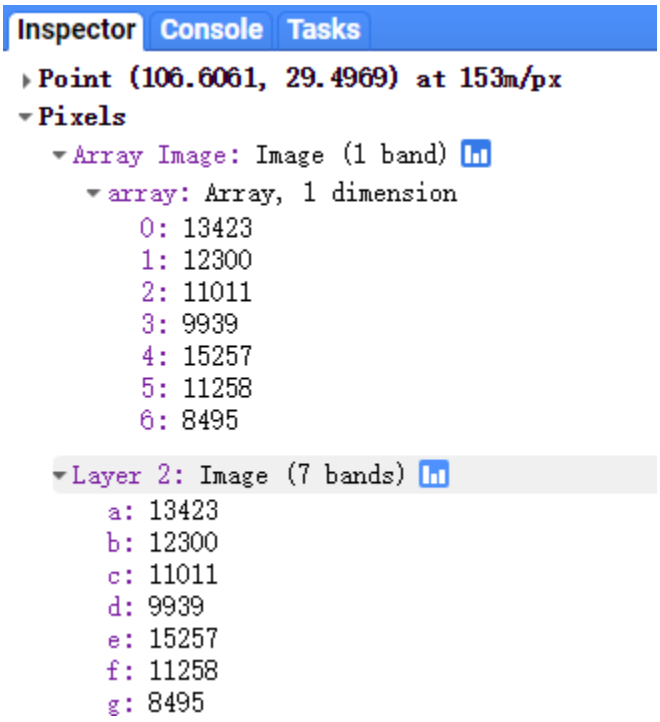

图 7.37 栅格的转一维矩阵操作

下边介绍栅格的区域统计操作，代码及执行效果如下：

```
var DEM = ee.Image("USGS/SRTMGL1_003");
var ROI = ee.Geometry.Polygon( [[[106.20650888671878, 29.570337903152346],
                                 [106.54159189453128, 29.56556010779314],
                                 [106.54159189453128, 29.725493070064537],
                                 [106.21612192382815, 29.71952998792036]]]);
var ROI_Mean = DEM.reduceRegion(ee.Reducer.mean(),ROI)
var ROI_Histo = DEM.reduceRegion(ee.Reducer.histogram(),ROI)

print(ROI_Mean,ROI_Histo)
Map.setCenter(106.4949, 29.458,10)
Map.addLayer(DEM,{min:0,max:2000},'DEM')
Map.addLayer(ROI)
```

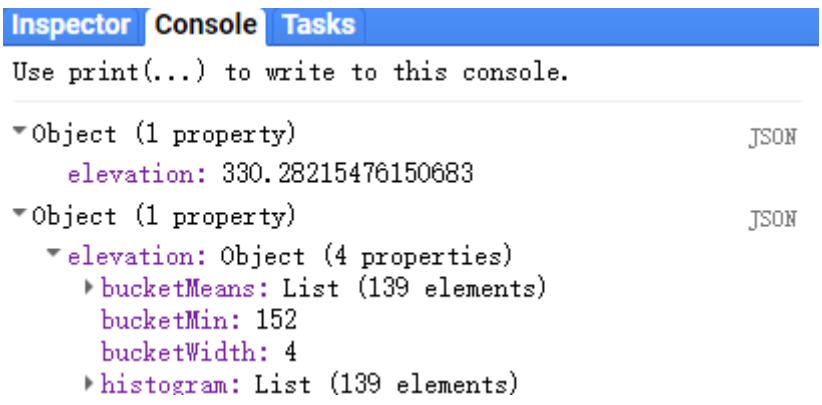

图 7.38 栅格的区域统计操作

下边介绍栅格的输出操作，代码及执行效果如下：

```
var DEM = ee.Image("USGS/SRTMGL1_003");
var ROI = ee.Geometry.Polygon(
            [[[106.20650888671878, 29.570337903152346],
              [106.54159189453128, 29.56556010779314],
              [106.54159189453128, 29.725493070064537],
              [106.21612192382815, 29.71952998792036]]]);
var DEM_Clip = DEM.clip(ROI)

Export.image.toDrive({
  image:DEM_Clip,
  region:ROI
  })
```

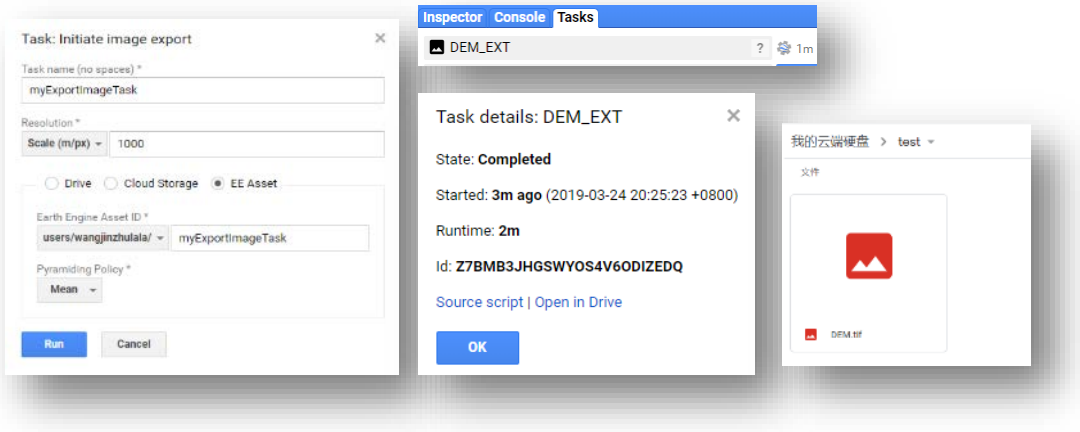

图 7.39 栅格的输出操作

这里需要注意，输出命令中建议指定的两个参数是 image 和 region，如果不指定 region，GEE 会默认的将目前视野中的图像输出出去，造成输出数据的缺失。

下边是本节介绍的有关 Image 的常用命令，尝试回忆其语法与功能。

ee.image()	ee.constant()	.mask()	.clip()	.select()	.slice()	addBands()
.reproject()	.rbgtohsv()	.hsvtorbg()	.unit8()	.cast()	.set()	.setMulti()
.remap()	.where()	.metadata()	.clamp()	.unitScale()	.interpolate()	.eq()
.abs()	.sin()	.bitWiseAnd()	.reduce()	.derivative()	ee.Terrain.products()	
ee.Algorithm.Terrain()		ee.Terrain.slope()		ee.Terrain.aspect()	ee.Terrain.fillMinima()	
ee.Terrain.hillshade()		ee.Terrain.hillshadow()		ee.Algorithm.Hillshadow()	.entropy()	
.Texture()	.zeroCrossing()	.focal_max()		.focal_min()	.focal_median()	
.focal_mode()	.convolve()	.reduceNeighborhood()		.ToVector()		
.ToArray()	.arrayFlatten()	.CrossCorrelation()		.distance()	Export.image()	
ee.Algorithms.CannyEdgeDetector()		ee.Algorithms.HoughTransform()				

7.2 Image Collection

GEE 作为一个遥感大数据平台，栅格图像是其核心数据。在 GEE 的代码中，Image Collection 与 Feature Collection 的命令有很多是重复的，因此可以将两者结合起来学习。另外，针对特定的 Landsat 系列数据，GEE 也给出了相应的命令以提高处理效率。最后 Confusion Marix 常用于遥感分类图像的验证。

下边介绍 Image Collection 的创建，代码及执行效果如下：

```
var Night_Light = ee.ImageCollection( 'NOAA/DMSP-OLS/NIGHTTIME_LIGHTS' );
var image_1 = ee.Image(1)
var image_2 = ee.Image(2)
var collection = ee.ImageCollection([image_1,image_2])
print(Night_Light,collection)

Map.setCenter( 107.54, 34.37, 4 );
Map.addLayer( Night_Light, {min:0, max:100, opacity:0.7} );
```

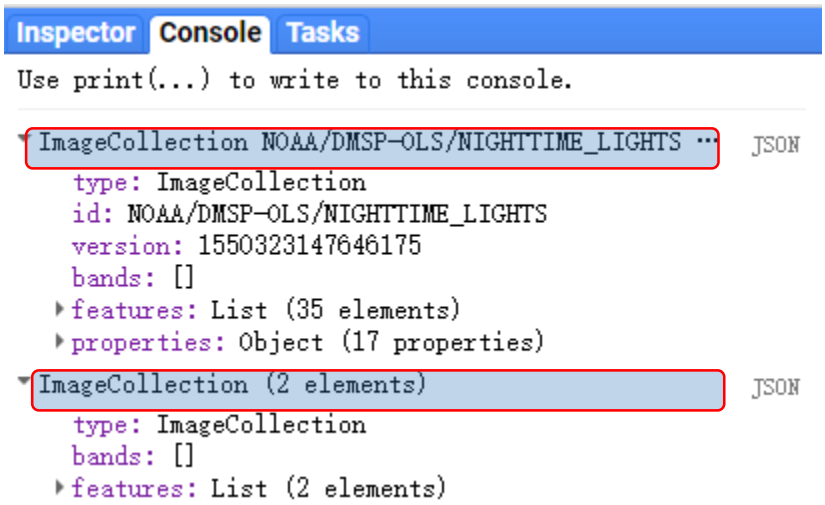

图 7.40 Image Collection 的创建

下边介绍 Image Collection 的筛选，代码及执行效果如下：

```
var China = ee.FeatureCollection("users/wangjinzhulala/China_Provinces")
var L8    = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                .filterBounds(China.geometry())
                .filterDate('2018-01-01','2018-12-31')
                .filterMetadata('CLOUD_COVER','less_than',0.1);

Map.setCenter( 107.753, 34.238, 4);
Map.addLayer( L8, {bands:'B4,B3,B2', min:0, max:0.2}, 'Original Images' );
```

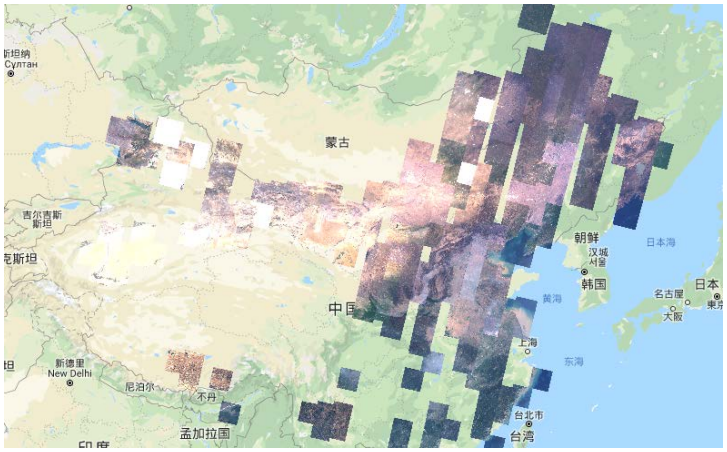

图 7.41 Image Collection 的筛选

应该指出的是，此处的筛选是“正常筛选”的简化模式，正常筛选参考下边的代码：

.filterBounds(China.geometry())	=	ee.Filter.Bounds(China.geometry())
.filterDate('2018-01-01','2018-12-31')	=	ee.Filter.Date('2018-01-01','2018-12-31')
.filterMetadata (‘CLOUD COVER’,‘less than’,0.1)	=	ee.Filter.Metadata(‘CLOUD COVER’,‘less than’,0.1)

下边介绍 Image Collection 的限制筛选，代码及执行效果如下：

```
var L8    = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                .filterBounds(ee.Geometry.Point(106.5487, 29.5444))
                .filterDate('2018-03-01','2019-03-01');

var L8_Limit    = L8.limit( 10 );
print( L8, L8_Limit );
```

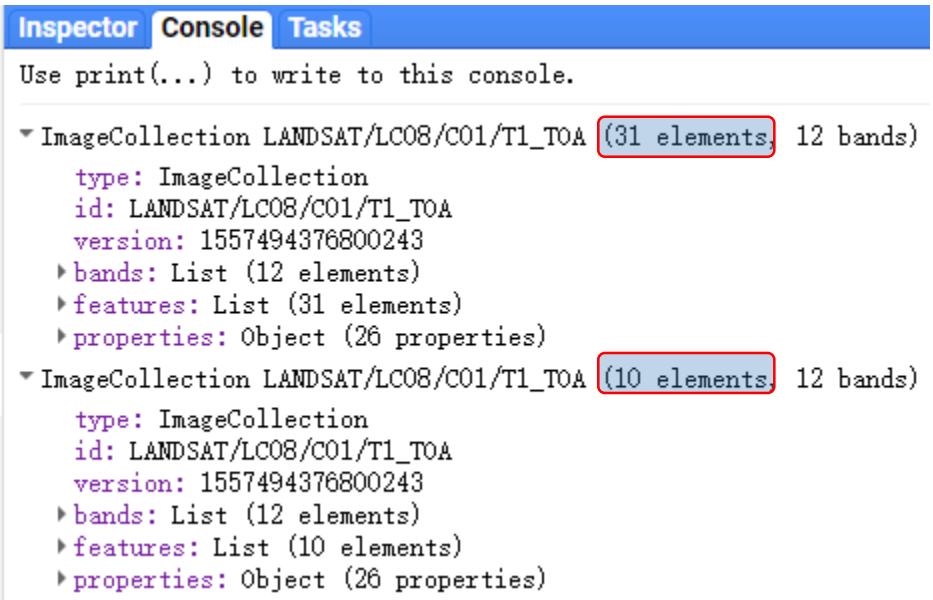

图 7.42 Image Collection 的限制筛选

下边介绍 Image Collection 的选择创建命令，代码及执行效果如下：

```
var L8    = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                .filterBounds(ee.Geometry.Point(106.5487, 29.5444))
                .filterDate('2018-03-01','2019-03-01')
                .limit(5);
var L8_Select = L8.select(['B5','B4','B3'], ['Near infrared','Red','Green']);

print( 'Original', L8 );
print( 'Selected', L8_Select );
```

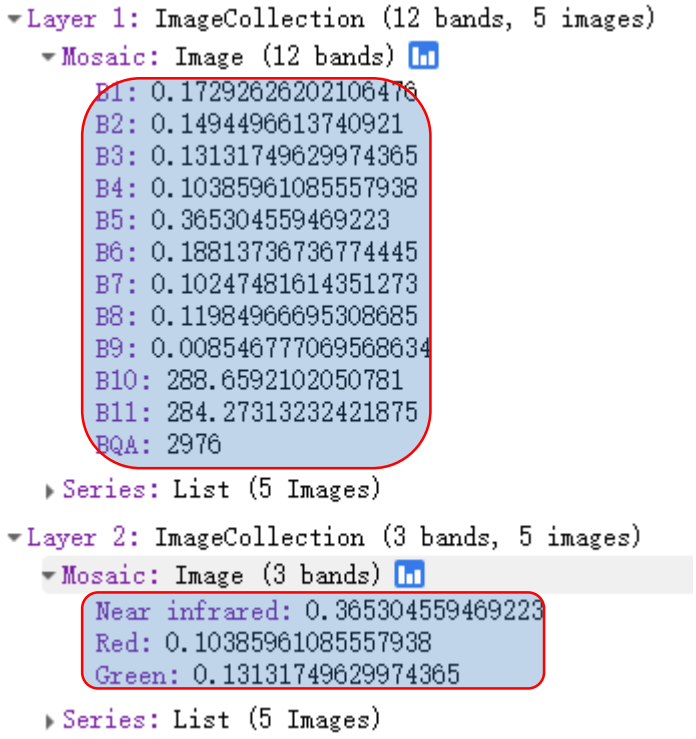

图 7.43 Image Collection 的选择创建

值得注意的是，选择创建命令.select()的功能是创建基于源数据的“副本”，同时，通过指定第二个参数值，还可以对副本数据属性进行重命名。

下边介绍 Image Collection 的选合并创建命令，代码及执行效果如下：

```
var L8    = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                .filterBounds(ee.Geometry.Point(106.5487, 29.5444))
                .filterDate('2018-03-01','2019-03-01')
                .limit(1);
var Combine_1 = L8.select(['B1'],['Coastal aerosol'])
var Combine_2 = L8.select(['B2'],['Blue'])
var Combine     = Combine_1.combine(Combine_2)

print(Combine_1,Combine_2,Combine)
```

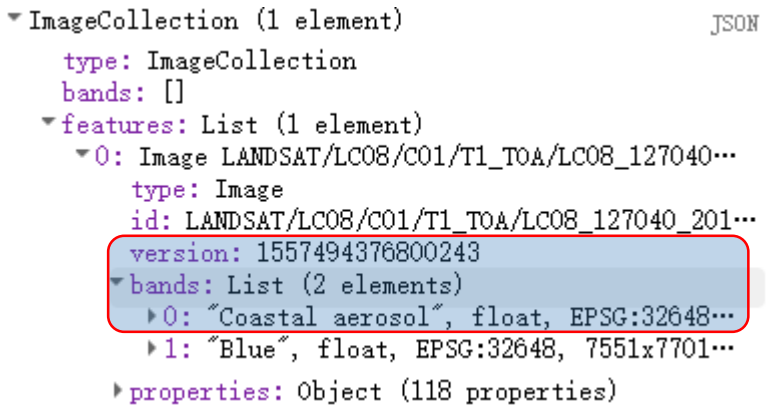

图 7.44 Image Collection 的合并创建

下边介绍 Image Collection 的数字格式转换，命令方式如下，具体操作可以参考前述例子。

.uint8	.uint16	.uint32	.uint64	.int8	.int16	.int32	.int64	.long	.float	.double
.toUnit8	toUint16	toUint32	toUint64	toInt8	toInt16	toInt32	toInt64	toLong	toFloat	toDouble

下边介绍 Image Collection 的属性改写命令，代码及执行效果如下：

```
var L8_1   = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_127041_20180303');
var L8_2   = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_127040_20180303');
var L8_3   = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_127039_20180303');
var L8_Collection = ee.ImageCollection([L8_1,L8_2,L8_2])

var Set_Property = L8_Collection.set('Creater','Jinzhu Wang')
                                .setMulti({'Creat time':'2019',
                                           His Cat' : '烧白' })

print(Set_Property)
```

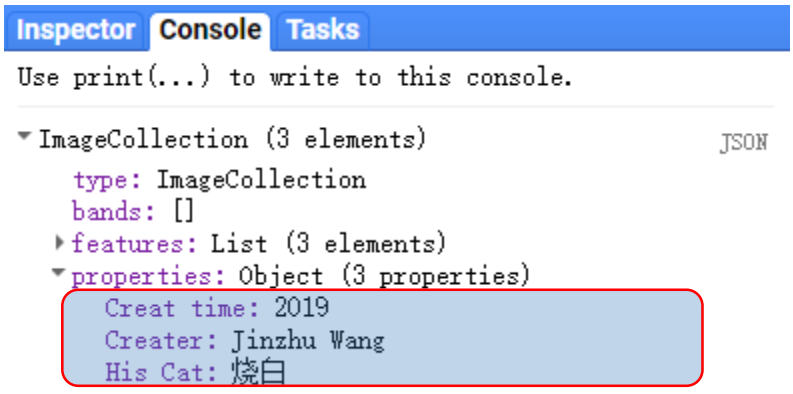

图 7.45 Image Collection 的属性改写

下边介绍 Image Collection 的拼接命令，代码及执行效果如下：

```
var L8_1 = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_127040_20180303').select('B5');
var L8_2 = ee.Image('LANDSAT/LC08/C01/T1_TOA/LC08_128039_20130413').select('B5');
var L8   = ee.ImageCollection([L8_1,L8_2]);

var L8_Mosaci = L8.mosaic()

print(L8_1,L8_2,L8_Mosaci)
Map.addLayer(L8_1)
Map.addLayer(L8_2)
Map.addLayer(L8_Mosaci)
```

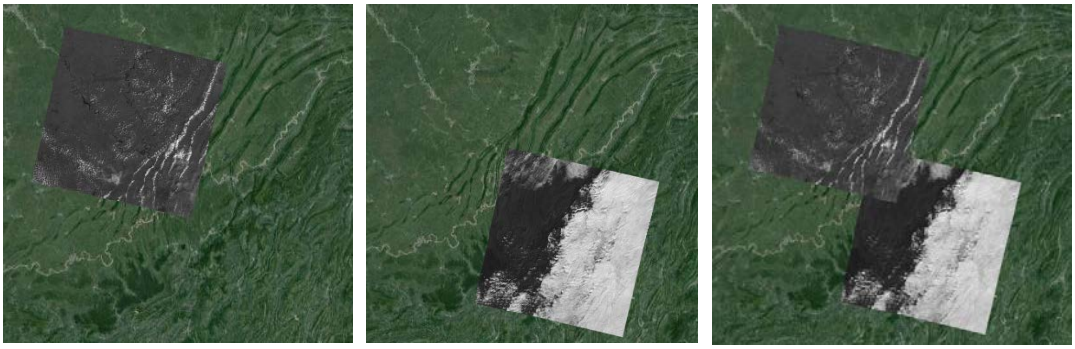

图 7.46 Image Collection 的拼接

下边介绍 Image Collection 的与或操作，代码及执行效果如下：

```
var Night_Light= ee.ImageCollection('NOAA/DMSP-OLS/NIGHTTIME_LIGHTS')
                  .select('stable_lights');
var Light_Or     = Night_Light.or();
var Light_And    = Night_Light.and();

Map.setCenter(105.13, 35.68, 4);
Map.addLayer( Light_Or,   {min: 0, max: 1, palette: ['000000','ffff99']}, 'Sometimes Lit');
Map.addLayer( Light_And, {min: 0, max: 1, palette: ['333333','ffff55']}, 'Always Lit');
```

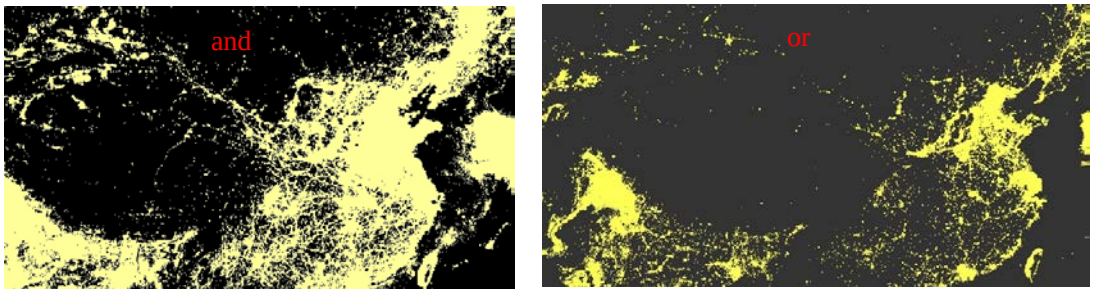

图 7.47 Image Collection 的与或操作

下边介绍 Image Collection 的函数操作，代码及执行效果如下：

```
var L8    = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                .filterBounds(ee.Geometry.Point(106.5206, 29.5618))
                .filterDate('2018-01-01','2018-12-31')
                .select('B[3-5]');

var L8_Sum          = L8.sum()
var L8_Product      = L8.product()
var L8_Max          = L8.max()
var L8_Min          = L8.min()
var L8_Mean         = L8.mean()

Map.addLayer(L8_Sum,{},'Sum')
Map.addLayer(L8_Product,{},'Product')
Map.addLayer(L8_Max,{},'Max')
Map.addLayer(L8_Min,{},'Min')
Map.addLayer(L8_Mean,{},'Mean')
```

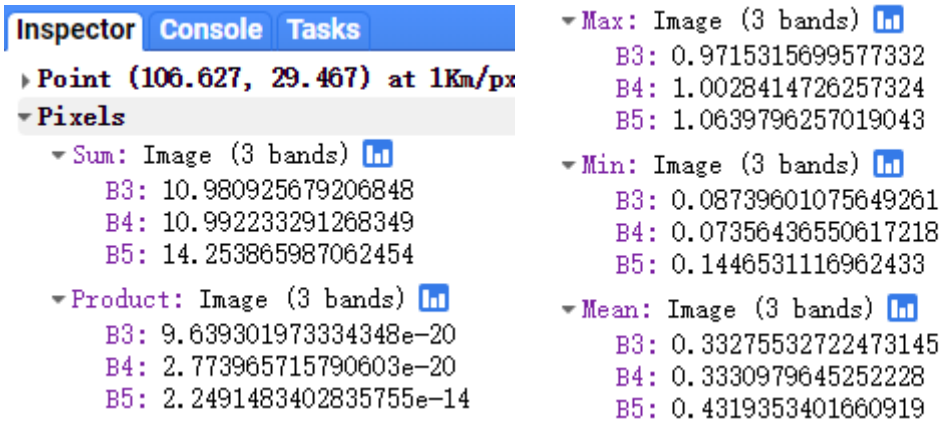

图 7.48 Image Collection 的函数操作

下边介绍 Image Collection 的取第一个操作，代码如下：

```
var L8    = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                .filterBounds(ee.Geometry.Point(106.5206, 29.5618))
                .filterDate('2018-01-01','2018-12-31')
                .select('B[3-5]')
                .sort('DATE_ACQUIRED');

var Time_No_1 = L8.first()

print(Time_No_1)
```

该操作的主要目的是观察 Image Collection 的结构。

下边介绍 Image Collection 的 toList 操作，代码及执行效果如下：

```
var L8    = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                .filterBounds(ee.Geometry.Point(106.5206, 29.5618))
                .filterDate('2018-01-01','2018-12-31')
                .select('B[3-5]')
                .sort('DATE_ACQUIRED');

var L8_List = L8.toList(5)
var Time_No_1 = ee.Image(L8_List.get(0))
var Time_No_2 = ee.Image(L8_List.get(1))
var Time_No_3 = ee.Image(L8_List.get(2))

print(Time_No_1,Time_No_2,Time_No_3)
```

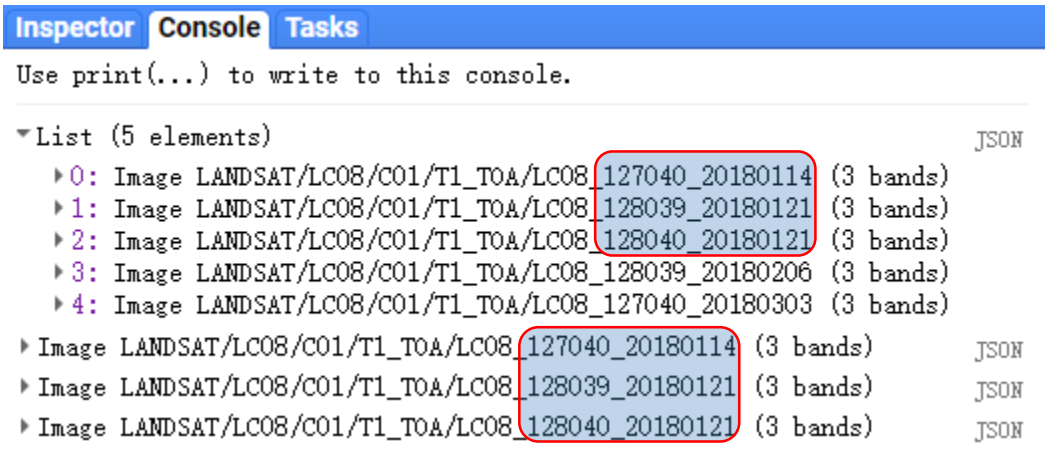

图 7.49 Image Collection 的 toList 操作

该操作的作用是将图像存储在“数据篮”(List)中，因此可以方便的提取需要的图像。

下边介绍 Image Collection 的 toArray 操作，代码及执行效果如下：

```
var L8    = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                .filterBounds(ee.Geometry.Point(106.5206, 29.5618))
                .filterDate('2018-01-01','2018-12-31')
                .select('B[3-5]')
                .sort('DATE_ACQUIRED');

var L8_List = L8.toList(5)
var Time_No_1 = ee.Image(L8_List.get(0))
var Time_No_2 = ee.Image(L8_List.get(1))
var Time_No_3 = ee.Image(L8_List.get(2))

print(Time_No_1,Time_No_2,Time_No_3)
```

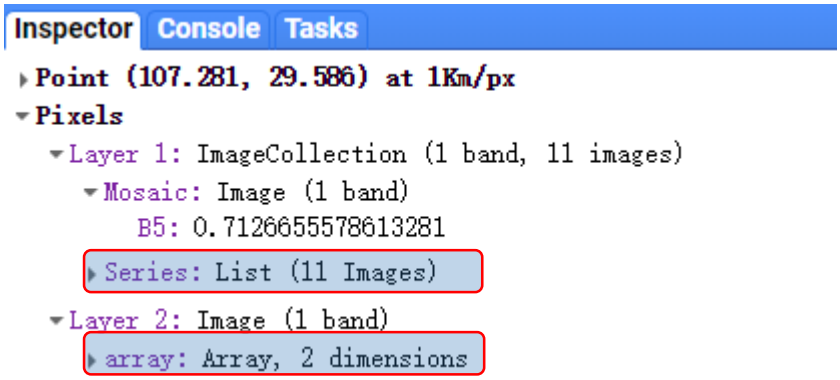

图 7.50 Image Collection 的 toArray 操作

Image Collection 的 toArray 和 toList 操作都是将原来的“图像集”转换成“数据集”的命令，但 toArray 的操作在转换的过程中带给了新数据“方向”的结构，因此使得转换后的数据更适合进行数学分析。

下边介绍 Image Collection 的 map 命令，代码及执行效果如下：

```
var L8    = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA")
                .filterBounds(ee.Geometry.Point(107.193, 29.1373))
                .filterDate('2018-01-01','2018-12-31')
                .select('B[4,5]')
                .limit(3);

function add_NDVI (image){
   var NDVI = image.normalizedDifference(['B5','B4'])
   return image.addBands(NDVI)
}

var L8_NDVI = L8.map(add_NDVI)

print(L8,L8_NDVI)
```

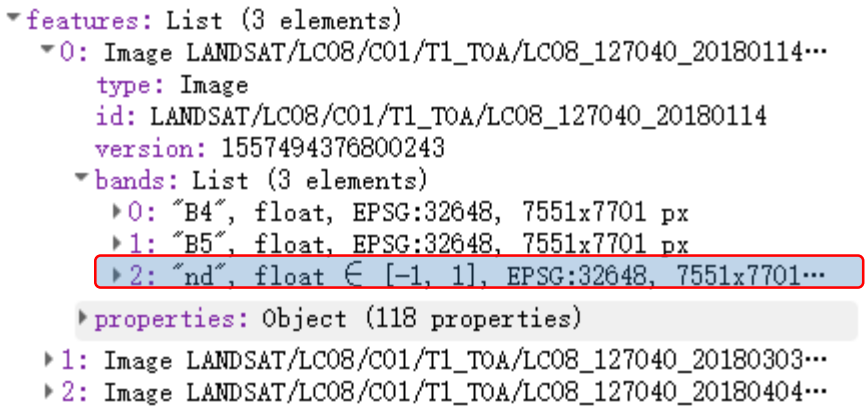

图 7.51 Image Collection 的 map 操作

与前边的例子相同，.map 命令是对矢量或者栅格数据集中的每一个数据都进行同样的操作。本例中，通过 function 命令编辑了添加 NDVI 的命令，然后利用.map 将这个命令统一运用到 Image Collection 里，达到给 Image Collection 里每个图像添加 NDVI 的目的。

下边是本节介绍的常用命令，尝试回忆其格式和功能。

ee.ImageCollection()	.limit()	.filterMetadata()	.filterDate()	.filterBounds()	
.filter()	.select()	.distinct()	.comnibe()	.unit8()	.set()
.setMulti()	.mosaic()	.and/or()	.sum/product/max/min/mean/mode/median/count()		
.first()	.toList()	.toArray()			

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