# 创建实体对象

学习如何用Entity API绘制空间数据，如点、标记、线、模型、多边形和三维体对象。

## 什么是Entity API

CesiumJS拥有丰富的空间数据API，可以将其分为两类：面向图形开发人员的低级Primitive API和用于数据驱动的可视化的高级Entity API。

低级Primitive API它的结构旨在为图形开发人员提供灵活的实现，而不是为了实现API一致性。 加载模型与创建广告牌不同，并且两者都与创建多边形完全不同。

每种可视化类型都有其自己独特的功能。此外，每个都有不同的性能特征，并且需要不同的最佳实践。尽管此方法功能强大且灵活，但可以为更高级别的抽象更好地服务于大多数应用程序。

EntityAPI公开了一组一致设计的高级对象，这些对象将相关的可视化和信息聚合到一个统一的数据结构中，我们称之为实体。

它使我们能够专注于数据的表示，而不必担心可视化的潜在机制。

它还提供了一些结构，可轻松构建复杂的，时动态的可视化文件，并使其自然地适合静态数据。

尽管Entity API实际上是在后台使用Primitive API，但这是我们（几乎）不必关心的实现细节。

通过将各种启发式方法应用于我们提供的数据，Entity API能够提供灵活的高性能可视化，同时公开一致，易学且易于使用的接口。.

## 第一个实体

通过以下代码，添加一个面。

var rect = viewer.entities.add({

id:"uniqueId",

polygon : {

hierarchy : Cesium.Cartesian3.fromDegreesArray([

109.080842,45.002073,

105.91517,45.002073,

104.058488,44.996596,

104.053011,43.002989,

104.053011,41.003906,

105.728954,40.998429,

107.919731,41.003906,

109.04798,40.998429,

111.047063,40.998429,

111.047063,42.000709,

111.047063,44.476286,

111.05254,45.002073]),

height : 0,

material : Cesium.Color.RED.withAlpha(0.5),

outline : true,

outlineColor : Cesium.Color.BLACK

}

});

viewer.zoomTo(rect);



上面的代码创建了viewer，它创建了地球和其他小部件。我们使用viewer.entities.add方法，传递给add的对象为实体提供初始值。返回值是实际的实体实例。

最后，viewer.zoomTo缩放摄影机以查看实体。可以为实体设置许多样式选项，我们为多边形填充指定了半透明的红色和黑色的轮廓。

## 图形和体

应用Entity API支持很多图形和体的绘制。这里不一一实现了，具体可查看官网教程：

https://cesium.com/docs/tutorials/creating-entities/



## 材质和轮廓

所有图形和体都有一些通用的属性来控制它们的外观。fill属性指定是否填充几何图形，outline属性指定几何图形是否轮廓。

当fill为true时，material属性决定填充的外观。下面将创建一个半透明的蓝色椭圆。

var entity = viewer.entities.add({

position: Cesium.Cartesian3.fromDegrees(-103.0, 40.0),

ellipse : {

semiMinorAxis : 250000.0,

semiMajorAxis : 400000.0,

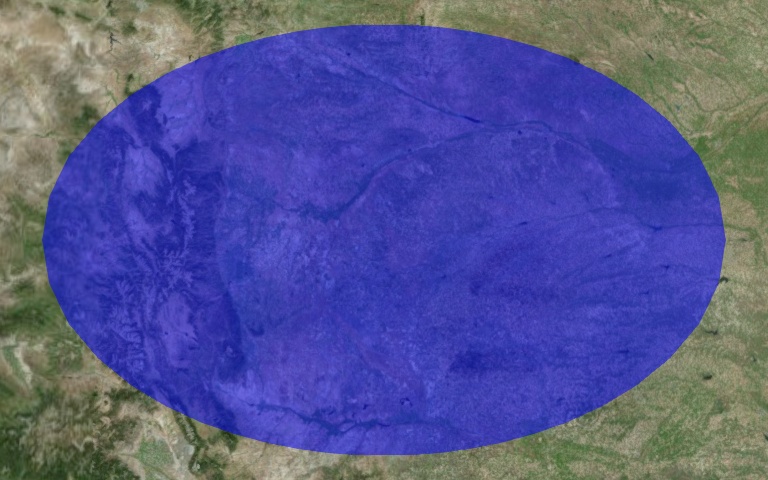
material : Cesium.Color.BLUE.withAlpha(0.5)

}

});

viewer.zoomTo(entity);

var ellipse = entity.ellipse; // For upcoming examples



## 轮廓

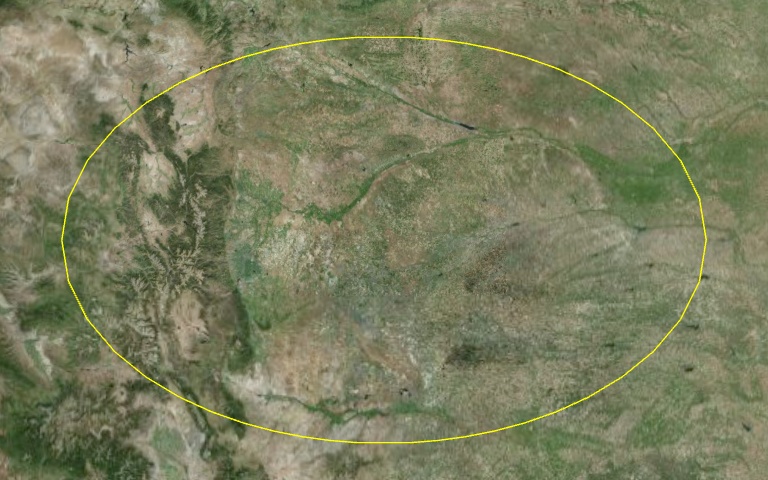
轮廓线依赖于outlineColor和outlineWidth属性。outlineWidth仅适用于非Windows系统，如Android、iOS、Linux和OS X。在Windows系统上，轮廓线的宽度始终为1。这是由于在Windows上执行WebGL的方式受到限制。 在上段代码基础上，执行以下代码，设置椭圆显示为轮廓线。

ellipse.fill = false;

ellipse.outline = true;

ellipse.outlineColor = Cesium.Color.YELLOW;

ellipse.outlineWidth = 2.0;



## 线

多段线是一种特殊情况，因为它们没有填充或轮廓特性。

他们依赖于特殊的材质属性，而不是颜色。由于这些特殊的材质设置，不同宽度和轮廓宽度的多段线将适用于所有系统。

var entity = viewer.entities.add({

polyline : {

positions : Cesium.Cartesian3.fromDegreesArray([-77, 35,

-77.1, 35]),

width : 5,

material : Cesium.Color.RED

}});

viewer.zoomTo(entity);

var polyline = entity.polyline // For upcoming examples



## 线的边框

设置边框宽度和颜色。

polyline.material = new Cesium.PolylineOutlineMaterialProperty({

color : Cesium.Color.ORANGE,

outlineWidth : 3,

outlineColor : Cesium.Color.BLACK

});



## 亮线

设置边框发亮。

polyline.material = new Cesium.PolylineGlowMaterialProperty({

glowPower : 0.2,

color : Cesium.Color.BLUE

});



## 高度延伸

平面的形状包括道路、椭圆、多边形和矩形，可以放置在高度上，也可以拉伸成一个体。在这两种情况下，平面图形或体仍将符合WGS84椭球的曲率。

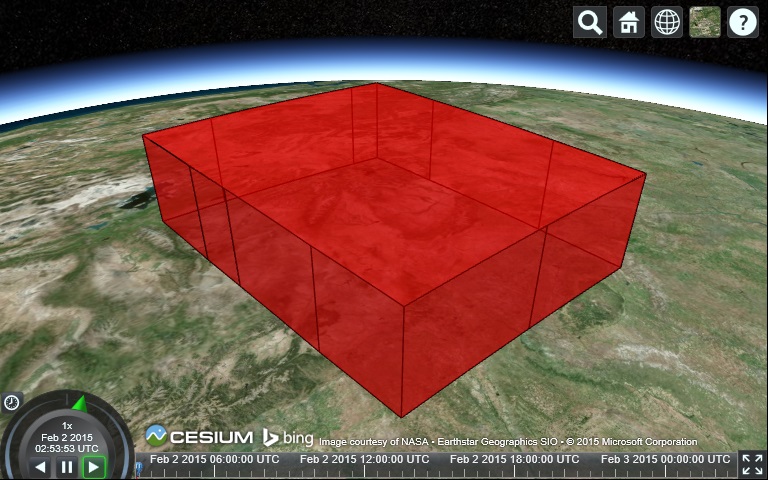
在相应的图形对象上设置高度属性（以米为单位）。下面的代码行将多边形提升到距地球20万米的高度。

若要将拉伸的体中，请设置“拉伸高度”属性。将在“高度”和“拉伸高度”之间创建体积。如果未定义高度，则体积从0开始。

下面是创建一个从200000米开始延伸到250000米的体积。

wyoming.polygon.height = 200000;

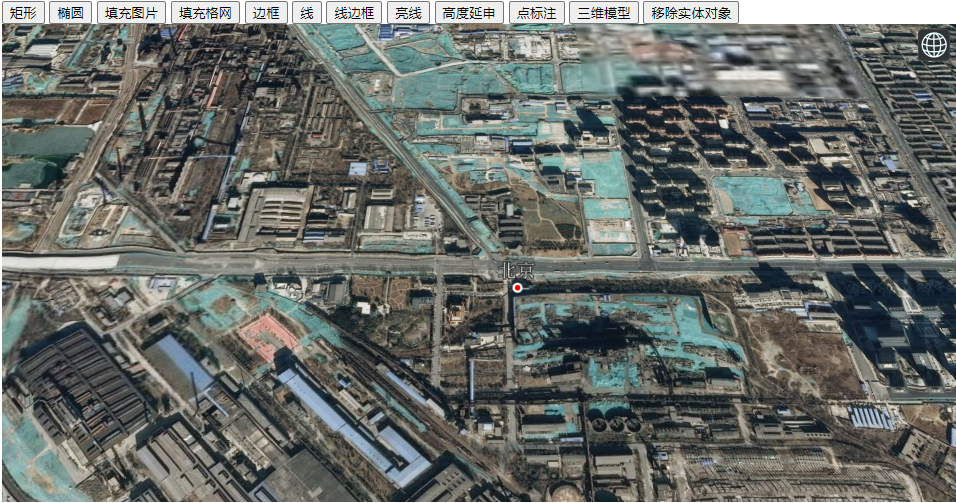
wyoming.polygon.extrudedHeight = 250000;



## 点标注

创建一个地理点或标注，例如位置的名称等。

**var** pointLabel = ***viewer***.**entities**.add({  
 **id**:**"uniqueId"**,  
 **name** : **'北京'**,  
 **position** : Cesium.Cartesian3.fromDegrees(116.166493, 39.9060534),  
 **point** : {  
 **pixelSize** : 5,  
 **color** : Cesium.Color.RED,  
 **outlineColor** : Cesium.Color.WHITE,  
 **outlineWidth** : 2  
 },  
 **label** : {  
 **text** : **'北京'**,  
 **font** : **'14pt monospace'**,  
 **style**: Cesium.LabelStyle.FILL\_AND\_OUTLINE,  
 **outlineWidth** : 2,  
 **verticalOrigin** : Cesium.VerticalOrigin.BOTTOM,  
 **pixelOffset** : **new** Cesium.Cartesian2(0, -9)  
 }  
});  
  
***viewer***.zoomTo(pointLabel);



## 3D模型

CesiumJS 支持三维模型 glTF，你可以在Sandcastle示例中找到相关示例程序。

设置模型路径和位置，就可以创建一个模型。

**var** entity = ***viewer***.**entities**.add({  
 **position** : Cesium.Cartesian3.fromDegrees(-123.0744619, 44.0503706),  
 **model** : {  
 **uri** : **'./data/Cesium\_Air.glb'** }  
});  
***viewer***.**trackedEntity** = entity;



## 相机控制

使用 viewer.zoomTo方法定位到指定的entity实体。viewer.flyTo 方法是飞行到entity。 两种方法都可以定位到Entity实体, EntityCollection实体集合, DataSource数据源,或者一个实体列表。

两种方法都计算所有提供的实体的视图。默认情况下，相机的方向为北，从45度角向下看。通过HeadingPitchRange方法，设置一个俯仰范围内参数来控制俯仰角度。

var heading = Cesium.Math.toRadians(90);

var pitch = Cesium.Math.toRadians(-30);

viewer.zoomTo(wyoming, new Cesium.HeadingPitchRange(heading, pitch));



zoomTo和flyTo都是异步函数；也就是说，不能保证它们在返回之前已经完成。例如，飞行到一个实体发生在许多动画帧上。这两个功能都可以返回Promise对象，用于执行完成后的操作。用下面的代码片段替换示例中的zoomTo。它飞到矩形实体时，在飞行结束后选择它。

viewer.flyTo(rect).then(function(result){

if (result) {

viewer.selectedEntity = rect;

}

});

如果飞行功能成功完成，则传递给回调函数的结果参数为true；如果飞行功能取消，即用户在该飞行操作完成之前启动了另一个飞行或缩放操作，或者因为目标没有相应的可视化成功，即没有可定位的对象，则传递给回调函数的结果参数为false。

有时，特别是在处理动态数据时，我们希望相机保持在实体上，而不是地球上。这是通过设置viewer.trackedEntity来实现， 需要设置跟踪实体的位置。

rect.position = Cesium.Cartesian3.fromDegrees(109.080842,45.002073);

viewer.trackedEntity = rect;

通过设置viewer.trackedEntity为undefined停止跟踪实体，或通过双击其他位置离开实体。调用zoomTo或flyTo也会取消跟踪。

## Entities管理

EntityCollection是用于管理和监视一组实体的集合。viewer.entities是一个实体集合。EntityCollection包括用于管理实体的add、remove和removeAll等方法。

有时我们需要更新以前创建的实体。所有实体实例都有一个可用于从集合中检索实体的唯一id。我们可以指定一个ID，否则将自动生成一个ID。

viewer.entities.add({

id : 'uniqueId'

});

使用getById检索实体。如果没有提供ID的实体存在，则返回undefined。

var entity = viewer.entities.getById('uniqueId');

如果实体不存在，则使用getorcreateone来创建新并获取新的实体。

var entity = viewer.entities.getOrCreateEntity('uniqueId');

手动创建一个新实体，并使用add将其添加到集合中。如果集合中已存在具有相同id的实体，则此方法将抛出异常。

var entity = new Entity({

id : 'uniqueId'

});

viewer.entities.add(entity);

使用collectionChanged事件，当在集合中添加、删除或更新实体时，它会通知侦听器。

以下是示例代码：

function onChanged(collection, added, removed, changed){

var msg = 'Added ids';

for(var i = 0; i < added.length; i++) {

msg += '\n' + added[i].id;

}

console.log(msg);

}

viewer.entities.collectionChanged.addEventListener(onChanged);

你将会看到在console中输出65个信息，因为没调用一次viewer.entities.add就会回调一次。

当一次更新大量实体时，可以在更改最后发送一个事件，这样更有效率。在viewer.entities.add之前调用viewer.entities.suspendEvents，然后在这个例子的最后调用viewer.entities.resumeEvents。当我们再次运行演示时，我们现在得到一个包含所有65个实体的事件。这些调用是引用计数的，因此可以嵌套多个suspend和resume调用。

## Picking拾取实体

拾取（点击选择一个对象）是我们需要与原始API进行简单交互的一个方面。使用scene.pick以及scene.drillPick获得实体。

function pickEntity(viewer, windowPosition) {

var picked = viewer.scene.pick(windowPosition);

if (defined(picked)) {

var id = Cesium.defaultValue(picked.id, picked.primitive.id);

if (id instanceof Cesium.Entity) {

return id;

}

}

return undefined;

};

function drillPickEntities(viewer, windowPosition) {

var i;

var entity;

var picked;

var pickedPrimitives = viewer.scene.drillPick(windowPosition);

var length = pickedPrimitives.length;

var result = [];

var hash = {};

for (i = 0; i < length; i++) {

picked = pickedPrimitives[i];

entity = Cesium.defaultValue(picked.id, picked.primitive.id);

if (entity instanceof Cesium.Entity &&

!Cesium.defined(hash[entity.id])) {

result.push(entity);

hash[entity.id] = true;

}

}

return result;

};

*//////////////////////////////////////////////////////////////////////////  
//首先要绑定鼠标左键事件。  
//参数movement获取屏幕点击坐标，剩下的就交给cesium了。  
//id是添加entity时赋予的id属性值  
//////////////////////////////////////////////////////////////////////////***var *handler*** = **new** Cesium.ScreenSpaceEventHandler(***viewer***.**canvas**);  
***handler***.setInputAction(**function** (movement) {  
 **var** tr = *pickEntity*(***viewer***,movement.**position**);  
 **if**(tr.**id**==**"entity"**){  
 *alert*(**'picked entity'**);  
 }  
  
 *// var pick = viewer.scene.pick(movement.position);  
 // if (Cesium.defined(pick) && (pick.id.id === 'entity')) {  
 // alert('picked!');  
 // }*}, Cesium.ScreenSpaceEventType.LEFT\_CLICK);

## The property system（属性系统）

我们为实体定义的所有值都存储为属性对象。如面的边框等。

outline是ConstantProperty的一个实例。本教程使用一种称为隐式属性转换（implicit property conversion）的速记形式，它自动获取原始值并在后台创建相应的属性。如果没有这个速记，我们将不得不编写一个更长版本的初始示例：

var wyoming = new Cesium.Entity();

wyoming.name = 'Wyoming';

var polygon = new Cesium.PolygonGraphics();

polygon.material = new Cesium.ColorMaterialProperty(Cesium.Color.RED.withAlpha(0.5));

polygon.outline = new Cesium.ConstantProperty(true);

polygon.outlineColor = new Cesium.ConstantProperty(Cesium.Color.BLACK);

wyoming.polygon = polygon;

viewer.entities.add(wyoming);

使用属性是因为实体API不仅能够表示常量值，还可以表示随时间变化的值。请参阅回调属性和插值Sandcastle示例来了解一些时间动态属性。