

简介及环境配置

Cesium是什么？

Cesium ['si:ziəm]是JavaScript开源库，通过Cesium，实现无插件的创建三维球和二维地图。它是通过WebGL技术实现图形的硬件加速，并且跨平台，跨浏览器，并提供动态数据的可视化展现。

cesium 功能

- 使用3d tiles 格式流式加载各种不同的3d数据，包含倾斜摄影模型、三维建筑物、CAD和BIM的外部 and 内部，点云数据。并支持样式配置和用户交互操作。
- 全球高精度地形数据可视化，支持地形夸张效果、以及可编程实现的等高线和坡度分析效果。
- 支持多种资源的图像图层，包括WMS，TMS，WMTS以及时序图像。图像支持透明度叠加、亮度、对比度、GAMMA、色调、饱和度都可以动态调整。支持图像的卷帘对比。
- 支持标准的矢量格式KML、GeoJSON、TopoJSON，以及矢量的贴地效果。
- 三维模型支持glTF2.0标准的PBR材质、动画、蒙皮和变形效果。贴地以及高亮效果。
- 使用CZML支持动态时序数据的展示。
- 支持各种几何体：点、线、面、标注、广告牌、立方体、球体、椭球体、圆柱体、走廊(corridors)、管径、墙体
- 可视化效果包括：基于太阳位置的阴影、自身阴影、柔和阴影。大气、雾、太阳、阳光、月亮、星星、水面。
- 粒子特效：烟、火、火花。
- 地形、模型、3d tiles模型的面裁剪。
- 对象点选和地形点选。
- 支持鼠标和触摸操作的缩放、渲染、惯性平移、飞行、任意视角、地形碰撞检测。
- 支持3d地球、2d地图、2.5d哥伦布模式。3d视图可以使用透视和正视两种投影方式。
- 支持点、标注、广告牌的聚集效果。

资料:

- [cesium官网](#) 
- [cesium github](#) 

更多参考资料见 [参考资料](#)

开始cesium前，先下载cesium源码，可以从官方网站 [下载](#)  也可以到 [cesium github](#)  clone。

需要安装 [node.js](#) 

编译源码

js是解释型语言，本不需要编译。但cesium是由众多模块组成，编译是为了把cesium各个模块源码打包生成统一cesium.js，对于cesium的打包命令见 [Cesium打包命令总结](#) 。

```
npm install #安装cesium开发和运行中依赖的第三方node.js包
```

sh

```
npm run release #创建`Build`目录，把cesium各个模块源码打包生成统一cesium.js，生成文档
```

sh

```
npm start #开启一个本地http server
```

sh

执行完会提示打开一个本地http地址：

```
Cesium development server running locally. Connect to http://localhost:8080/
```

其中：

- [Sandcastle](#) [在线地址](#) 包含众多cesium示例，开发者经常光顾
- [Documentation](#) [在线地址](#) cesium API文档，开发必备

我这里选择基于node依赖安装，前提需要安装 [node](#) 。

IDE: [Visual Studio Code](#)

服务器: [live-server](#) (基于node)

先执行

```
npm init
```

sh

配置生成 `package.json`：

```
Administrator@WIN-IQLNPTL95TO MINGW64 /e/workspace/Gis/sogrey/Cesium-start-Example (master) sh
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (cesium-start-example)
version: (1.0.0)
description: cesium 入门示例
entry point: (index.js)
test command:
git repository: (https://github.com/Sogrey/Cesium-start-Example.git)
keywords: cesium examples
author: Sogrey
license: (ISC) MIT
About to write to E:\workspace\Gis\sogrey\Cesium-start-Example\package.json:

{
  "name": "cesium-start-example",
  "version": "1.0.0",
  "description": "cesium 入门示例",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/Sogrey/Cesium-start-Example.git"
  },
  "keywords": [
    "cesium",
    "examples"
  ],
  "author": "Sogrey",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/Sogrey/Cesium-start-Example/issues"
  },
  "homepage": "https://github.com/Sogrey/Cesium-start-Example#readme"
}

Is this OK? (yes) y
```

如上，学习过程中的示例存放在 [Cesium-start-Example](#) ，再执行

```
npm i cesium
```

安装cesium依赖，完成后自动多出一个目录 `node_modules` 。查看 `node_modules` 下 `cesium` 的目录结构：



其中

- `Build` 目录下是打包后的，
 - `Cesium` 目录下是压缩好的，用于生产
 - `CesiumUnminified` 是未压缩的，可用于开发调试
- `Source` 为源码

自此，环境配置就基本完成了。

第一个cesium应用-Hello world

前面已经下载了cesium依赖，存放在 `node_modules` 目录下，本地运行没任何问题，在上传github加载时似乎对于 `node_modules` 有隔阂，重命名为 `libs` ，如果你是手动下载的cesium源码或release包则不会有这样的问题。

现在我们实现第一个cesium应用-Hello world。

新建一个 `hello-world.html` :

```
html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Cesium 入门 - Hello worrld</title>
</head>
<body>

</body>
</html>
```

引入cesium组件样式:

```
html
<style>
  @import url(libs/cesium/Build/CesiumUnminified/Widgets/widgets.css);
  html,
  body {
    height: 100%;
    margin: 0;
    padding: 0;
  }
</style>
```

`body` 标签中新建一个 `div` ,设置其id为 `cesiumContainer` ，并引入 `cesium.js` :

```
html
<div id="cesiumContainer" style="height: 100%;"></div>
<script src="libs/cesium/Build/CesiumUnminified/Cesium.js"></script>
```

下面就准备写下我们第一行cesium代码:

```
html
<script>
  var viewer = new Cesium.Viewer("cesiumContainer");
</script>
```

[完整代码](#) 

先预览一下吧：

[在线预览](#) 

一个圆润的地球映入眼帘，到此第一个应用就完成了。

Viewer 以及一些有用的组件

在前面我们创建了 第一个简单的cesium应用Hello world ,如下图:



[在线预览](#)

而我们的代码仅仅一行:

```
var viewer = new Cesium.Viewer("cesiumContainer");
```

js

这是我们接触到的第一个cesium API,也是最基础的。

创建viewer

任何Cesium应用的基础都是Viewer, 一个交互的三维地球仪。创建一个Viewer, 并指定一个id为 `cesiumContainer` 的div容器, cesium将在该容器中创建画布, 绘制渲染三维场景。

```
var viewer = new Cesium.Viewer("cesiumContainer");
```

js

默认, 场景能够处理鼠标和触控输入事件, 如相机控制:

- 鼠标左键单击和拖动 - 在地球表面移动相机.
- 鼠标右键单击和拖动 - 放大、缩小(相机拉近或拉远)
- 鼠标中键滚轮 - 放大、缩小
- 鼠标中键单击和拖动 - 以地球表面某个点旋转相机

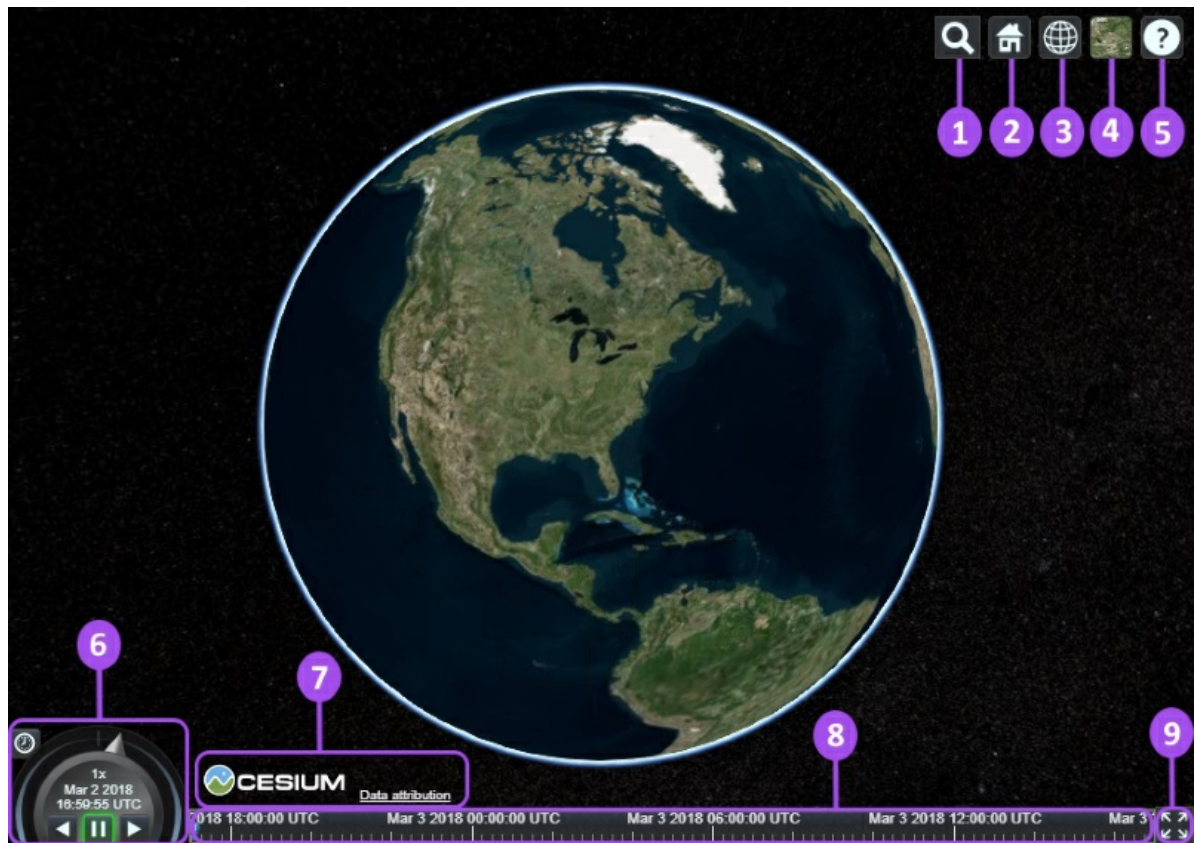
触控事件:

- 一个手指拖曳- 平移视图(One finger drag - Pan view)
- 两个手指捏放- 缩放视图(Two finger pinch - Zoom view)
- 两个手指拖曳, 相同方向- 俯仰视图(Two finger drag, same direction - Tilt view)
- 两个手指拖曳, 相反方向- 旋转视图(Two finger drag, opposite direction - Rotate view)



Viewer的一些有用的组件

观察场景我们发现除了中心一个地球仪外, 还有很多有用的组件。



1. [Geocoder](#) 地名搜索: 地名搜索工具, 相机飞行到查询地点. 默认使用Bing Maps数据.
2. [HomeButton](#) 默认视图: 视图复位, 回到默认视图.
3. [SceneModePicker](#) 场景模式: 切换模式3D, 2D 或2.5D (Columbus View).
4. [BaseLayerPicker](#) 基础图层: 选择影像或地形图层.
5. [NavigationHelpButton](#) 帮助: 帮助, 提供默认相机控制方法.
6. [Animation](#) 动画: 控制动画播放速度.
7. [CreditsDisplay](#) 鸣谢: 显示数据归属.
8. [Timeline](#) 时间线: 指示当前时间, 允许用户跳到指定时间.
9. [FullscreenButton](#) 全屏: 全屏.

我们之前仅一行代码创建了这个场景, 实际上是采用了默认配置, 查看API文档 [Viewer doc](#), `Viewer` 有两个参数, 第一个是我们已经使用过的 `container`, 传入一个指定容器的id; 第二个是配置, 以上提到的组件均可在这配置显示与否。

若要去除左下角和右上角的其他标注或按钮, 直接修改option的参数:


```
js
var defaultOption = {
  animation:false,//左下角控制动画
  baseLayerPicker:false,//右上角图层选择器
  fullscreenButton:false, //右下角全屏按钮
  geocoder:false,//右上角搜索
  homeButton:false, //home键，点击回到默认视角
  infoBox:false, //点击模型不显示cesium自带的信息框
  //scene3DOnly:false,//仅仅显示3d,可隐藏右上角2d和3d按钮
  selectionIndicator: false, //点击模型不显示cesium自带的绿色选中框
  timeline:false,//最下面时间轴
  navigationHelpButton:false,//右上角帮助按钮
  navigationInstructionsInitiallyVisible:false,
  useDefaultRenderLoop:true,
  showRenderLoopErrors:true,
  projectionPicker:false,//投影选择器
};
var viewer = new Cesium.Viewer("cesiumContainer", defaultOption);
```

查看 [这里](#) ↗

去除版权信息

js 方式:

```
js
viewer._cesiumWidget._creditContainer.parentNode.removeChild(
  viewer._cesiumWidget._creditContainer); //去掉版权信息
```

或

```
js
viewer._cesiumWidget._creditContainer.style.display = "none"; //去掉版权信息
```

css方式:

```
js
.cesium-widget-credits{
  display:none !important;
}
```

Cesium ion

Cesium ion是一个三维数据切片和存储的平台，这里使用ion平台上存储的Sentinal-2 影像和Cesium World Terrain. 需要访问 <https://cesium.com/ion/> ↗ 注册一个免费账户，并在Access Tokens页面创建访问令牌。Cesium ion默认提供5GB存储空间。

在创建viewer之前设置访问令牌:

```
js
Cesium.Ion.defaultAccessToken = '<YOUR ACCESS TOKEN HERE>';
```

添加影像图层

Cesium支持影像图层的添加、删除、排序、调整。

每个图层的亮度(brightness)、对比度(contrast)、灰度(gamma)、色相(Hue)、饱和度(Saturation)都支持动态调整。

Cesium提供影像图层操作的很多方法，包括颜色调整(color adjustment)、图层混合(layer blending)等。代码示例：

- [添加基础影像](#)
- [调整影像颜色](#)
- [影像图层控制和排序](#)
- [Splitting imagery layers](#)

Cesium提供了多个影像图层提供者，包括：

- WMS - OGC标准， [WebMapServiceImageryProvider](#)
- TMS - 访问地图瓦片的REST接口，可以使用 [MapTiler](#) or [GDAL2Tiles](#) .I 生成，参见 [createTileMapServiceImageryProvider](#)
- WMTS(with time dynamic imagery) -OGC标准， [WebMapTileServiceImageryProvider](#)
- ArcGIS - [ArcGIS Server REST API](#) ， 见 [ArcGisMapServerImageryProvider](#)
- Bing Maps - [Bing Maps REST Services](#) ， 需要 [Bing Maps key](#) , [BingMapsImageryProvider](#)
- Google Earth - [Google Earth Enterprise server](#) 发布的数据， 见 [GoogleEarthEnterpriseImageryProvider](#)
- Mapbox - 需要token, [MapboxImageryProvider](#)
- Open Street Map -访问OSM或 [Slippy map tiles](#) ， 参见 [createOpenStreetMapImageryProvider](#)
- Cesium Ion平台 - [IonImageryProvider](#)

其他内置影像图层提供者

- [GridImageryProvider](#)
- [ImageryProvider](#)
- [SingleTileImageryProvider](#) - 从一张图片中创建瓦片
- [TileCoordinatesImageryProvider](#)
- [UrlTemplateImageryProvider](#) - 自定义瓦片切片方案，如

```
//cesiumjs.org/tilesets/imagery/naturalearthii/{z}/{x}/{reverseY}.jpg
```

当然，也可以通过实现 [ImageryProvider](#)接口 定义新的影像接入方式。

举例，加载 [GoogleEarthEnterpriseImageryProvider](#)：

```
var geeMetadata = new GoogleEarthEnterpriseMetadata('http://www.earthenterprise.org/3d');
var gee = new Cesium.GoogleEarthEnterpriseImageryProvider({
  metadata : geeMetadata
});

var viewer = new Cesium.Viewer("cesiumContainer",{
  baseLayerPicker:false,//关闭基本图层
  imageryProvider:gee,
});
```

加载谷歌卫星影像：

```
js
var google=new Cesium.UrlTemplateImageryProvider({
  url:'http://www.google.cn/maps/vt?lyrs=s@800&x={x}&y={y}&z={z}',
  tilingScheme:new Cesium.WebMercatorTilingScheme(),
  minimumLevel:1,
  maximumLevel:20
});
var viewer = new Cesium.Viewer("cesiumContainer",{
  baseLayerPicker:false,//关闭基本图层
  imageryProvider:google,
});
```

加载arcGis:

```
js
var viewer = new Cesium.Viewer("cesiumContainer", {
  baseLayerPicker: false, //关闭基本图层
  imageryProvider: new Cesium.ArcGisMapServerImageryProvider({
    url: 'https://services.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer'
  }),
});
```

Cesium加载全球地形图:

```
js
var terrain=new Cesium.createWorldTerrain({
  requestWaterMask:true,
  requestVertexNormals:true
});
viewer.terrainProvider=terrain;//加入世界地形图
```

高德影像底图:

```
viewer = new Cesium.Viewer("cesiumContainer", {
  animation: false, //是否显示动画控件
  baseLayerPicker: false, //是否显示图层选择控件
  geocoder: true, //是否显示地名查找控件
  timeline: false, //是否显示时间线控件
  sceneModePicker: true, //是否显示投影方式控件
  navigationHelpButton: false, //是否显示帮助信息控件
  infoBox: true, //是否显示点击要素之后显示的信息
  imageryProvider: new Cesium.UrlTemplateImageryProvider({
    url: "https://webst02.is.autonavi.com/appmaptile?style=6&x={x}&y={y}&z={z}",
    //layer: "tdtVecBasicLayer",
    //style: "default",
    //format: "image/png",
    //tileMatrixSetID: "GoogleMapsCompatible",
    //show: false
  })
});
viewer.imageryLayers.addImageryProvider(new Cesium.UrlTemplateImageryProvider({
  url: "http://webst02.is.autonavi.com/appmaptile?x={x}&y={y}&z={z}&lang=zh_cn&size=1&scale=1&style=8&x={x}&y={y}&z={z}",
  //layer: "tdtAnnoLayer",
  //style: "default",
  //format: "image/jpeg",
  //tileMatrixSetID: "GoogleMapsCompatible"
}));
```

高德街道底图:

```
viewer = new Cesium.Viewer("cesiumContainer", {
  animation: false, //是否显示动画控件
  baseLayerPicker: false, //是否显示图层选择控件
  geocoder: true, //是否显示地名查找控件
  timeline: false, //是否显示时间线控件
  sceneModePicker: true, //是否显示投影方式控件
  navigationHelpButton: false, //是否显示帮助信息控件
  infoBox: true, //是否显示点击要素之后显示的信息
  imageryProvider: new Cesium.UrlTemplateImageryProvider({
    url: "http://webrd02.is.autonavi.com/appmaptile?lang=zh_cn&size=1&scale=1&style=8&x={x}&y={y}&z={z}",
    //layer: "tdtVecBasicLayer",
    //style: "default",
    //format: "image/png",
    //tileMatrixSetID: "GoogleMapsCompatible",
    //show: false
  })
});
viewer.imageryLayers.addImageryProvider(new Cesium.UrlTemplateImageryProvider({
  url: "http://webst02.is.autonavi.com/appmaptile?x={x}&y={y}&z={z}&lang=zh_cn&size=1&scale=1&style=8&x={x}&y={y}&z={z}",
  //layer: "tdtAnnoLayer",
  //style: "default",
  //format: "image/jpeg",
  //tileMatrixSetID: "GoogleMapsCompatible"
}));
```

更多影像地图查看 [国家地理信息公共服务平台](#) 

配置地形

Viewer除了 `imageryProvider` 影像外，还有 `terrainProvider` 地形，默认为 `EllipsoidTerrainProvider`

举例：

```
// Create Cesium World Terrain with default settings
var viewer = new Cesium.Viewer('cesiumContainer', {
  terrainProvider : Cesium.createWorldTerrain();
});
```

```
// Create Cesium World Terrain with water and normals.
var viewer = new Cesium.Viewer('cesiumContainer', {
  terrainProvider : Cesium.createWorldTerrain({
    requestWaterMask : true,
    requestVertexNormals : true
  });
});
```

配置场景

将场景配置为基于太阳的位置启用照明。

```
// Enable lighting based on sun/moon positions
viewer.scene.globe.enableLighting = true;
```

这将使我们场景中的照明随一天的时间而变化。如果缩小，您会看到地球的一部分很暗，因为太阳已经落在世界的那一部分。

一些基本的Cesium类型:

- `Cartesian3`：3D直角坐标—使用地球固定框架（ECEF）相对于地球中心（以米为单位）时，
- `Cartographic`：由经度，纬度（以弧度表示）和距WGS84椭球面的高度定义的位置
- `HeadingPitchRoll`：围绕东西向北框架中的局部轴的旋转（以弧度为单位）。航向是绕负z轴的旋转。螺旋是绕负y轴的旋转。滚动是绕正x轴的旋转。
- `Quaternion`：表示为4D坐标的3D旋转。

相机控制

该 `Camera` 是属性 `viewer.scene` 和控制什么是目前可见的。我们可以直接设置摄像机的位置和方向，也可以使用Cesium Camera API来控制摄像机，该API旨在指定摄像机随时间的位置和方向。

一些最常用的方法是：

- `Camera.setView(options)`：立即将相机设置在特定的位置和方向
- `Camera.zoomIn(amount)`：沿着视图矢量向前移动相机
- `Camera.zoomOut(amount)`：沿着视图矢量向后移动相机

- `Camera.flyTo(options)` [🔗](#)：创建从当前摄像机位置到新位置的动画摄像机飞行
- `Camera.lookAt(target, offset)` [🔗](#)：定位和定位相机，以给定偏移量瞄准目标点
- `Camera.move(direction, amount)` [🔗](#)：沿任何方向移动相机
- `Camera.rotate(axis, angle)` [🔗](#)：围绕任何轴旋转相机

要了解API的功能，请查看以下相机演示：

- [相机API演示](#) [🔗](#)
- [自定义相机控件演示](#) [🔗](#)

让我们将相机移至纽约，尝试其中一种方法。`camera.setView()` [🔗](#) 使用 `Cartesian3` [🔗](#) 和设置初始视图，使用 `a` 和 `HeadingPitchRoll` [🔗](#) 来定位和定向：

```
js
// Create an initial camera view
var initialPosition = new Cesium.Cartesian3.fromDegrees(-73.998114468289017509, 40.6745128956466, 161.0);
var initialOrientation = new Cesium.HeadingPitchRoll.fromDegrees(7.1077496389876024807, -31.9872, 0);
var homeCameraView = {
  destination : initialPosition,
  orientation : {
    heading : initialOrientation.heading,
    pitch : initialOrientation.pitch,
    roll : initialOrientation.roll
  }
};
// Set the initial view
viewer.scene.camera.setView(homeCameraView);
```

现在将摄像机定位并定向为向下看向曼哈顿，并且我们的视图参数保存在一个对象中，我们可以将其传递给其他摄像机方法。

实际上，我们可以使用同一视图来更新按下主屏幕按钮的效果。与其让我们从远处返回到地球的默认视图，不如覆盖按钮以将我们带到曼哈顿的初始视图。我们可以通过添加更多选项来调整动画，然后添加事件监听器以取消默认排序，并调用 `flyTo()` [🔗](#) 新的主视图：

```
js
// Add some camera flight animation options
homeCameraView.duration = 2.0;
homeCameraView.maximumHeight = 2000;
homeCameraView.pitchAdjustHeight = 2000;
homeCameraView.endTransform = Cesium.Matrix4.IDENTITY;
// Override the default home button
viewer.homeButton.viewModel.command.beforeExecute.addEventListener(function (e) {
  e.cancel = true;
  viewer.scene.camera.flyTo(homeCameraView);
});
```

有关基本相机控制的更多信息，请查看官方的 [相机教程](#) [🔗](#)。

时钟控制

配置查看器 `Clock` [🔗](#) 并 `Timeline` [🔗](#) 控制场景中时间的流逝。

[Clock API](#) [🔗](#)

当使用特定时间时，Cesium使用该 `JulianDate` [🔗](#) 类型，该类型存储自1月1日正午-4712（公元前4713年）以来的天数。为了提高精度，此类将日期的整数部分和日期的秒数部分存储在单独的组件中。为了安全进行算术运算并表示leap秒，该日期始终存储在国际原子时间标准中。

这是我们如何设置场景时间选项的示例：

```
// Set up clock and timeline.
viewer.clock.shouldAnimate = true; // make the animation play when the viewer starts
viewer.clock.startTime = Cesium.JulianDate.fromIso8601("2017-07-11T16:00:00Z");
viewer.clock.stopTime = Cesium.JulianDate.fromIso8601("2017-07-11T16:20:00Z");
viewer.clock.currentTime = Cesium.JulianDate.fromIso8601("2017-07-11T16:00:00Z");
viewer.clock.multiplier = 2; // sets a speedup
viewer.clock.clockStep = Cesium.ClockStep.SYSTEM_CLOCK_MULTIPLIER; // tick computation mode
viewer.clock.clockRange = Cesium.ClockRange.LOOP_STOP; // loop at the end
viewer.timeline.zoomTo(viewer.clock.startTime, viewer.clock.stopTime); // set visible range
```

这将设置场景动画的速率，开始和停止时间，并在达到停止时间时告诉时钟返回到开始。还将时间线小部件设置为适当的时间范围。查看此 [时钟示例代码](#) [🔗](#) 以尝试时钟设置。

参考

- <https://cesium.com/docs/tutorials/cesium-workshop/#creating-the-viewer> [🔗](#)
- <https://www.jianshu.com/p/f66caf4cb43f> [🔗](#)

cesium 坐标系统

WGS84坐标系 和 笛卡尔空间Cartesian3直角坐标系

- WGS84 坐标系
- 笛卡尔空间(Cartesian3)直角坐标系

地球仪长半轴：6378137.0米

- Cartographic 制图坐标 (longitude, latitude, height)，对应经纬度坐标，弧度制，主要用在用户接口上。方便理解、直观。
- Cartesian2 平面坐标系
- Cartesian3 笛卡尔直角坐标系 (x,y,z) 做空间计算用
- Cartesian4 几乎用不到

笛卡尔空间坐标的原点就是椭球的中心。单位米。

Cesium的坐标是以地心为原点，一向指向南美洲(X 经度0)，一向指向亚洲(Y)，一向指向北极州(Z)

- 从经纬度经 fromDegrees 函数转换成 Cartesian3 世界坐标。

fromDegrees()方法，将经纬度和高程转换为世界坐标

```
Cesium.Cartesian3.fromDegrees(-117.16, 32.71, 15000.0)
//等同于
//new Cesium.Cartesian3(-2457919.937615054, -4790818.832832404, 3435047.293539871)
```

- 直接new Cartesian3:

```
new Cesium.Cartesian3(-2457919.937615054, -4790818.832832404, 3435047.293539871)
```

坐标转换

构造定义：

1. Cartesian2 : new Cesium.Cartesian2(x, y)
2. Cartesian3 : new Cesium.Cartesian3(x, y, z)
3. Cartographic: new Cesium.Cartographic(longitude, latitude, height) 注：经纬度为弧度单位

角度与弧度转换


```

/**
 * 弧度转角度
 */
function radian2Degrees(radian) {
    // 角度 = 弧度 * 180 / Math.PI;
    return radian * 180 / Math.PI;
}
/**
 * 角度转弧度
 */
function degrees2Radian(degrees) {
    // 弧度= 角度 * Math.PI / 180;
    return degrees * Math.PI / 180;
}

```

```

//将弧度转换为度。
Cesium.Math.toDegrees(radian);
//将度数转换为弧度。
Cesium.Math.toRadians(degrees);

```

获取鼠标点在屏幕中的坐标

```

// 获取画布
var canvas = viewer.scene.canvas;

var mouseHandler = new Cesium.ScreenSpaceEventHandler(canvas);

// 绑定鼠标左点击事件
mouseHandler.setInputAction(function (event){
    // 获取鼠标点的windowPosition
    var windowPosition = event.position;
    }, Cesium.ScreenSpaceEventType.LEFT_CLICK);

```

屏幕坐标转换为笛卡尔空间直角坐标

```

var ray = viewer.camera.getPickRay(windowPosition);

var cartesian3 = viewer.scene.globe.pick(ray, viewer.scene);

```

三维笛卡尔空间直角坐标转换为地理坐标(弧度)

```

var ellipsoid=viewer.scene.globe.ellipsoid;

var cartographic=ellipsoid.cartesianToCartographic(cartesian);

```

三维笛卡尔空间直角坐标转换为地理坐标(经纬度)

```
var ellipsoid=viewer.scene.globe.ellipsoid;

var cartographic=ellipsoid.cartesianToCartographic(cartesian);

var lat=Cesium.Math.toDegrees(cartographic.latitude);
var lng=Cesium.Math.toDegrees(cartographic.longitude);
var alt=cartographic.height;
```

经纬度转换为笛卡尔空间直角坐标

直接转

```
# Cesium.Cartesian3.fromDegrees(longitude, latitude, height, ellipsoid, result)
var position = Cesium.Cartesian3.fromDegrees(-115.0, 37.0);
```

先转换为弧度，再进行转换

```
var ellipsoid=viewer.scene.globe.ellipsoid;
var cartographic=Cesium.Cartographic.fromDegrees(lng, lat, alt);
var cartesian = ellipsoid.cartographicToCartesian(cartographic);
```

笛卡尔空间直角坐标转换为屏幕坐标

```
var pick = Cesium.SceneTransforms.wgs84ToWindowCoordinates(viewer.scene, cartesian);
```

三维笛卡尔坐标转换为二维笛卡尔坐标

```
Cesium.Cartesian2.fromCartesian3(cartesian, result)
```

弧度与经纬度的相互转换

经纬度转换为弧度


```
Cesium.CesiumMath.toRadians(degrees)
```

弧度转换为经纬度






```
Cesium.CesiumMath.toDegrees(radians)
```

大体总结：`Cartographic` 与 `Cartesian3`、`Cartesian2`、经纬度坐标（WGS84）坐标转换：

- `Cartographic` → `Cartesian3` : `Cartographic.toCartesian` [↗](#)
- `Cartesian3` → `Cartographic` : `Cesium.Cartographic.fromCartesian(cartesian, ellipsoid, result)` [↗](#)
- `Cartesian3` → `Cartesian2`: `Cesium.Cartesian2.fromCartesian3(cartesian, result)` [↗](#)



- 经纬度坐标（WGS84）→ Cartesian3: [Cesium.Cartesian3.fromDegrees\(longitude, latitude, height, ellipsoid, result\)](#) 
- 弧度坐标 → Cartesian3: [Cesium.Cartesian3.fromRadians\(longitude, latitude, height, ellipsoid, result\)](#) 
- 经纬度坐标（WGS84）→ Cartographic: [Cesium.Cartographic.fromDegrees\(longitude, latitude, height, result\)](#) 

Cartesian3一些常用API:

- [Cartesian3.clone](#)  复制Cartesian3实例。
- [Cartesian3.distance](#)  计算两点之间的距离。
- [Cartesian3.dot](#)  计算两个笛卡尔的点（标量）乘积。
- [Cartesian3.cross](#)  计算两个笛卡尔的叉（外）乘积。
- [Cartesian3.normalize](#)  计算提供的笛卡尔坐标系的标准化形式，归一化

注意传入参数末尾参数result，为了优化内存使用，传入result，计算后结果赋值给该result，可复用，不传则会创建一个result。

参考

- [关于Cesium中的常用坐标系及说明](#) 
- [Cesium中的坐标的转化](#) 
- [Cesium中的几种坐标和相互转换](#) 
- [Cesium.Cartesian3 和经纬度以及屏幕坐标等之间的转换](#) 

Viewer类-一切API的入口

Camera 相机及视角

Camera类-去哪儿，随心所欲

cesium中的相机：

`Cesium.Viewer.camera` : [Camera](#)

Camera常用属性：

- [position](#) 相机在世界坐标中的位置， [direction](#) 相机的观看方向， [right](#) 相机的朝右方向？ [up](#) 相机的向上方向？



- [heading](#) (朝向)、 [pitch](#) (俯仰)、 [roll](#) (翻滚)



图中 `g` 是重力方向与 `z` 相反。

盗个图：



盗图选自 [@installing](#) : [Cesium类HeadingPitchRoll及heading、pitch、roll等参数详解](#)

Camera有几个常用API：

- `[setView(options)]`(<https://cesium.com/docs/cesiumjs- startref-doc/Camera.html#setView>) Sets the camera position, orientation and transform. 设置相机的位置、方向和变换。
- [flyTo\(options\)](#) Flies the camera from its current position to a new position. 使相机从当前位置飞到新位置。
- [HeadingPitchRange\(heading, pitch, range\)](#)
- [lookAt\(target, offset\)](#)

setView(options)

官方示例：

```

// 1. Set position with a top-down view 设置相机位置
viewer.camera.setView({
  destination : Cesium.Cartesian3.fromDegrees(-117.16, 32.71, 15000.0)
});

// 2 Set view with heading, pitch and roll
viewer.camera.setView({
  destination : cartesianPosition,
  orientation: {
    heading : Cesium.Math.toRadians(90.0), // east, default value is 0.0 (north) 左右摆头
    pitch : Cesium.Math.toRadians(-90),    // default value (looking down) 上下抬头
    roll : 0.0                             // default value
  }
});

// 3. Change heading, pitch and roll with the camera position remaining the same.
viewer.camera.setView({
  orientation: {
    heading : Cesium.Math.toRadians(90.0), // east, default value is 0.0 (north)
    pitch : Cesium.Math.toRadians(-90),    // default value (looking down)
    roll : 0.0                             // default value
  }
});

// 4. View rectangle with a top-down view
viewer.camera.setView({
  destination : Cesium.Rectangle.fromDegrees(west, south, east, north)
});

// 5. Set position with an orientation using unit vectors.
viewer.camera.setView({
  destination : Cesium.Cartesian3.fromDegrees(-122.19, 46.25, 5000.0),
  orientation : {
    direction : new Cesium.Cartesian3(-0.04231243104240401, -0.20123236049443421, -0.9786292),
    up : new Cesium.Cartesian3(-0.47934589305293746, -0.8553216253114552, 0.1966022179118339)
  }
});

```

查看北京城:

```

viewer.camera.setView({
  destination: Cesium.Cartesian3.fromDegrees(116.39, 39.9, 15000.0),
  orientation: {
    heading: Cesium.Math.toRadians(0.0), // east, default value is 0.0 (north)
    pitch: Cesium.Math.toRadians(-90), // default value (looking down)
    roll: 0.0 // default value
  }
});

```

setView 之 rectangle 方式

设置目标为一个矩形区域。定位到范围参考 [这里](#) 

```
view.camera.setView({
  destination: Cesium.Rectangle.fromDegrees(0.0,20.0,10.0,30.0), //west, south, east, north
  orientation: {
    heading : Cesium.Math.toRadians(20.0), // 方向
    pitch : Cesium.Math.toRadians(-90.0), // 倾斜角度
    roll : 0
  }
});
```

flyTo(options)

官方示例:

```
// 1. Fly to a position with a top-down view
viewer.camera.flyTo({
  destination : Cesium.Cartesian3.fromDegrees(-117.16, 32.71, 15000.0)
});

// 2. Fly to a Rectangle with a top-down view
viewer.camera.flyTo({
  destination : Cesium.Rectangle.fromDegrees(west, south, east, north)
});

// 3. Fly to a position with an orientation using unit vectors.
viewer.camera.flyTo({
  destination : Cesium.Cartesian3.fromDegrees(-122.19, 46.25, 5000.0),
  orientation : {
    direction : new Cesium.Cartesian3(-0.04231243104240401, -0.20123236049443421, -0.9786292),
    up : new Cesium.Cartesian3(-0.47934589305293746, -0.8553216253114552, 0.1966022179118339)
  }
});

// 4. Fly to a position with an orientation using heading, pitch and roll.
viewer.camera.flyTo({
  destination : Cesium.Cartesian3.fromDegrees(-122.19, 46.25, 5000.0),
  orientation : {
    heading : Cesium.Math.toRadians(175.0),
    pitch : Cesium.Math.toRadians(-35.0),
    roll : 0.0
  }
});
```

查看北京城:

```
viewer.camera.flyTo({
  destination: Cesium.Cartesian3.fromDegrees(116.39, 39.9, 15000.0),
  orientation: {
    heading: Cesium.Math.toRadians(0.0), // east, default value is 0.0 (north)
    pitch: Cesium.Math.toRadians(-90), // default value (looking down)
    roll: 0.0 // default value
  }
});
```

HeadingPitchRange(heading, pitch, range)

在局部框架中定义 `航向角`，`俯仰角` 和 `范围`。

- 航向是从局部北向旋转，其中正角向东增加。间距是从局部xy平面旋转的角度。
- 正俯仰角在平面上方。负俯仰角在平面下方。
- 范围是距框架中心的距离。

参数：

- `heading` 航向角，以弧度为单位。航向方向的右侧为正
- `pitch` 俯仰角（以弧度为单位）。仰为正，俯为负
- `range` 距中心的距离，以米为单位。

Using a HeadingPitchRange offset

```
var center = Cesium.Cartesian3.fromDegrees(116.39, 39.9);
var heading = Cesium.Math.toRadians(50.0);
var pitch = Cesium.Math.toRadians(-20.0);
var range = 5000.0;
viewer.camera.lookAt(center, new Cesium.HeadingPitchRange(heading, pitch, range));
```

js

lookAt(target, offset)

使用 `目标` 和 `偏移量` 设置摄像机的 `位置` 和 `方向`。目标必须以世界坐标给出。偏移可以是 `笛卡尔坐标系`，也可以是以目标为中心的局部北北上参考系中的 `航向` / `俯仰` / `范围`。

如果偏移量是笛卡尔坐标，则它是相对于转换矩阵定义的参考帧中心的偏移量。

如果偏移为航向/俯仰/范围，则航向和俯仰角在变换矩阵定义的参考帧中定义。

航向是从y轴到x轴的角度。间距是从xy平面开始的旋转。

正俯仰角在平面下方。负俯仰角在平面上方。范围是距中心的距离。

在2D模式下，必须有一个俯视图。摄像机将被放置在目标上方并向上看。目标上方的高度将是 `偏移量`。航向将根据偏移量确定。如果无法从偏移量确定航向，则航向将为北。

- `target` 世界坐标中的目标位置。
- `offset` 在局部东北向上参考系中与目标的偏移，以目标为中心。

Using a cartesian offset

```
var center = Cesium.Cartesian3.fromDegrees(116.39, 39.9);
viewer.camera.lookAt(center, new Cesium.Cartesian3(0.0, -4790000.0, 3930000.0));
```

js

[在线预览](#) 

记录视角

同理，想要标记某个位置和角度，下次直接进入，可以在选好的角度上按 **F12** 进入开发者工具输入

- `viewer.camera.heading`
- `viewer.camera.pitch`
- `viewer.camera.position`

回车可以得到信息, 获取到的是弧度, 角度与弧度转换

```
//获取视角
function getCamera() {
  return {
    position: viewer.camera.position,
    heading: viewer.camera.heading,
    pitch: viewer.camera.pitch
  }
}
```

```
//设置视角
viewer.camera.flyTo({
  destination: getCamera().position,
  orientation: {
    heading: getCamera().heading, // east, default value is 0.0 (north)
    pitch: getCamera().pitch, // default value (looking down)
    roll: 0.0 // default value
  }
});
```

[在线预览-查看北京城](#) [在线预览-地球自转](#)

设置默认视角

cesium默认视角定位在美国，也就是点击 `HomeButton` 转向的视角，怎么修改默认视角呢？

需要在 创建 `Viewer` 之前 执行下面代码：

```
//设置默认视角在中国
var china = Cesium.Rectangle.fromDegrees(100,10,120,70);
Cesium.Camera.DEFAULT_VIEW_RECTANGLE = china;

var viewer = new Cesium.Viewer("cesiumdiv");
```

Cartesian3和Cartographic

- `Cartographic` 制图坐标 (longitude, latitude, height)，对应经纬度坐标，弧度制，主要用在用户接口上。方便理解、直观。
- `Cartesian3` 笛卡尔直角坐标系 (x,y,z) 做空间计算用

坐标转换:

- Cartographic -> Cartesian3 : [Cartographic.toCartesian](#)
- Cartesian3 -> Cartographic : [Cartographic.fromCartesian](#)

Cartesian3一些常用API:

- [Cartesian3.clone](#) 复制Cartesian3实例。
- [Cartesian3.distance](#) 计算两点之间的距离。
- [Cartesian3.dot](#) 计算两个笛卡尔的点（标量）乘积。
- [Cartesian3.cross](#) 计算两个笛卡尔的叉（外）乘积。
- [Cartesian3.normalize](#) 笛卡尔标准化，归一化

获取相机朝向在水平面上的投影的方向

应用场景：做第一人称漫游时，调整相机方向朝向地下，前进 [camera.moveForward](#) 和后退 [camera.moveBackward](#) 时出现上天入地的情况，为模拟人物在水平面行走，这种情况是不允许的，不论相机朝天朝地都应该在水平面平行方向运动。

由 [camera.lookUp](#) 的源码有感而发。

```
var pitch = viewer.camera.pitch;
viewer.camera.look(viewer.camera.right,pitch);//调整相机朝向水平
//相机朝向在水平面上的投影的方向
console.log(viewer.camera.direction.clone());
viewer.camera.look(viewer.camera.right,-pitch);//恢复相机朝向为之前的方向
```

js

查看 [示例](#)

参考

- [\[官方示例\]Camera Tutorial](#)
- [Cesium相机](#)
- [Cesium中级教程3 - Camera - 相机（摄像机）](#)
- [cesium中定位方法使用](#)
- [cesium中的定位方法比较](#)

ImageryLayer类-影像图层，给地球换个皮肤

ImageryLayer类

ImageryProvider类

TerrainProvider类-地形，让三维表现更立体

sampleTerrain

Entity-API 与地球交互

Entity [↗](#)

Option

属性:

名称	类型	描述
<code>id</code>	String	可选 此对象的唯一标识符。如果未提供，则将生成GUID。
<code>name</code>	String	可选 显示给用户的人类可读名称。它不必是唯一的。
<code>availability</code>	TimeIntervalCollection ↗	可选 与此对象关联的可用性（如果有）。
<code>show</code>	boolean	可选 的布尔值，指示是否显示实体及其子级。
<code>description</code>	Property ↗	可选 字符串属性，用于为此实体指定HTML描述。
<code>position</code>	PositionProperty ↗	可选 一个指定实体位置的属性。
<code>orientation</code>	Property ↗	可选 一个指定实体方向的属性。

可添加的Graphics 图案:

option	类型	描述
billboard	BillboardGraphics	可选 与此广告实体关联的广告牌。
box	BoxGraphics	可选 与此实体关联的框。
corridor	CorridorGraphics	可选 与此实体关联的走廊。
cylinder	CylinderGraphics	可选 要与此实体关联的圆柱体。
ellipse	EllipseGraphics	可选 与此实体关联的椭圆。
ellipsoid	EllipsoidGraphics	可选 与此实体关联的椭球。
label	LabelGraphics	可选 的options.label与此实体关联。
model	ModelGraphics	可选 与该实体关联的模型。
path	PathGraphics	可选 与此实体关联的路径。
plane	PlaneGraphics	可选 与此实体关联的平面。
point	PointGraphics	可选 与此实体关联的点。
polygon	PolygonGraphics	可选 要与此实体关联的多边形。
polyline	PolylineGraphics	可选 与此实体关联的折线。
polylineVolume	PolylineVolumeGraphics	可选 polylineVolume与此实体关联。
rectangle	RectangleGraphics	可选 要与此实体关联的矩形。
wall	WallGraphics	可选 与此实体关联的墙。

举例：

官方例子 [Points](#)

```
viewer.entities.add({
  position : Cesium.Cartesian3.fromDegrees(-75.59777, 40.03883),
  point : {
    pixelSize : 10,
    color : Cesium.Color.YELLOW
  }
});
```

上面示例中,使用了viewer的entities添加Entity实体，entities实际是entity的集合对象。其中：

- `id` : 表示唯一标识符
- `name` : 表示名字，可以不设置
- `position` : 点在场景中位置
- `point` : 指明该entity对象为point类型 [PointGraphics](#)，其中大小为10、颜色为黄色。

同样，添加面对象和多边形雷同，具体查询对应API文档。

```

var videoElement = document.getElementById('trailer'); //获得video对象
var sphere = viewer.entities.add({ //添加实体
  name: 'video plane outline',
  position: Cesium.Cartesian3.fromDegrees(-79, 39, 0),
  plane: { //面对象
    plane: new Cesium.Plane(Cesium.Cartesian3.UNIT_Z, 0.0),
    dimensions: new Cesium.Cartesian2(400.0, 217.5),
    fill: true,
    outline: true,
    material: videoElement //指定材质
  },
  polygon: { //不规则多边形
    hierarchy: Cesium.Cartesian3.fromDegreesArray([
      -79, 39,
      -79.015, 39,
      -79.02, 39.01,
      -79, 39.01
    ]),
    material: videoElement //指定材质
  }
});

```

[查看 示例](#)

材质

空间对象可视化，不仅需要知道对象的空间位置，还需要知道对象的显示样式。显示样式就是通过材质来控制，比如说颜色、透明度、纹理贴图、更高级的光照等等。我们常用到就是颜色和透明度。

以下代码为绘制一个半透明的红色椭圆，设置material为Cesium.Color.RED.withAlpha(0.5)透明度为0.5的红色：

```

viewer.entities.add({
  position: Cesium.Cartesian3.fromDegrees(103.0, 40.0),
  name: 'Red ellipse on surface with outline',
  ellipse: {
    semiMinorAxis: 250000.0,
    semiMajorAxis: 400000.0,
    material: Cesium.Color.RED.withAlpha(0.5),
  }
});

```

填充和边框

填充和边框共同组成了面状对象的样式，通过制定属性 `fill`（默认为 `true`）和 `outline`（默认为 `false`）来确定是否显示填充和边框，`material` 对应填充样式，`outlineColor` 和 `outlineWidth` 对应边框的颜色和宽度。如一下内容绘制一个填充半透明红色边框为蓝色的椭圆：

```
viewer.entities.add({
  position:Cesium.Cartesian3.fromDegrees(103.0, 40.0),
  name:'Red ellipse on surface with outline',
  ellipse:{
    semiMinorAxis:300000.0,
    semiMajorAxis:300000.0,
    height:200000.0,
    fill:true,
    material:Cesium.Color.RED.withAlpha(0.5),
    outline:true, //必须设置height, 否则outline无法显示
    outlineColor:Cesium.Color.BLUE.withAlpha(0.5),
    outlineWidth:10.0//不能设置, 固定为1
  }
});
```

贴图

通过设置material为图片url, 可以将图片填充到对象中:

```
viewer.entities.add({
  position:Cesium.Cartesian3.fromDegrees(103.0, 40.0),
  name:'Red ellipse on surface with outline',
  ellipse:{
    semiMinorAxis:250000.0,
    semiMajorAxis:400000.0,
    height:200000.0,
    fill:true,
    material:"./sampledata/images/globe.jpg",
    outline:true, //必须设置height, 否则outline无法显示
    outlineColor:Cesium.Color.BLUE.withAlpha(0.5),
    outlineWidth:10.0//windows系统下不能设置固定为1
  }
});
```

垂直拉伸

有时候我们需要将面在垂直方向进行拉伸形成体, 通过extrudedHeight即可实现这种效果, 形成的体积任然符合它拉伸面的地球曲率。


```
viewer.entities.add({
  position:Cesium.Cartesian3.fromDegrees(103.0, 40.0),
  name:'Red ellipse on surface with outline',
  ellipse:{
    semiMinorAxis:250000.0,
    semiMajorAxis:400000.0,
    height:200000.0,
    extrudedHeight:400000.0,
    fill:true,
    material:Cesium.Color.RED.withAlpha(0.5),
    outline:true, //必须设置height, 否则outline无法显示
    outlineColor:Cesium.Color.BLUE.withAlpha(0.5),
    outlineWidth:10.0//windows系统下不能设置固定为1
  }
});
```

场景中entity管理

viewer.entities属性实际上是一个EntityCollection对象，是entity的一个集合，提供了add、remove、removeAll等等接口来管理场景中的entity。

查看帮助文档，提供一下接口：

- [Cesium.EntityCollection.collectionChangedEventCallback\(collection,added, removed, changed\)](#) [↗](#) The signature of the event generated by [EntityCollection#collectionChanged](#) [↗](#) .
- [add\(entity\) → Entity](#) [↗](#) 添加实体
- [computeAvailability\(\) → TimeInterval](#) [↗](#) 计算集合中实体的最大可用性。如果集合包含无限可用数据和无限数据的混合，则它将返回仅与非无限数据有关的时间间隔。如果所有数据都是无限的，则将返回无限的间隔。
- [contains\(entity\) → Boolean](#) [↗](#) 是否有包含
- [getById\(id\) → Entity](#) [↗](#) 通过ID查询实体
- [getOrCreateEntity\(id\) → Entity](#) [↗](#) 获取具有指定ID的实体或创建它，如果不存在，则将其添加到集合中。
- [remove\(entity\) → Boolean](#) [↗](#) 移除指定实体
- [removeAll\(\)](#) [↗](#) 移除全部，清空
- [removeById\(id\) → Boolean](#) [↗](#) 移除指定ID的实体
- [resumeEvents\(\)](#) [↗](#) [EntityCollection#collectionChanged](#) [↗](#) 添加或删除项目后立即 恢复引发事件。调用此函数时，在事件暂停期间进行的任何修改将作为单个事件触发。该函数是按引用计数的，只要有对的相应调用，就可以安全地多次调用 [EntityCollection#resumeEvents](#) [↗](#) 。
- [suspendEvents\(\)](#) [↗](#) [EntityCollection#collectionChanged](#) [↗](#) 在引发到的相应调用之前， 防止引发事件 [EntityCollection#resumeEvents](#) [↗](#) ，此时将引发涵盖所有已暂停操作的单个事件。这允许有效地添加和删除许多项目。只要有相应的调用，就可以安全地多次调用此函数 [EntityCollection#resumeEvents](#) [↗](#) 。

DataSource

说到dataSourceDisplay，或许初学者会有点儿陌生。但是说到viewer.entities，可能大家都很熟悉了。实际上我们平常通过viewer.entities加入到场景中的各种对象，等同于加入到dataSourceDisplay当中。

dataSourceDisplay实际上内部管理着一堆dataSource对象，其中有一个比较特殊的dataSource，名字叫defaultDataSource。它的内部和其他类型的dataSource也很相似，都是由一堆entity组成的entities。entity代表一个实体。我们平时使用的viewer.entities，其实只是一个快捷方式。它真正调用的是

```
dataSourceDisplay.defaultDataSource.entities
```

。

我们通过viewer.entities增加到场景中的实体，实际上是由dataSourceDisplay来管理的。

defaultDataSource相当于Cesium为我们内置的一个dataSource，不需要我们手动创建，只需要调用viewer.entities直接加载三维实体就好。

参考

- <https://zhuanlan.zhihu.com/p/80904975> 

Scene

Scene是用来管理三维场景的各种对象实体的核心类。

globe用来表示整个地球的表皮，地球表皮的绘制需要两样东西，地形高程和影像数据。Cesium的地形高程数据只能有一套，而影像数据可以由多层，多层可以相互叠加。

primitives、groundPrimitives则是表示加入三维场景中的各种三维对象了。groundPrimitives用来表示贴地的三维对象。我们之前通过viewer.entities加入场景中的三维实体，大多会转化成primitives和groundPrimitives。

这里面有一个值得注意的问题：经常有开发者会调用 `scene.primitives.removeAll()` 来清空所有三维场景对象。这个操作是破坏性的。因为viewer.entities可以自己管理和自身相关的primitive的，也就是它会自动调用scene.primitives的 `add` 和 `remove` 方法，来进行primitive的增删操作。然而此时因为 `removeAll` 的操作，却也被强制删掉了，从而导致viewer.entities失效。 `removeAll` 并非不能用，我们在接下来的Primitive类中论述。

最后剩下的就是一堆地球周边的环境对象了，比如 `skyBox`（用来表示星空）、`skyAtmosphere`（用来表示大气）、`sun`（表示太阳）、`moon`（表示月亮）等等。

- <https://zhuanlan.zhihu.com/p/80904975> 

Cesium的Property机制



参考

- [Cesium的Property机制总结](#) 

Cesium3DTileset 让场景更细致更真实

3d tile特点

- 3d tiles的特点 <https://cesium.com/blog/2015/08/10/introducing-3d-tiles/>
- 协议完全开放：任何组织机构都可以用此标准来定义自己的数据。
- 渐进加载和渲染：这是3dtiles的主要目的，采用HLOD技术，保证只加载和渲染和当前精度匹配的数据。
- 面向三维空间：定义在三维空间中，不仅仅是点、线、面等常规二维数据
- 可交互：支持鼠标选择和高亮
- 样式可配置：根据对象属性修改对象的显示样式。
- 更强的适应性：空间索引不仅仅支持常规四叉树，可以根据数据内容动态构建索引树。
- 更强的灵活性：动态调整数据加载精度
- 更广泛的数据支持：点云(points)、三维模型(b3dm,i3dm)、甚至地形、矢量(vctr)都可以用3d tiles格式定义。
- 精度：使用矩阵偏移解决大坐标值的漂移问题
- 实时的：支持动态数据

Cesium3DTile 类

点云(pnts)、三维模型(b3dm,i3dm)、甚至地形、矢量(vctr)都可以用3d tiles格式定义。

Cesium3DTileset 类

举例：

```
var tileset = scene.primitives.add(new Cesium.Cesium3DTileset({
    url : 'http://localhost:8002/tilesets/Seattle/tileset.json'
}));
viewer.scene.primitives.add(tileset);
viewer.flyTo(tileset);
```

js

高度调整

```
//readyPromise 异步
tileset.readyPromise.then(function (argument) {
    var heightOffset = 20.0;//高度 抬升
    var boundingSphere = tileset.boundingSphere;//包围球
    var cartographic = Cesium.Cartographic.fromCartesian(boundingSphere.center);//包围球中心点
    var surface = Cesium.Cartesian3.fromRadians(cartographic.longitude, cartographic.latitude, 0);
    var offset = Cesium.Cartesian3.fromRadians(cartographic.longitude, cartographic.latitude, heightOffset);
    var translation = Cesium.Cartesian3.subtract(offset, surface, new Cesium.Cartesian3());
    tileset.modelMatrix = Cesium.Matrix4.fromTranslation(translation);
});
```

```
//经纬度高度 一起调整 仅限原数据中 root.transform 使用这种方式计算的
tileset.readyPromise.then(function (argument) {
    var position = Cesium.Cartesian3.fromDegrees(106.540585, 29.558622, 20);
    //中心位置调整
    var mat = Cesium.Transforms.eastNorthUpToFixedFrame(position);
    tileset._root.transform = mat;
});
```

Cesium3DTileStyle [🔗](#) 类

举例：

```
tileset.style = new Cesium.Cesium3DTileStyle({
    color : {
        conditions : [
            ['${Height} >= 100', 'color("purple", 0.5)'],
            ['${Height} >= 50', 'color("red")'],
            ['true', 'color("blue")']
        ]
    },
    show : '${Height} > 0',
    meta : {
        description : '"Building id ${id} has height ${Height}."'
    }
});
```

参考

- [CESIUM3DTITLE 调整位置](#) [🔗](#)
- [Cesium学习笔记（五）：3D 模型](#) [🔗](#)

Primitive-API 性能好，才是真的好

GeometryInstance类

Geometry类

Fabric 玩点高级

Appearance类

Material类

ParticleSystem 粒子系统

什么是粒子系统？

粒子系统是一种图形技术，可以模拟复杂的基于物理的效果。粒子系统是小图像的集合，当一起查看时，它们会形成更复杂的“模糊”对象，例如火，烟，天气或烟火。通过使用诸如初始位置，速度和寿命之类的属性指定单个粒子的行为，可以控制这些复杂的效果。

cesium 中粒子效果相关：

- [Particle](#) 粒子系统发射的粒子。
- [ParticleBurst](#) 表示在系统生命周期中给定时间的粒子系统中的粒子爆发。
- [ParticleEmitter](#) 一个从ParticleSystem初始化Particle的对象。此类型描述一个接口，不能直接实例化。
- [ParticleSystem](#) 粒子系统管理粒子集合的更新和显示。

一个基本粒子系统代码：

```
var particleSystem = viewer.scene.primitives.add(new Cesium.ParticleSystem({  
  image : '../SampleData/smoke.png', // 图片资源  
  imageSize : new Cesium.Cartesian2(20, 20), // 图像大小  
  startScale : 1.0, // 开始缩放比  
  endScale : 4.0, // 死亡时缩放比  
  particleLife : 1.0,  
  speed : 5.0, // 速度  
  emitter : new Cesium.CircleEmitter(0.5), // 发射器 圆形粒子发射器  
  emissionRate : 5.0,  
  modelMatrix : entity.computeModelMatrix(viewer.clock.startTime, new Cesium.Matrix4()),  
  lifetime : 16.0 // 寿命  
}));
```

Particle 类

粒子系统发射的粒子。

构造函数

```
new Cesium.Particle(options)
```

可选参数 `options` :

名称	类型	默认值	描述
<code>mass</code>	Number	<code>1.0</code>	<code>可选</code> 粒子的质量（以千克为单位）。
<code>position</code>	Cartesian3	<code>Cartesian3.ZERO</code>	<code>可选</code> 粒子在世界坐标中的初始位置。
<code>velocity</code>	Cartesian3	<code>Cartesian3.ZERO</code>	<code>可选</code> 世界坐标系中粒子的速度向量。
<code>life</code>	Number	<code>Number.MAX_VALUE</code>	<code>可选</code> 世界坐标系中粒子的速度向量。
<code>image</code>	Object		<code>可选</code> 用于广告牌的 URI，HTMLImageElement或 HTMLCanvasElement。
<code>startColor</code>	Color	<code>Color.WHITE</code>	<code>可选</code> 粒子诞生时的颜色。
<code>endColor</code>	Color	<code>Color.WHITE</code>	<code>可选</code> 粒子死亡时的颜色。
<code>startScale</code>	Number	<code>1.0</code>	<code>可选</code> 粒子诞生时的比例。
<code>endScale</code>	Number	<code>1.0</code>	<code>可选</code> 死亡时粒子的比例。
<code>imageSize</code>	Cartesian2	<code>new Cartesian2(1.0, 1.0)</code>	<code>可选</code> 尺寸，宽度乘高度，以像素为单位缩放粒子图像。

ParticleBurst 类

表示在系统生命周期中给定时间的粒子系统中的粒子爆发。控制每秒发射多少粒子，从而改变系统中粒子的密度。

构造函数

```
new Cesium.ParticleBurst(options)
```

js

参数 `options`：

名称	类型	默认值	描述
<code>time</code>	Number	<code>0.0</code>	<code>可选</code> 粒子系统生命周期开始后发生爆发的时间（以秒为单位）。
<code>minimum</code>	Number	<code>0.0</code>	<code>可选</code> 突发中发射的最小粒子数。
<code>maximum</code>	Number	<code>50.0</code>	<code>可选</code> 爆发中发射的最大粒子数。

指定一个 `burst` 对象数组，以在指定的时间发射粒子爆发（如上面的动画所示）。这会增加粒子系统的多样性或爆炸性。

将此属性添加到您的中 `particleSystem`。


```

bursts : [
  new Cesium.ParticleBurst({time : 5.0, minimum : 300, maximum : 500}),
  new Cesium.ParticleBurst({time : 10.0, minimum : 50, maximum : 100}),
  new Cesium.ParticleBurst({time : 15.0, minimum : 200, maximum : 300})
]

```

在给定的时间，这些脉冲将在最小和最大粒子之间发射。

ParticleEmitter 类

一个从 `ParticleSystem` 初始化 `Particle` 的对象。此类型描述一个接口，不能直接实例化。

当粒子诞生时，其初始位置和速度矢量受的控制 `ParticleEmitter`。发射器每秒会产生由 `emissionRate` 参数指定的一定数量的粒子，并根据发射器类型使用随机速度进行初始化。

对应 `ParticleSystem` 的 `emitter`：

```

var fireSystem = new Cesium.ParticleSystem({
  image: '../assets/img/fire.png',
  emitter: new Cesium.ConeEmitter(Cesium.Math.toRadians(20.0)), // 粒子发射器
  imageSize: new Cesium.Cartesian2(40, 40),
  emissionRate: 15.0,
  lifetime: 16.0,
  modelMatrix: computeModelMatrix(modelEntity, Cesium.JulianDate.now()), // 位置
});
scene.primitives.add(fireSystem);

```

内置的几个发射器对象：

- **BoxEmitter** 在箱子内发射粒子的粒子发射器。粒子将随机放置在盒子中，并且具有从盒子中心发出的初始速度。在一个盒子内随机采样的位置初始化粒子，并将它们从六个盒子面之一中引出。它接受一个 `Cartesian3` 参数，该参数指定盒子的宽度，高度和深度尺寸。

参数：

- `dimensions` : `Cartesian3` 盒子的宽度，高度和深度尺寸。正方体盒子

```

// 默认值
new Cartesian3(1.0, 1.0, 1.0)

```

- **CircleEmitter** 从圆形发射粒子的粒子发射器。粒子将定位在一个圆内，并且具有沿z向量移动的初始速度。

参数：

- `radius` : Number 圆的半径，以米为单位。

```

// 默认值 1.0
new Cesium.CircleEmitter(1.0)

```

- **ConeEmitter** 在圆锥体内发射粒子的粒子发射器。粒子将位于圆锥体的尖端，并且初始速度朝向底部。

参数：

- `angle` :Number 圆锥角（以弧度为单位）。

```
//默认值 30度角对应弧度
new Cesium.ConeEmitter(Cesium.Math.toRadians(30.0))
```

- **SphereEmitter** 在球体内发射粒子的粒子发射器。粒子将随机放置在球体内，并且具有从球体中心发出的初始速度。

参数：

- `radius` :Number 球体的半径，以米为单位。默认1.0

```
//默认值1.0
new Cesium.SphereEmitter(1.0)
```

ParticleSystem 类

粒子系统管理粒子集合的更新和显示。

粒子发射速率

`emissionRate` 属性控制每秒生成多少个粒子，用来调整粒子密度。值越大每秒产生粒子数越多，粒子越密集。

```
new Cesium.ParticleSystem({
  emissionRate: 15.0,
  //...
});
```

粒子/粒子系统的生命周期

一些参数控制了粒子系统的 **生命周期**，默认粒子系统一直运行。设置 `lifetime` 属性控制粒子的持续时间，同时需要设置 `loop` 属性为 `false`。 `loop` 设为 `true`，一直循环执行下去。比如设定一个粒子系统运行5秒，5秒后粒子效果结束：

```
new Cesium.ParticleSystem({
  lifetime: 5.0,
  loop: false,
  //...
});
```

为了每个粒子都有一个随机生命周期，我们可以设置 `minimumParticleLife` 和 `maximumParticleLife`。比如下面的代码设置了粒子生命周期在5秒和10秒之间：

```
new Cesium.ParticleSystem({
  minimumParticleLife: 5.0,
  maximumParticleLife: 10.0,
  //particleLife: 1.0,
  //...
});
```

设置 `particleLife` 属性为5.0 表示设置每个粒子的生命周期是5秒。如果设置 `particleLife` , `minimumParticleLife` 和 `maximumParticleLife` 将失效。

粒子样式

贴图(image):

纹理贴图。为粒子 `Particle` 换个皮肤。

```
new Cesium.ParticleSystem({
  image: '../assets/img/fire.png',
  //...
});
```

颜色 ([Color](#))

除了设定image属性来控制粒子的纹理外，还可以设定一个颜色值，这个值可以在粒子的生命周期内变化。

下面代码使火焰粒子产生的时候是淡红色，消亡的时候是半透明黄色。

```
new Cesium.ParticleSystem({
  //color:Cesium.Color.RED,
  startColor: Cesium.Color.YELLOW.withAlpha(0.5),
  endColor: Cesium.Color.RED.withAlpha(0.7),
  //...
})
```

大小 (Size)

通常粒子大小通过 `imageSize` 属性控制。如果想设置一个随机大小，每个粒子的宽度在 `minimumImageSize.x` 和 `maximumImageSize.x` 之间随机，高度在 `minimumImageSize.y` 和 `maximumImageSize.y` 之间随机，单位为像素。

```
new Cesium.ParticleSystem({
  minimumImageSize : new Cesium.Cartesian2(30.0, 30.0),
  maximumImageSize : new Cesium.Cartesian2(60.0, 60.0),
  //...
});
```

和颜色一样，粒子大小的倍率在粒子整个生命周期内，会在 `startScale` 和 `endScale` 属性之间插值。这个会导致你的粒子随着时间变大或者缩小。

```
new Cesium.ParticleSystem({
  startScale: 1.0,
  endScale: 4.0,
  //...
});
```

运行速度(Speed)

发射器控制了粒子的位置和方向，速度通过 `speed` 参数或者 `minimumSpeed` 和 `maximumSpeed` 参数来控制。

速度越大相同生命周期内运动越远。

```
new Cesium.ParticleSystem({
  //speed: 5.0,
  minimumSpeed: 5.0,
  maximumSpeed: 10.0,
  //...
});
```

如果设置 `speed` 将覆盖 `minimumSpeed` 和 `maximumSpeed`。

位置

粒子系统使用两个转换 [矩阵](#) 来定位:

- `modelMatrix` : 把粒子系统从模型坐标系转到世界坐标系。
- `emitterModelMatrix` : 在粒子系统的局部坐标系内变换粒子发射器。

我们提供两个属性也是为了方便，当然可以仅仅设置一个，把另一个设置为单位矩阵。为了学习创建这个矩阵，我们尝试把我们的粒子系统相对另一个 `entity`。

```
new Cesium.ParticleSystem({
  //粒子发生位置,相对于modelEntity
  modelMatrix: computeModelMatrix(modelEntity, Cesium.JulianDate.now()),
  //...
});
```

颗粒质量 (mass)

- `mass` 以千克为单位设置最小和最大颗粒质量。默认1.0kg
- `minimumMass` 设置粒子质量的最小范围（以千克为单位）。粒子的实际质量将被选择为高于此值的随机量。
- `maximumMass` 以千克为单位设置最大粒子质量。粒子的实际质量将选择为低于此值的随机量。

方法 (Methods)

- `destroy()` 销毁
- `isDestroyed()` → Boolean 是否被销毁

事件（Events）

完成事件（complete）

当粒子系统达到其生命周期尽头时触发事件。

更新回调（UpdateCallback [🔗](#)）

为了提升仿真效果，粒子系统有一个更新函数。这个是个手动更新器，比如对每个粒子模拟重力或者风力的影响，或者除了线性插值之外的颜色插值方式等等。每个粒子系统在仿真过程中，都会调用更新回调函数来修改粒子的属性。回调函数传过两个参数，一个是粒子本身，另一个是仿真时间步长。大部分物理效果都会修改速率向量来改变方向或者速度。下面是一个粒子响应重力的示例代码：

```
var gravityScratch = new Cesium.Cartesian3();  
/**  
 * 用于在每个时间步修改粒子属性的函数。这可以包括力的修改，颜色，大小等。  
 * @param {Particle} particle 正在更新的粒子。  
 * @param {Number} dt 自上次更新以来的时间（以秒为单位）。  
 */  
function applyGravity(particle, dt) {  
    // 计算每个粒子的向上向量（相对地心）  
    var position = particle.position;  
    Cesium.Cartesian3.normalize(position, gravityScratch);  
    Cesium.Cartesian3.multiplyByScalar(gravityScratch, viewModel.gravity * dt, gravityScratch);  
    particle.velocity = Cesium.Cartesian3.add(particle.velocity, gravityScratch, particle.velocity);  
}
```

这个函数计算了一个重力方向，然后使用重力加速度（-9.8米每秒平方）去修改粒子的速度方向。

设置粒子系统的更新函数：

```
new Cesium.ParticleSystem({  
    updateCallback: applyGravity,  
    //...  
});
```

额外的天气效应

使用雾和大气效果来增强可视化效果，并匹配我们试图复制的天气类型。











`hueShift` 沿着颜色光谱改变颜色，`saturationShift` 改变了视觉实际需要的颜色与黑白的对比程度，`brightnessShift` 改变了颜色的生动程度。

雾密度 `density` 改变了地球上覆盖物与雾的颜色之间的不透明程度。雾的 `minimumBrightness` 用来使雾变暗。

```
scene.skyAtmosphere.hueShift = -0.8;  
scene.skyAtmosphere.saturationShift = -0.7;  
scene.skyAtmosphere.brightnessShift = -0.33;  
  
scene.fog.density = 0.001;  
scene.fog.minimumBrightness = 0.8;
```

[查看 示例](#) 

参考

- [\[官方\]Introduction to Particle Systems](#) 
- [\[官方\]Advanced Particle System Effects](#) 
- [\[官方示例\]Particle System 小车尾气](#) 
- [\[官方示例\]Particle System Fireworks 烟花](#) 
- [\[官方示例\]天气](#) 
- [\[官方示例\]拖尾](#) 
- [Cesium官方教程9--粒子系统](#) 
- [CESIUM实时目标跟踪最新特效教程系列2—粒子系统（实时发射波束跟踪目标）](#) 
- [CESIUM粒子特效笔记](#) 
- [CESIUM中级教程9 - ADVANCED PARTICLE SYSTEM EFFECTS 高级粒子系统效应](#) 

鼠标交互

拾取技术（picking）能够根据一个屏幕上的像素位置返回三维场景中的对象信息。

有好几种拾取：

- `Scene.pick` [🔗](#) : 返回窗口坐标对应的图元的第一个对象。
- `Scene.drillPick` [🔗](#) : 返回窗口坐标对应的所有对象列表。
- `Globe.pick` [🔗](#) : 返回一条射线和地形的相交位置点。

官方示例:

- [拾取示例](#) [🔗](#)
- [3D Tiles 对象拾取](#) [🔗](#)

因为我们想实现鼠标滑过的高亮效果，首先需要创建一个鼠标事件处理器。 `ScreenSpaceEventHandler` [🔗](#) 是可以处理一系列的用户输入事件的处理器。 `ScreenSpaceEventHandler.setInputAction()` [🔗](#) 监听某类型的用户输入事件 -- `ScreenSpaceEventType` [🔗](#) 用户输入事件类型，做为一个参数传递过去。这里我们设置一个回调函数来接受鼠标移动事件:

```
var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function(movement) {
    //...
}, Cesium.ScreenSpaceEventType.MOUSE_MOVE); // 鼠标移动事件
```

举例,拾取构件查看属性:

```
var scene = viewer.scene;
// 添加一个左键点击事件
viewer.screenSpaceEventHandler.setInputAction(function (movement) {
    // 拾取
    var feature = scene.pick(movement.position);
    if (feature instanceof Cesium.Cesium3DTileFeature) {
        // 查看拾取到构件属性
        var propertyNames = feature.getPropertyNames();
        var length = propertyNames.length;
        for (var i = 0; i < length; ++i) {
            var propertyName = propertyNames[i];
            console.log(propertyName + ': ' + feature.getProperty(propertyName));
        }
    }
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

[查看 示例](#) [🔗](#)

Cesium获取鼠标点击位置

屏幕坐标（鼠标点击位置距离canvas左上角的像素值）

通过: `movement.position` 获取

```
var viewer = new Cesium.Viewer('cesiumContainer');

var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (movement) {
    console.log(movement.position);
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

世界坐标（Cartesian3）

通过: `Camera.pickEllipsoid(windowPosition, ellipsoid, result) → Cartesian3` 拾取, 可以获取当前点击视线与椭球面相交处的坐标, 其中ellipsoid是当前地球使用的椭球对象:

`viewer.scene.globe.ellipsoid`。

```
var viewer = new Cesium.Viewer('cesiumContainer');

var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (movement) {
    var position = viewer.scene.camera.pickEllipsoid(movement.position, viewer.scene.globe.ellipsoid);
    console.log(position);
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

场景坐标

通过: `viewer.scene.pickPosition(movement.position)` 获取, 根据窗口坐标, 从场景的深度缓冲区中拾取相应的位置, 返回笛卡尔坐标。

```
var viewer = new Cesium.Viewer('cesiumContainer');

var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (movement) {
    var position = viewer.scene.pickPosition(movement.position);
    console.log(position);
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

地标坐标

通过: `viewer.scene.globe.pick(ray, scene, result)` 获取, 可以获取点击处地球表面的世界坐标, 不包括模型、倾斜摄影表面。其中`ray=viewer.camera.getPickRay(movement.position)`。


```
js
var viewer = new Cesium.Viewer('cesiumContainer');

var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (movement) {
    var ray=viewer.camera.getPickRay(movement.position);
    var position = viewer.scene.globe.pick(ray, viewer.scene);
    console.log(position);
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

限制鼠标的视图控制

```
js
// 禁用放大缩小和自由旋转视图
viewer.scene.screenSpaceCameraController.enableZoom = false;
viewer.scene.screenSpaceCameraController.enableTilt = false;
```

修改视图默认鼠标操作方式

```
js
// 修改默认的鼠标视图控制方式。
viewer.scene.screenSpaceCameraController.zoomEventTypes = [Cesium.CameraEventType.WHEEL, Cesium.CameraEventType.PINCH];
viewer.scene.screenSpaceCameraController.tiltEventTypes = [Cesium.CameraEventType.PINCH, Cesium.CameraEventType.WHEEL];
```

添加自定义鼠标事件（1），实现点击、双击、右键点击等事件

```
js
// 添加鼠标点击事件。
// 可以通过Cesium.ScreenSpaceEventType类实现不同的触发条件
viewer.screenSpaceEventHandler.setInputAction(function(click) {
    // 处理鼠标按下事件，获取鼠标当前位置
    var feature = viewer.scene.pick(click.position);
    //选中某模型
    if (feature && feature instanceof Cesium.Cesium3DTileFeature) {
        console.log(feature);
    }
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);



// 移除事件
viewer.screenSpaceEventHandler.removeInputAction(Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

添加自定义鼠标事件（2），实现点击、双击、右键点击等事件。本质来讲和上面是一样的，只是写法不同。

```
// 添加事件
// 可以通过Cesium.ScreenSpaceEventType类实现不同的触发条件
var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function(click){
    console.log('左键单击事件: ',click.position);
},Cesium.ScreenSpaceEventType.LEFT_CLICK);

// 移除事件
handler.removeInputAction(Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

参考

- [Cesium获取鼠标点击位置（PickPosition）](#) 
- [Cesium鼠标事件汇总](#) 

自定义气泡

js

```
var info = document.getElementById("info");
function showInfo(entity) {
    info.innerHTML = entity.name + '<br>' + entity.description;
    info.style.display = 'block';
}
function hideInfo() {
    info.style.display = 'none';
}
var scene = viewer.scene;
var pickPosition;
viewer.screenSpaceEventHandler.setInputAction(function onLeftClick(movement) {
    var picked = scene.pick(movement.position);
    if (picked) {
        if (picked.id == model) {

            pickPosition = scene.pickPosition(movement.position);
            showInfo(model);

        }
    } else {
        hideInfo();
    }
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
var removeChanged = scene.postRender.addEventListener(function (percentage) {

    //转换到屏幕坐标
    if (pickPosition && info.style.display == 'block') {
        var winpos = scene.cartesianToCanvasCoordinates(pickPosition);
        if (winpos) {

            info.style.left = (winpos.x - 100 / 2).toFixed(0) + 'px';
            info.style.top = (winpos.y - 100).toFixed(0) + 'px';

        }
    }
});
```

[在线预览](#) 

Cesium 中的pick

在cesium中，想获取不同的对象，需要通过pick方法来进行拾取，但是Cesium中有多种pick的方法，例如：

- scene中有pick、pickPosition、及drillPick等
- camera中有getPickRay、pickEllipsoid等
- globe中有pick

先来分类说一下各个pick的作用：

scene中（一般用来获取entity对象）：

- pick: `scene.pick` 可以通过此方法获取到pick对象，通过pick.id即可拾取当前的entity对象，也可以获取Cesium3DTileFeature对象；
- drillPick: `scene.drillPick(click.position)` 是从当前鼠标点击位置获取entity的集合，然后通过for循环可以获取当前坐标下的所有entity；
- pickPosition: 通过 `viewer.scene.pickPosition(movement.position)` 获取，可以获取场中任意点击处的对应的世界坐标。（高程不精确）

pick与drillPick的区别：pick只可获取一个entity对象（如该位置存在多个entity，哪怕面点线不在同一高度，面entity都可能会盖住点线entity），但drillPick可获取当前坐标下的多个对象；

camera和globe中的pick：

这两个里面的pick一般搭配使用，通过camera中的getPickRay获取ray（射线），然后通过globe中的pick方法，获取世界坐标，如下面的地形坐标的获取：

1. 通过pick进行地形上的坐标的获取

这个是常用的方法，当你想获取当前鼠标位置的三维坐标时，经常使用到这个方法：

第一步：通过camera的getPickRay，将当前的屏幕坐标转为ray（射线）；

```
viewer.camera.getPickRay(windowCoordinates);
```

js

第二步：找出ray和地形的交点，即可求出三维世界坐标

```
globe.pick(ray, scene);
```

js

2. 通过pick获取entity

```
handler.setInputAction(function (movement) {  
    var pick = viewer.scene.pick(movement.endPosition); //获取的pick对象  
    var pickedEntity = Cesium.defined(pick) ? pick.id : undefined; //pick.id即为entity  
}, Cesium.ScreenSpaceEventType.MOUSE_MOVE);
```

js

参考

- [Cesium 中的pick讲解](#) 

Cesium 各高度的获取

1. 地形高度的获取

- 方法a: 通过事件获取到像素坐标，然后转为世界坐标，再求地形高度

```
var handler = new Cesium.ScreenSpaceEventHandler(scene.canvas);
handler.setInputAction(function(evt) {
    var ray=viewer.camera.getPickRay(evt.position);
    var cartesian=viewer.scene.globe.pick(ray,viewer.scene);
    var cartographic=Cesium.Cartographic.fromCartesian(cartesian);
    var height = cartographic.height的值为地形高度。
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

- 方法b: 先转为经纬度，通过viewer.scene.globe.getHeight(cartographic) 直接求地形高度，cartographic.height可以为0
- 方法c: Cesium.sampleTerrain 获取简单地形高度；（异步）
- 方法d: Cesium.sampleTerrainMostDetailed 获取精确地形高度；（异步）

地形高度获取小结:如果你是在事件里获取可用a方法，如果你要是想实时获取可用b，其它情形可用c

2. 模型表面高度的获取

```
var handler = new Cesium.ScreenSpaceEventHandler(scene.canvas);
handler.setInputAction(function(evt) {
    var scene = viewer.scene;
    if (scene.mode !== Cesium.SceneMode.MORPHING) {
        var pickedObject = scene.pick(evt.position);
        if (scene.pickPositionSupported && Cesium.defined(pickedObject) && pickedObject.node) {
            var cartesian = viewer.scene.pickPosition(evt.position);
            if (Cesium.defined(cartesian)) {
                var cartographic = Cesium.Cartographic.fromCartesian(cartesian);
                var lng = Cesium.Math.toDegrees(cartographic.longitude);
                var lat = Cesium.Math.toDegrees(cartographic.latitude);
                var height = cartographic.height;//模型高度
                mapPosition={x:lng,y:lat,z:height}
            }
        }
    }
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

参考

- [Cesium 各高度的获取](#) 

dae转gltf/bgltf

使用COLLADA2GLTF-bin.exe把dae数据转为gltf格式。通过cmd进入到colladaToglft.exe所在的文件夹，使用如下命令：

```
COLLADA2GLTF-bin.exe -f daePath -e
```

sh

或者

```
COLLADA2GLTF-bin.exe -f daePath -o gltfPath
```

sh

这里的daePath为dae文件的全路径，比如 `C:\Test.dae`

gltf的转换工具可以在 <https://github.com/KhronosGroup/glTF/wiki/Converter-builds> 获取——
COLLADA2GLTF-bin.exe

参考：

- [\[github\]COLLADA to glTF converter](#)
- [dae转gltf](#)
- [Cesium中导入三维模型方法（dae到glft/bglft）](#)

跨域问题

```
Access to image at 'http://localhost:8082/error/cesiumla_cros.html:1
b.png' from origin 'http://localhost:8081' has been blocked by CORS
policy: No 'Access-Control-Allow-Origin' header is present on the
requested resource.
```

- 症状：浏览器输出CORS policy错误，所加载的对象没显示
- 问题定位：web前端开发，与cesium无关
- 问题复现：页面引用的一些不在当前页面地址的资源
- 问题解决：
 - 跨域问题的终极解决方法在服务端
 - 若服务端代码可改：添加跨域头
 - 若服务端不可控：添加代理服务器



模型漂移

- 症状：随着视角旋转，模型并不能在中心位置

- 问题定位：图形学，与cesium有关

- 问题复现：模型和地形的相对高度不一致

- 问题解决：

- 1. `viewer.scene.globe.depthTestAgainstTerrain=true;`//打开地形深度检测
- 2. 调节对象高度；
- 3. `viewer.scene.globe.depthTestAgainstTerrain=false;`//关闭地形深度检测

底图偏移

- 症状：换了底图之后，影像和实体有较大偏差
- 问题定位：GIS数据源，与cesium有关
- 问题复现：国内公开访问的底图（除天地图）都有火星偏移
- 问题解决：
 - 不要采用有偏移的底图数据（可以使用天地图底图）
 - 或者实时编译修正