简介及环境配置

> **Cesium是什么？**
>
> Cesium ['si:zɪəm]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的外部和内部，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、公告牌、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、公告牌的聚集效果。

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 。

```sh
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
```

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

```sh
npm init
```

配置生成 `package.json` :

```sh
Administrator@WIN-IQLNPTL95TO MINGW64 /e/workspace/Gis/sogrey/Cesium-start-Example (master)
$ 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

```sh
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

一个圆润的地球引入眼帘，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:

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

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。

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

默认，场景能够处理鼠标和触控输入事件，如相机控制:

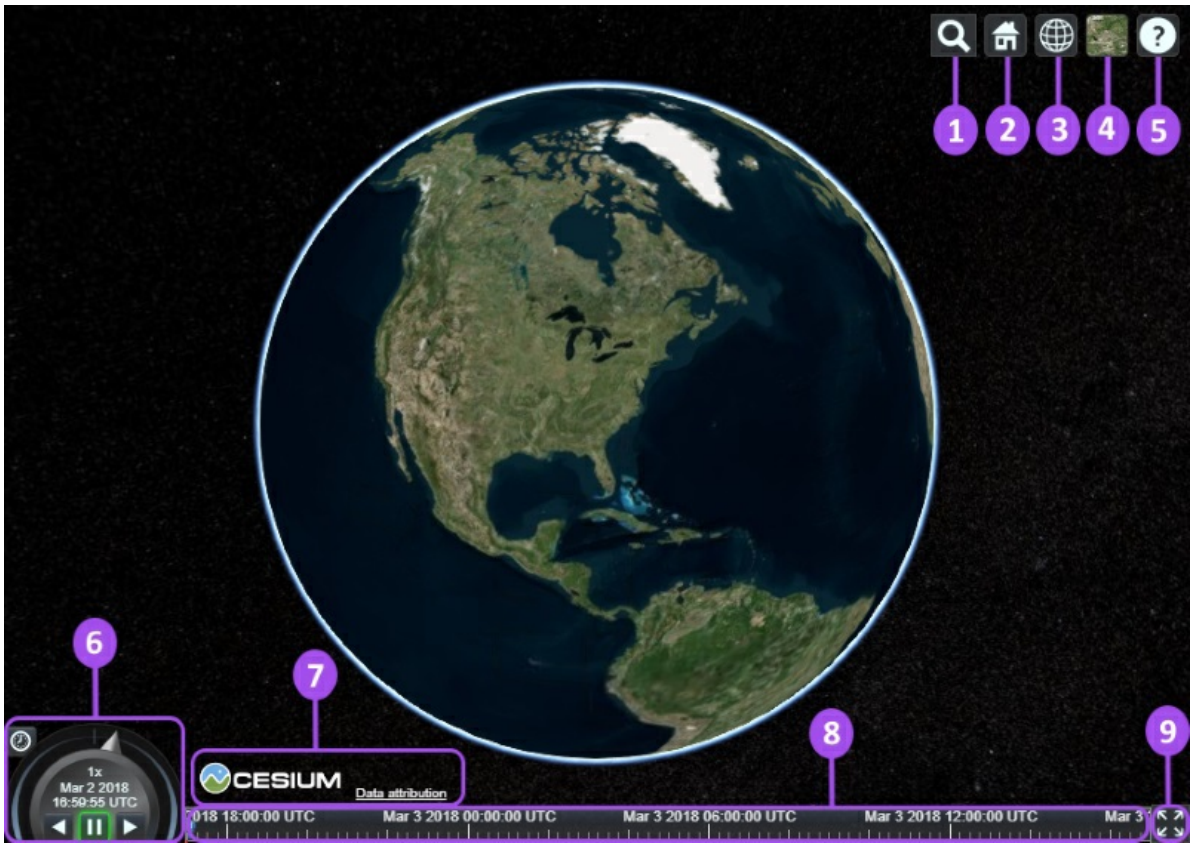
- 鼠标左键单击和拖动 - 在地球表面移动相机.
- 鼠标右键单击和拖动 - 放大、缩小(相机拉近或拉远)
- 鼠标中键滚轮 - 放大、缩小
- 鼠标中键单击和拖动 - 以地球表面某个点旋转相机

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,
 geocoder:false,//右上角搜索
 homeButton:false,
 infoBox:false,
 scene3DOnly:false,//仅仅显示3d,可隐藏右上角2d和3d按钮
 selectionoIndicatr:false,
 timeline:false,//最下面时间轴
 navigationHelpButton:false,//右上角帮助按钮
 navigationInstructionsInitiallyVisibl: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._creditContair
```

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 :

```js
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:

```js
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&
    //layer: "tdtAnnoLayer",
    //style: "default",
    //format: "image/jpeg",
    //tileMatrixSetID: "GoogleMapsCompatible"
}));
```

高德街道底图:

```js
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}&
        //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&
    //layer: "tdtAnnoLayer",
    //style: "default",
    //format: "image/jpeg",
    //tileMatrixSetID: "GoogleMapsCompatible"
}));
```

更多影响地图查看 国家地理信息公共服务平台

配置地形

Viewer除了 `imageryProvider` 影响外，还有 `terrainProvider` 地形，默认为 EllipsoidTerrainProvider

举例：

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

```js
// 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。

```js
// 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。螺距是绕负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` 是的属性 `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](https://sandcastle.cesium.com/?src=Camera Tutorial.html&label=Tutorials)

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a `HeadingPitchRoll` 来定位和定向：

```js
// Create an initial camera view
var initialPosition = new Cesium.Cartesian3.fromDegrees(-73.998114468289017509, 40.6745128956466
var initialOrientation = new Cesium.HeadingPitchRoll.fromDegrees(7.1077496389876024807, -31.9872
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期，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，请查看我们的 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：

```js
// 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

- 长半轴：6378137.0
- 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 -> Cartographic.toCartesian : Cartesian3
- Cartesian3 -> Cartographic.fromCartesian : Cartographic

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。

cesium 坐标系中心点在地心原点。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

```js
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：

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

Viewer类-一切API的入口

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)、 pith (俯仰) 、 roll (翻滚)

图中 `g` 是重力方向与 `z` 相反。

Camera有几个常用API：

- setView(options) 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：

```js
// 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, -0.978629
        up : new Cesium.Cartesian3(-0.47934589305293746, -0.8553216253114552, 0.196602217911833
    }
});
```

查看北京城：

```js
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
    }
});
```

flyTo(options)

官方示例：

```js
// 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：

```js
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
    }
});
```

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

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

```js
//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

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：

```js
//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 -> Cartographic.toCartesian : Cartesian3
- Cartesian3 -> Cartographic.fromCartesian : Cartographic

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

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

属性：

名称	类型	描述
`id`	String	可选此对象的唯一标识符。如果未提供，则将生成GUID。
`name`	String	可选显示给用户的人类可读名称。它不必是唯一的。
`availability`	TimeIntervalCollection	可选与此对象关联的可用性（如果有）。
`show`	boolean	可选的布尔值，指示是否显示实体及其子级。
`description`	Property	可选的字符串属性，用于为此实体指定HTML描述。
`position`	PositionProperty	可选一个指定实体位置的属性。
`orientation	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	可选的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：

```js
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
    }
});
```

DataSource

Scene.pick

```js
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);
```

Property

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(pnts)、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：

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

高度调整

```js
//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);//包围求中心点
    var surface = Cesium.Cartesian3.fromRadians(cartographic.longitude, cartographic.latitude, 0
    var offset = Cesium.Cartesian3.fromRadians(cartographic.longitude, cartographic.latitude, he
    var translation = Cesium.Cartesian3.subtract(offset, surface, new Cesium.Cartesian3());
    tileset.modelMatrix = Cesium.Matrix4.fromTranslation(translation);
});
```

```js
//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：

```js
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}."'
    }
});
```

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

Particle类

ParticleEmitter类

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:

```js
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：

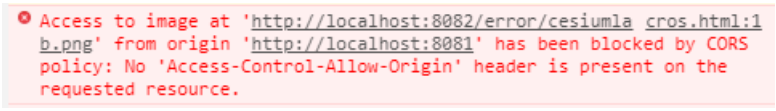
```js
var scene = viewer.scene;
//添加一个左键点击事件
viewer.screenSpaceEventHandler.setInputAction(function (movement) {
    //拾取
    var feature = scene.pick(movement.position);
    if (feature instanceof Cesium.Cesium3DTileFeature) {
        //查看拾取到构件属性
        var propertyNames = feature.getPropertyNames();
        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);
```

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

跨域问题



- 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