CesiumExampleCollection

学习合集：

WebGL & Cesium & Three_最凶残的小海豹的博客-CSDN博客

cesium entity 和primitive 绘制对象 primitive合并 但是有不同的外观

Gis - 随笔分类 - 3D入魔 - 博客园 (cnblogs.com)

FreeGIS.org

cesium_A873054267的博客-CSDN博客

sky..... cesium,webgl,web前端-CSDN博客

Cesium框架的示例集合

搭建环境

在这里我默认已经安装 nodejs 并配置好环境。我目前使用的版本是 v16.20.0

- 使用 vite 脚手架构建 vue3 项目。Vite 需要 Node.js 版本 >= 12.0.0。
    - 使用 $ pnpm create vite 指令安装
- 在 public 文件夹中创建 lib 文件夹，在其中拷贝一份 cesium 库文件。
- 替换 src 文件夹。

教程 - 在 Vue3+Ts 中引入 CesiumJS 的最佳实践@2023 - 知乎 (zhihu.com)

重要类

着重关心：地球，时间轴控件，各类实体以及他们的样式变化。对应到代码中就是

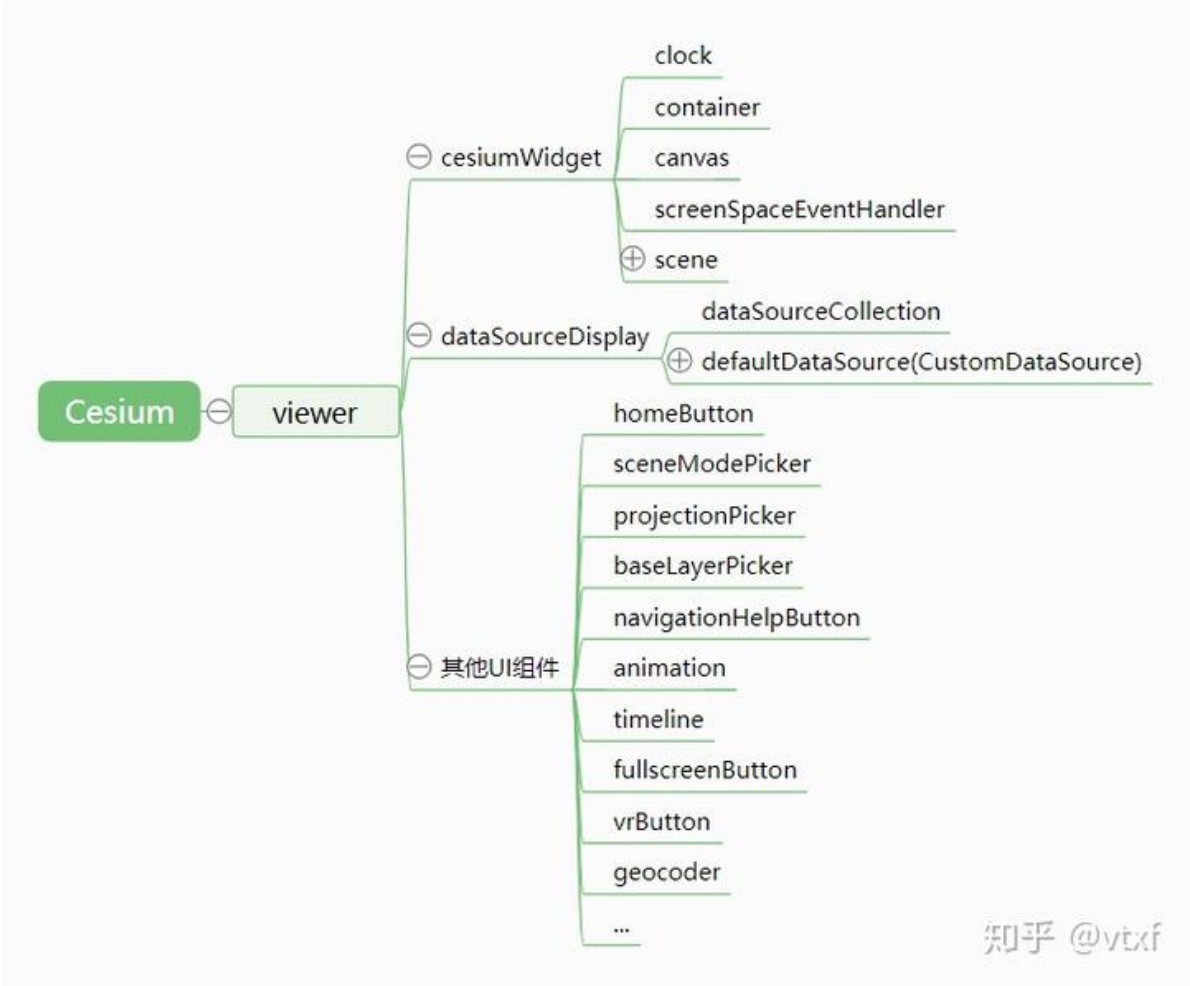
Viewer 目录

cesiumWidget

dataSourceDisplay

其他 UI 组件

但凡创建任何一个三维窗口可能都逃脱不了Viewer类的使用。Viewer基本上也就代表这一个Cesium的**三维窗口的所有**（有待理解）。因此我们首先介绍一些Viewer的组成。当我们使用 `var viewer = new Cesium.Viewer('cesiumContainer');` 创建一个三维窗口以后，它的内部是怎样组成的呢？请看下图：



可以看到viewer其实就是一堆UI的组合。

1. cesiumWidget

最顶上的**cesiumWidget**是核心的三维窗口所在。这里面不仅仅包含创建三维窗口所需要的canvas、还有scene用来管理三维场景中的所有三维对象。其中的一个异类可能就是用来表示时间的clock了，因为这玩意儿和窗口真没啥关系。

2. 其他UI组件

最下方的**其他UI组件**包含了各种按钮、时间轴等UI组件。这些UI组件都是Cesium为我们预先创建好的，可以直接进行交互调用的工具。它们纯粹就是div的堆叠，当然内部会调用一些改变三维场景的命令，从而影响着三维窗口的显示内容。

3. dataSourceDisplay

可以看到Viewer类几乎都是这些UI的组合，当然唯一例外的是中间的**dataSourceDisplay**，这是viewer向三维场景中添加三维对象的接口。刚才已经说了，scene是管理三维场景中的对象的。那么它和这个datasourceDisplay是否会有冲突？答案是否定的。datasourceDisplay所做的事情就是把外部数据资源，比如kml、geojson、各种内部的几何体转化成scene能识别的三维场景对象，再通过内部一些命令，加入到scene当中去的。所以可以说，dataSourceDisplay中管理的对象，实际上和scene中的一些对象是对应的关系。但是它再将三维场景对象加入scene中去以后，还是会继续管理这些对象。比如您在dataSourceDisplay中删除某些对象以后，scene当中的某些对象也会被删除掉。

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。这话怎么说呢？请看下图：

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`。

所以说嘛，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。

而其他dataSource就没有这么好运了，在Cesium的API文档中搜索可以发现：

这里的GeoJsonDataSource、KmlDataSource、CzmlDataSource相当于可以引用外部资源，然后自己内部**自动转换**成一个一个的entity，不需要我们做特殊的操作。当然我们也可以 借助返回的 Promise 来对这些 entity 做一些适当的 **修改** ，Cesium的Sandcastle中有相应的示例。

Entity表示一个实体对象，准确的讲，应该是一个可以随时间动态变化的实体对象。为什么这样说呢？Cesium为了让Entity能够赋予时间的动态特性，把其属性都仔细设计了一番，特别引入了Property这个类。比如position本来用经纬度表示一下就ok了，结果现在它被设计成Property 类型。好处是这个Property可以记录某某时间段在某个位置，然后另外一个时间段，则在另外一个位置。也就是说position这个Property已不单纯指表示某个位置了，被赋予了时间的动态特性，内部的结构可以很复杂，不同的时间在不同的位置。

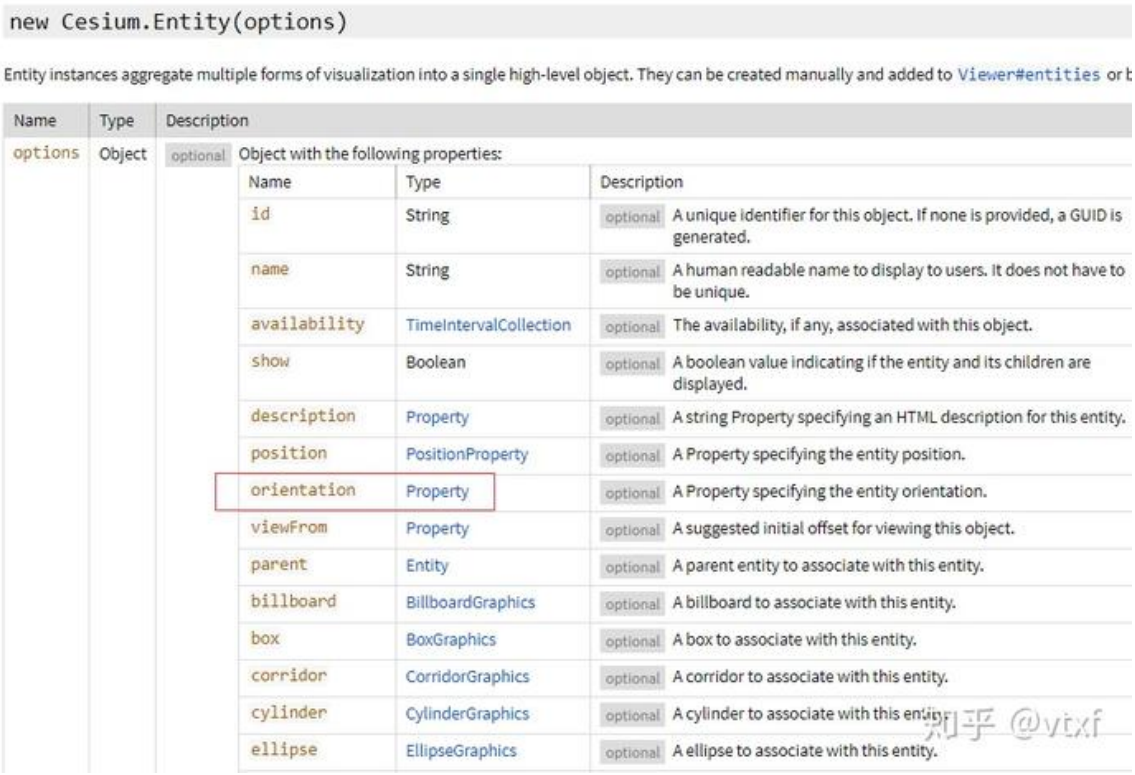
Entity还有一个其他的基本属性：

id表示唯一标识符。

name表示名字，可以不设置。

orientation表示实体的姿态变化（旋转方向）。 这个属性的内部是用四元数(Quaternion)表示的，Property中内部的基本属性类型是Quaternion，即某个时间段是这个Quaternion，另外一个时间段，又是另外一个Quaternion。Quaternion表示四元数，可能很多前端工程师并不熟悉，如果Cesium把它换成欧拉角，会让大家好理解一点。

另外一个需要吐槽的地方在于，orientation的类型是一个Property，并不能由此推断出这和个Property内部需要使用Quaternion作为基本类型来用。所以Cesium的API文档在这里是描述得不太清楚的。我也是在看Cesium的示例才知道orientation需要赋值为一个Quaternion对象。而且不止于此，很多其他Property属性也有着同样的问题，所以有时候只能看Cesium的源码才能了解该如何操作。



new Cesium.Entity(options)

Entity instances aggregate multiple forms of visualization into a single high-level object. They can be created manually and added to Viewer#entities or b

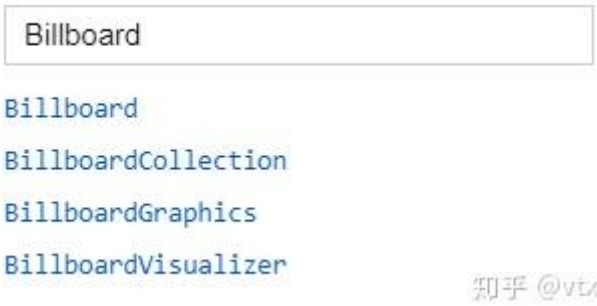
Name	Type	Description
options	Object	optional Object with the following properties:

Name	Type	Description
id	String	optional A unique identifier for this object. If none is provided, a GUID is generated.
name	String	optional A human readable name to display to users. It does not have to be unique.
availability	TimeIntervalCollection	optional The availability, if any, associated with this object.
show	Boolean	optional A boolean value indicating if the entity and its children are displayed.
description	Property	optional A string Property specifying an HTML description for this entity.
position	PositionProperty	optional A Property specifying the entity position.
orientation	Property	optional A Property specifying the entity orientation.
viewFrom	Property	optional A suggested initial offset for viewing this object.
parent	Entity	optional A parent entity to associate with this entity.
billboard	BillboardGraphics	optional A billboard to associate with this entity.
box	BoxGraphics	optional A box to associate with this entity.
corridor	CorridorGraphics	optional A corridor to associate with this entity.
cylinder	CylinderGraphics	optional A cylinder to associate with this entity.
ellipse	EllipseGraphics	optional A ellipse to associate with this entity.

Entity除了这些基本属性之外，还有很多几何图形类，比如billboard、box、corridor等等。这些属性的类型都是XXXGraphics。XXXGraphics内部仍然是由一堆Property类型的属性组成。

由此可见Entity从里到外，都是特别崇尚Property的设计。Property无处不在。之前我写过一篇文章叫Cesium的Property机制总结，如果您希望对Property有更多的了解，可以参考一下。

还有一个地方需要开发者特别注意：但凡遇到 XXXProperty、XXXGraphics 的类，就要明白，它一定是和Entity相关的。Cesium有很多名称类似的类，但是他们的使用场景差别很大，千万不要搞混了。



Billboard

Billboard

BillboardCollection

BillboardGraphics

BillboardVisualizer

比如，API文档中搜索Billboard就会发现有：Billboard、BillboardCollection、BillboardGrahpics等类。其中BillboardGrahpics就属于Entity API中使用的类，而Billboard、BillboardCollection这些不带有Propery后缀、Graphics后缀的，一般都是Primitive API的一员。二者是绝对不能混用的。

Scene

Scene也是我们使用Cesium时无法跳过的一个重要的类。

它是用来管理三维场景的各种对象实体的核心类。其中：

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。

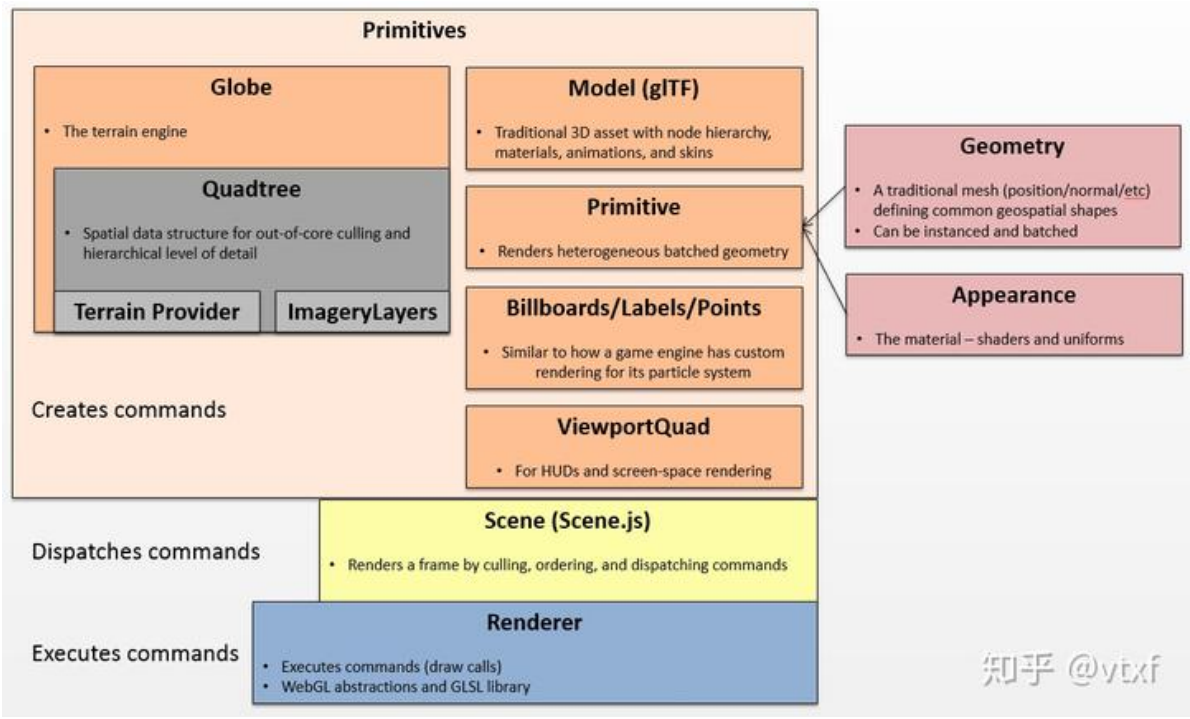
最后剩下的就是一堆地球周边的环境对象了，比如天空盒（用来表示星空）、skyAtmosphere（用来表示大气）、sun（表示太阳）、moon（表示月亮）等等。

Primitive & PrimitiveCollection

Primitive是Primitive API的核心类，它代表Cesium的三维场景中的一个基本绘制图元。

然而Cesium在这个地方搞得让人有点儿混淆：就是Cesium的**Primitive类**和**Primitive类型**并非一个东西。。哎。。实在是有一些拗口。。

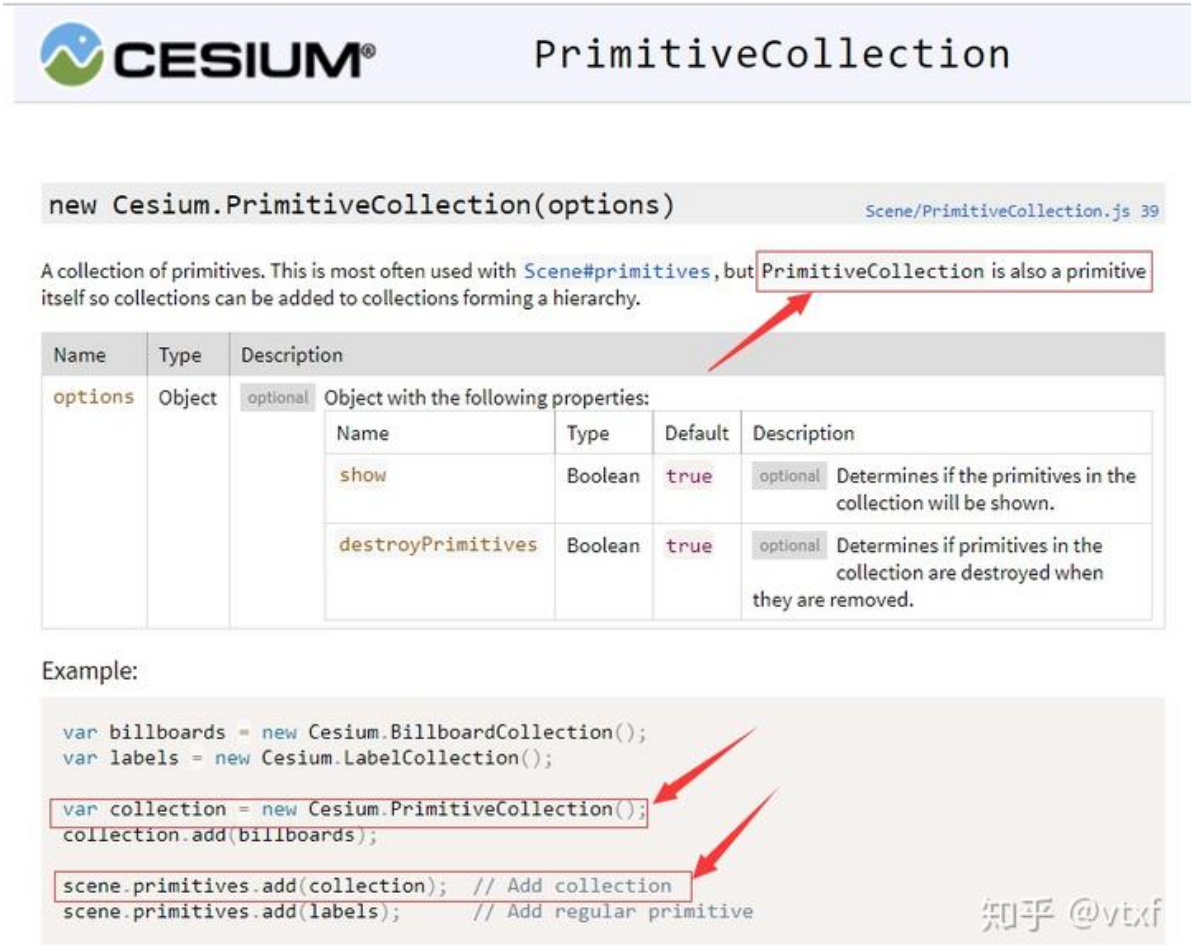
暂且我们把**Primitive类型**，称为图元类型。那么Cesium中属于图元类型的类有很多：比如Globe、Model、Primitive、Billboards、Labels、Points、ViewportQuad。注意：这里面的Primitive只是图元类型的一种而已！我是觉得Cesium的Primitive类这个名字取得很不好，很容易让人误解。如果改成CustomPrimitive（类似CustomDataSource）或许能更好地让人理解。不过Cesium就这样，又能咋地。。ヽ(￣Д￣)ﾉ



务必注意：Cesium用PrimitiveCollection类来作为图元类型的集合容器，**不仅仅是承载Cesium的Primitive类**，同时也可以承载Model、ViewportQuad类图元。

甚至我们可以自定义图元类型，存放到PrimitiveColloection当中去。我在github上开源了一些自定义Primitive类的方法和示例，如果有兴趣，也可以了解一下： https://github.com/cesiumlab/cesium-custom-primitive。

我们再来聊一下PrimitiveCollection这个类。这个类也很有意思，不仅表示一个图元类型的集合，本身也是一个**图元类型**！



Cesium的API文档里面早有说明，而且还给出了示例(注意示例中的scene.primitives也是一个PrimitiveCollection类型的对象)。于是，就产生了各种奇妙的可能了。我们可以利用PrimitiveCollection来构建一棵具有层级结构的场景树。

上图中，scene.primitives相当于场景树的根节点，我们在根节点上可以挂接各种Primitive对象，同时也可以挂接PrimitiveCollection对象，然后PrimitiveCollection对象下面又可以挂接N个不同的Primitive对象。

很自然，为了方便清空某个PrimitiveCollection对象下的所有对象，自然需要removeAll()这个方法了。

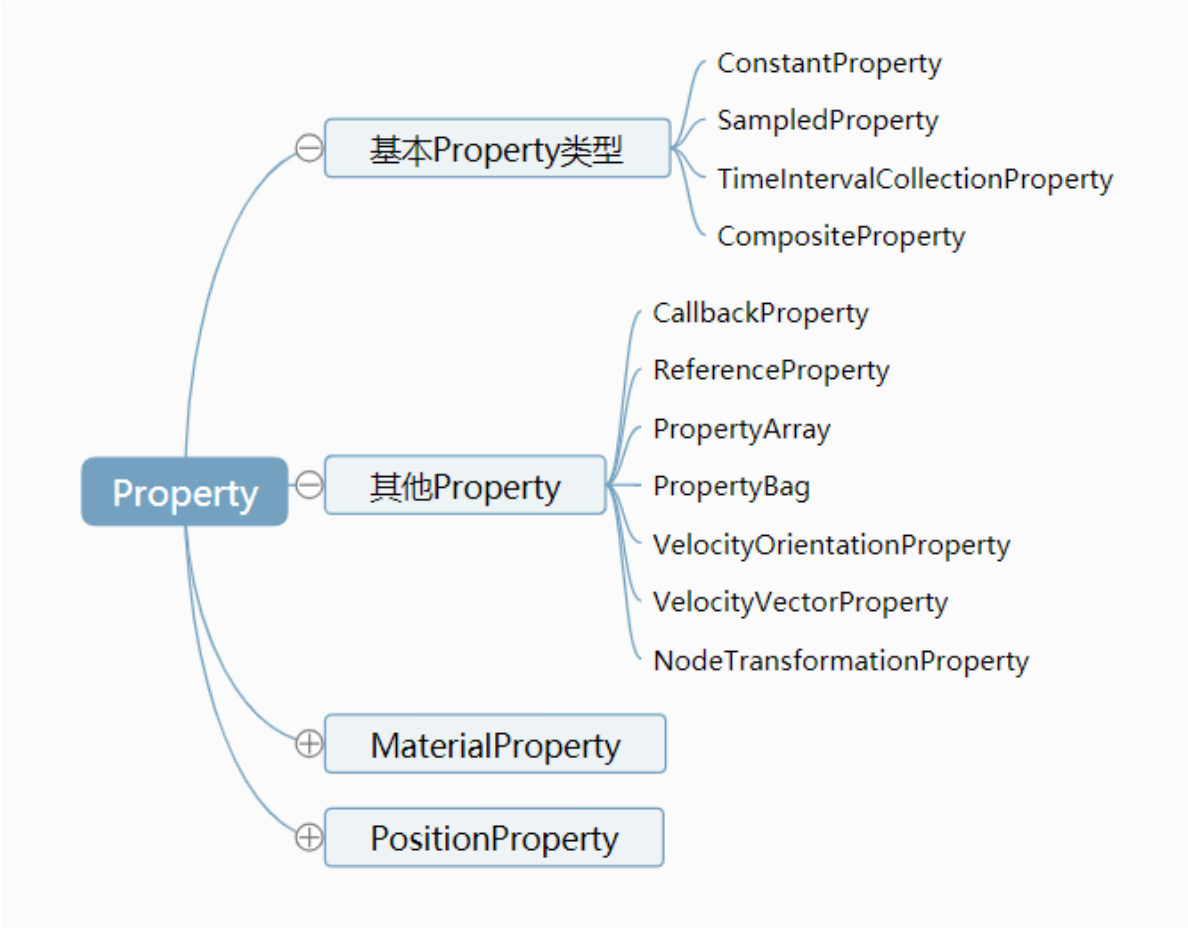
但是需要注意的是，这个方法可以用在任何一个自定义的PrimitiveCollection对象上，但是我们不能直接在scene.primities上使用。如前所述，因为viewer.entities会在scene.primitives上偷偷挂接一些它管理的primitive对象。如果我们直接调用scene.primitives.removeAll()，相当于把viewer.entities也给删除了。

所以removeAll()方法的使用是需要注意场合的。

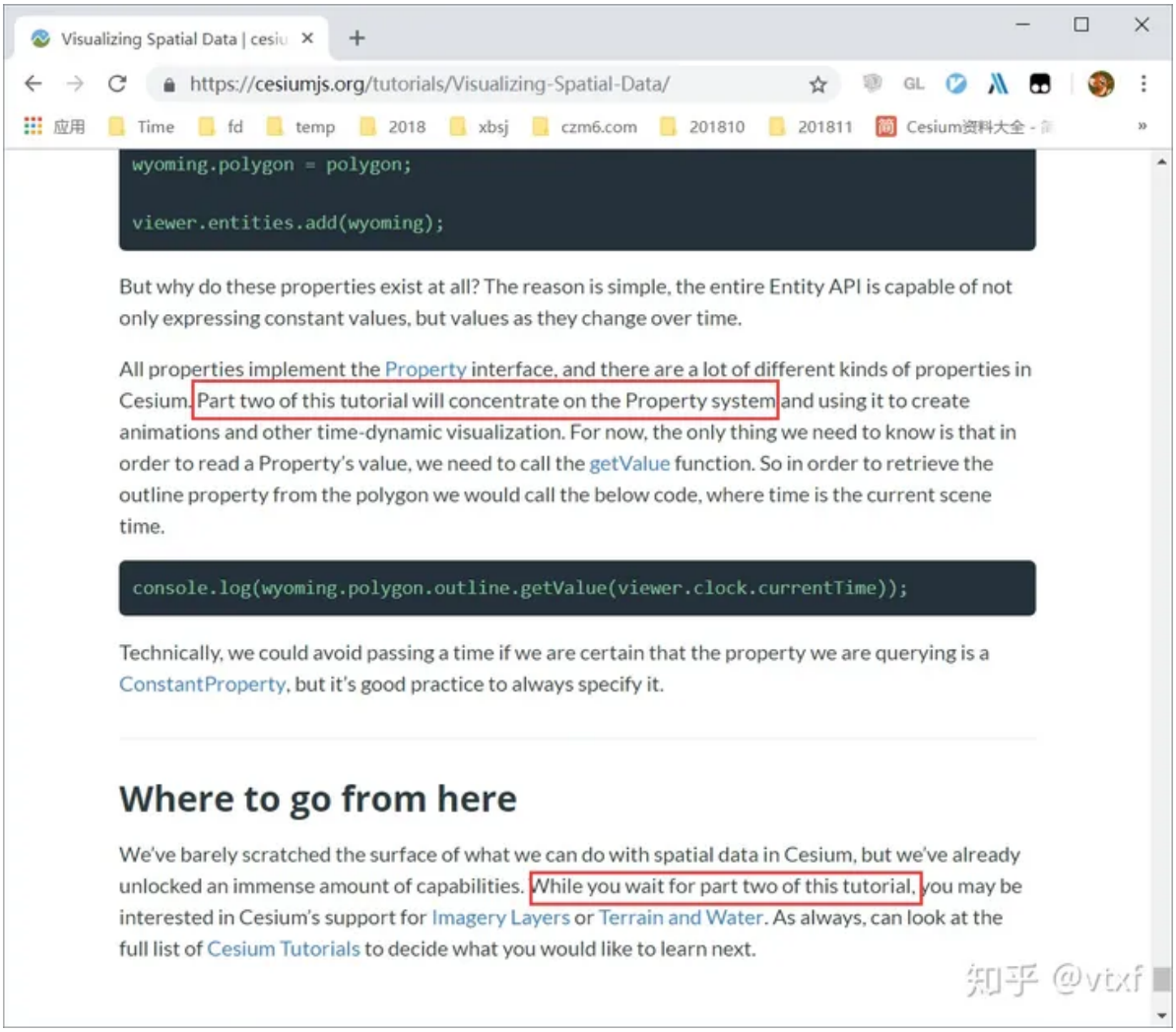
总结

至此，Cesium几个重要的类我们都一一做了介绍，这几个类也是Cesium最核心的类，相信由此出发，读者可以更好地掌握Cesium的API使用方法。以后有机会，我们会再介绍一下Cesium API中一些约定俗称的，但是前端开发者又比较难理解的API调用方法。

Property 动态改变



前言Cesium官方教程中有一篇叫《空间数据可视化》(Visualizing Spatial Data)。该文文末简单提到了Cesium的Property机制，然后话锋一转，宣告此教程的第二部分将重点讲解Property机制。但是呢，第二部分还没有写好，说在等待的过程中，可以先看下Cesium对影像和地形的支持。。可以看官方教程中的说法，如下图所示：



于是，我苦等了一年啦。。官方教程的第二部分还是没能看到。。毕竟这是Cesium官方推荐使用的Entity API中最重要的部分之一。。居然这么久了也不给更新下。。我想还是自己总结一下得好。。为什么要用Property？还是举个例子来说吧。

不变的长方体 ConstantProperty

比如我想在地球上的某个位置加一个盒子，可以这样写代码：

```
// 创建盒子
var blueBox = viewer.entities.add({
    name : 'Blue box',
    position: Cesium.Cartesian3.fromDegrees(-114.0, 40.0, 300000.0),
    box : {
        dimensions : new Cesium.Cartesian3(400000.0, 300000.0, 500000.0),
        material : Cesium.Color.BLUE,
        outline: true,
    }
});
```

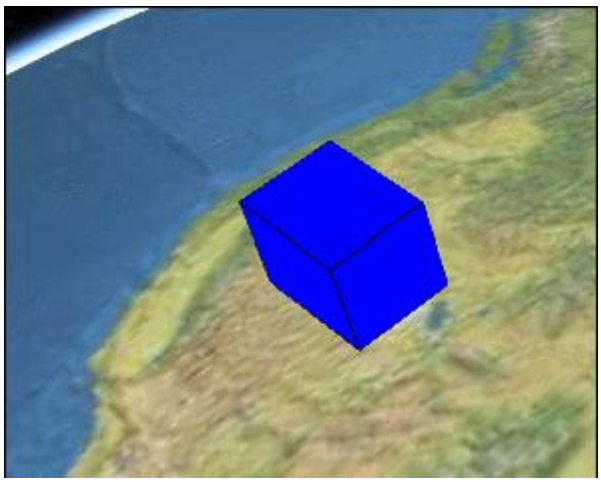
下面的两句代码实现的效果一致且意义也一致

```
blueBox.box.dimensions = new Cesium.Cartesian3(400000.0, 300000.0, 200000.0);
```

```
blueBox.box.dimensions = new ConstantProperty(new Cesium.Cartesian3(400000.0,
300000.0, 200000.0));
```

Entity的 `box.dimensions` 类型并不是 `Cartesian3`，而是一个 `Property` 。虽然我们赋值了一个 `Cartesian3` ，但是 Cesium 内部会隐晦地转化成了一个`ConstantProperty` 。注意只会隐晦地转化成`ConstantProperty` ，而不是`SampleProperty` ，更不是`TimeIntervalCollectionProperty` 。虽然叫`ConstantProperty`，但是，这里Constant的意思并不是说这个Property不可改变，而是说它**不会随时间发生变化。**

最终的效果如图所示：



匀速式变长的长方体 SampledProperty

但是呢，如果我想让这个盒子逐渐变长，该怎么操作呢？方法是有的，就是可以不停地去修改blueBox.position，类似这样：

```
setInterval(function(){ blueBox.box.dimensions = xxx; }, 3000);
```

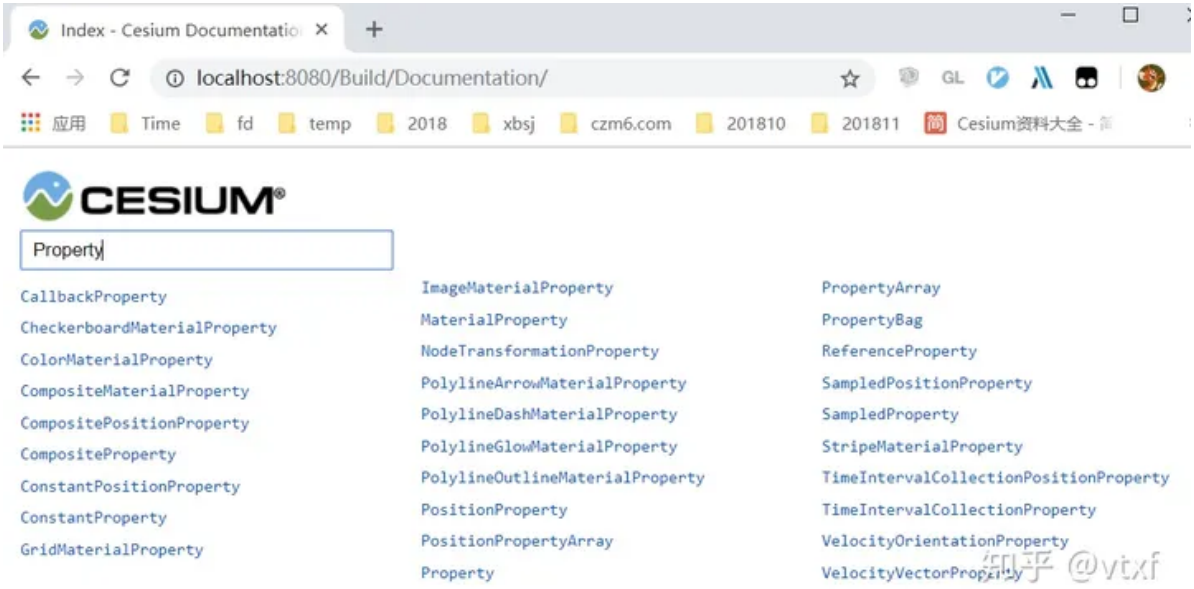
如果场景中有很多物体，在不同的时间段要发生各种走走停停地运动时，这样操作可能会很累人。那么**Cesium就提供一种机制，让dimensions可以随时间自动发生变化，自动赋予不同的数值（位置）。这也就是property的作用**了。

以下代码的加入，就可以让盒子如上图所示做线性运动了。

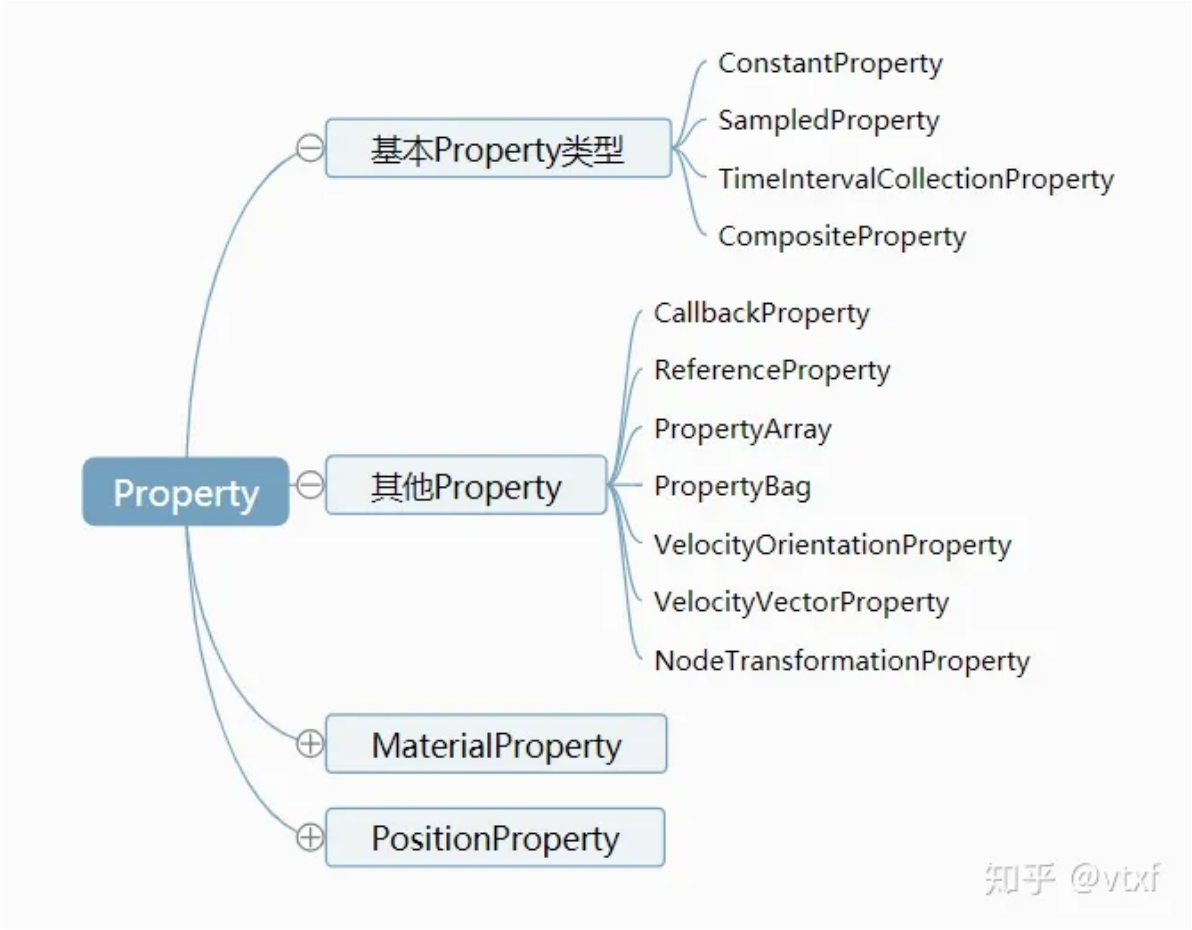
```
// 装配一个 SampledProperty 对象
var property = new Cesium.SampledProperty(Cesium.Cartesian3);
property.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-01T00:00:00.00Z'),
    new Cesium.Cartesian3(400000.0, 300000.0, 200000.0));
property.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-03T00:00:00.00Z'),
    new Cesium.Cartesian3(400000.0, 300000.0, 700000.0));
// 使用这个 SampledProperty 对象
blueBox.box.dimensions = property;
```

以上代码的意思就是在两个不同的时间点分别赋予不同的位置，用SampledProperty包装成一个property，最后赋给blueBox.box.dimensions。由此可见，**Property最大的特点是和时间相互关联，在不同的时间可以动态地返回不同的属性值。**而Entity则可以感知这些Property的变化，在不同的时间驱动物体进行动态展示。Cesium宣称自己是数据驱动和time-dynamic visualization，这些可都是仰仗Property系统来实现的。当然，Property可不只是这么简单，以下再详细论述。

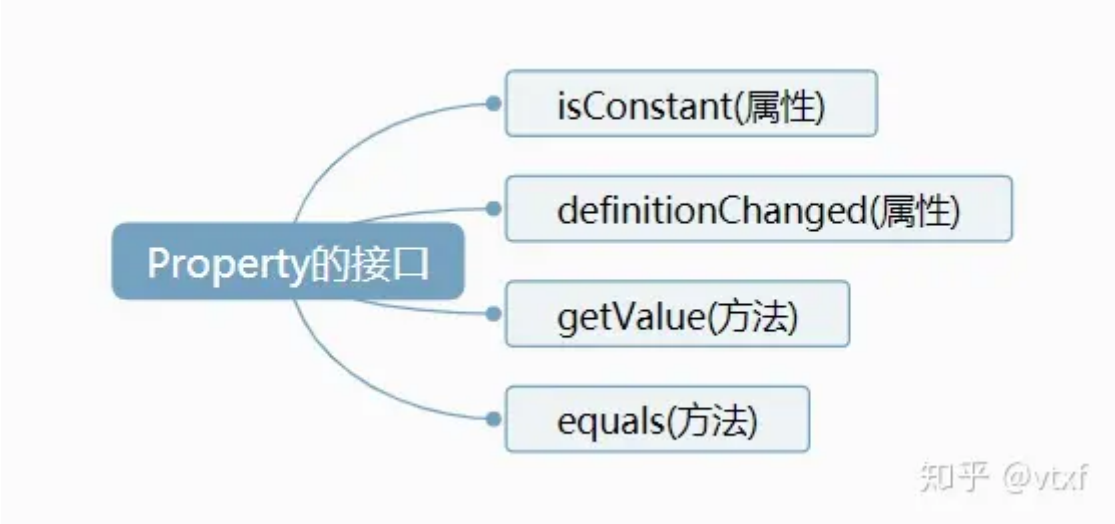
Property的分类Cesium的Property不止有刚才示例代码中的SampleProperty，还有很多其他的类型。如果搜索一下Cesium的API文档，会有很多。。如下图所示：



我们简单分类一下



Property虚基类Property是所有Property类型的虚基类。它定义了以下接口。



- **getValue** 是一个方法，用来获取某个时间点的特定属性值。它有两个参数：第一个是time，用来传递一个时间点；第二个是result，用来存储属性值，当然也可以是undefined。这个result是Cesium的scratch机制，主要是用来**避免频繁创建和销毁对象而导致内存碎片。Cesium就是通过调用getValue类似的一些函数来感知Property的变化的**，当然这个方法我们在外部也是可以使用的。
- **isConstant** 用来判断该属性是否会随时间变化，是一个布尔值。Cesium会通过这个变量来决定是否需要在场景更新的每一帧中都获取该属性的数值，从而来更新三维场景中的物体。如果isConstant为true，则只会获取一次数值，除非definitionChanged事件被触发。
- **definitionChanged** 是一个事件，可以通过该事件，来监听该Property自身所发生的变化，比如数值发生修改。
- **equals** 是一个方法，用来检测属性值是否相等。

跳跃式变长的长方体 TimeIntervalCollectionProperty

基本Property类型 `SampleProperty` 我们最早在上述示例中使用的就是它，用来通过给定多个不同时间点的Sample，然后在每两个时间点之间进行线性插值的一种Property。代码写法如下：

```
var property = new Cesium.SampledProperty(Cesium.Cartesian3);
property.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-01T00:00:00.00Z'),
    new Cesium.Cartesian3(400000.0, 300000.0, 200000.0)
);
property.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-03T00:00:00.00Z'),
    new Cesium.Cartesian3(400000.0, 300000.0, 700000.0)
);
blueBox.box.dimensions = property;
```

TimeIntervalCollectionProperty该Property用来指定各个具体的时间段的属性值，每个时间段内的属性值是恒定的，并不会发生变化，除非已经进入到下一个时间段。拿创建的盒子示例来说，表现出来的特点就是盒子尺寸的变化时跳跃式的。
代码如下：

```
// 装配一个 TimeIntervalCollectionProperty
var property = new Cesium.TimeIntervalCollectionProperty(Cesium.Cartesian3);
property.intervals.addInterval(
    Cesium.TimeInterval.fromIso8601({
        iso8601 : '2019-01-01T00:00:00.00Z/2019-01-01T12:00:00.00Z',
```

```
        isStartIncluded : true,
        isStopIncluded : false,
        data : new Cesium.Cartesian3(400000.0, 300000.0, 200000.0)
    })
);
property.intervals.addInterval(
    Cesium.TimeInterval.fromIso8601({
        iso8601 : '2019-01-01T12:00:01.00Z/2019-01-02T00:00:00.00Z',
        isStartIncluded : true,
        isStopIncluded : false,
        data : new Cesium.Cartesian3(400000.0, 300000.0, 400000.0)
    })
);
property.intervals.addInterval(
    Cesium.TimeInterval.fromIso8601({
        iso8601 : '2019-01-02T00:00:01.00Z/2019-01-02T12:00:00.00Z',
        isStartIncluded : true,
        isStopIncluded : false,
        data : new Cesium.Cartesian3(400000.0, 300000.0, 500000.0)
    })
);
property.intervals.addInterval(
    Cesium.TimeInterval.fromIso8601({
        iso8601 : '2019-01-02T12:00:01.00Z/2019-01-03T00:00:00.00Z',
        isStartIncluded : true,
        isStopIncluded : true,
        data : new Cesium.Cartesian3(400000.0, 300000.0, 700000.0)
    })
);
// 使用 TimeIntervalCollectionProperty
blueBox.box.dimensions = property;
```

ConstantProperty

通过对TimeIntervalCollectionProperty和SampleProperty的描述，读者应该基本了解Property的特点。我们回过头来说下ConstantProperty，其实这才是最常用的Property。示例代码如下：`blueBox.box.dimensions = new Cesium.Cartesian3(400000.0, 300000.0, 200000.0);`以上代码貌似没有使用ConstantProperty，实际上他是等同于：`blueBox.box.dimensions = new ConstantProperty(new Cesium.Cartesian3(400000.0, 300000.0, 200000.0));`也就是举个例子，

可以通过 property.getValue(viewer.clock.currentTime) 方法来获取某个时间点property的属性值。如果property是SampleProperty或者TimeIntervalCollectionProperty的话，不同的时间点，可能getValue出不同的数值。但是如果这个property是ConstantProperty，那么无论什么时间（getValue的第一个参数不起作用），最后返回的数值都是一样的。但是不会随时间变化，并不代表不可改变。ConstantProperty还有一个setValue的方法，开发者可以通过调用它，来在适当的时候改变property的值。比如，我可以通过点击按钮来修改ConstantProperty，代码如下：`blueBox.box.dimensions.setValue(new Cesium.Cartesian3(400000.0, 300000.0, 700000.0));`需要注意的是，虽然最终效果一样，但是以下两种写法的意义是不一样的。`blueBox.box.dimensions = new Cesium.Cartesian3(400000.0, 300000.0, 200000.0);``blueBox.box.dimensions.setValue(new Cesium.Cartesian3(400000.0, 300000.0, 700000.0));`前者会创建一个新的ConstantProperty，后者则会修改原有的ConstantProperty的值。

组合前几种变化 CompositePropertyCompositeProperty

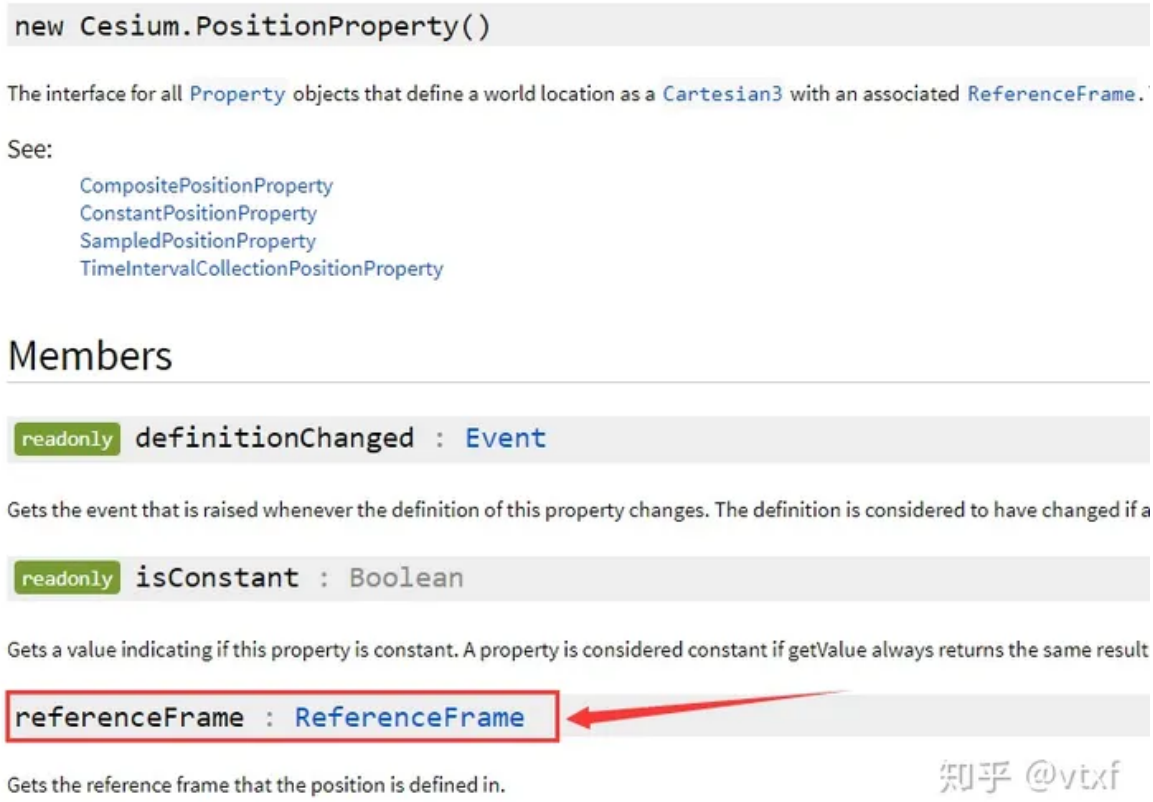
组合的Property，可以把多种不同类型的ConstantProperty、SampleProperty、TimeIntervalCollectionProperty等Property组合在一起来操作。比如前一个时间段需要线性运动，后一段时间再跳跃式运动。则可以使用类似下面这段代码来实现。

```
// 1 sampledProperty
var sampledProperty = new Cesium.SampledProperty(Cesium.Cartesian3);
sampledProperty.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-01T00:00:00.00Z'),
    new Cesium.Cartesian3(400000.0, 300000.0, 200000.0)
);
sampledProperty.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-02T00:00:00.00Z'),
    new Cesium.Cartesian3(400000.0, 300000.0, 400000.0)
);
// 2 ticProperty
var ticProperty = new Cesium.TimeIntervalCollectionProperty();
ticProperty.intervals.addInterval(Cesium.TimeInterval.fromIso8601({
 iso8601 : '2019-01-02T00:00:00.00Z/2019-01-02T06:00:00.00Z',
 isStartIncluded : true,        isStopIncluded : false,        data : new
Cesium.Cartesian3(400000.0, 300000.0, 400000.0)    }));
ticProperty.intervals.addInterval(Cesium.TimeInterval.fromIso8601({
 iso8601 : '2019-01-02T06:00:00.00Z/2019-01-02T12:00:00.00Z',
 isStartIncluded : true,        isStopIncluded : false,        data : new
Cesium.Cartesian3(400000.0, 300000.0, 500000.0)    }));
ticProperty.intervals.addInterval(Cesium.TimeInterval.fromIso8601({
 iso8601 : '2019-01-02T12:00:00.00Z/2019-01-02T18:00:00.00Z',
 isStartIncluded : true,        isStopIncluded : false,        data : new
Cesium.Cartesian3(400000.0, 300000.0, 600000.0)    }));
ticProperty.intervals.addInterval(Cesium.TimeInterval.fromIso8601({
 iso8601 : '2019-01-02T18:00:00.00Z/2019-01-03T23:00:00.00Z',
 isStartIncluded : true,        isStopIncluded : true,        data : new
Cesium.Cartesian3(400000.0, 300000.0, 700000.0)    }));
// 3 compositeProperty
var compositeProperty = new Cesium.CompositeProperty();
compositeProperty.intervals.addInterval(
    Cesium.TimeInterval.fromIso8601({
        iso8601 : '2019-01-01T00:00:00.00Z/2019-01-02T00:00:00.00Z',
        data : sampledProperty
    })
);
compositeProperty.intervals.addInterval(
    Cesium.TimeInterval.fromIso8601({
        iso8601 : '2019-01-02T00:00:00.00Z/2019-01-03T00:00:00.00Z',
        isStartIncluded : false,
        isStopIncluded : false,
        data : ticProperty
    })
);
// 4 设置position
blueBox.box.dimensions = compositeProperty;
```

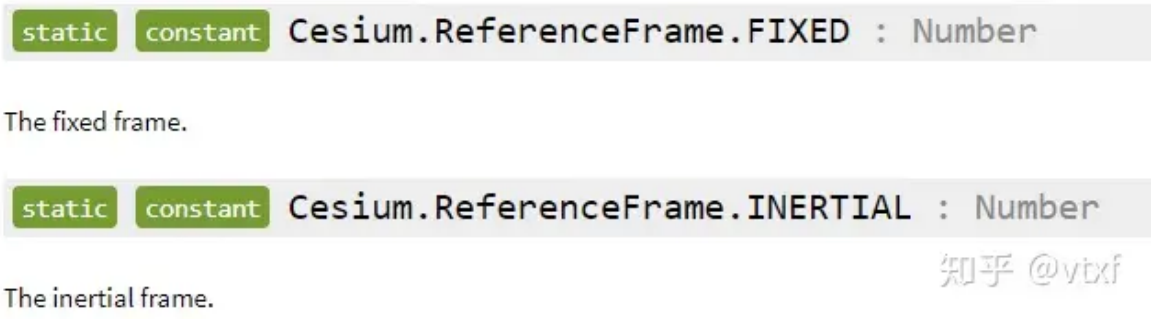
我们一直在用SampledProperty、ConstantProperty等来修改Entity的box.dimensions属性。基本上可以得出结论：大部分Property都是可以赋值给Entity的box.dimensions的。

线性改变位置 PositionProperty

以上示例可以看到，我们一直在用SampledProperty、ConstantProperty等来修改Entity的box.dimensions属性。基本上可以得出结论：大部分Property都是可以赋值给Entity的box.dimensions的。PositionProperty和Property一样，是一个虚类，并不能直接实例化，他扩展了Property的接口，增加了referenceFrame，同时只能用来表示position。



referenceFrame是用来表示position的参考架。目前Cesium有以下两种参考架。



我们常用的是FIXED这种默认类型，它相当于以地球的中心作为坐标系的原点，x轴正向指向赤道和本初子午线的交点。（可能描述不准确。。）这样我们给定一个笛卡尔坐标(x, y, z)，它在地球上的位置是固定的。而INERTIAL这种类型，则相当于以太阳系的质心为原点的坐标架偏移到地球的中心来，如果给定一个笛卡尔坐标(x, y, z)，那么它在不同的时间表示的是地球上的不同位置。。（我的理解，可能有误。。）一般情况下，我们用不上INERTIAL。但是如果真的给定了INERTIAL下的坐标点，Cesium内部会通过PositionProperty，把它转成同一个FIXED下的坐标点来使用，这些不需要我们操作。但是，因为普通的Property是没有办法进行这种参考架的自动转换的，所以Cesium派生了一批PositionProperty类型。基于PositionProperty的类型有以下几种：

- CompositePositionProperty
- ConstantPositionProperty
- PositionProperty
- PositionPropertyArray

- SampledPositionProperty
- TimeIntervalCollectionPositionProperty

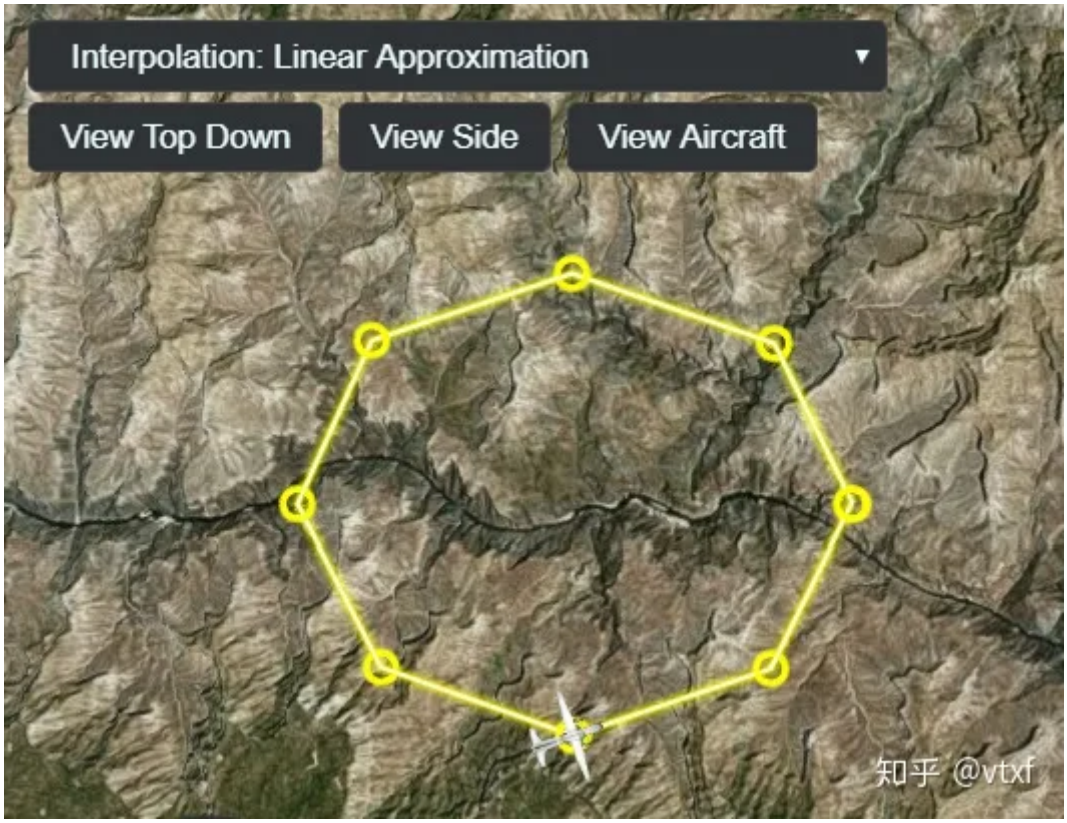
稍加留意，就会发现，和普通的Property相比，只是多了一个Position，所以用法上也大同小异，只不过他们是用来专门表示位置的。SampledPositionProperty的用法，不多解释，直接看代码吧：

```
var property = new Cesium.SampledPositionProperty();
property.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-01T00:00:00.00Z'),
    Cesium.Cartesian3.fromDegrees(-114.0, 40.0, 300000.0)
);
property.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-03T00:00:00.00Z'),
    Cesium.Cartesian3.fromDegrees(-114.0, 45.0, 300000.0)
);
blueBox.position = property;
```

插值改变位置 setInterpolationOptions

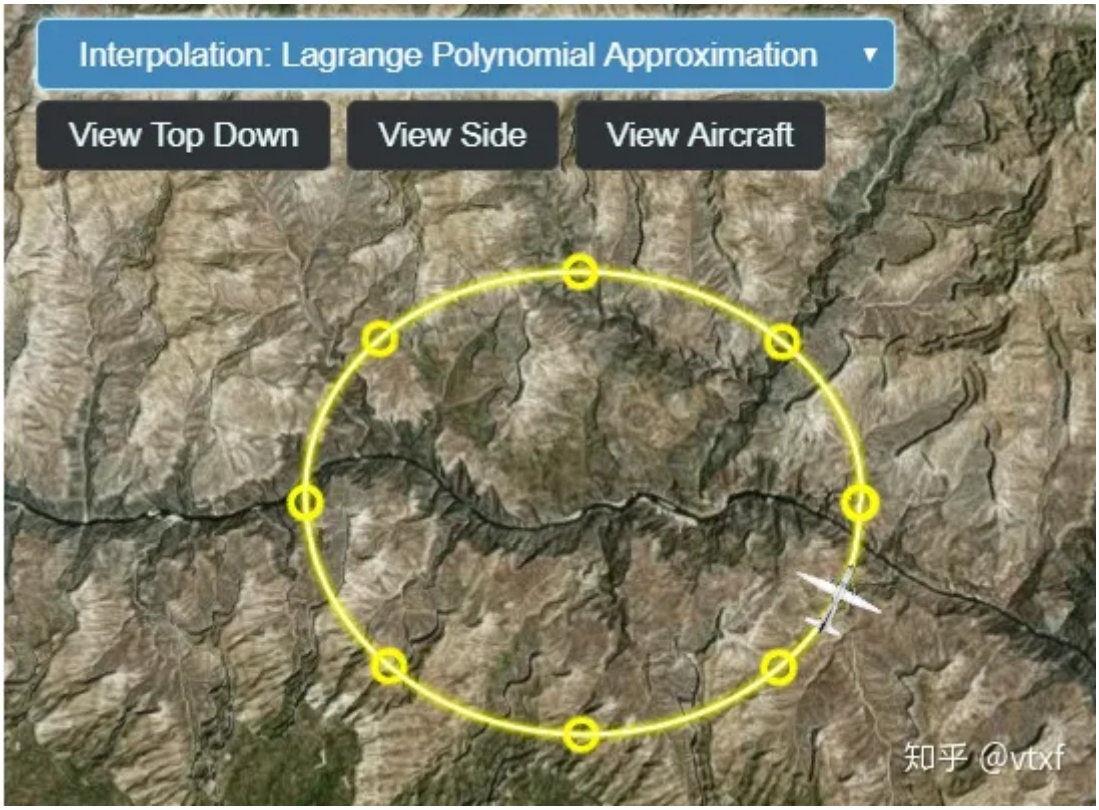
`SampleProperty` 和 `SampledPositionProperty` 有一个特有的方法：`setInterpolationOptions`，用来修改不同的插值方式。以下是以Cesium的Interpolation示例中的截图来说明他们的不同之处。

线性插值



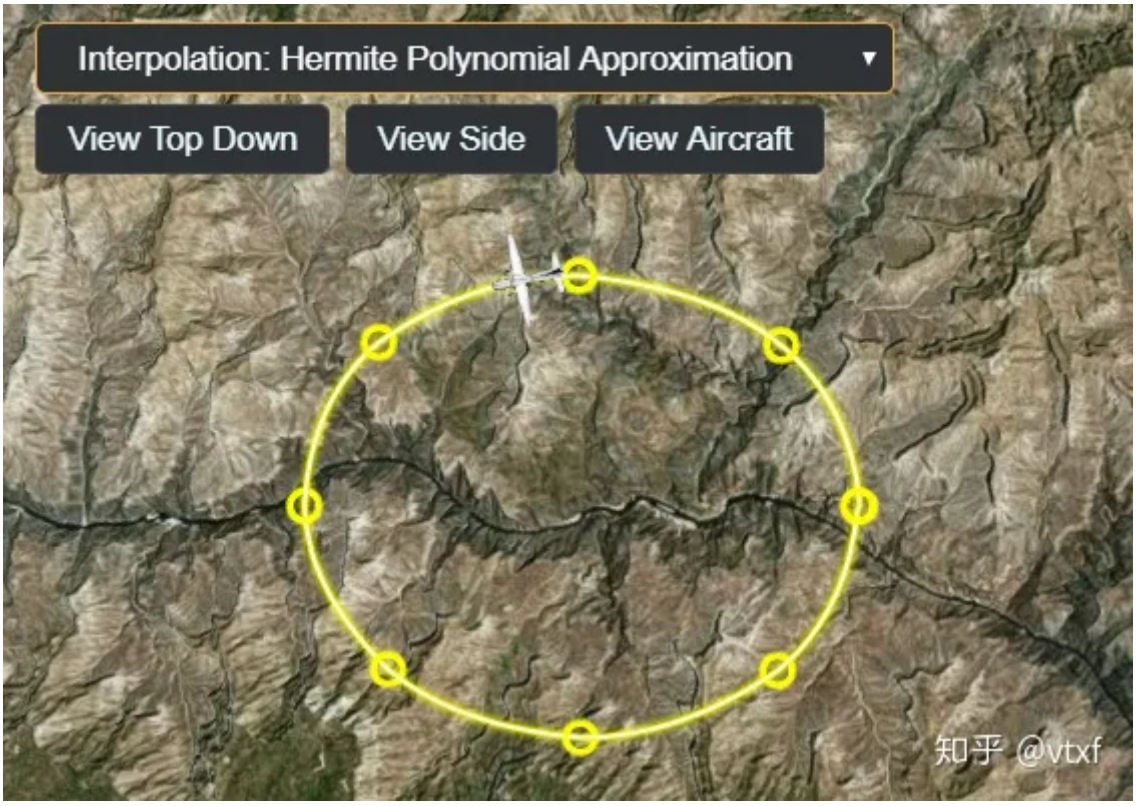
代码写法如下：`entity.position.setInterpolationOptions({ interpolationDegree : 1, interpolationAlgorithm : Cesium.LinearApproximation });`

Lagrange插值




```
entity.position.setInterpolationOptions({ interpolationDegree : 5,
interpolationAlgorithm : Cesium.LagrangePolynomialApproximation });
```

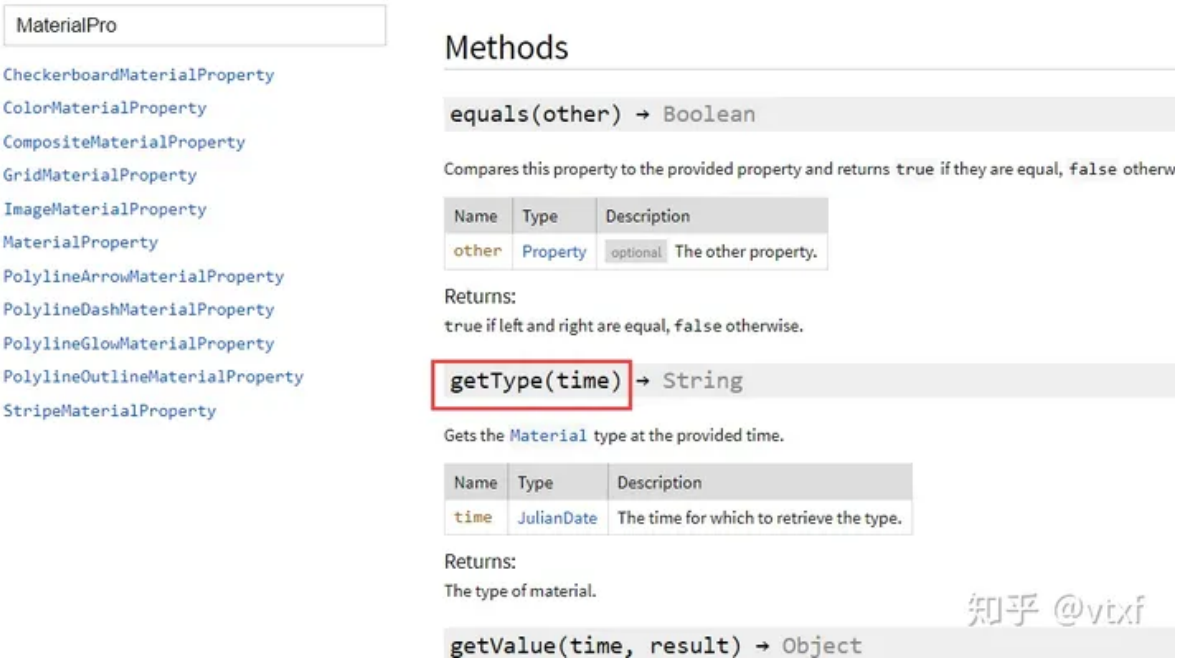
Hermite插值




```
entity.position.setInterpolationOptions({ interpolationDegree : 2,
interpolationAlgorithm : Cesium.HermitePolynomialApproximation });
```

改变长方体的外观 MaterialProperty

MaterialPropertyMaterialProperty是用来专门表示材质的Property，它对Property进行了扩展，增加了getType方法，用来获取材质类型。



MaterialProperty也是一个虚基类，派生类有：

- CheckerboardMaterialProperty
- ColorMaterialProperty
- CompositeMaterialProperty
- GridMaterialProperty
- ImageMaterialProperty
- MaterialProperty
- PolylineArrowMaterialProperty
- PolylineDashMaterialProperty
- PolylineGlowMaterialProperty
- PolylineOutlineMaterialProperty
- StripeMaterialProperty

使用上大同小异，我们以ColorMaterialProperty来说明一下。

改变长方体颜色 ColorMaterialProperty

```
blueBox.box.material = new Cesium.ColorMaterialProperty(new Cesium.Color(0, 1, 0));
// 以上代码等同于
// blueBox.box.material = new Cesium.Color(0, 1, 0);
```

ColorMaterialProperty的动态变化如果希望Color动起来的话，也是可以的。ColorMaterialProperty的内部有一个color属性，可以赋予一个SampledProperty来实现动态效果。

```
var colorProperty = new Cesium.SampledProperty(Cesium.Color);
colorProperty.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-01T00:00:00.00Z'),
    new Cesium.Color(0, 1, 0)
);
colorProperty.addSample(
    Cesium.JulianDate.fromIso8601('2019-01-03T00:00:00.00Z'),
    new Cesium.Color(0, 0, 1)
);
blueBox.box.material = new Cesium.ColorMaterialProperty(colorProperty);
```

✨自定义改变各种属性CallbackProperty

自由度最高的一种Property，让用户通过自定义，回调函数，来返回需要的值。回调函数中，用户可以使用time来给定value，也可以以自己的方式给给定。以下代码就是不通过time，自己手动调整dimension的示例。

```
var l = 200000.0;
var property = new Cesium.CallbackProperty(function (time, result) {
    result = result || new Cesium.Cartesian3(0, 0, 0);
    l += 10000.0;
    if (l > 700000.0) {
        l = 200000.0;
    }
    result.x = 400000.0;
    result.y = 300000.0;
    result.z = l;
    return result;
}, false);
blueBox.box.dimensions = property;
```

🔗链接属性ReferenceProperty

该Property可以直接链接到别的对象的Property上，相当于引用，省得自己构建了。比如这里我创建了一个红色的盒子redBox，希望它和之前的蓝色盒子一起变大。那么可以使用以下代码：

```
var collection = viewer.entities;  redBox.box.dimensions = new Cesium.ReferenceProperty(collection, blueBox.id, ['box', 'dimensions']);
```

```
new Cesium.ReferenceProperty(targetCollection, targetId, targetPropertyNames)
```

A Property which transparently links to another property on a provided object.

Name	Type	Description
targetCollection	EntityCollection	The entity collection which will be used to resolve the reference.
targetId	String	The id of the entity which is being referenced.
targetPropertyNames	Array.<String>	The names of the property on the target entity which we will use.

ReferenceProperty构造函数的参数有三个。

第一个参数用来指定需要引用的对象所属的collection，如果没有自己专门创建EntityCollection的话，可以直接使用viewer.entities。

第二个参数传递所指对象的id。

第三个参数指定属性的位置的数组，如果是有层级的属性，可以依次写入。比如 `['billboard', 'scale']` 指定的是entity.billboard.scale 属性。当然还有其他设置方式，可以参见Cesium的api文档。

📌独配 PropertyBag

PropertyBag 虽然不是以Property结尾，但实际上也是一个Property。它的特点是可以包装一个对象(JS中的对象概念)，该对象的每一个属性(JS中的属性概念)，都可以作为一个动态的Property。比如之前修改dimensions的话，dimensions是作为一个Cartesian3类型变量整体封装到Property中去的，如果我们只想修改dimensions的x。则可以使用PropertyBag来实现，代码如下：

```
var zp = new Cesium.SampledProperty(Number);
zp.addSample(Cesium.JulianDate.fromIso8601('2019-01-01T00:00:00.00Z'),
200000.0);
zp.addSample(Cesium.JulianDate.fromIso8601('2019-01-03T00:00:00.00Z'),
700000.0);
// 为 dimension 属性配置一个 PropertyBag 实例，为 z 值单独配置一个SampledProperty来控制
z 值的随时间改变
blueBox.box.dimensions = new Cesium.PropertyBag({
    x: 400000.0,
    y: 300000.0,
    z: zp
});
```

效果和 sampleProperty 类似，但是修改的只是 dimensions 的 z 。PropertyArrayPropertyArray和上述的PropertyBag类似，只是其内部封装了一个数组而已。这里不再赘述。

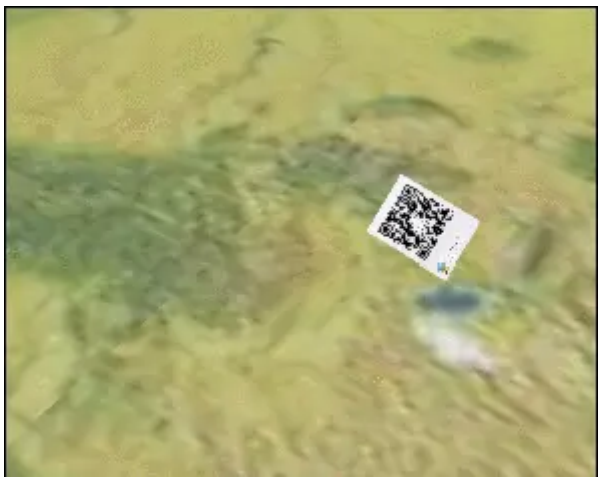
☑物体朝向与速度方向

VelocityOrientationProperty该Property用来Entity的position的位置变化，来计算出移动的方向，最后把速度方向输出成Orientation。Cesium自带的示例中有一个Interpolation中有其用法，不再赘述。

VelocityVectorProperty与上面的Property类似，把速度方向转成Vector。使用示例如下：

```
blueBox.box.show = false;
blueBox.billboard = {
    scale: 0.05,
    image : 'https://upload-images.jianshu.io/upload_images/80648-
5dfe8a3ea2c250be.png?imageMogr2/auto-
orient/strip%7CimageView2/2/w/540/format/webp',
    alignedAxis : new Cesium.VelocityVectorProperty(blueBox.position, true) //
alignedAxis must be a unit vector
};
```

可见图像的摆放方向和位置移动的方向保持一致。效果如下：



附录

本文在github上的源码

ImageryLayer 载入影像

为什么cesium只有imageryLayer？ - 知乎 (zhihu.com)

与 Widgets 组件类中的 BaseLayerPicker

Viewer创建中指定

```
this.map = new Viewer(withKeyId, {
            imageryProvider: new ArcGisMapServerImageryProvider({
                url:
'https://services.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer'
            }),
});
```

后续添加

这个记录全是错的，因为我忘记加底图地址😂

记录：

情况	如何添加 imageryProvider	是否设置 Ion	BaseLayerPicker	显示结果
	直接添加 { }	否	没有这个图层的图标，显示为黑色	正常
	不加 Viewer.ConstructorOptions { }	否	不可使用 BaseLayerPicker 中的 Cesium Ion 组中的数据源	不显示地球但显示星空
		是	可以正常使用所有数据源	正常
	间接加 { }	否	按钮消失	不显示地球但显示星空
		是	按钮消失	不显示地球但显示星空

1. 在 viewer 中直接添加一个 imageryProvider 不需要设置 Ion ，加载正常，但是在 BaseLayerPicker 组件中是异常的，没有这个图层的图标，显示为黑色。
2. 在不传入 Viewer.ConstructorOptions 这个配置时
    1. 有 Ion 配置，可以正常使用 BaseLayerPicker 所有数据源。
    2. 没有 Ion 配置，不可使用 BaseLayerPicker 中的 Cesium Ion 组中的数据源，且在首屏时不显示地球但显示星空。
    3. 总结：需要注意 BaseLayerPicker 组件如何修改，后面需要对其改写配置。
3.

载入GeoJson-GeoJsonDataSource

需要配合 axios 来得到 json 文件。那么要先书写 axios 相关的逻辑代码。

相关文件 `http.ts  code.ts  @/utils/format`

- 在 http.ts 和 env.d.ts配置一个 `readonly VITE_API_BASEURL: string;` 什么意思?
    - 在 vite.config.ts 中配置一个 `base: viteEnv.VITE_BASE,` 这个配置base配置项可以让打包后的 index.html 中 href="/index.f6170881.css" --> href="./index.f6170881.css" 那么对于 development 模式无影响吗?
    - 但是我配置了 vite 中的代理服务器 且 在.env总加了 `VITE_API_BASEURL =  '/api'` 后就可以了。解释在代码注释中。

- ```
  // 简略版
  Cesium.Viewer.dataSources.add(
    // 使用 静态方法
      Cesium.GeoJsonDataSource.load(data, {
                  stroke: Cesium.Color.HOTPINK,
                  fill: Cesium.Color.fromAlpha(Color.RED, 0.3),
                  strokeWidth: 3
      })
  );
  ```

☑总结用法

为了在界面上显示 GeoJson 文件所存储的地理信息，需要以下几步（会有失偏颇）：

1. 获取当前 Cesium 实例的 Viewer，例如 `const viewer = new Cesium.Viewer('cesiumContainer');`
2. 准备一个 GeoJsonDataSource 实例（Source）。`const g = new GeoJsonDataSource('itsName');`
3. 使用 load 方法或者 process 方法为该 GeoJsonDataSource 提供数据源。

   ```
   g.process('./src/components/Map/data.json', styleOptions || {
           stroke: Color.HOTPINK,
           fill: Color.fromAlpha(Color.RED, 0.5),
           strokeWidth: 3
       }
   )
   ```

4. 在 viewer 中添加该 GeoJsonDataSource 实例。`viewer.dataSources.add(g);`

📌注意

- Cesium.GeoJsonDataSource.load(data, options) 是该类的静态方法，用于简写。
  - 源码中是这样的：

    ```
    GeoJsonDataSource.load = function (data, options) {
      return new GeoJsonDataSource().load(data, options);
    };
    ```

- load(data, options) → Promise. 意义与上面的静态方法一样。重复使用会覆盖原来的数据。
  - 源码

    ```
    GeoJsonDataSource.prototype.load = function (data, options) {
      return preload(this, data, options, true);
    };
    ```

- process(data, options) → Promise
  - 源码

    ```
    GeoJsonDataSource.prototype.process = function (data, options) {
      return preload(this, data, options, false);
    };
    ```
    ```

- 通过上面的三个 API 可以看出都是使用了 preload 方法，不过在该类中不暴露出来。
    - https://github.com/CesiumGS/cesium/blob/1.110/packages/engine/Source/DataSources/GeoJsonDataSource.js#L909
    - 其中有一个`clear`是一个布尔值，用于指定是否在加载新数据之前清除之前的数据。load 和 process 的参数区别在这。

❓

其实我还是有很多疑问：

- 有几种Source，和 dataSourceDisplay 有关系吗？我如何查阅 geojson 加载到哪个实例中，是 GeoJsonDataSource 实例中吗，还是说被安排在其他某个地方？使用 process 加载到同一个 GeoJsonDataSource 实例中会不会不太好？
- 下面的代码预期是只有第一个 process 的数据会有高度，但是实际结果是两个都拉升了。

```
            g.process('./src/components/Map/data.json', styleOptions || {
                stroke: Color.HOTPINK,
                fill: Color.fromAlpha(Color.RED, 0.5),
                strokeWidth: 3
            })
            .then((e)=>{
                // this.viewer.dataSources.add(e);
                let entities = e.entities.values;
                for (let i = 0; i < entities.length; i++) {
                    // entity[i].polygon.outline = false;
                    entities[i].polygon.extrudedHeight = 1000;
                }
            })
            g.process(data, styleOptions || {
                stroke: Color.GREEN,
                fill: Color.fromAlpha(Color.GREEN, 0.5),
                strokeWidth: 3
            })

            this.viewer.dataSources.add(g);
```

解释：异步加载问题。先加载的 第二个 ，后加载的第一个 😂。因为data是已经加载好了的数据。

- 为什么 GeoJsonDataSource 实例中 name 为 itsName， 但是控制台打印的 显示为 data2.json？？？？

键鼠事件

Cesium 中的pick讲解_cesuim中pickposition的原理?-CSDN博客

Cesium 拾取有多个方法，下面就分别说明一下几种常用方法都是做什么用的，在什么场景下使用。

1. **viewer.scene.pick** ✅

通过坐标位置，拾取 Entity 、 Primitive、3D-Tiles（Cesium3DTileFeature）对象。例如获取 Entity，通过position（坐标位置）获取到 pick 对象，通过pick.id即可拾取当前的entity对象。 `var pick = viewer.scene.pick(position);`

注意： scene.pick 只能获取一个对象，并且获取的是最顶部的对象。如果拾取点没有对象，则为 undefined

使用场景： viewer.scene.pick 主要是用来拾取 Entity 、 Primitive、3D-Tiles。拾取后，可以用于**改变对象的属性参数。**

示例代码： 点击获取图标，修改图标的图片

```
var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
// 设置左键点击事件
handler.setInputAction(function (event) {
  // 获取 pick 拾取对象
  var pick = viewer.scene.pick(event.position);
  // 判断是否获取到了 pick
  if (Cesium.defined(pick)) {
    // 修改拾取到的entity的样式
    pick.id.billboard.image = "xxx.png"
  }
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

2. **viewer.scene.pickPosition**

主要是用于拾取对应位置的世界坐标，例如：拾取点击模型表面的坐标。

注意： 一定要开启深度检测（`viewer.scene.globe.depthTestAgainstTerrain = true;`），否则在没有3dTile模型的情况下，会出现空间坐标不准的问题，如果不开启深度检测，只能在3dTile模型上获取准确的空间坐标。

使用场景： 适用于模型表面位置的选取，拾取三维物体的坐标等。

示例

```
var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (event) {
     var position = viewer.scene.pickPosition(event.position);
     console.log("获取到的坐标：", positionposition);
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

3. **viewer.scene.drillPick** ✅

可以理解为**穿透拾取**，是从当前拾取位置获取所有对象的对象列表（entity的集合），列表按其在场景中的视觉顺序（从前到后）排序（通过for循环可以获取当前坐标下的所有entity）。

注意： drillPick 和 Pick 不同，Pick 只能拾取一个对象，而 drillPick 可以拾取多个对象，并且 drillPick 可以设置 limit 参数，limit 参数可以控制获取几个对象，超出的就不获取了。

使用场景： 适用于多个对象重叠在一个位置，并且要获取到多个对象的情况。

示例

```
var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (event) {
    var pickedObjects = scene.drillPick(event.position);
    // pickedObjects 使用for循环 可以拿到所有entity
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

4. **viewer.scene.globe.pick**

找到射线与渲染的地球表面之间的交点（射线必须以世界坐标给出），返回的是世界坐标。一般用来获取加载地形后的经纬度和高程。

注意： 一定要开启深度检测（viewer.scene.globe.depthTestAgainstTerrain = true;）

使用场景： 一般用于获取点击处地球表面的世界坐标（有地形），注意：不包括模型、倾斜摄影表面。

示例

```
var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (event) {
     var ray = viewer.camera.getPickRay(event.position);
     var position = viewer.scene.globe.pick(ray, viewer.scene);
     console.log("获取到的坐标：", position);
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

5. **viewer.scene.camera.pickEllipsoid**

通过一个必选的屏幕坐标，获取椭圆球体表面的一个Cartesian3坐标。适用于裸球表面的选取，是基于数学模型的椭圆球体。

注意： pickEllipsoid在加载地形的情况下有误差，地形凹凸程度越大，误差越大，所以不要用来获取有地形的坐标。

使用场景： 主要用于获取椭球面的位置。

示例

```
var handler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (event) {
     var position = viewer.scene.camera.pickEllipsoid(event.position,
viewer.scene.globe.ellipsoid);
     console.log("获取到的坐标：", position);
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

参考文档：

Cesium 点击获取/拾取（PICK）的不同用法*cesium* 拾取最凶残的小海豹的博客-CSDN博客

https://www.cnblogs.com/airduce/p/14663927.html

Cesium 各类键鼠事件

90 cesium的四种点击拾取方法 矩阵变换 本地坐标转世界 禁止场景/鼠标左右拖动、禁用中键 修改点击infoBox内容 3Dtile性能优化 cesium+vue_cesium 拾取点_smallcelebration的博客-CSDN博客

Cesium 拾取 API，按拾取物体来分类总结

Cesium：四种拾取pick 以及原理(cha138.com)

按拾取对象分类

拾取坐标

1. 仅拾取椭球体表面坐标 pickEllipsoid

使用 `Camera.prototype.pickEllipsoid` 方法，接受一个必选的屏幕坐标，返回一个三维世界坐标 `Cartesian3`。

2. 拾取带地形高度的地表坐标

使用 `Globe.prototype.pick` 方法。需要事先使用 `Camera.prototype.getPickRay` 创建射线。接受一个必选的射线对象，一个必选的 `Scene` 对象，返回一个三维世界坐标 `Cartesian3`。

3. 拾取三维物体的坐标

使用 `Scene.prototype.pickPosition` 方法。

> 拓展阅读
> `Scene.prototype.pickPositionSupported`，只读字段，表示当前 Scene 是否支持拾取坐标
> `Scene.prototype.pickTranslucentDepth`，Boolean 类型字段，使用它的前提是设置 `Scene.prototype.useDepthPicking` 为 true，这个 **会增加性能消耗**，来判断透明物体的深度

拾取三维物体

拾取 Entity 和 Primitive（包括 3D-Tiles）

使用 `Scene.prototype.pick` 方法，返回一个对象：

```
{
  primitive: Primitive | GroundPrimitive | Cesium3DTileContent | ...
  id?: Entity
}
```

若拾取到的是 Entity，那么返回的对象的 id 字段将为此 Entity，否则为 undefined.

还有一个 `Scene.prototype.drillPick`，穿透拾取的意思，与 pick 的区别就是能拾取多个点击点的三维物体。

拾取 DataSource 加载的数据

一样通过 `Scene.prototype.pick` 和 `drillPick` 方法拾取，接受二维屏幕坐标 `Cartesian2`。

拾取图层

这个功能正在推进，届时可能在 1.84 版本的 Cesium 会加入拾取图层的功能。
API 或为 `ImageryLayerCollection.prototype.pickImageryLayers`，参数同 `Globe.prototype.pick`，也是射线求交的一种，返回一个 `ImageryLayerCollection` 或 undefined.

原理

在 Cesium 的场景组织中，有那么几个容器构成了三维世界：

Scene：包括了 Globe，除了 Globe 的元素外，还加上了 Primitive、Entity、DataSource 等三维物件
Globe：包括了 Ellipsoid，还包括了所有的影像图层、地形瓦片，可以算是椭球体上面的皮肤
Ellipsoid：一个数学公式所定义的旋转椭球体，代表一个纯粹的地球椭球形状

所以，针对不同的容器，就有不同的拾取。

拾取不准确的问题：开启深度检测

`Scene.prototype.pickPosition`、`Scene.prototype.pick` 和 `Globe.prototype.pickRay` 的准确性受深度缓存影响，所以，在深度检测不开启时，拾取的坐标会不准确。

建议开启。

场景触发事件

场景中一些变化触发的监听事件，随着Cesium中一些对象实例化而产生的事件。常用的场景触发事件有：

- Camera.changed：相机发生变化时触发
- Camera.moveEnd：相机停止移动时触发
- Camera.moveStart：相机开始移动时触发
- Scene.preUpdate：场景更新之前触发
- Scene.postUpdate：场景更新之后立即触发
- Scene.preRender：场景更新之后渲染之前触发
- Scene.postRender：场景渲染之后触发
- Scene.terrainProviderChanged：地形提供器发生变化时触发
- Viewer.trackedEntityChanged：entity的属性发生变化时触发
- Cesium3DTileset.allTilesLoaded：所有3dtiles数据加载完成以后触发
- Cesium3DTileset.loadProgress：3dtiles初始化加载过程中触发
- Cesium3DTileset.tileFailed：3dtiles瓦片加载失败时触发
- Globe.imageryLayersUpdatedEvent：地球加载图层更新时触发

```
// 需要回调的函数
function callbackFunc(event){
    cosnole.log(event)
}
// 渲染之前执行
viewer.scene.preRender.addEventListener(callbackFunc);
viewer.scene.preRender.removeEventListener(callbackFunc);

// 更新之前执行
viewer.scene.preUpdate.addEventListener(callbackFunc);
viewer.scene.preUpdate.removeEventListener(callbackFunc);

// 实时渲染执行
viewer.scene.postRender.addEventListener(callbackFunc);
viewer.scene.postRender.removeEventListener(callbackFunc);

// 实时更新执行
viewer.scene.postUpdate.addEventListener(callbackFunc);
viewer.scene.postUpdate.removeEventListener(callbackFunc);
```

dataSourceDisplay，entity，dataSource

dataSourceDisplay 是对数据进行处理的一类 API ， entity 表示的是处理完的数据在 Cesium 中的一个个实体或者自定义的实体。dataSource 是泛指 GeoJsonDataSource 等这类数据或者 Wall 这类自定义实体，当然也有这个 API ，不过一般作为 定义数据源的接口，该接口将任意数据转换为EntityCollection以供一般使用。该对象是用于文档目的的接口，不建议直接实例化来使用。

分类

学习新事物，分类是帮助我们构建体系最好的方法。

dataSourceDisplay

成员：

- dataSources : DataSourceCollection 一个容器，用于存放 dataSource 实例。
- defaultDataSource : CustomDataSource 接口类型为 new Cesium.CustomDataSource(name)。一种 dataSource 的实现类型，可用于手动管理一组实体。
- ready 只读，指示数据源中的所有实体是否就绪。
- scence 略

例子：

```
// 1. 实例化一种 dataSource 类型，这里实例化 CustomDataSource ；
const dataSource = new Cesium.CustomDataSource('myData');
// 2. 实例化的 CustomDataSource 中添加一个 entity ；
const entity = dataSource.entities.add({
   position : Cesium.Cartesian3.fromDegrees(1, 2, 0),
   billboard : {
       image : 'image.png'
   }
});
// 3. 实例化的 CustomDataSource 添加到 实例化的 dataSources 中（使用的是实例化的
dataSources 的 add 方法）。
viewer.dataSources.add(dataSource);
```

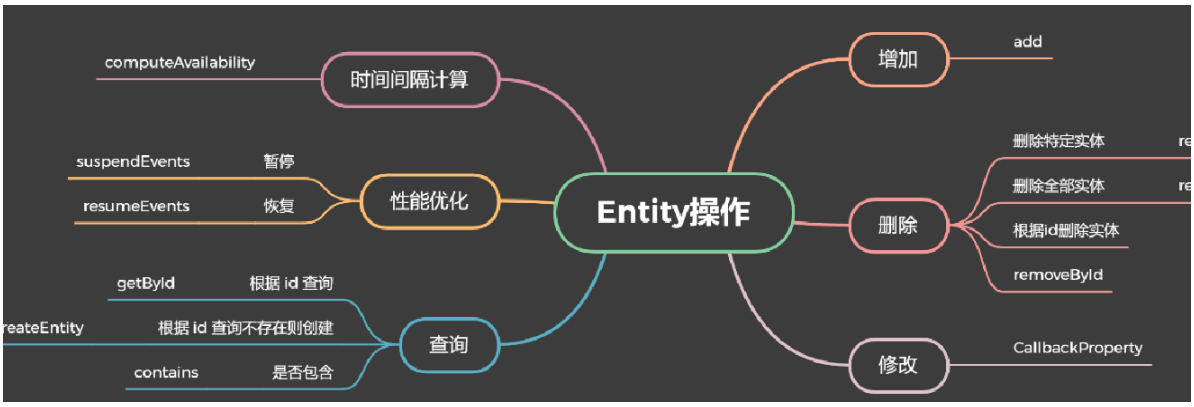
entity

成员：

- 不一一列举了。

方法：

- 几乎都是用来控制实体的属性之类的方法



4.Cesium中实体Entity的增删改查及性能优化（超详细）-阿里云开发者社区 (aliyun.com)

对比 entity， entities，EntityCollecton，EntityCluster：

entity 是一个类，可以实例化的对象，实例后的对象被赋予一些属性表现出不同的“性状”，即空间对象可视化，需要给定对象的空间位置和显示样式。

entities 作为 某个实例化对象的 属性名 例如 `viewer.entities`，一般为 EntityCollecton 。

EntityCollecton 是 entity 容器，可以放置多个 entity 实例，也提供了`add、remove、removeAll`等等接口来管理场景中的entity。

EntityCluster 是Cesium中的实体聚合类，用于对多个实体 **Billboard**、**Label** 、**Point** 进行聚合和分组，以便更好地管理和控制它们。EntityCluster可以帮助您组织和优化您的场景，并为用户提供更好的交互体验。https://sandcastle.cesium.com/index.html?src=Clustering.html

例子：

```
      const anEntity = new Entity({
            id:'obj_id_110',
            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ouline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
            }
        });

        viewer.entities.add(anEntity);

        varhandler = new Cesium.ScreenSpaceEventHandler(viewer.scene.canvas);
        handler.setInputAction(function(movement) {
            varpick = viewer.scene.pick(movement.position);
            if(Cesium.defined(pick) && (pick.id.id === 'obj_id_110')) {
                alert('picked!');
            }
        },Cesium.ScreenSpaceEventType.LEFT_CLICK);
```

给定 **空间位置和显示样式** 创建 entity ， 使用 add 方法添加到 EntityCollecton 实现场景中 entity 的管理，并借助ScreenSpaceEventHandler 实现场景中 entity 的拾取。

✅高级功能实现

动态更新：Entity支持实时动态更新，可以通过设置位置、方向等属性的回调函数来实现地理实体的实时更新。这样，我们可以根据实时数据或交互操作来改变实体的状态，从而实现实时演示、动画效果等。需要借助 Property 实现。

扩展与自定义：通过继承Entity类，我们可以扩展和自定义实体的功能。可以通过添加新的属性、方法和事件处理函数，来满足特定需求或实现定制化的地理实体。**?????**

水体模型的前端展示构建流程

数据源制作

arcgis工具

geojson面的高度改变

Cesium 中 entity 简单移动_cesium entity移动-CSDN博客

打算使用加载点的方式来模拟水体

为什么 GeoJsonDataSource 实例没有 position：因为这是一个 collection ， 一个 entity 的 collection 。但每个 entity 有 position ， 每个 entity 有 polygon ， polygon 有 height 。

GeoJsonDataSource 实例中有 show 属性 为 true，GeoJsonDataSource 实例中的 entities 属性中也有 show 属性 为 true ， 但 entities 中的 polygon 中 show 属性为 undefined 。

tips

> 带有透明度的颜色的图层叠加后是会让颜色也叠加的

> 在vue组件的setup中得到 window 中自定义属性会出现 undefined ， 但是在 onmounted 钩子中读取该属性则正常？？？？？

> 一般加载数据的方法都是异步的，而且会返回一个 Promise ， 可以数据源这个 Promise 来进行更进一步的数据处理，详细可以去看 《 载入GeoJson-GeoJsonDataSource # 疑问 》

> Cesium通过 feature ids来操作3dtiles瓦片集中的要素 - 知乎 (zhihu.com)
>
> Cesium如何用射线相交排除已经交到的模型 - 我爱学习网 (5axxw.com)
>
> cesium drillPick实现原理_A873054267的博客-CSDN博客

> Cesium阴影技术_cesium shadowmap-CSDN博客
>
> Cesium 高性能扩展之DrawCommand（三）：显隐和点选 - 简书 (jianshu.com)

Cesium提供了射线查询功能，可用于检测与射线相交的模型。你可以将射线从相机位置向前发射，然后用射线查询功能检测在射线路径上的所有对象。如果一个对象在之前的射线路径中已经被交到过，可以通过记录已被处理的对象以及它们的位置来实现排除。

具体实现可以参考以下代码：

```
var ray = new Cesium.Ray(camera.position, camera.direction);
var pickedObjects = scene.drillPick(ray); // 检测射线路径上的所有对象
var intersectedPositions = []; // 记录已经交到的所有位置
for (var i = 0; i < pickedObjects.length; ++i) {
    var object = pickedObjects[i].primitive;
    var position = object.position; // 获取对象的位置
    // 如果当前位置已经被处理过，则排除当前对象
    if (intersectedPositions.indexOf(position) !== -1) {
        continue;
    }
    // 处理当前对象
    // ...
    intersectedPositions.push(position); // 记录已处理的位置
}
```

在实际应用中，由于射线查询是一个相对耗时的操作，为了提高性能，可以使用空间分割算法（如八叉树）来对空间进行划分，只检测射线路径经过的空间，避免不必要的查询。