Search

TWAVER DOCUMENT CENTER

- 开发指南 – TWaver HTML5 3D
  - 功能列表及性能指标
  - 3D实战：一步一步学3D开发
  - 3D开发基础
  - 3D对象详解
  - 3D高级开发
  - 附录
    - 3D对象格式说明
    - 常见问题解答
    - API文档

TWAVER DOCUMENT CENTER › 开发指南 – TWaver HTML5 3D › 3D开发基础

第一个简单例子

为了让开发者快速理解mono的开发方法，下面是一个简单的例子，通过几分钟即可完成。之后，您可以继续深入学习mono详细的开发方法。通过这个简单的例子，您可以体验使用mono进行开发是多么方便和快捷。

1.创建HTML文件

创建一个最简单的mono程序，只需t.js这一个库文件即可。用编辑工具创建一个test.html文件，内容如下：

```
<html>
<head>
  <meta charset="utf-8"/>
  <title>Hello MONO</title>
  <script type="text/javascript" src = "t.js"></script>
  <script type="text/javascript">
    //your code here
  </script>
</head>

<body onload = "load()">
  <div id = "mainDiv">
    <canvas id="myCanvas" /
  </div>
</body>
</html>
```

以上HTML代码创建了一个空白页面，并在页面加载结束后调用load函数，在load函数中填写代码即可。其中t.js包通过一个script标签进行引入。浏览器访问该页面，会显示一个空白页面。

接下来，我们在load函数中创建一个3D场景。在load函数中，我们先创建一个3D的基本场景。一个简单的3D场景，需要有一些基本的元素，包括数据容器、3D视图、镜头、灯光、3D物体等。下文中我们会逐个逐步详细说明。

2.创建DataBox

在mono中，所有的3D物体都需要放入一个DataBox容器进行统一管理。这一设计方式和TWaver的2D产品（如TWaver Java、TWaver HTML5等）保持一致，也是基本的MVC设计准则。Mono中，数据放入数据容器进行管理（DataBox代表MVC中的Model层，与TWaver 2D中DataBox类似），3D的数据绘制和展示放在Network中管理（Network代表MVC中的View层，与TWaver 2D中的Network类似）。

需要留意的是，这里的DataBox和Network都是mono中的类，位于mono.*包下，和TWaver HTML5中的twaver.*包不同，两者代码也是完全分离的。

所以，创建3D场景首先需要创建一个DataBox容器，用来容纳3D场景中的各种物体。代码非常简单，直接new一个mono.DataBox即可：

```
1 var box = new mono.DataBox();
```

3.创建镜头

有了DataBox后，需要创建一个Network对象来显示3D场景。在创建Network之前，我们首先需要创建一个镜头对象来构造Network。

镜头（mono.Camera）是一个代表用户第一视角的对象，它类似我们的眼睛。和2D不同，3D世界中我们看到的场景会随着我们的“眼睛”的位置的不同而变化。物体的远近、大小、角度、光照等，都与镜头有直接的关系，因此每一个Network必须有一个镜头与之对应。

镜头有许多参数，主要是对镜头透视的定义。此处先不展开介绍，开发时使用最常见的参数即可：

```
1 var camera = new mono.PerspectiveCamera();
2 camera.setPosition(50,80,100);
```

上面代码中，第一句话表示创建镜头对象，第二句话设置镜头的空间位置（x、y、z）。

4.创建Network

Network是mono中3D的绘制“画布”，它负责把DataBox中的3D数据通过WebGL技术渲染绘制出来，并接受用户的各种交互（鼠标、键盘、触控等）。

```
1 var network= new mono.Network3D(box, camera, myCanvas);
```

上面的代码创建了一个Network对象。其中第一个参数是前面创建的DataBox对象，第二个参数是前面创建的镜头对象，第三个参数是HTML中定义的canvas对象的id值，也就是这句中定义的canvas对象：

```
1 <canvas id="myCanvas"/>
```

创建Network后，一般还需要设置一些交互模式，否则这个Network只能显示3D场景，而不能响应鼠标等交互动作（如旋转、缩放等）。和TWaver 2D中类似，Network的交互模式都被封装在各种Interaction中，直接创建并设置给Network即可。这里我们给Network设置了一个默认的交互模式：mono.DefaultInteraction。它为Network提供了最常用、基本的交互模式，包括鼠标和触控的旋转、缩放等操作。

```
var interaction = new mono.DefaultInteraction(network);
interaction.zoomSpeed = 30;
network.setInteractions([interaction]);
```

上面代码先创建了一个默认Interaction，并为它设置了缩放操作的相应速度（如鼠标滚轮），最后放在一个数组中设置给Network。Network可以接受一个Interaction数组，这些Interaction可以通过自由组合添加到Network中，这些Interaction会同时生效，以满足各种不同的交互要求。

例如，可以在数组中再添加EditInteraction、SelectionInteraction等，就可以增加编辑、选中等交互功能。

接下来，我们添加一行代码，让Network始终保持全屏充满页面空间，即使改变页面窗口大小也会保持这个效果，用户视觉体验会更加直观，使用也比较方便。

```
mono.Utils.autoAdjustNetworkBounds(network,document.documentElement,'clientWidth','clientHeight');
```

以上就完成了Network的基本设置。

5.创建灯光

一般而言，在3D场景中模拟真实世界的场景，光照是一个非常重要的元素。在一个完全没有光照的3D场景中，所有物体都是黑色的，就和现实世界中一个完全黑暗的房间一样。为了让物体反射光线到我们的眼睛里，除了需要给物体设置材质之外，首先要给场景添加光源。

在mono中，有多种类型的光源可供选择。最常见的有两个：环境光（mono.AmbientLight）和点光源（mono.PointLight）。环境光类似现实世界中漫反射的环境光，它没有固定统一的方向，所有物体的所有面都会接收到环境光发来的光源。点光源是一个发光点，以其位置为中心向四周发射光照。

一般3D场景中，我们可以同时使用这两种光源，来模拟真实场景。例如环境光模拟现实世界环境光，点光源模拟现实中的太阳光或室内的灯光。如果需要，还可以设置多个点光源，来模拟多个位置的灯光。

光照最主要的属性有光的颜色、强度等。点光源还需要设置其位置。

光源对象和3D物体对象类似，也要放入DataBox中。下面代码创建了一个点光源和环境光，并加入DataBox中：

```
//create point light.
var pointLight = new mono.PointLight(0xFFFFFF,1);
pointLight.setPosition(10000,10000,10000);
box.add(pointLight);

//create ambient light.
box.add(new mono.AmbientLight(0x888888));
```

6.创建3D物体

通过以上的代码，我们就搭建好了3D场景，但尚未放入任何3D物体。最后，我们来创建一个简单的3D立方体，并放入场景中。Mono中提供了很多基本3D形状，如立方体（mono.Cube）、圆柱体（mono.Cylinder）等等。每个物体可以设置其大小、位置、旋转角度、缩放比例、材质等属性。更多物体类型和属性的介绍请看后续章节及apidoc。

在这里，我们创建一个长宽高分别为25、25、25的立方体，并设置其位置（x、y、z）为55、50、55。然后设置其材质图片、材质类型等属性。最后将其置入DataBox中进行显示。代码如下：

```
1 var node=new mono.Cube(25,25,25);
2 node.setPosition(55, 50, 55);
3 node.setStyle('m.texture.image','../images/fence.png').setStyle('m.type','phong');
4 box.add(node);
```

请注意，上述代码中，'m.texture.image'属性是材质图片，可传入png、jpg等图片的url地址；'m.type'是材质的类型，分为'basic'和'phong'两种。其中，'phong'类型支持高光光照效果，而'basic'类型不支持光照效果。

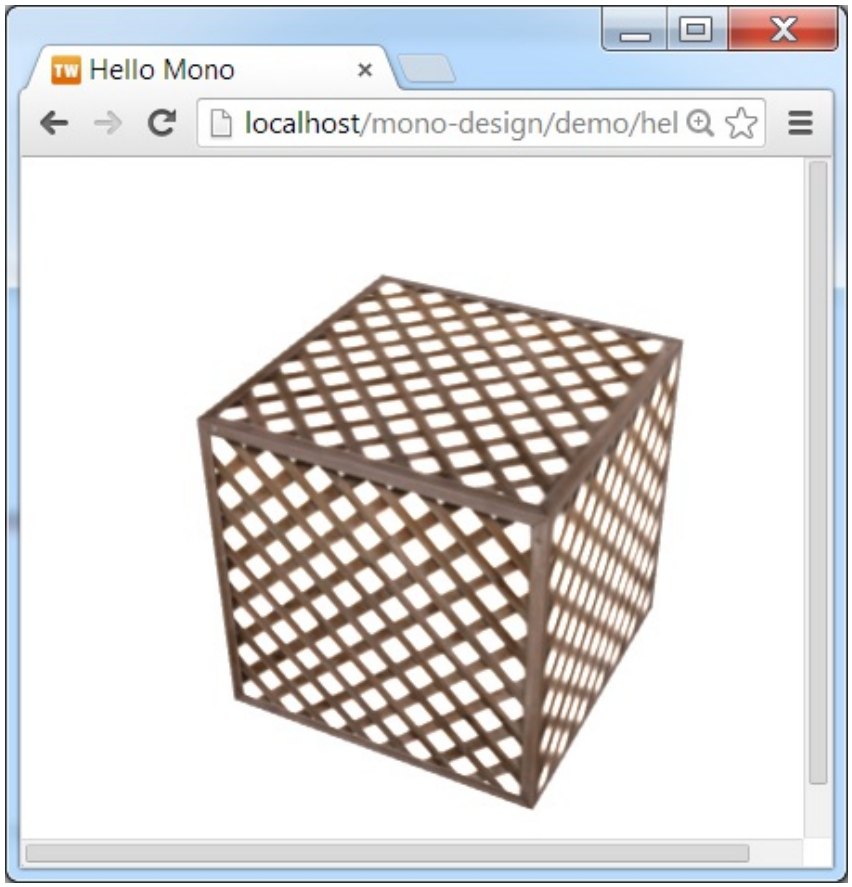
7.完整代码

以上介绍的完整代码如下：

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4     <title>Hello MONO</title>
5     <script type="text/javascript" src = "t.js"></script>
6     <script type="text/javascript">
7        function load(){
8
9           var box = new mono.DataBox();
10          var camera = new mono.PerspectiveCamera(30, 1.5, 0.1, 10000);
11          camera.setPosition(50,80,100);
12
13          var network= new mono.Network3D(box, camera, myCanvas);
14          var interaction = new mono.DefaultInteraction(network);
15          interaction.zoomSpeed = 30;
16          network.setInteractions([interaction]);
17          mono.Utils.autoAdjustNetworkBounds(network,document.documentElement,'clientWidth','clientHeight');
18
19          var pointLight = new mono.PointLight(0xFFFFFF,1);
20          pointLight.setPosition(10000,10000,10000);
21          box.add(pointLight);
22          box.add(new mono.AmbientLight(0x888888));
```

```
23
24          var node=new mono.Cylinder(25,25,25);
25          node.setPosition(55, 50, 55);
26          node.setStyle('m.texture.image','../images/fence.png').setStyle('m.type','phong');
27          box.add(node);
28
29       }
30    </script>
31 </head>
32 <body onload = 'load()'>
33    <div id = "mainDiv">
34       <canvas id="myCanvas"/>
35    </div>
36 </body>
37 </html>
```

用浏览器访问该页面，显示效果如下：



基础知识

概述

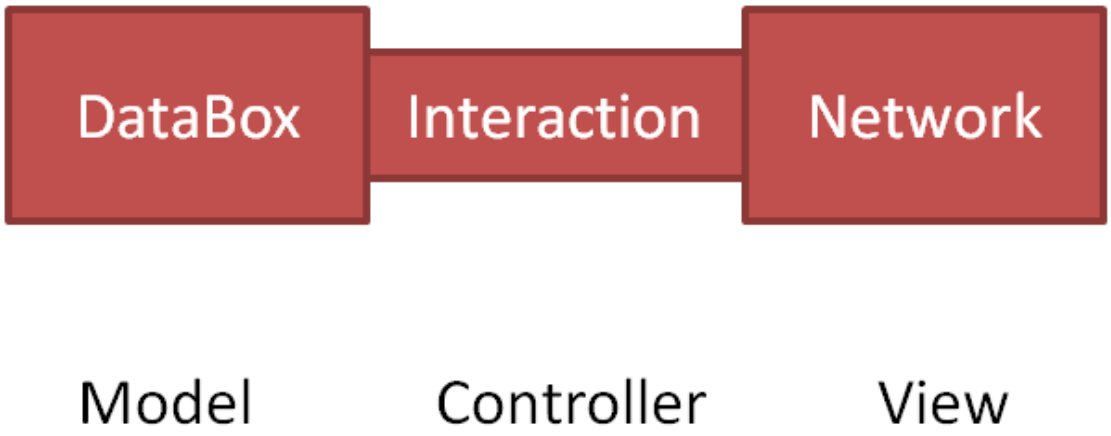
欢迎使用TWaver MONO Design！MONO Design是TWaver 3D产品统称，包括3D建模编辑工具、模板库以及mono开发包。开发者可以通过直接使用mono开发包以及API来创建各种3D应用程序。MONO Design基于WebGL及HTML5，开发者需要掌握JavaScript语言和基本的HTML知识。

Mono开发所需的js库文件主要包括：

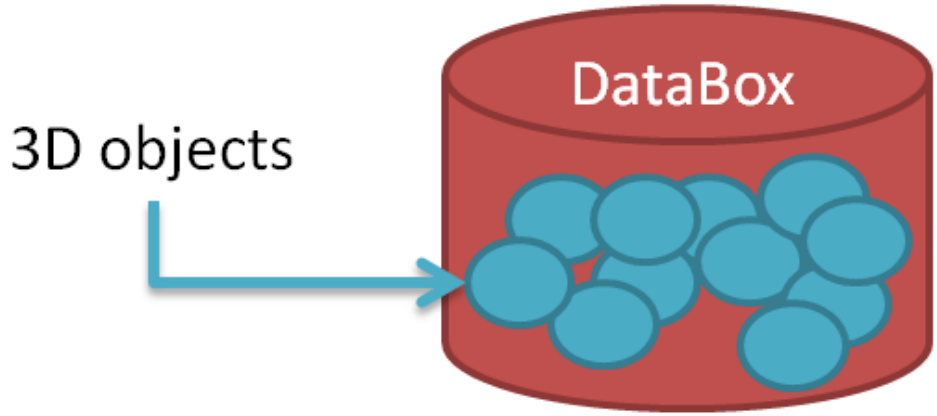
- t.js：mono核心库，包括3D模型及基础API；
- twaver.js：twaver html5 2d的库文件；
- mono_toolkits.js：模板操作API，提供数据模板的导入及导出功能

MVC设计架构

Mono采用了TWaver一贯的MVC（Model,View,Controller）架构。DataBox作为Model层，是一个容器管理对象，用来管理所有的3D对象。Network是3D的呈现渲染画布，也是MVC中的View层。而交互及基于事件监听机制则代表了控制层Controller。Mono的MVC结构如下图：



DataBox是一个对象管理容器。所有要显示的3D对象都放置在这个容器中统一管理。容器会负责3D对象的增、删、查等工作，同时监听数据的变化并通过事件监听机制进行广播通知。Network则是负责具体显示的画布，它必须连接到一个具体DataBox容器，并将容器中的数据绘制到canvas画布上。DataBox和Network作为数据管理者和画布渲染者独立工作、各司其职。

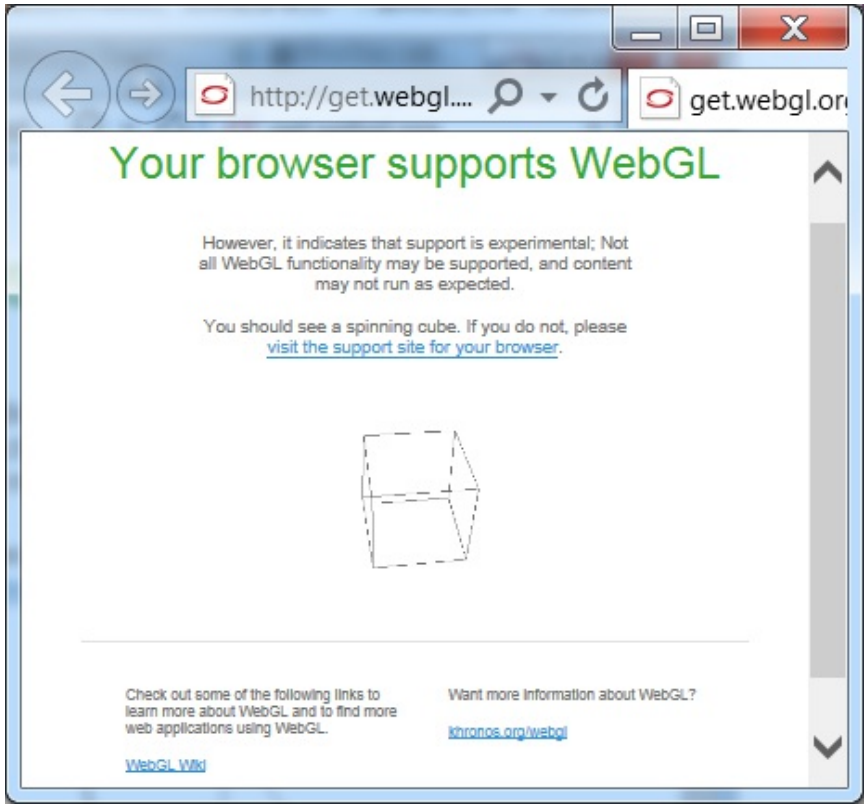


支持浏览器

开发者需要在支持WebGL的浏览器上调试运行mono程序。包括：

- Chrome v30及以上
- IE v11及以上
- Safari
- FireFox
- Opera

可以点击网址get.webgl.org来检查浏览器是否支持WebGL。如果能够看到一个旋转的立方体，则说明浏览器支持WebGL（如下图）。



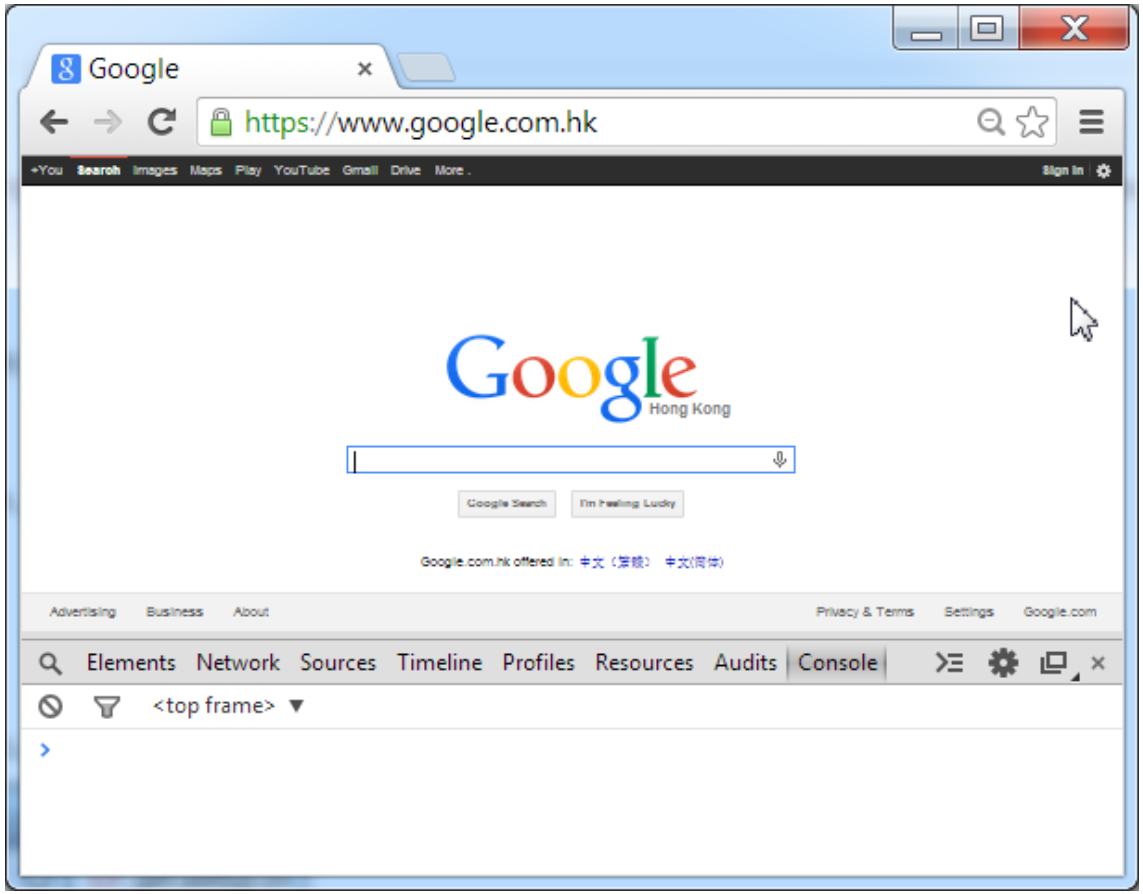
开发工具及调试

JavaScript开发可以选择使用Eclipse、NetBeans、Visual Studio等大型IDE工具，也可以选择直接使用文本编辑工具如EditPlus、UltraEdit等。前者可以提供一些自动提示、集成调试等功能，缺点是程序比较庞大笨重。后者则轻量快速直接，但无自动提示等功能。开发者可以根据自身情况合理选择代码编辑工具。

对于开发测试浏览器，推荐使用Google的Chrome。Chrome速度快、对WebGL支持好、调试方便，是WebGL开发者首选浏览器。

在部署测试页面程序时，可以选择用IDE中的内置的调试方法，也可以直接放入如Tomcat等Web服务器中并用浏览器直接访问。

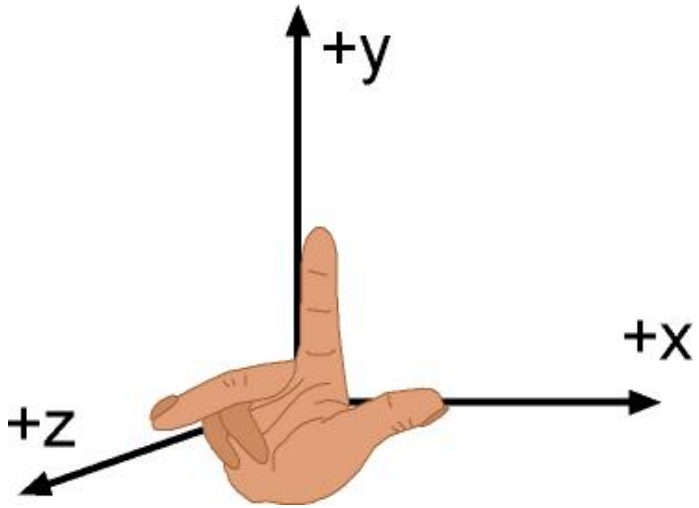
Chrome浏览器提供了内置的调试工具，非常方便。可以直接按F12打开开发工具窗口，其中提供了控制台、查看页面元素、断点跟踪、性能测试等功能（见下图）。



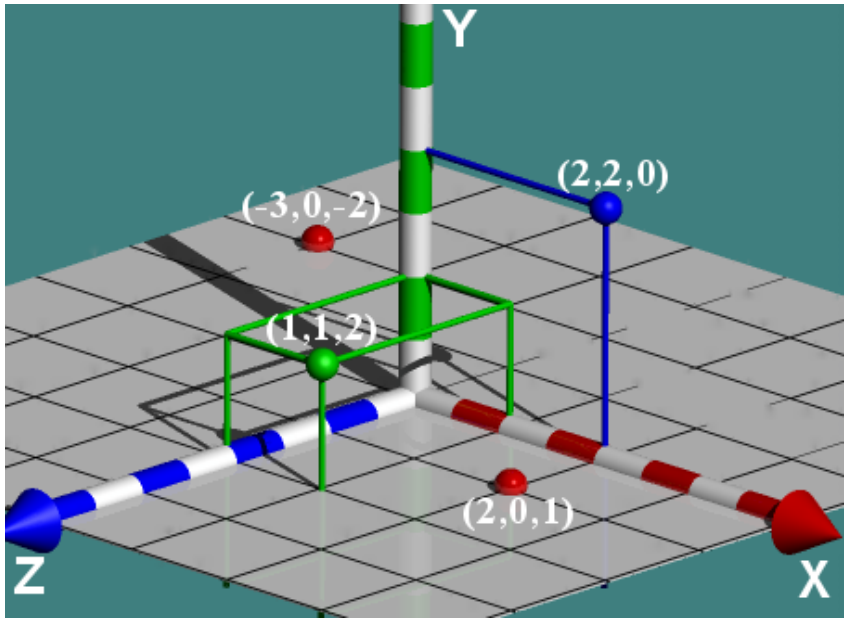
更多关于Chrome开发工具的功能介绍，请参考Google官方文档：https://developers.google.com/chrome-developer-tools/

坐标系统

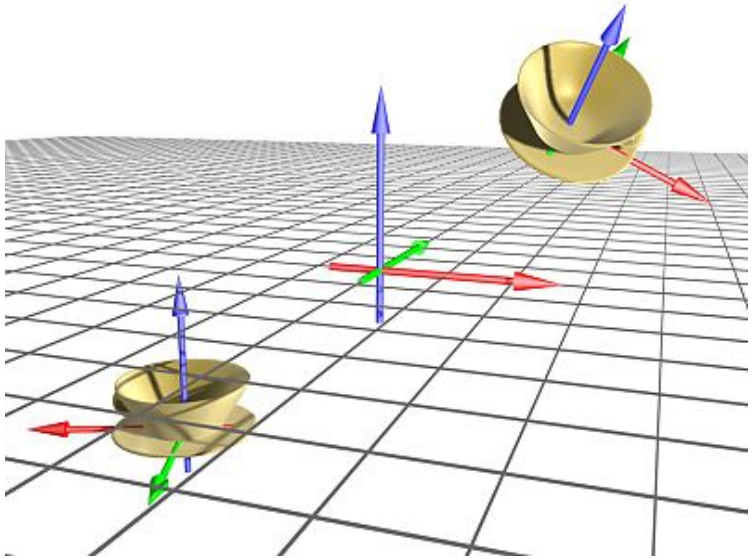
在mono中，3D场景的坐标是一个空间直角坐标系。直角坐标系是指在原点O，做三条互相垂直的数轴，它们都以O为原点，分别叫做x轴(横轴）、y轴(纵轴)、z轴(竖轴)，统称坐标轴。在mono中，坐标系方向遵循右手法则，即以右手拇指、食指、中指相互交叉，拇指指向x轴正方向，食指指向y轴正方向，则中指指向的即是z轴正方向。



在3D中，空间的一个点由(x, y, z)三个数值组成。下图展示了不同数值其具体表示的点的位置。熟练掌握坐标点的位置，对mono开发具有重要的作用。



此外，mono中还有世界坐标系和本地坐标系之分。世界坐标系是指所有3D物体所在的空间的大坐标系，而本地坐标系则是指以某一物体自己位置为原点的局部坐标系。例如一个飞速行走中的汽车，汽车的位置信息可以通过世界坐标系来定义其在地面空间的位置，而汽车内部的某个点则可以通过汽车的本地坐标系进行指定，而它和汽车本身的位置、旋转角度无关。



角度单位

在mono中，所有的角度数值都统一用弧度表示。即：Math.PI表示180度。例如：node.setRotation(Math.PI, Math.PI/2, Math.PI);

3D对象种类

Mono中，3D场景中的物体都是由各种3D基本形状的物体组成的。例如立方体、球体、圆柱体等等。这些最基本形状的物体由mono直接提供，它们可以通过进一步组合、运算等，形成现实世界中更复杂的物体。所以，mono开发中首先需要了解基本3D物体的种类和其特性。

目前，mono支持的基本3D物体有：

- mono.Cube：立方体。由6个面组成的立方体是最常见的3D物体。典型用法：设备、机架、仪表等
- mono.Sphere：球体。球体表面是一个曲面，由长宽高控制其尺寸。其曲面是由一系列纵向+横向均匀切分的小平面拼接而成，圆滑度取决于切分横向和纵向的分片数量，数量越大曲面越圆滑但性能越低。典型用法：灯泡、球形装饰物等
- mono.Cylinder：圆柱体。圆柱体表面是一个纵向直线、横向圆形的曲面组成，由长宽高控制其尺寸，还可设置顶面和底面是否封闭。需要留意的是，**圆柱的顶面和底面的尺寸可以不同**。侧面圆滑度由分片数量进行控制，分片数量越大侧面越圆滑但性能越低。圆柱可以有一些特别用法：圆锥（顶面半径为0）、金字塔形状（顶面半径0、侧面分片为4）、半金字塔形状（顶面半径为底面半径的一半、侧面分片为4）等等。可以通过灵活设置上下面的半径、侧面分片数量，来创建很多变种的圆柱体。典型用法：建筑物支撑柱、各种圆柱形物体
- mono.Torus：圆环。圆环是一个类似“手镯”形状的圆环物体。它的切面和形状都是圆形，其圆滑度都可通过切片数量进行

控制。此外，圆环还可以通过角度控制其是否封闭。360度的圆环是一个封闭的圆环，而180度的圆环则是一个被切去一半的半个圆环残体。如果将切面切片和环形切片都设置成3，则会形成一个类似三角管体组成的三角形形状体。典型用法：弧形门把手、弯管连接头体、各种圆环形状体等

- mono.PathNode：路径体。这是一种复杂的形状体，由两个任意形状进行控制：切面形状，以及前进走向。最终形状是该切面形状沿着前进path走向进行移动而形成的物体。例如，一个圆形切面沿着一个多边形移动，就会形成一个复杂的管线物体。这种形状还可以控制两个端头是否封闭、封闭的形状和尺寸，横切方向是否闭合、闭合角度、闭合样式等。通过控制这些参数，可以创建例如管线、弯管、香肠体、切开的管线等。典型用法：管线
- mono.TextNode：文字。将一串文字进行建模并形成一个复杂的3D物体。文字在3D中并没有直接的支持方式，所以和其他复杂形状一样，需要给出其具体的形状然后进行切片、分片来模拟。由于3D中没有直接的字体数据，需要额外提供字体的具体形状。在mono中没有内置任何字体信息，需要开发者额外去创建并引入到程序中。在MONO Design编辑器中，提供了几种英文字体形状信息，存储在类似“***_regular.typeface.js”的js文件中。例如，“helvetiker_regular.typeface.js”文件中存储了helvetiker字体的正常体信息，“helvetiker_bold.typeface.js”文件中存储了helvetiker字体的粗体信息，等等。如需要更多的字体，可以到网站http://typeface.neocracy.org/在线提交、生成和下载。一般一个英文字体对应的js文件大约在数百kb左右。由于中文字体形状复杂、字符数量巨大，一般不建议使用中文字符。典型用法：各种文字标识、设备标签
- mono.ShapeNode：形状块。一个任意Shape形状加厚度形成的实心物体。典型用法：地板、3D地图块图等
- mono.PathCube：路径方块。它的纵切面是一个矩形，并沿着一个路径方向进行建模。常见用法：房间的墙体等
- mono.Particle：粒子系统。一个粒子系统是一个特别的3D物体，它由很多内部的点组成，每个点有一个贴图。通过控制点的数量和位置，可以实现一种复杂运动的粒子效果。例如：模拟烟雾、火焰、喷水等。典型用法：设备烟雾、火焰、液体喷射物等
- mono.CSG：运算组合体。运算组合体是由多个3D物体进行组合运算得出的物体。例如立方体A合并球体B、球体A减掉圆柱体B。通过多次运算，可以创建以上3D物体无法实现的复杂物体。典型用法：现实世界中各种复杂不规则物体
- mono.Billboard：公告牌。公告牌是一种特殊的3D物体，它只有一个图片组成，而且这张图片会永远朝前面向镜头（也就是我们用户的眼睛），而无论场景如何变化。在游戏软件场景中，经常会有这样的场景：一个移动的物体（人、机器等）会在上方显示一个图片，图片上显示了文字、状态信息等，这个图片会永远朝向用户的眼睛，而无论物体如何移动。Billboard的这一特性使得它非常合适制作3D场景中的各种信息展示、文字说明、提示警告等。典型用法：设备告警、设备状态信息等

此外，mono还会持续的增加更多的基本3D物体。3D物体通过进一步的组合，可以形成更复杂的形状，更逼真的模拟现实世界中的各种物体。

通用对象属性

所有的3D物体都从mono.Element->mono.Node方向进行继承，所以它们有很多共同的基本属性。

常见的有：

位置、角度、比例

每一个3D对象都有空间位置、角度、比例属性。

- position：物体位置信息，通过get/set方法操作。默认位置为（0, 0, 0）。不同物体自身的原点位置略有差异，一般而言都是在物体的自身的中心点。例如立方体的原点在自己内部中心点。所以，当新建一个立方体并设置坐标为(0, 0, 0)时，立方体将有一半在水平坐标下方。用法举例：node.setPosition(100, 200, 300)
- rotation：物体在x、y、z轴向的旋转角度，通过get/set方法操作。默认值为（0, 0, 0）。例如，node.setRotation(0, Math.PI, 0)表示node在y轴方向旋转180度
- scale：物体在x、y、z轴方向的缩放比例，通过get/set方法操作。默认值为（1, 1, 1）。例如，node.setScale(1, 2, 1)表示node在y轴方向上放大到2倍。此时node的高度会变大（但是通过getHeight()获取的高度值height并未改变，仅仅是高度的比例增加了一倍）

Style样式

每个对象都有很多的样式属性，定义了它的外观、材质、颜色等等参数信息。mono中支持两种属性设置方法，一是通过setStyle/getStyle和一个key值方法进行存取。例如：node.setStyle('m.color', 'red')。二是使用CSS样式的设置方法，例如： node.s({'m.color' : '#FF6666','m.wireframe' : true});最常用的style是和材质相关的一些属性，这些属性都以'm.'前缀开头。以下列出了常用的一些style。

m.texture.image

材质贴图图片url。例如：node.setStyle('m.texture.image','../images/default_texture.png')。

m.type

材质的类型。可选'basic'或'phong'。其中，basic不支持光照效果，但效率高；phong支持光照效果，效率略低。

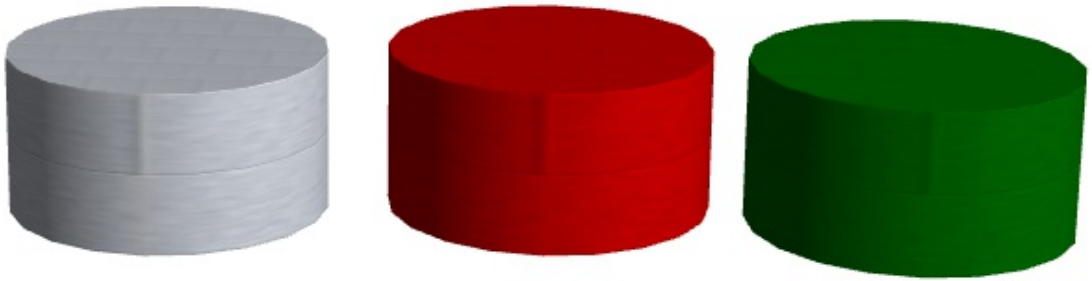
m.color

材质的染色，默认值为空。设置后，可以将物体以指定的颜色进行渲染。如果指定了贴图，则会用该颜色将贴图重新"染色"。注意：为了获得更透彻的染色效果，应同时设置'm.ambient'属性。

m.ambient

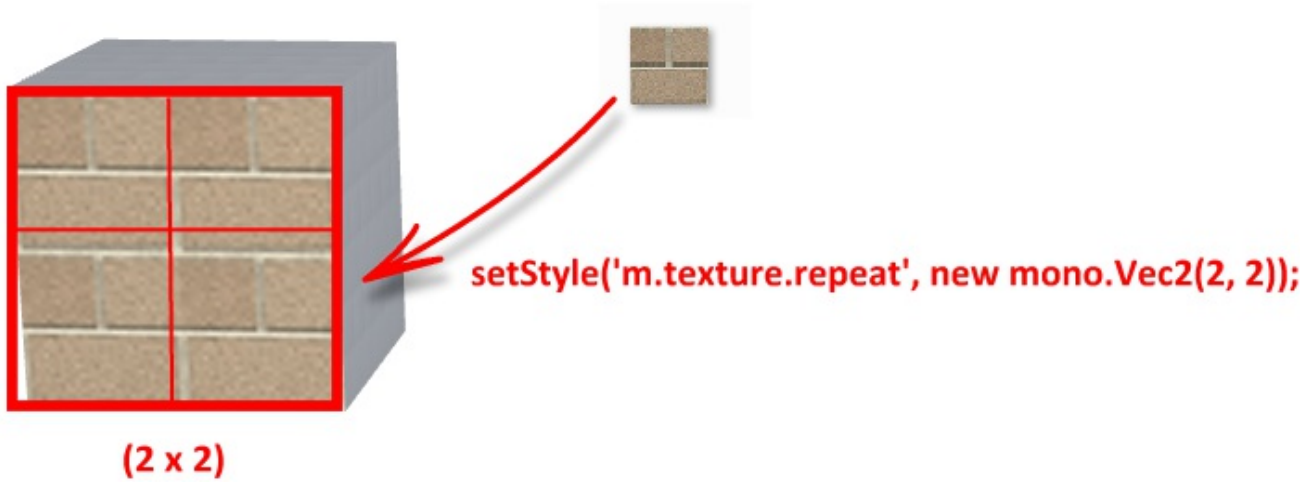
材质的光照反射颜色，默认为空。当未设置时，材质将用自然颜色反射环境光。例如，将此属性设置为绿色，则该物体将会反射出绿色的环境光。注意：为了获得更透彻的染色效果，应同时设置'm.color'属性。

下图是同时设置了m.color和m.ambient后展示的染色效果：



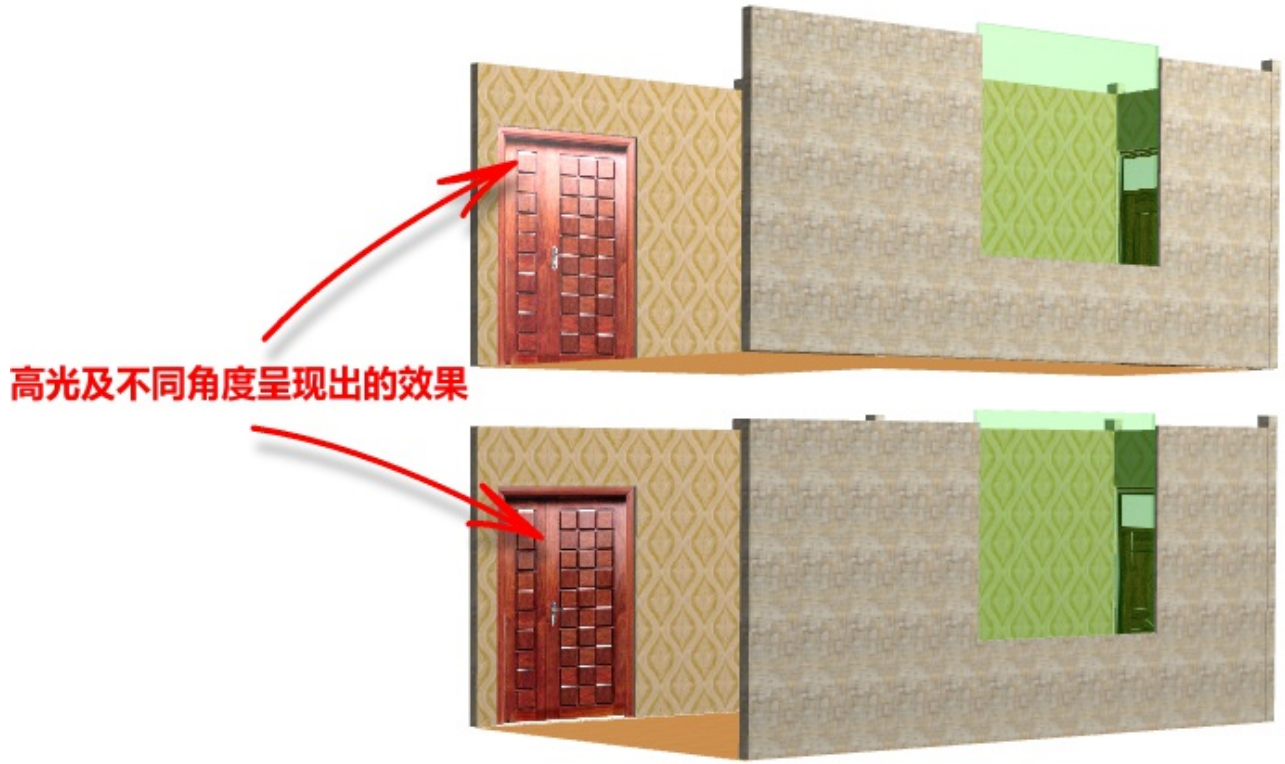
m.texture.repeat

材质的纹理重复次数，类型为mono.Vec2。默认情况下，贴图会在物体表面上进行横向和纵向的100%拉伸贴图显示。而设置了这一属性后，可以指定在横向和纵向上按照指定次数进行“重复”性的纹理绘制。例如，墙面、地板等物体，都可以通过这种方法对一张小图进行重复纹理。举例：node.setStyle(‘m.texture.repeat’,new mono.Vec2(10, 5))可将贴图在横向和纵向分别重复10次和5次。**注意：当使用本方法启用纹理贴图时，贴图图片的宽和高的尺寸规格必须为像素数为2^n（2的n次方，例如32、64、128、256、512、1024….）的数字，例如128×128或32×64的png图片。否则纹理会显示不正常。**



m.specularStrength

物体材质的镜面反射强度。当一个物体有一定的镜面反射效果时（例如半透明的玻璃、光亮的金属表面），可适当设置这一数值。数值越大，产生的镜面反射效果越强烈。对于一些无镜面反射的表面（例如水泥、木材等粗糙表面），应避免设置这一属性。

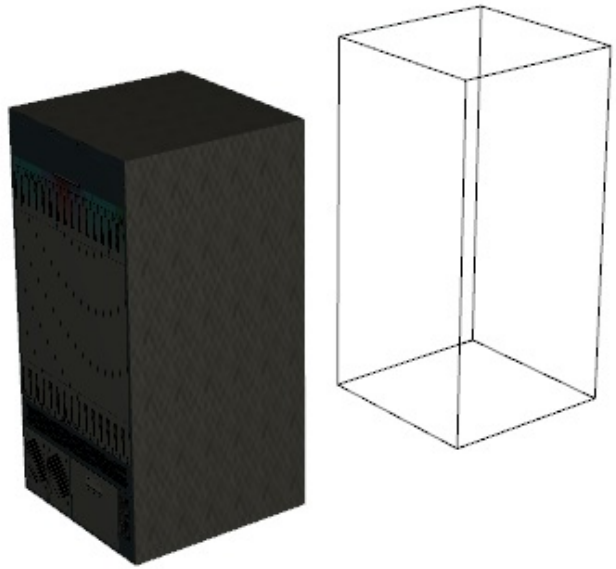


m.side

设置贴图为内面贴图、外面贴图还是双面贴图。默认为外面贴图。可选值为：mono.FrontSide外面贴图、mono.BackSide内面贴图、mono.DoubleSide双面贴图,双面贴图会在内，外两面都贴上图，会消耗更长的时间，如果3D对象不需要显示内外两面，可不设置为双面贴图。

m.wireframe

设置3D物体为线框类型，例如：node.setStyle('m.wireframe',true);



其他通用属性

所有3D对象还有很多其他通用属性，以下列出一些常见部分：

- setSelectable(selectable) – 是否对象可以鼠标点击选中

综合演示

以下代码综合演示了如何为对象设置一些常见属性：

```
1 var node = new mono.Node();
2 node.setPosition(10,10,10);
3 node.setRotation(Math.PI,0,0);
4 node.setScale(2,2,2);
5
6 node.setStyle('m.texture.image',pic);
7 node.setStyle('m.type','basic');
8 node.setStyle('m.side', TGL.DoubleSide);
```

使用基础工具类

在开发之前，开发者可以先熟悉一下mono中提供的一些很实用的基础类。这些类提供了一些基础的数学运算和数据操作方法，在以后使用时可以查阅。

t.js中提供的基础类介绍

- mono.Utils.createElement(element1,element2,material,materialIndexOffset) 在element1和element2的基础上创建1个新的Element对象
- mono.Utils.mergeElement(element1, element2, materialIndexOffset) 把element2合并到element1
- mono.Utils.autoAdjustNetworkBounds(network, o, w, h, left, top) 调整Network的Bounds
- mono.Utils.validateLicense(license) 验证license信息
- mono.Utils.isSame(element1,element2) 判断两个对象是否为同一对象
- mon.Mat3(n11, n12, n13, n21, n22, n23, n31, n32, n33) 3 x 3 矩阵
- mon.Mat4(n11, n12, n13, n14, n21, n22, n23, n24, n31, n32, n33, n34, n41, n42, n43, n44) 4 x 4矩阵
- mono.Vec2(x,y)二维向量
- mono.Vec3(x,y,z)三维向量

mono_toolkits.js提供的基础类介绍

mono_toolkits.js中的基础类为开发者提供了便捷操作3D对象，管理3D场景，以及应用3D特效的函数和对象并提供模板的导入，导出功能：

- mono.Toolkits.loadGraph(network,graphContents,parent,isAppend) 将房间图纸的json内容导入到3D场景中
- mono.Toolkits.loadGraphUrl(network,url,parent,isAppend,scope,callback) 通过url路径导入房间的图纸
- mono.Toolkits.loadTemplate(box3d, templateContents, parent) 将组件模板的json内容导入到box3d中
- mono.Toolkits.loadTemplateUrl(box3d, url, parent, scope, callback) 通过组件模板的url，将模板导到box3d中
- TemperatureBoard.addPoint(x,y,value)向云图组件注入采样数据
- TemperatureBoard.getTemperatureBoard()获得将采样点数据渲染为云图后的mono.Plane对象
- TemperatureBoard.getImage()获得采样点数据渲染为云图后的图片数据。可以直接用于Image对象的src属性
- LEDDisplay.display(value)显示指定的数字
- LEDDisplay.getView()获得LED数字模式组件的DOM对象

DataBox与Network

使用DataBox

DataBox是管理所有数据对象的容器，在MVC框架中处于M（模型）层，它是视图的数据提供者，它的主要功能是对数据进行装载、卸载，并对数据元素的变化进行监听。

数据操作

DataBox中的数据可以通过API创建，也可以通过XML或JSON的方式进行序列化和反序列化。DataBox中提供了许多对数据操作的API，以下列出了常用的API：

- add(data, index) 往数据容器中添加一个数据
- addByDescendant(data) 如果节点有孩子，就把节点的孩子也添加到数据容器中
- remove(data) 从数据容器中删除某个数据
- removeById(id) 通过数据的ID编号从数据容器中删除该数据
- contains(data) 判断是否包含某个数据
- containsById(id) 通过数据ID判断是否包含某个数据
- getDataById(id) 根据数据的ID编号获取对应的数据
- getDatas() 获取数据容器中的所有数据
- getDataAt(index) 获取数据容器中某个序号上的数据
- size() 数据容器的大小，也就是容器中数据的数量
- lightsSize() 数据容器中光源的数量
- getLights() 获取数据容器中所有的光源
- clear() 清空数据容器中所有的数据
- removeSelection 删除所有选中的数据
- clearSelection 对所有数据取消选中
- clearEditing 对所有数据取消编辑

以下一段程序展示了对Databox中数据的增删改查：

```
1  var box = new mono.DataBox();
2  for(var i = 0; i < 3; i++) {
3     var element = new mono.Element("element_" + i);
4     element.setName("element_" + i);
5     box.add(element);
6  }
7  box.forEach(function(element) {
8     console.log(element.getName()); //element_0,element_1,element_2
9  });
10 var element0 = box.getDataAt(0);
```

```
element0.setName("newElement");
console.log(element0.getName());//newElement
box.remove(element0);
console.log(box.size());//3
box.clear();
console.log(box.size());//0
```

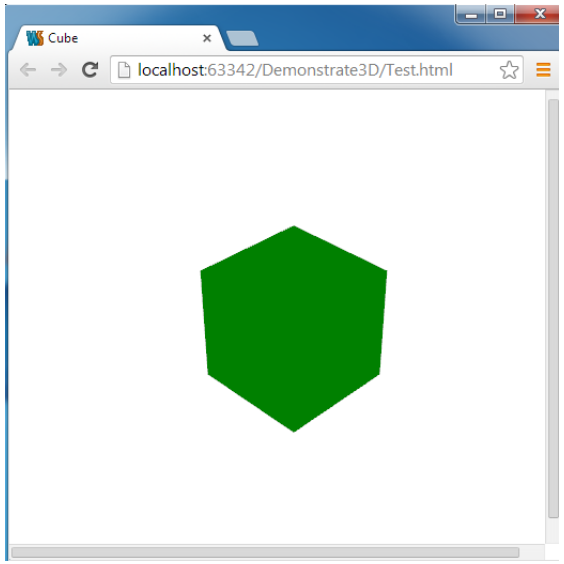
如果想要呈现优美的3D场景，还需要DataBox和Network（视图层）关联，通过模型驱动视图来展现3D场景：

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>Cube</title>
  <script type="text/javascript" src = "t.js"></script>
  <script type="text/javascript">
    function init(){
      var box = new mono.DataBox();
      var camera = new mono.PerspectiveCamera(30, 1.5, 0.1, 10000);
      camera.setPosition(-800,800,-800);
      camera.look(new mono.Vec3(0, 0, 0));

      var network= new mono.Network3D(box, camera, myCanvas);
      var interaction2 = new mono.SelectionInteraction(network);
      var interaction=new mono.DefaultInteraction(network);
      interaction.zoomSpeed=5;
      var interactions = [interaction, interaction2];
      network.setInteractions(interactions);
      mono.Utils.autoAdjustNetworkBounds(network,document.documentElement,'clientWidth','clientHeight');

      var cube = new mono.Cube(200,200,200);
      cube.setStyle('m.type','basic');
      cube.setStyle('m.color', "green");
      cube.setPosition(0,0,0);
      box.add(cube);
    }
  </script>
</head>
<body onload = "init()">
<div id = "mainDiv"><canvas id="myCanvas" /></div>
</body>
</html>
```

运行效果图：



事件监听

DataBox可以对容器中的数据对象的变化进行监听，DataBox提供了以下的事件监听器供用户使用：

- addDataBoxChangeListener(listener, scope, ahead) 当数据容器中的数据发生改变时（增加，删除，清空），就可以通过此方法监听
- removeDataBoxChangeListener(listener, scope) 移除数据容器的数据增减变化的监听器
- addDataPropertyChangeListener(listener,scope,ahead) 当数据容器中的数据属性发生变化时，比如网元位置，网元名称等，都可以通过此方法来监听
- removeDataPropertyChangeListener(listener, scope) 移除数据容器中数据的属性更改事件的监听器

以下一段程序展示了监听器的使用：

```
1 var box = new mono.DataBox();
2 var element = new mono.Element();
3 box.addDataBoxChangeListener(function (e) {
4   console.log(e.kind); // "add","remove"
5 });
6 var element2 = new mono.Element();
7 box.add(element2);
8 box.remove(element);
```

使用Network组件

Network3D是一个用于交互的视图组件，可以展示3D场景，并实现用户和3D场景之间的交互，比如旋转镜头，选中3D对

象，通过鼠标或键盘移动3D对象等。

在网页中添加Network

在网页中插入Network3D，首先自定义一个div，并创建一张画布canvas:

```
<div id = "mainDiv" >
    <canvas id="myCanvas" width="800" height="800"/>
</div>
```

Network3D需要由一个与之关联的DataBox驱动，显示DataBox中的网元。当初始化Network3D时，DataBox默认会绑定在Network3D上。通过network3D构造方法创建network3D，并与DataBox、Camera和canvas绑定：

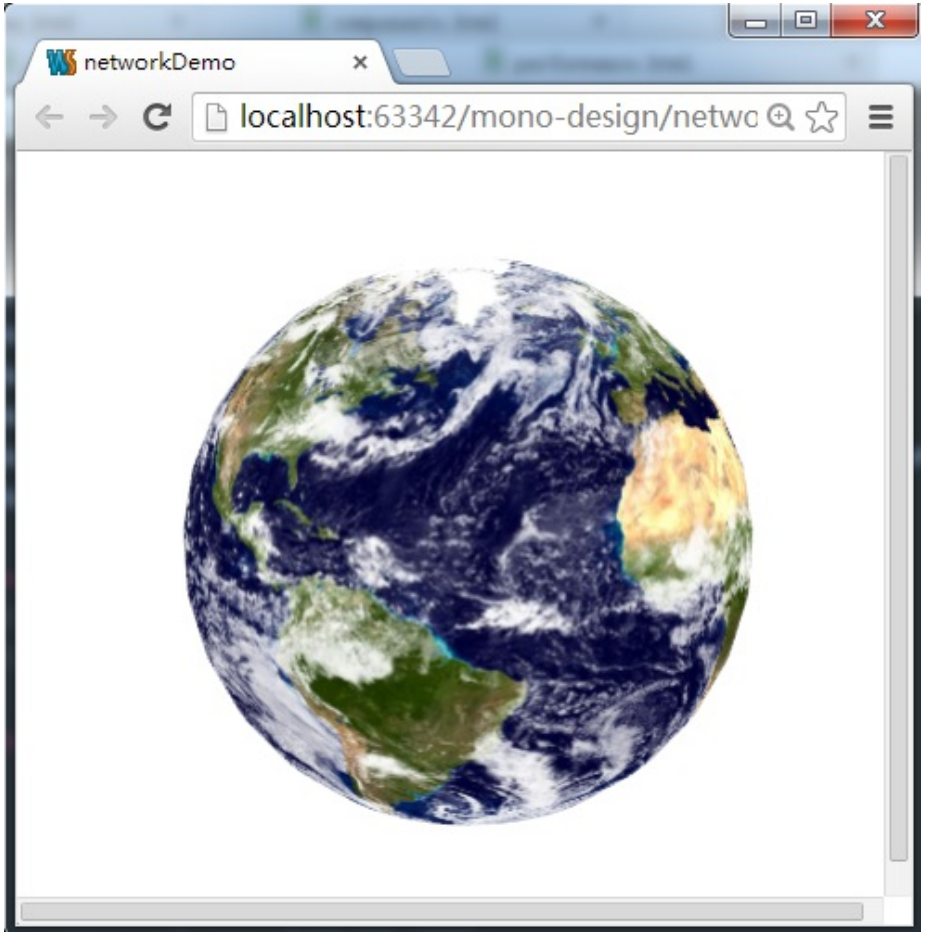
```
//构造方法
Network3D=function(dataBox,camera,canvas,parameters){};
```

下面的例子在html中创建了一个network3D，并展现出一个3D物体。

```
<!DOCTYPE html>
<html>
<head>
<title>Room Inspection</title>
<script type="text/javascript" src = "libs/t.js"></script>
<script type="text/javascript" src = "libs/twaver.js"></script>
<script type="text/javascript">
var network, interaction;
function load(){
    var box = new mono.DataBox();
    var camera = new mono.PerspectiveCamera(30, 1.5, 0.1, 10000);
    var target = new mono.Vec3(150,50,150);
    camera.setPosition(1000,500,1000);
    camera.lookAt(target);
    network = new mono.Network3D(box, camera, myCanvas);
    mono.Utils.autoAdjustNetworkBounds(network,document.documentElement,'clientWidth','clientHeight');
    box.add(new mono.AmbientLight(0xffffff));
    createBall(box);
}
function createBall(box){
    var ball=new mono.Sphere(300,100);
    ball.setStyle('m.texture.image','images/earth2.png');
    ball.setStyle('m.type','phong');
    ball.setStyle('m.texture.repeat',new mono.Vec2(1,1));
    ball.setStyle('m.specularStrength',100);
    box.add(ball);
}
    </script>
```

```
</head>
<body onload = 'load()'>
    <div id = "mainDiv" >
        <canvas id="myCanvas" width="800" height="800"/>
    </div>
</body>
</html>
```

执行代码后运行效果如图。



Interaction交互模式

mono中提供了DefaultInteraction、SelectIonInteraction、EditInteraction等三种交互模式。DefaultInteraction交互模式包括可以在3D场景中旋转镜头，通过鼠标滚轮缩放镜头，键盘操作镜头等。设置SelectionInteraction交互模式，3D场景中的对象可以被选中，也可以按住ctrl键进行框选；在EditInteraction模式下可以编辑3D对象，例如平移，缩

放，旋转等。

也可以通过setInteractions方法为network同时设置多种交互模式。

- DefaultInteraction.getRotateSpeed():获取镜头旋转的速度
- DefaultInteraction.setRotateSpeed(rotateSpeed):设置镜头旋转的速度
- DefaultInteraction.getZoomSpeed():获取镜头缩放的速度
- DefaultInteraction.setZoomSpeed(zoomSpeed):设置镜头缩放的速度
- DefaultInteraction.getPanSpeed():获取镜头平移的速度
- DefaultInteraction.setPanSpeed(panSpeed)设置镜头平移的速度
- EditInteraction.setShowHelpers(showHelpers):设置是否显示帮助的组件
- EditInteraction.setScaleable(scaleable):设置是否可缩放
- EditInteraction.setRotateable(rotateable):设置是否可旋转
- EditInteraction.setTranslateable(translateable):设置是否可平移
- EditInteraction.setDefaultMode(defaultMode):设置默认的模式，mono中提供了三种模式：translate,rotate和scale
- EditInteraction.setScaleRate(scaleRate):设置缩放率
- EditInteraction.setForceSameScale(forceSameScale):设置是否需要等比例缩放

下面代码为network设置了DefaultInteraction和SelectionInteraction两种交互模式。

```
1 network= new mono.Network3D(box, camera, myCanvas);
2 interaction = new MONO.DefaultInteraction(network);
3 interaction.zoomSpeed = 30;
4 interaction.yLowerLimitAngle = Math.PI/180;
5 interaction.yUpLimitAngle = Math.PI / 3;
6 network.setInteractions([new MONO.SelectionInteraction(network), interaction]);
```

自适应缩放

自适应缩放会自动调整Network3D的Bounds值，使3D物体在network中的呈现更加友善。可通过如下方法实现：

```
1 autoAdjustNetworkBounds = function(network, o, w, h, left, top) {}
```

鼠标键盘交互事件

mono中为我们提供了非常友善的用户鼠标键盘交互事件，下面列出常用的事件。

鼠标交互事件

- 按住左键移动：旋转物体
- 按住右键移动：平移物体

- 滚轴：缩放物体

键盘交互事件

- pageup:沿着z轴正方向移动
- pagedown:沿着z轴负方向移动
- left:沿着x轴正方向移动
- up:沿着x轴负方向移动
- right:沿着y轴正方向移动
- down:沿着y轴负方向移动
- ctrl+A:全选
- ctrl+C:复制
- ctrl+V:粘贴

自动渲染

我们可以通过setRenderSelectFunction()方法过滤哪些选中的网元需要绘制选择边框，哪些不需要绘制边框。

```
1 setRenderSelectFunction = function(f) {}
```

我们还可以在分别监听到渲染前后，使用setBeforeRenderFunction()设置渲染Network3D前的执行方法，使用setRenderCallback()来设置渲染Network3D之后的回调方法。方法原型如下：

```
1  mono.Network3D.prototype.setRenderCallback = function(f) {}
2  //调用setRenderCallback方法
3  network.setRenderCallback(function(){
4     //do something
5  });
6
7  mono.Network3D.prototype.setBeforeRenderFunction = function(f) {}
8  //调用setBeforeRenderFunction方法
9  network.setBeforeRenderFunction(function(){
10 //do something
11 });
```

说点什么吧...

发布

TWaver Documents正在使用多说