

---

# Table of Contents

简介	1.1
鉴权	1.2
认证	1.2.1
告警API	1.3
增加告警	1.3.1
更新告警	1.3.2
删除告警	1.3.3
指标数据API	1.4
推送指标数据	1.4.1
添加湿度监控	1.4.2
资产API	1.5
增加资产	1.5.1
更新资产	1.5.2
删除资产	1.5.3
查询资产	1.5.4
获取资产及扩展字段	1.5.5
同步资产方案	1.6
增加临时资产	1.6.1
删除临时资产	1.6.2
附录一	1.7

# 简介

文档中代码是Javascript, Ajax访问API, \$代表JQuery, demo中的代码是放在3D机房系统发布包的resource/public下访问

3D机房系统提供了标准的（RESTfulwebservice）开放式的应用程序接口（API），供第三方调用 RESTfulwebservice：通过统一资源标示符（URL）来识别和定位资源，并且针对这些资源而执行的操作是通过HTTP规范定义。其核心操作只有GET、PUT、POST、DELETE。

接口原则：

1. 允许数据使用者，通过标准的接口访问3D机房系统数据
2. 允许数据提供者，通过标准的接口增加数据到3D机房系统

3D机房系统包含资产管理模块，动力环境监控模块功能模块。

# 鉴权

系统的安全性，数据的安全一直是用户关心的问题。由于3D可视化系统主要负责的是呈现，并多在内网中使用，所以这部分一直没有开放出来。最近有客户提到，就把这块慢慢完善起来。

接口部分使用了JWT(JSON Web Token)认证机制，在访问接口之前，需要先认证获取token（参考认证章节），并在之后的所有接口访问中带上这个token，为了获得最大的可扩展性，我们允许客户端使用一下3个方法附加我们的token：作为请求链接（query）的参数，作为主体的参数（body），和作为请求头（Header）的参数。前两种参数名为access-token，最后一个，将使用Header x-access-token。

如果没有认证，在访问接口的时候会返回：

```
{"error": "must be authorized"}
```

如果认证过期，在访问接口的时候会返回：

```
{"error": "jwt expired"}
```

服务器并没有做refresh token，所以当token过期后需要再次认证

接口的鉴权默认并没有启用，但是在后端提供开关，config.js中auth属性，如果需要启用，将auth值设为true

## 总结使用方法：

1. 在后端config.js中将auth值设为true，开启鉴权开关
2. 通过认证接口，认证并获取token
3. 访问其他接口并带上token，携带token三种方式，query、body、Header（x-access-token）

# 认证

**POST** `http://{serverhost:serverport}/api/monitor/auth`

## Body

用户名和密码

Accept: application/json

```
{"username": "", "password": ""}
```

## Returns

Content-Type: application/json

```
//成功
{"access_token": ""}

//失败
{error: "失败原因"}
```

# 告警API

告警API包含：增加、更新、删除、查询。模拟需求场景，对4种API进行说明。

需求场景：编号为rack84\_E01设备，可能产生三种告警：温度告警，网络告警，物理告警

## 告警字段列表

字段	类型	必需	描述
realId	string	是	告警的唯一标识，与接口中alarmID对应
dataId	string	是	发生告警的设备编号，与接口中deviceID对应
alarmTypeId	string	是	对应告警类型中的编号，参考告警类型模块，与接口中alarmType对应
level	string	是	对应告警级别中的编号，参考告警级别模块
devIp	string		发生告警的IP
time	date		告警发生时间
ackTime	date		告警确认时间
ackNotice	string		确认告警时备注
description	string		描述
client	Object		扩展的字段

# 增加告警

**POST** `http://{serverhost:serverport}/api/monitor/alarms`

## Body

一个或多个告警实例

Accept: application/json

```
{  
    module: string, 模块名, 保留字段, 缺省值“PEMS”,  
    data: [  
        {  
            alarmId: string, required, 告警的唯一标识,  
            deviceId : string, required, 发生告警的设备,  
            alarmType: string, required, 对应告警类型中的编号, 参考告警类型模块,  
            level: string, required, 对应告警级别中的编号, 参考告警级别模块,  
            description: string, 告警描述,  
            time: date, 告警发生时间,  
            ackTime: date, 告警确认时间,  
            ackNotice: string, 确认告警时备注,  
            devIp: string, 发生告警的IP,  
            client: {} Object, 扩展的字段  
        }  
    ]  
}
```

## Returns

异步操作, 返回结果仅表示推送到3D机房系统后端

Content-Type: application/json

验证成功

```
{  
    value:"success",  
    error: null  
}
```

某种原因验证失败

## 增加告警

```
{  
    value: "fail",  
    error: "原因",  
    items: [{}],数组元素为，错误的告警对象，但是不一定会有，只有在添加告警做验证不通过时有值  
}
```

## 多种原因验证失败

```
{  
    value: "fail",  
    error: "verification failed",  
    items: [{  
        reason: '',  
        items: [{}],数组元素为，错误的告警对象，但是不一定会有，只有在添加告警做验证不通过时有值  
    }]  
}
```

## Demo

```
var data = {  
    module: "PEMS",  
    data: [  
        {  
            alarmId: '123',  
            deviceId : 'rack84_E01',  
            alarmType: '温度告警',  
            level: 'critical',  
            description: '湿度过高',  
            time: new Date(),  
            devIp:'127.0.0.1',  
            client: {'告警值':70}  
        },  
        {  
            alarmId: '234',  
            deviceId : 'rack84_E01',  
            alarmType: '网络告警',  
            level: 'critical',  
            description: '网络端口',  
            time: new Date(),  
            devIp:'127.0.0.1',  
            client: {'告警值':'disconnect'}  
        },  
    ]  
};  
$.post('/api/monitor/alarms', data, function(data, textStatus, xhr) {  
    /*optional stuff to do after success */  
});
```

增加告警

---

# 更新告警

PUT

http://{serverhost:serverport}/api/monitor/alarm/ [id]

## Path parameters

name	type	description
id	string	推送方的告警的唯一标识

## Body

要更新的告警属性，通常在确认告警或修改告警状态时，通过更新方法

Accept: application/json

```
{  
    deviceId : string, required, 发生告警的设备,  
    alarmType: string, required, 对应告警类型中的编号, 参考告警类型模块,  
    level: string, required, 对应告警级别中的编号, 参考告警级别模块,  
    description: string, 告警描述,  
    time: date, 告警发生时间,  
    ack_time: date, 告警确认时间,  
    ack_notice: string, 确认告警时备注,  
    devIp: string, 发生告警的IP,  
    client: {} Object, 扩展的字段  
}
```

## Returns

异步操作，返回结果仅表示推送到3D机房系统后端

Content-Type: application/json

```
{value:"success", error: null}
```

## Demo

以增加告警章节demo中添加的编号为123的告警为例

```
var data = {
    "ackTime": "2016-12-30",
    "ackNotice": "已转交其他同事处理",
    "client": {
        "确认时间": "2016-12-30 9",
        "告警确认人": "admin",
        "确认内容": "已转交其他同事处理"
    },
    "status": "ack"
}
$.ajax({
    url: '/api/monitor/alarm/123',
    type: 'PUT',
    data: data,
    success: function(result) {
        // Do something with the result
    }
});
```

# 删除告警

**DELETE**

`http://{serverhost:serverport}/api/monitor/alarm/ [id]`

## Path parameters

<b>name</b>	<b>type</b>	<b>description</b>
<code>id</code>	<code>string</code>	推送方的告警的唯一标识

## Body

无内容

## Returns

异步操作，返回结果仅表示推送到3D机房系统后端

Content-Type: application/json

```
{value:"success", error:null}
```

## Demo

以增加告警章节demo中添加的编号为123的告警为例

```
$.ajax({
  url: '/api/monitor/alarm/123',
  type: 'DELETE',
  success: function(result) {
    // Do something with the result
  }
});
```

# 指标数据API

指标数据是指传感器或设备上的监控数据，例如温湿度、电流电压、各种开关量等数据。3D机房系统用户可以在系统查看此类数据，并且实时变化。

与3D机房系统进行数据交互有两种方式，过程如下：

- 动环系统 -----推-----> 3D机房系统后端 -----推-----> 3D机房系统前端 ----->  
动态刷新
- 动环系统 <-----拉----- 3D机房系统后端 -----推-----> 3D机房系统前端 ----->  
动态刷新

# 推送指标数据

**POST** `http://{serverhost:serverport}/api/monitor/data`

## Body

一个或多个业务对象指标数据， deviceId作为一个对象，由其指标属性和值组成键值对

Accept: application/json

```
{  
    module: string, 模块名, 保留字段, 缺省值“PEMS”,  
    data: {  
        'deviceId1': {  
            "property1": value1,  
            "property2": value2,  
            "property3": value3,  
            ... ...  
        },  
        'deviceId2': {  
            "property1": value1,  
            "property2": value2,  
            "property3": value3,  
            ... ...  
        }  
    }  
}
```

- **deviceId**: 指推送的业务对象唯一标识或资产对象唯一标识，确切的来说都称为业务对象唯一标示，只是在特定的情况直接将资产对象看作为业务对象。更多细节参考业务对象模块
  - **property**: 指业务对象的属性，也就是指标属性，
  - **value**: 指对应的指标属性值

## Returns

异步操作，返回结果仅表示推送到3D机房系统后端

Content-Type: application/json

```
{value:"success", error: null}
```

## Demo

```
var getData = function(){
    return {
        module: "PEMS",
        data: {
            'b8f7th02': {
                'humidity':Math.floor(Math.random()*10),
                'temperature':Math.floor(Math.random()*10)
            },
            'b8r701c06hr01': {
                'humidity':Math.floor(Math.random()*10),
                'temperature':Math.floor(Math.random()*10),
                "网络":{
                    "状态":"接通",
                    "速度":"100m/s"
                }
            }
        }
    };
}

var data = getData();
$.post('api/monitor/data', data, function(data, textStatus, xhr) {
    /*optional stuff to do after success */
});
```

# 添加湿度监控

先描述下3D机房系统的流程，当显示湿度信息的时候，

1. 找到当前场景中资产对象
2. 找到资产对象的传感器，并且传感器的类型为湿度，资产对象跟传感器为父子关系
3. 根据传感器的ID找到其关联关系，将关联关系中需要的业务对象及其属性组织为订阅订单
4. 将订阅订单发送服务器端订阅数据，等待服务器推送
5. 服务器响应后，根据缓存反向查寻，定位传感器，并更新传感器的值

综上所述，分析得知需要添加的数据有：

## 1. 资产对象

湿度是房间的属性，创建一个房间的资产对象

```
{  
  id:"f2-r1",  
  description:"二楼房间一",  
  parentId:"floor2",  
  dataTypeId:"area01"  
}
```

## 2. 传感器

添加一个湿度传感器

```
{  
  id:"humidity01",  
  description:"湿度传感器",  
  parentId:"f2-r1",  
  type:"humidity",  
  position:{x:3,y:100,z:300}  
}
```

## 3. 业务对象及其属性

添加一个湿度传感器的业务对象

```
{  
  id:"hum01",
```

## 添加湿度监控

```
description:"湿度传感器",
type:"SRH"
}
```

## 添加业务对象属性

```
{
  type:"SRH",
  property:"humidity",
  valueType:"number",
  unit:"%rh"
}
```

## 4. 关联关系

### 添加传感器与业务对象之间的关联关系

```
{
  id:"humidity01",
  relations:[{"bid":"hum01","property":"humidity"}]
}
```

## 5. 第三方推送的JSON数据格式:

```
{
  module:"PEMS",
  data:{
    "hum01":{
      "humidity":60
    }
  }
}
```

# 资产API

对于资产的管理，大部分的资产数据在初始化的系统的时候，会将资产数据整理导入系统。对于系统上线后，对于少量的资产操作可以使用资产API，通过资产API对资产进行操作可以实时同步到前端浏览器

## 资产字段列表

字段	类型	必需	描述
id	string	是	编号
name	string	是	名称
description	string		描述
position	string		参考附录一：location、position、空间规划
position2d	string		2D中的物理坐标
rotation	string		表示资产对象的旋转值，{x:0,y:0,z:0}，x、y、z分别表示三个方向上的旋转角度
location	string		参考附录一：location、position、空间规划
parentId	string	是	父对象，例如设备的父对象为机柜
dataTypeld	string	是	资产模型编号
weight	int		资产重量

# 增加资产

**POST** `http://{serverhost:serverport}/api/monitor/asset`

## Body

一个或多个资产实例

Accept: application/json

```
{  
  data: [  
    {  
      id: string, required, 编号,  
      name : string, required, 名称,  
      description: string, 描述,  
      position: string, 物理坐标, 值是{x:0,y:0,z:0}的JSON字符串, 参考附录一: location、position、空间规划,  
      position2d: string, 2D中的物理坐标,  
      rotation: string, 表示资产对象的旋转值, {x:0,y:0,z:0}, x、y、z分别表示三个方向上的  
      旋转角度,  
      location: string, 逻辑坐标, 值是{x:0,y:0,z:0}的JSON字符串, , 参考附录一: location  
      、position、空间规划,  
      parentId: string, required, 父对象, 例如设备的父对象为机柜,  
      dataTypeId: string, required, 资产模型编号,  
      weight: int, 资产重量,  
      customField: {}, 资产的扩展字段, 扩展字段是存在根据资产分类创建的扩展表中。customFi  
eld中内容为扩展表中的字段  
    }  
  ]  
}
```

## Returns

Content-Type: application/json

添加成功

```
{  
  value: 添加的资产对象(JSON格式),  
  error: null  
}
```

## 增加资产

---

### 添加失败

```
{  
    value: "fail",  
    error: "原因",  
    item: 错误的资产对象，但是不一定会有，只有在添加资产做验证不通过时有值  
}
```

## Demo

```
var data = {  
    module: "PEMS",  
    data: [  
        {  
            id: 'b8r201c04r23e01',  
            name: 'b8r201c04r23e01',  
            location: {"y":28,"z":"pos_pos"},  
            parentId: 'b8r201c04r23',  
            dataTypeId: 'equipment2',  
            customField: {  
                ipAddr: '192.168.1.11',  
                brand: '华为'  
            }  
        }  
    ]  
};  
$.post('/api/monitor/asset', data, function(data, textStatus, xhr) {  
    /*optional stuff to do after success */  
});
```

# 更新资产

**PUT**

http://{serverhost:serverport}/api/monitor/asset/ [id]

## Path parameters

name	type	description
id	string	资产的唯一标识

## Body

要更新的告警属性，通常在确认告警或修改告警状态时，通过更新方法

Accept: application/json

```
{
  name : string, required, 名称,
  description: string, 描述,
  position: string, 参考附录一: location、position、空间规划,
  position2d: string, 2D中的物理坐标,
  rotation: string, 表示资产对象的旋转值, {x:0,y:0,z:0}, x、y、z分别表示三个方向上的旋转角度,
  location: string, 参考附录一: location、position、空间规划,
  parentId: string, required, 父对象, 例如设备的父对象为机柜,
  dataTypeId: string, required, 资产模型编号,
  weight: int, 资产重量
}
```

## Returns

Content-Type: application/json

```
//成功
{value:'success', error: null}
//失败
{value:更新的资产对象(JSON格式), error:'原因'}
```

## Demo

以增加资产章节demo中添加的编号为b8r201c04r23e01的资产为例

```
var data = {
    name: 'b8r201c04r23e01-update',
    description: 'b8r201c04r23e01-update',
    location: {"y":10,"z":"pos_pos"},
    parentId: 'b8r201c04r23',
    dataTypeId: 'equipment2'
}
$.ajax({
    url: '/api/monitor/asset/b8r201c04r23e01',
    type: 'PUT',
    data: data,
    success: function(result) {
        // Do something with the result
    }
});
```

# 删除资产

**DELETE**

`http://{serverhost:serverport}/api/monitor/asset/ [id]`

## Path parameters

name	type	description
id	string	资产的唯一标识

## Body

无内容

## Returns

Content-Type: application/json

```
{value:删除的资产对象(JSON格式), error:null或原因}
```

## Demo

以增加资产章节demo中添加的编号为b8r201c04r23e01的资产为例

```
$.ajax({
  url: '/api/monitor/asset/b8r201c04r23e01',
  type: 'DELETE',
  success: function(result) {

  }
});
```

# 获取资产

**GET** `http://{serverhost:serverport}/api/monitor/asset`

## Query parameters

<b>name</b>	<b>type</b>	<b>description</b>
id	string	资产的唯一标识
name	string	名称
description	string	描述
parentId	string	父对象, 例如设备的父对象为机柜
dataTypeId	string	资产模型编号
categoryId	string	分类编号

## Body

无内容

## Returns

Content-Type: application/json

```
{value:{count:'查询结果条目数',rows:array}, error: null或原因}
```

array为数组, 数组元素包含资产的所有字段, 以及资产的所有扩展字段, 由于扩展字段由资产分类来决定, 所以资产的扩展字段不尽相同

## Demo

```
$.ajax({
  url: '/api/monitor/asset',
  data: {categoryId:'airConditioning', parentId:'r101'},
  type: 'GET',
```

## 查询资产

---

```
success: function(result) {  
    console.log(result);  
}  
});
```

# 获取资产及扩展字段

**GET**

`http://{serverhost:serverport}/api/monitor/asset/customField`

## Query parameters

名称	类型	描述
category	string	资产的分类，资产的扩展字段是根据资产分类来设置的，相同分类的资产的扩展字段相同。
assetProps	JSON string	资产的查询条件，包含：id、name、description、parent_id、data_type_id、business_type_id
customProps	JSON string	扩展字段的查询条件，能包含的查询条件为具体的扩展字段，例如扩展字段有ip字段，就可以将ip作为查询条件

## Body

无内容

## Returns

Content-Type: application/json

```
//成功
{value: array, error: null}
//失败
{value: null, error: '原因'}
```

array为数组，数组元素包含资产的所有字段，以及资产的所有扩展字段，由于扩展字段由资产分类来决定，所以资产的扩展字段不尽相同

## Demo

以增加资产章节demo中添加的编号为b8r201c04r23e01的资产为例

```
$ajax({
  url: '/api/monitor/asset/customField',
  data: {assetProps:JSON.stringify({id:'b8r201c04r23e01'})},
  type: 'GET',
  success: function(result) {
    console.log(result);
  }
});
```

# 同步资产方案

客户在用3D可视化系统做展示的时候，可能已经存在一个运维系统，3D系统对接的数据主要来源于这个运维系统。那么这里就有了资产同步的问题，下面是资产同步方案：

## 1. 3D可视化系统同步给客户的系统

实现机制是3D可视化系统在对资产的操作会通过websocket发出事件，客户监听此事件进行同步。但是有些客户的习惯是从他们系统同步到3D系统，不接受这个方案

## 2. 客户的系统同步给3D可视化系统

实现方式是客户直接调用3D可视化系统的接口，3D可视化系统同步展示。但是3D可视化系统在展示资产的时候需要位置、模型、父子关系等，这些信息客户不方便维护

由于以上两种方案存在的不足，所以提出第三种同步方案，在此方案中跟方案二一样还是客户把资产数据同步到3D系统，但是位置、模型、父子关系信息不需要客户提供，而是到在可视化的3D界面界面中选择模型，拖拽定位。

同步方案目前仅针对添加资产，删除资产可以直接调用删除资产的接口即可

# 增加临时资产

**POST**

`http://{serverhost:serverport}/api/monitor/tempAsset`

## Body

一个或多个需要同步到系统中的资产实例

Accept: application/json

```
{  
  data: [  
    {  
      id: string, required, 编号,  
      name : string, required, 名称,  
      description: string, 描述,  
      location: int, 给设备使用, 描述设备所在的U位,  
      parentId: string, 父对象, 给设备使用,  
      height: int, 给设备使用, 表示设备所占的U数,  
      isEquipment: boolean, 所添加资产是否为设备  
      extend: {}, 扩展的字段, 这些扩展字段最终将存到对应的Category的扩展表中  
    }  
  ]  
}
```

## Returns

Content-Type: application/json

添加成功

```
{  
  value: 添加的资产对象(JSON格式),  
  error: null  
}
```

添加失败

```
{  
  value: "fail",  
  error: "原因",
```

```
    item: 错误的资产对象，但是不一定会有，只有在添加资产做验证不通过时有值
}
```

## Demo

```
var data = {
  module: "PEMS",
  data: [
    {
      id: 'b8r201c04r23e01',
      name: 'b8r201c04r23e01',
      extend: {
        ipAddr: '192.168.1.11',
        brand: '华为'
      }
    }
  ]
};

$.post('/api/monitor/tempAsset', data, function(data, textStatus, xhr) {
  /*optional stuff to do after success */
});
```

# 删除临时资产

**DELETE**

http://{serverhost:serverport}/api/monitor/tempAsset/ [ [id](#) ]

## Path parameters

name	type	description
id	string	资产的唯一标识

## Body

无内容

## Returns

Content-Type: application/json

{value:删除的资产对象(JSON格式), error:null或原因}

## Demo

以增加资产章节demo中添加的编号为b8r201c04r23e01的资产为例

```
$.ajax({
  url: '/api/monitor/tempAsset/b8r201c04r23e01',
  type: 'DELETE',
  success: function(result) {
    }
});
```

# 附录一

## location、position、空间规划

### 物理坐标 – position

任意一个3D对象在三维直角坐标系中显示时，必定会有物理坐标信息，反应为投射到每个轴上的位置。在3D机房系统中，资产对象是一个3D对象，该对象有一个position属性（position有三个值，分别是X, Y, Z），代表在空间直角坐标系中的三个轴上的位置。

### 逻辑坐标 – location

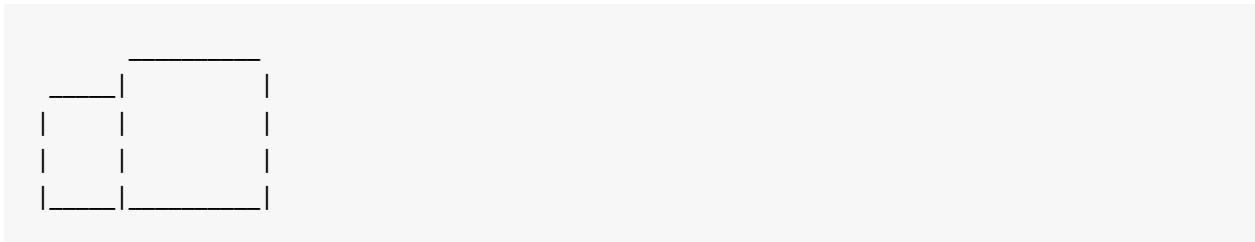
通过前面介绍的position属性，我们可以任意设定一个3D对象的位置信息，然而实际生活中，我们的被管对象往往是有规划的摆放。例如机房中可以摆4行10列机柜、机柜中可以摆下42个的1U的服务器。这些数据决定了空间坐标中的显示位置，然后又没有跟空间坐标的直接关联。在3D机房系统中这种逻辑坐标称为location。系统会将逻辑坐标location计算出一个物理坐标position，用来摆放显示。那3D机房系统是怎么计算出的position的呢？

我们还需要引入一个概念：空间规划，例如前面提到的机房里面可以摆放4行10列，机柜可以摆放42个1U服务器。在资产模型中的size和childrenSize属性表示空间规划概念，后面会详细介绍。例如上面的举例，机房的childrenSize值是{"x": 10, "z": 4}，42U的机柜的size值是{"x": 1, "z": 1}，它的childrenSize值是{y:42}。到此已经很清楚计算过程了，把机房外观尺寸（1000, 1000, 1000）的x方向分成10等份，每一份是100，z方向分成4等份，每一份是250来计算机柜的位置，例如机柜的location是（2, 0, 1），计算出的position值为（1002, 0, 1250）。那么设备的位置就是把机柜的y（高度）除了42，来计算单个设备的位置，例如机柜高度是H，一个1U的设备摆放在4U的位置，那么postion属性的y值等于H / 42 \* 4。房间四周可能需要空留部分通道，不能用来摆放机柜、设备的x和z方向因为跟机柜对齐没有设置值，具体会在后面的内容介绍。

location属性介绍：x, y, z分别表示3个方位的布局，支持数字和对齐方式  
( ['posneg', 'center\_neg', 'neg\_neg', 'center', 'pos\_pos',  
, 'center\_pos', 'neg\_pos'] ) 两种方式。对齐方式说明：“前面对应的是自身，'\_'后面对应的是父亲

如以x轴为例，即x:'pos\_neg'...共七中情况

1. pos\_neg: 表示的是自己在x轴最大值处于父亲在x轴最小值处对齐；
2. center\_neg: 表示的是自己在x轴的中心点和父亲在x轴最小处值对齐；
3. neg\_neg: 表示的是自己在X轴最小处和父亲在x轴最小处对齐；
4. center: 自己在x轴的中心点和父亲在x轴的中心点对齐；
5. pos\_pos: 自己在x轴的最大值和父亲在x轴的最大值处对齐；
6. center\_pos: 自己在x轴的中心点和父亲在x轴的最大值处对齐；
7. neg\_pos: 自己在x轴的最小处和父亲在x轴的最大值对齐； 例如机柜的location位置是(2, 0, 1), x = 2表示第二列, z = 1表示在第一排, y值"neg\_pos"表示一种对齐方式, 表示机柜的最下面和机房模型的最上面对齐。如下图：前面的矩形代表自己，后面的代表父亲，并且向右代表正方向，那么下面这种情况就是：“pos\_neg”



## 空间规则

在上面的介绍中引入了空间规划的概念，空间规划的本质是父对象和子对象之间的物理位置重叠，本着更符合人机工程思维，将物理位置抽象出来，将物理单位转换成逻辑单位，简单理解为将空间或平面按最小单位划分成很多小格子，一个子对象最少要占用一个格子或整数倍。例如前面的房间尺寸是(1000, 0, 1000) 柜子的尺寸是(100, 100, 250)，那么房间抽象出来的容积为(10, 0, 4)，即可以摆放4行10列柜子，容积是 $4 \times 10 = 40$ ，如果有四个柜子的location是(2, 0, 1)、(3, 0, 1)、(4, 0, 1)、(5, 0, 1)，那么房间利用率就是 $4 / 40$  (房间容积) \* 100% = 10%，剩余空间率为90%。包括在计算整个机房内机柜的空间占用情况时，将会更加直观给出结果。

空间规划相关的两个属性：size和childrenSize。

size：表示自身所占的位置，属性包括x、y、z，分别表示三个方向的尺寸。例如2U的服务器，需要占用2个1U的高度。那么值为：{y: 2} 例如

```
var size = {"x": 1, "z" : 1};
```

childrenSize：属性包括：x、y、z、xPadding、yPadding、zPadding。其中x、y和z表示其中x、y和z表示可以容纳子对象的数量，xPadding、yPadding和zPadding的值是一个二维数组，默认是[0, 0]，表示三个方向相需要扣除的间隙。例如42U的机柜，有42个U的高度，如果装1U的设备可以容纳42个，如果装2U的设备则只能容纳21个。那么值为：{y: 42}

```
var childrenSize={  
    y:42,  
    yPadding:[5.545,5.545],  
    zPadding:[-0.5,-0.5],  
    xPadding:[5.545,5.545]  
};
```

空间规划在计算空间利用率等指标时，会根据size和childrenSize来计算剩余空间和已经使用的空间。