

Table of Contents

简介	1.1
实施向导	1.2
用户管理	1.3
资产管理	1.4
资产	1.4.1
资产模型	1.4.2
业务类型	1.4.3
资产分类	1.4.4
虚拟机	1.4.5
场景	1.4.6
灯光	1.4.7
页面配置	1.5
提示信息	1.5.1
弹出面板	1.5.2
过滤器	1.5.3
右键菜单	1.5.4
模版数据	1.5.5
管线	1.6
漏水绳	1.7
制冷管道	1.8
巡检	1.9
巡检路径	1.9.1
巡检点	1.9.2
巡检报告	1.9.3
巡检区域	1.9.4
巡检对象	1.9.5
巡检指标	1.9.6

温湿度	1.10
温度云图	1.10.1
传感器	1.10.2
数据绑定	1.11
业务对象	1.11.1
业务属性	1.11.2
关联关系	1.11.3
告警设置	1.12
告警类型	1.12.1
告警级别	1.12.2
告警状态	1.12.3
采集	1.13
通信规约	1.13.1
解析协议	1.13.2
通道	1.13.3
视频监控	1.14
操作日志	1.15
镜头	1.16
镜头动画	1.16.1
镜头动画动作	1.16.2
扩展字段	1.17
业务数据管理	1.17.1
导入	1.18
导入机柜	1.18.1
导入设备	1.18.2
导入管线	1.18.3
更新资产编号	1.18.4
导入扩展字段	1.18.5
附录一	1.19

简介

此文档是，TWaver 3D机房可视化系统(后称3D机房)的后台管理说明文档，针对3D机房的后台各个模块进行详细说明。

实施向导

用户管理

资产管理

资产

资产对象管理的是资产实例，与实际资产对应，每个资产对象依据资产模型确定资产的外形，并在资产对象管理中设置资产的位置信息，从属关系。

资产对象中包含的字段：

字段	描述	数据类型	必填
id	资产对象编号	string	true
name	资产对象名称	string	true
description	数据描述	string	
location	参考附录一：location、position、空间规划	json 字符串	
position2d	2D中的物理坐标描述	json 字符串	
rotation	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	json 字符串	
position	参考附录一：location、position、空间规划	json 字符串	
parentId	父对象，例如设备的父对象为机柜	string	
dataTypeld	资产模型编号	string	
weight	资产重量	int	true
businessTypeld	业务类型编号	string	

资产模型

资产模型是根据资产的外观来进行区分的，例如机柜，不同的厂家的机柜外观不同，这些就需要添加多个资产模型

资产模型包含字段：

```

id: required, 资产模型编号
categoryId: required, 资产分类编号, 来自于资产分类模块的编号
description: 资产模型描述
size: 参考附录一: location、position、空间规划
childrenSize: ; 参考附录一: location、position、空间规划
model: required, 在3D场景中, 资产模型对应的3D模型, 模型ID注册在TWaver Make中, 在3D场景中使用, 必须有值
modelParameters: 3D模型参数
simpleModel1: 简单模型, 资产模型在延迟加载时使用的模型对象, 注册在TWaver Make中
simpleModelParameters: 简单模型参数
model2d: 在2D场景中, 资产模型对应的2D模型, 模型ID注册在TWaver Make中, 在2D场景中使用, 必须有值
model2dParameters: 2D模型参数
batchable: 默认值: false, 保留字段, 暂未使用
lazyable: 默认值: false, 是否延迟加载
stopAlarmPropagationable: 默认值: false, 阻止告警传播
rotationExp: 高级用法, 旋转表达式
positionExp: ; 高级用法, 位置表达式{"z": "(data.parentCategory&&data.parentCategory.getId()&&data.parentCategory.getId().toLowerCase().indexOf('channel') >=0)?(z%2?-3:-5):z"}}
system: 默认值: false, 类型标识, 如果为true标识为系统内置分类, 添加分类时忽略此属性, Boolean, 内置的资产模型有设备、机柜、板卡、空调、端口、指示灯、电力系统、显示部件、各种柜子、通道、环境检测、办公、建筑、数据中心、列头柜、电视机、灭火器、灭火器组、门禁、地球
powerRating: 默认值: 0, 用电额定功率
weightRating: 默认值: 0, 承重
subtype: 子类型, 例如:设备分为服务器(server)和网络设备(network)
noPrefab: 是否使用clonePrefab方法复制相同的的对象,可以提高渲染效率
noVirtualOther: 当lookAt此类型模型时, 不虚化其他模型
searchFilter: 默认值: true, 在下拉列表中, 是否被过滤, 如果值为true, 显示, false, 不显示

```

添加资产模型

POST <http://{serverhost:serverport}/api/datatype/add>

Body

Accept: application/json

```
{  
    id: '',  
    categoryId: '',  
    //... ... 资产模型其他不是必须字段  
}
```

Returns

返回添加时传递过去的属性值， 及其它默认值不为null的属性

Content-Type: application/json

```
//成功  
{  
    value: {  
        "id": "",  
        "categoryId": "",  
        //... ... 其他添加时传递过去的属性值， 及其它默认值不为null的属性  
    },  
    error: null  
}  
//失败  
{error: 'error message' 或 {message:""} }
```

更新资产模型

POST <http://{serverhost:serverport}/api/datatype/update>

Body

Accept: application/json

```
{  
    //需要更新的值  
    value: {  
        categoryId: '',  
    },  
    // 所有的更新条件  
    options: {  
        id: '',  
    }  
}
```

Returns

返回被更新资产模型的所有字段

Content-Type: application/json

```
//成功
{
  value: {
    //资产模型的所有字段
  },
  error: null
}
//失败
{error: 'error message' 或 {message:""} }
```

删除资产模型

POST <http://{serverhost:serverport}/api/datatype/remove>

Body

Accept: application/json

```
{
  // 所有的删除条件
  id: '',
}
```

Returns

被删除资产模型的所有字段

Content-Type: application/json

```
//成功
{
  value: {
    //被删除资产模型的所有字段
  },
  error: null
}
//失败
{error: 'error message' 或 {message:""} }
```

查看资产模型

POST http://{serverhost:serverport}/api/datatype/get

Body

Accept: application/json

```
{  
    // 所有的查询条件  
    id: '',  
}
```

Returns

返回符合查询条件的第一个资产模型的所有字段

Content-Type: application/json

```
//成功  
{  
    value: {  
        //符合查询条件的第一个资产模型的所有字段  
    },  
    error: null  
}  
//失败  
{error: 'error message' 或 {message:""} }
```

搜索资产模型

POST http://{serverhost:serverport}/api/datatype/search

Body

Accept: application/json

```
{  
    // 所有的搜索条件  
    id: '',  
}
```

Returns

返回符合搜索条件的所有资产模型的所有属性

Content-Type: application/json

```
//成功
{
    //符合搜索条件的所有资产模型
    value: [
        //资产模型的所有属性
    ],
    error: null
}
//失败
{error: 'error message' 或 {message:""} }
```

业务类型

资产分类

在系统中可能某些功能针对某类资产进行处理，所以有了资产分类管理。例如在资产模型中的位置表达式中，如下：

```
{"z": "(data.parentCategory&&data.parentCategory.getId()&&data.parentCategory.getId().toLowerCase().indexOf('channel') >=0)?(z%2?-3:-5):z"  
}
```

其中就用到了分类为channel的资产。通道可能存在多种样子，在资产类型中添加多个，这里要指定多个类型就不合理，最佳方案是根据分类来处理

资产分类包含字段：

```
id: 分类编号  
description: 分类描述  
stopAlarmPropagationable: 阻止告警传播, Boolean  
visible: 此分类的资产对象是否可见  
system: 分类标识, 如果为true标识为系统内置分类, 添加分类时忽略此属性, Boolean, 内置分类有地球、数据中心、楼、楼层、机房、通道、机柜、列头柜、空调、发电机、设备、板卡、端口、指示灯、连线、部件、门禁、环境系统、消防、办公用品、电视机、普通柜子（配电柜、开关柜等）
```

虚拟机

场景

页面配置

提示信息

弹出面板

弹出面板是3D机房系统中当选择一个3D对象时，显示的html tab组件，弹出面板包含多个tab，其中这里配置的URL所指向的页面会作为一个tab显示，系统会将选中的3D对象ID传递给URL指向的页面 弹出面板包含字段

`id: 3D对象的分类编号，即category ID`

`props: 拓展属性，暂未使用`

`url: 指向的页面路径，例如：id为rack，url为https://www.baidu.com/，那么当选中机柜的时候百度的首
页会作为一个`

`tab, 在弹出的面板中`

過濾器

右键菜单

模版数据

管线

管线指的是两个资产对象之间的连线

```
id: 唯一标示符, 类型为string  
dataTypeId: 管线的资产模型, 资产模型表中的ID, 用于控制管线的显示效果  
name: 管线名称  
fromId: 管线起始资产的ID, 资产表中的ID, 例如设备A连接到设备B, fromId是A的id  
fromPortId: 如果管线是从设备上端口上开始, 例如设备A的端口a1连接到设备B的端口 b1.fromPortId指a  
1的资产id  
fromOffset: 指管线的起始端离起始资产直接的偏移量, 如果为0, 两者连接在一起  
toId: 管线结束资产的ID, 资产表中的ID, 例如设备A连接到设备B, toId是的id  
toPortId: 如果管线是从设备上端口上开始, 例如设备A的端口a1连接到设备B的端口b1.toPortId指b1的资产i  
d  
toOffset: 指管线的结束端离结束资产直接的偏移量, 如果为0, 两者连接在一起  
type: 当type为“link”时就创建mono.Link, 否则创建mono.PathLink  
path: 管线的路径  
description: 描述
```

漏水绳

制冷管道

巡检

巡检是运维产品必不可少的功能，是一个想对告警来说，检查频率想对较低，涉及到指标想对较少，有固定轨迹，固定时间轮训检查硬件状态的功能。

巡检报告相关字段

```
inspectionId: 巡检路径的编号 string
startTime: 开始巡检时刻yyyy-MM-ddHH:mm:ss
endTime: 结束巡检时刻yyyy-MM-ddHH:mm:ss
remark: 本次巡检的备注信息 string巡检结果相关字段
dateTime: 巡检时刻yyyy-MM-ddHH:mm:ss
target: 巡检资产对象的编号 string
type: 巡检结果类型，目前只支持text，后续可能会增加url等方式，可以在系统播放时，显示对应的url界面
。
content: 巡检结果，查看巡检报告时，会弹出对话框，显示该字段里面的内容，type的只为url时，这里传入
对应的url地址。
```

巡检路径

巡检点

巡检报告

巡检区域

巡检对象

巡检指标

温湿度

温度云图

传感器

资产对象显示的属性靠传感器驱动，例如显示房间的某一位置的湿度的Billboard，需有一个湿度传感器，并设置传感器相对房间的位置，传感器的父对象是房间，类型是湿度。传感器的位置就是Billboard的显示位置

传感器包含字段：

```
id: 传感器编号  
description: 传感器描述  
parentId: 传感器父对象，传感器的位置是相对父对象的  
position: 物理位置  
location: 逻辑位置  
type: 传感器类型，例如温度、湿度
```

数据绑定

在数据绑定时候，两个使用场景介绍

在看下面的使用场景之前，先明白业务对象、业务对象属性、关联关系三个模块的作用，以及它们之间的关系

场景一

简介: 客户推送自己的ID，推送ID为"TH001"监控对象的属性值，此监控对象包含两个属性，"温度"、"湿度"

数据使用方法: 将温湿度的值分别赋值给对应的采集器，由采集器负责显示

3D系统为了能接收数据并使用数据，需要做的设置:

- 1、在业务对象属性模块中添加类型为"TH",属性分别为"温度"、"湿度"的两条记录。类型"TH"是自定义出来，而"温度"、"湿度"来源于监控对象包含的属性
- 2、在业务对象模块添加记录，ID为"TH001"，类型为"TH"。类型"TH"来源于业务对象属性模块，ID"TH001"来源于客户推送的ID
- 3、在采集器模块，添加两个采集器"t001"和"h001"，"t001"负责消费温度数据、"h001"负责消费湿度数据
- 4、在关联关系模块添加数据

```
id | relations
----|-----
t001 | {bid:"TH001",property:"温度"}
h001 | {bid:"TH001",property:"湿度"}
```

场景二

简介: 客户推送3D系统中的ID，推送ID为“SOE01”监控对象的属性值，此监控对象包含多个属性，"运行状态"、"A相电压"、"B相电压"、"C相电压"等

数据使用方法: 所有值以key-value方式展示

由于直接展示所有属性，属性消费者不需要知道有什么属性。客户直接推送的是3D系统中的ID，不需要转换。那么3D系统为了能接收数据并使用数据，需要做的设置只有一部：

1、在关联关系模块添加数据

```
id | relations
----|-----
S0E01 | ""
```

在关联关系中有解释这种设置如何映射。大家可能有疑问，既然不需要知道有什么属性，也不需要ID映射，为什么还需要在关联关系中添加一条记录？这由订阅逻辑决定的，在订阅逻辑中依据是否有relation来判断是否有推送的数据。或许这里可以优化相关逻辑，连这一步配置都省略。

总结

从以上的两个场景可以看出，如何做数据绑定，主要看两个方面： 1、客户推送的ID是否需要转换 2、3D系统是否要展示特定的属性

- 两者都需要：配置业务对象及属性，在relation中绑定bid和属性
- 仅需要前者：不需要配置业务对象及属性，在relation中仅绑定bid
- 仅需要后者：配置业务对象及属性，在relation中绑定bid和属性
- 两者都不需要：不需要配置业务对象及属性，在relation中不绑定任何东西

业务对象

业务对象的功能

业务对象的功能主要在两点：

一、映射客户资产编号与3D可视化系统中资产编号的关系。

在一个系统中每个资产必然存在一个唯一标识符即ID，即同样一个设备，在客户的系统有一个ID，在3D系统中有一个ID。那么问题就来了，在客户系统推给3D系统展示数据的时候，如何决定被推送的数据由哪个资产来展示，要解决这个问题就需要一个描述两个系统之间两个ID关系的地方。那么这些描述关系到底在哪？

具体放在哪，我们看看有几种解决方案，最直接的两个方案：

1. 把3D系统的ID放在客户的系统中，客户那里描述映射关系，推送数据的时候，包含的是3D系统中的ID，那么3D系统接收到数据后，直接根据此ID做后续工作。
2. 把客户系统中ID放在3D系统中，并描述客户系统ID与3D系统ID之间的映射关系，客户推送数据的时候，包含的是客户的系统ID，那么3D系统接收到数据后，通过映射关系找到3D系统中的ID，然后再做后续的工作。

对于方案1，在实施的时候，势必会增加客户的工作量，也可能客户的系统已经稳定不愿再开发，甚至客户系统是第三方开发，无法继续开发，等等原因不能维护3D系统的ID。为了防止这些原因导致项目不能继续，项目不能继续就没有合同，没有合同就没有盈利，没有盈利就没有奖金，所以扯这么多就是为了钱，我们选择方案2。（理由绝对充分）

那么在方案2中，有两个部分：

1. 维护客户系统ID的地方
2. 维护客户系统ID与3D系统ID的地方

业务对象就是为完成第一部分工作而生的，维护客户系统ID的表。

关联关系就是为完成第二部分工作而生的，维护客户系统ID与3D系统ID的表，具体参考“关联关系”模块介绍

二、定义客户推送的数据中的属性

3D系统如果想使用客户推送的数据，那就需要知道客户推送的是什么数据。例如在展示温度云图的时候，温度云图中包含的每个点，都需要客户推送的温度值，那么如何知道业务对象包含哪些属性呢？

决定业务对象包含哪些属性的是业务对象类型，在业务对象类型对应了多个属性。描述类型与属性的表是业务对象属性表，具体参考业务对象属性介绍。

给业务对象设置了业务对象类型就决定此业务对象拥有此类型所包含的所有属性。

业务对象字段

业务对象包含字段：

- `id`: 业务对象的编号，保证唯一性
- `description`: 对业务对象的描述
- `type`: 业务对象的类型，业务对象类型决定了所属此业务对象的监控属性。业务对象属性管理中，属性是与类型关联的

通过以上的描述，可以得知业务对象ID就是上面所说的客户系统中的ID。不过换一种说法应该更贴切，就是客户推送给3D系统的业务对象ID，因为推送过来的ID不一定就是客户系统中的ID。为什么这么说，在数据绑定中有给出使用场景。

业务对象属性

业务对象属性功能

主要功能：定义客户推送的数据中的属性

3D系统如果想使用客户推送的数据，那就需要知道客户推送的是什么数据。例如在展示温度云图的时候，温度云图中包含的每个点，都需要客户推送的温度值，那么如何知道客户推送的数据中，哪个属性是温度？

那就需要一个地方描述客户推送数据中的属性，业务对象属性就是描述这些属性的地方。

业务对象属性中定义了属性名，属性描述，数据类型，单位等基础信息。除了这些外，业务对象属性中还抽象出类型，把相同类型的资产包含的属性归为一组，方便业务对象的使用。

例如：温湿度传感器包含温度和湿度两个属性，那么定义业务对象类型为“温湿度传感器”，属性有：“温度”、“湿度”。那么在业务对象中，ID为“1#温湿度传感器”的业务对象的类型为“温湿度传感器”，ID为“2#温湿度传感器”的业务对象的类型为“温湿度传感器”。那么“1#温湿度传感器”和“2#温湿度传感器”两个业务对象分别拥有“温度”和“湿度”两个属性。

在业务对象管理中有介绍，业务对象的属性是通过业务对象的类型确认的，业务对象属性中就管理了业务对象类型所包含的属性，并定义了属性的数据类型，单位。

业务对象属性字段

业务对象属性包含字段：

- type：属性所属的业务对象类型
- property：属性名
- valueType：属性值的数据类型，取值：number、date、boolean
- unit：属性单位

关联关系

通过业务对象中的介绍，了解到关联关系起着维护客户系统ID与3D系统ID之间映射关系的作用。

仅仅是两个ID之间的映射那会比较简单。在业务对象属性中有举例，在展示温度云图的时候，温度云图中包含的每个点，都需要客户推送的温度值”。业务对象属性解决了“客户推送的业务对象，有哪些属性，哪个属性是温度？”的问题。那么关联关系就要解决“温度云图上的温度点要显示哪个业务对象的哪个属性的问题？”的问题。因此关联关系不仅仅维护ID之间的关系，还涉及到ID对应的属性。

关联关系的一条记录中包含3D系统的ID，客户系统的ID，属性。对应的列：

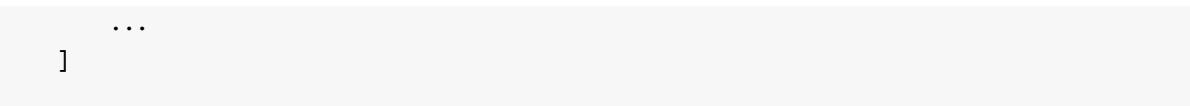
- id：3D系统的ID
- relations：客户系统的ID，属性。结构为{bid:"",property:""},bid对应客户系统的ID，由于客户系统的ID在业务对象中定义，因此这里用bid表示。3D系统的ID与客户系统的ID的关系可以是一对多的关系，所以relations的值可以是个数组。relations多种数据结构解释：
 1. {bid:"A",property:"a1"},关联A下面的a1属性
 2. {bid:"A",property:["a1","a1"]},关联A下面的a1、a2属性
 3. {bid:"A"},关联A下面的所有属性
 4. [{bid:"A",property:"a1"},{bid:"B",property:"b1"}],关联A下面的a1属性和B下面的b1
 5. 如果是数组累计关联每个元素中内容，规则符合1、2、3
 6. "", 空字符串，管理bid为id的所有属性，此情况可以不存在业务对象和业务对象属性

关联关系管理，关联关系指的是传感器与业务模型之间的关系，由于服务端是根据业务对象来缓存和推送数据的，所以当前端接收到数据后，再根据关联关系中的模型数据ID找到前端模型，赋值并做处理

关联关系单据管理包含字段：

- id：传感器编号
- relations：表述关联关系，JSON格式，具体格式为

```
[  
  {  
    bid:"业务对象ID",  
    property:"业务对象属性",  
  },
```



告警设置

告警类型

管理系统中的告警类型，自定义告警类型

告警类型包含字段：

`id`: 告警类型编号

`name`: 告警类型名称

`level`: 告警类型级别，系统提供的告警级别有，`indeterminate`、`warning`、`minor`、`major`、`critical`，暂不支持扩展，

`description`: 告警类型描述

告警级别

告警状态

采集

通信规约

解析协议

通道

视频监控

操作日志

镜头

镜头动画

镜头动画动作

扩展字段

业务数据管理

导入

导入机柜

导入设备

导入管线

更新资产编号

导入扩展字段

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

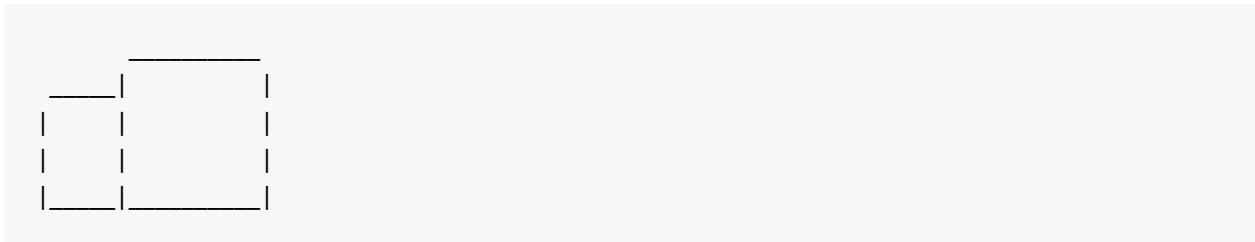
通过前面介绍的position属性，我们可以任意设定一个3D对象的位置信息，然而实际生活中，我们的被管对象往往是有规划的摆放。例如机房中可以摆4行10列机柜、机柜中可以摆下42个的1U的服务器。这些数据决定了空间坐标中的显示位置，然后又没有跟空间坐标的直接关联。在3D机房系统中这种逻辑坐标称为location。系统会将逻辑坐标location计算出一个物理坐标position，用来摆放显示。那3D机房系统是怎么计算出的position的呢？

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。