分类号_____ 学号 M201176027

学校代码 10487 密级_____

华中科技大学

硕士学位论文

面向 Web 的网元三维面板的设计与实现

学位申请人：秦　浩

学 科 专 业：软件工程

指 导 教 师：黄立群　副教授

答 辩 日 期：2014.1.7

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree for the Master of Engineering

The Design and Implementation of Web-oriented Three-dimensional of the Network Element System

Candidate : Qin Hao

Major : Software Engineering

Supervisor : Assoc. Prof. Huang Liqun

Huazhong University of Science and Technology

Wuhan 430074, P. R. China

January, 2014

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，　　在_____年解密后适用本授权书。

　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　月　日　　日期：　　年　月　日

摘 要

随着网络规模的急剧膨胀，网络复杂度的不断加深，网络的高效性、安全性和可靠性需求变得越来越难满足，网络管理也越发得到重视。网管系统可以大致分成对网络的管理和对网元的管理两部分。网元管理是对设备本身的管理，与设备间的相互关系、分配调度等无关。网元面板管理是网元管理的一个子系统，用户通过它查看网元设备面板，直观地观察设备运行状态，是与用户息息相关的一个功能组件。

传统的网元面板管理系统，能够对添加进该系统的所有网元设备面板进行管理同时提供展示功能，该设备面板往往是以二维图片的形式向用户展示网元的运行状态。二维的设备面板图对于没有见过实际网元的用户来说往往较抽象，不是很好的用户体验。而与传统的二维设备面板管理系统不同，网元三维模型管理系统提供对添加进该系统的所有网元设备面板进行管理，但是向用户展示的是高仿真度的网元三维模型。网元三维模型管理系统，通过 SNMP 通信从网络设备的 Mib 信息库取得网元的实体 Mib 信息。结合预先定义好的网元静态结构配置信息（坐标、图片等），在系统后台构建网元逻辑模型，并将构建好的逻辑模型转换成前台可解析的 Json 结构传递给前台页面。前台页面收到 Json 数据，解析并通过 WebGL 前端三维渲染技术将网元的模拟三维模型渲染出来，添加上与用户交互的动作。

网元三维模型管理系统的设计与实现，为网元模型管理的开发与研究做出了探索。在网管系统中引入 WebGL 技术，利用三维模拟使得网元面板更加贴近真实，让用户对网元设备有更直观的体验。

关键词：网元管理　　网元三维模型　　前端三维渲染技术

Abstract

With the rapid expansion of network, the requirement for network getting increasingly difficult to meet, network management attracts more attention. Network management system can be divided into two parts: the network management and network element management. NE management is the management of the device itself, the relationship between devices and distribution-independent scheduling. NE panel management is a network element management subsystem, through which users view network element equipment panel, visually observe the equipment running, is a functional component is closely related with the user.

Traditional NE panel management system is able to a provide panel display functions of the managed device, but the device panel is usually displayed by two-dimensional picture. Two-dimensional devices panel is too abstract for the users who haven't seen the real device, not a good user experience. Different from the traditional two-dimensional device panel systems, NE three- dimensional device model systems show the user a high degree of simulation three-dimensional model of the network element. NE three-dimensional device model systems obtain information on the NE entity Mib via SNMP communication. Combining predefined static structure configuration information (coordinates, pictures, etc.) to build the logic model network elements in the background, then change the logic model into JSON which foreground resolvable and pass to the front page. Front page receive JSON data and analysis it, through WebGL technology will simulate a three-dimensional model of a network element rendered, at last add the function of user interaction.

Design and implementation of the NE three- dimensional device model systems made a exploration for the development and research NE management model. For the first time, WebGL technology is used in the network management system. The use of three-dimensional simulations make NE panel closer to real NE device, allowing users have a more intuitive experience to network elements.

Key words: Network element management WebGL Three.js

目 录

1 绪论

1.1 课题背景与研究意义

在信息爆炸的当今时代，网络规模不断膨胀，网络变得越来越复杂，而用户对网络的安全性及可靠性要求越来越高。网络管理技术已成为现代计算机网络技术中重要的研究课程。只有对网络进行有效的管理，才能确保网络的稳定运行与发展。当今众多网络硬件设备厂商，也都推出自己的网络管理软件。其中，iMC 是杭州华三通信技术公司（H3C）以业务管理为中心的网络智能化管理系统。它集鉴权、部署、审计、调度、监控等功能特性于一体，与 H3C 公司的网络硬件设备一起为用户提供网络管理的整套解决方案。

本课题来源于本人与杭州华三通信技术有限公司实习期间参加的网络设备管理(Device Manager)软件项目。网络设备（简称网元）管理软件是 iMC 网络智能管理系统下的一个重要组件，用于对网络设备本身的管理，与设备之间的相互关系、网络资源的分配调度等无关，是网络管理的基础[1]。网元面板管理系统作为其中的一个子系统，用于管理设备面板。用户通过它可以查看设备完整仿真的面板视图，方便用户直观地了解网络设备的运行状态，同时提供对设备及端口数据流量的统计和监视功能[2]。虽然网元设备面板管理只是网管系统中一个功能较单一的模块，但是往往却是用户关注最高的功能模块之一。因为对于大部分用户来说，网元设备面板给了用户最直观的体验，网络中的设备状态通过设备面板一目了然，哪些板卡正常使用，哪些端口 down 掉都能从设备面板中直观反馈给用户，甚至直接在面板界面上对设备进行下发配置。

传统网络设备面板是以二维图片的形式展示的，通常是使用 Applet 等技术将制作好的机框、板卡、端口等的二维图片有序叠加，完成二维的设备面板显示。然而，对于没有见过实际网元设备的用户来说，二维的设备面板仍然较抽象。与传统的网络设备面板管理不同，本课题的创新点在于对二维网元面板管理系统的改造，完成了以 WebGL 技术为基础的网络设备三维模型管理， 使得设备面板以更加直观和逼真的效果，将设备的实时状态展示给用户。

1.2 国内外研究现状分析

本系统三大技术要点，一是 Web 服务器后台与设备通信，二是 Web 服务器应用架构，最后是 Web 前端的 3D 渲染。Web 服务器后台与网络设备通信及 Web 服务器应用架构，两者都是趋向成熟的技术。两者都有各自开源的框架或类库。前者如 SNMP4J，封装了设备 SNMP 通信所需的基础类库，后者如 Struts、JavaServer Faces 等。

而 Web 前端的 3D 渲染技术——WebGL 则正处于百花争鸣、茁壮成长的阶段。WebGL 是新的 Web 上的 3D 图形规范。有了 WebGL，开发者仅仅通过 JavaScript、Web 浏览器和标准的 Web 技术的堆叠就可以充分驾驭计算机图形硬件的渲染能力[3]。在 WebGL 出现之前，开发者必须依靠插件或者本地应用程序，并且要求用户下载并安装软件来实现 3D 体验。WebGL 是 HTML5 技术大家庭中的一员。虽然没有写在官方的 HTML5 规范内，但大部分支持 HTML5 的浏览器都支持 WebGL。像 WebSocket、Web Workers 和其他 W3C 推荐之外的技术一样，WebGL 实质上已经成为现代浏览器转变为一流的应用平台的一部分。WebGL 可以在大部分主流的台式机及增长迅猛的移动浏览器中运行。目前安装了支持 WebGL 的浏览器用户已经超过数百万[4]。WebGL 正处于一个生机勃勃、茁壮成长的生态系统的中心，将使网络体验在视觉上更加丰富动人。目前，已经有数百个使用 WebGL 技术开发的网站应用和工具，从游戏到数据可视化、计算机辅助设计以及电子商务[5]。

随着 WebGL 标准的快速发展，WebGL 这一 Web 端的 3D 渲染技术也开始进入到各大网络设备厂商的视野，华为的 Esight、H3C 的 iMC 等纷纷投向 WebGL 的怀抱，开始研究 WebGL 所带来的更加丰富与立体的用户体验。首先与 WebGL 产生直观联系的就是网管软件中的设备面板管理系统了。目前，各大设备厂商仍未有成熟的应用 WebGL 技术的网管产品。虽然 WebGL 发展的如火如荼，但是将 WebGL 引入到网络管理系统中仍然是全新的尝试。我相信，随着网管产品的不断发展与成熟，用户很快就能体验到 3D 网元面板管理、3D 网络拓扑等带来的全新体验了。

1.3 研究内容及目标

本文研究网元三维模型管理系统的设计与实现。主要包括的内容是系统如何根据取得的设备实体 Mib 信息，构造设备实体信息的数据结构，将该数据结构转换成约定好的数据格式并传递给 Web 前端，前端收到的结构数据，渲染出该设备的三维模型展示给用户。本方案省略了从设备取的 Mib 数据的过程，重点关注两个方面，一如何通过取得的网元 Mib 信息构造出逻辑模型；二是如何通过浏览器渲染出网元 3D 模型。具体的研究内容包括：

（1）存储网元实体静态信息的 XML(Extensible Markup Language)配置文件设计与实现（以下简称 XML 配置文件）。一台网元设备实际生产出来时，其许多实体信息已经确定下来，如机框的大小、丝印纹理、板卡槽位的大小及坐标等等。这些信息我们统称为网元实体静态信息。这些实体静态信息都存储在 XML 配置文件里。除了网元设备，板卡和端口在生产出来那一刻，许多实体信息也是确定的，如板卡的大小、板卡的端口槽位坐标、槽位大小、板卡的丝印纹理等等，这些静态实体信息也存储在 XML 配置文件里。因此 XML 配置文件是一个分层的结构，在实现部分会详细讨论。

（2）设计实现动态构建网元逻辑模型。要构建网元逻辑模型，首先需要针对所有网元实体设计网元实体类。网元实体结构是一个类似于树的结构，设计好的网元实体类相当于树的节点，动态构建网元逻辑模型的过程就是利用节点动态创建树的过程。网元实体类的初始化或者说对象的创建，依赖于网元实体的信息。虽然 XML 已经存储了设备实体的静态信息，但这些信息只能构建出一个没有任何状态的机框，因为不知道槽位上是否插有板卡，端口的状态是什么。网元的实际状态信息存储在 Mib 信息库中，需要系统后台通过 SNMP 通信从物理设备中取数据。取出来的实际设备 Mib 信息与 XML 配置文件进行匹配。该匹配算法决定了哪些槽位插哪些板卡，机框和板卡上的哪些端口是什么状态，最终构造出网元逻辑模型。

（3）重点研究前台页面如何根据获得的网元结构数据渲染出仿真的网元三维模型。此处使用对 WebGL 进行高级封装的第三方三维引擎库 Three.js。研究的内容包括：三维场景渲染的几个基本要素（照相机、场景等）；基础几何物体主要是立方体

的渲染；如何利用二维画布向三维场景添加二维动态元素；响应用户的鼠标事件，用户拖拽鼠标物体旋转；物体的拾取，当用户点击模型上的实体时，系统捕捉该实体并产生选中效果等。

1.4 论文组织结构

本文通过章节组织内容，共分为六章，第一章：绪论。介绍课题的来源及意义，国内外研究现状分析、研究内容及章节结构。第二章：网元三维模型系统技术背景。介绍了 MIB、WebGL 等相关背景知识。第三章：网元三维模型系统需求分析。第四章：网元三维模型系统的总体设计，包括分层模块化结构、系统流程分析和关键技术。第五章：主要关注网元逻辑模型的构建，及前端的三维渲染。展示课题的研究成果。第六章：总结全文工作，指出当前实现存在的不足，提出系统进一步的改进方向。

2 相关技术简介

2.1 管理信息库 Mib

设备面板管理是网管系统的子系统，是对网络当前设备的实时状态的一个直观反映，需要采集当前网络设备的状态信息，这些状态信息都存在于网络设备的管理信息库 Mib 中。

2.1.1 Mib 相关概念

MIB 是 Management Information Base 的英文缩写。《TCP/IP 详解》中给出的定义是：就是所有代理进程包含的、并且能够被管理进程进行查询和设置的信息的集合[6]。MIB 信息中的被管资源都是对象，每个对象都代表了被管资源的某一种属性，这些对象组合在一起便成了 MIB 库[7]。Mib 库信息涵盖了网络管理中所需要的方方面面的网络信息，包括网络的状态、通信量的监控、通信失败的统计甚至内部结构的当前内容等[8]。网络管理员通过对 Mib 库的读写操作，来完成对网络的管理[9]。

MIB 文件是用 ASN.1 语法来描述的，用户可以自己定义 Mib，但是必须遵从 ASN.1 的语法和相关约束，因此用户需要参考一些 ASN.1 语法的有关文档如 RFC1155、RFC1212 [10]。ASN.1 是 Abstract Syntax Notation One 的缩写，以为抽象表示法，它是用文本来描述被管对象的，mib 文件的文件扩展名为“.mib”，用户可以使用任何文本编辑器来读写该文件[11]。

MIB 是以树状结构进行存储的，Mib 通过遵守 ASN.1 语法的树形结构来组织信息的，每一个树形结构上的节点代表了一片有用的信息[12]。每个节点包含两个内容：（1）该节点的对象标识符（2）该节点的简单描述[13]。对象标识符 OID(object identifier) 是资源对象的唯一标识，它是由一些权威机构进行管理和分配的。OID 是一组由“.”隔开的整数，每一个 OID 都代表了 MIB 树上的节点及其位置[14]。对象标识符从树的顶部开始，顶部没有标识，以 root 表示，如图 2-1 所示。

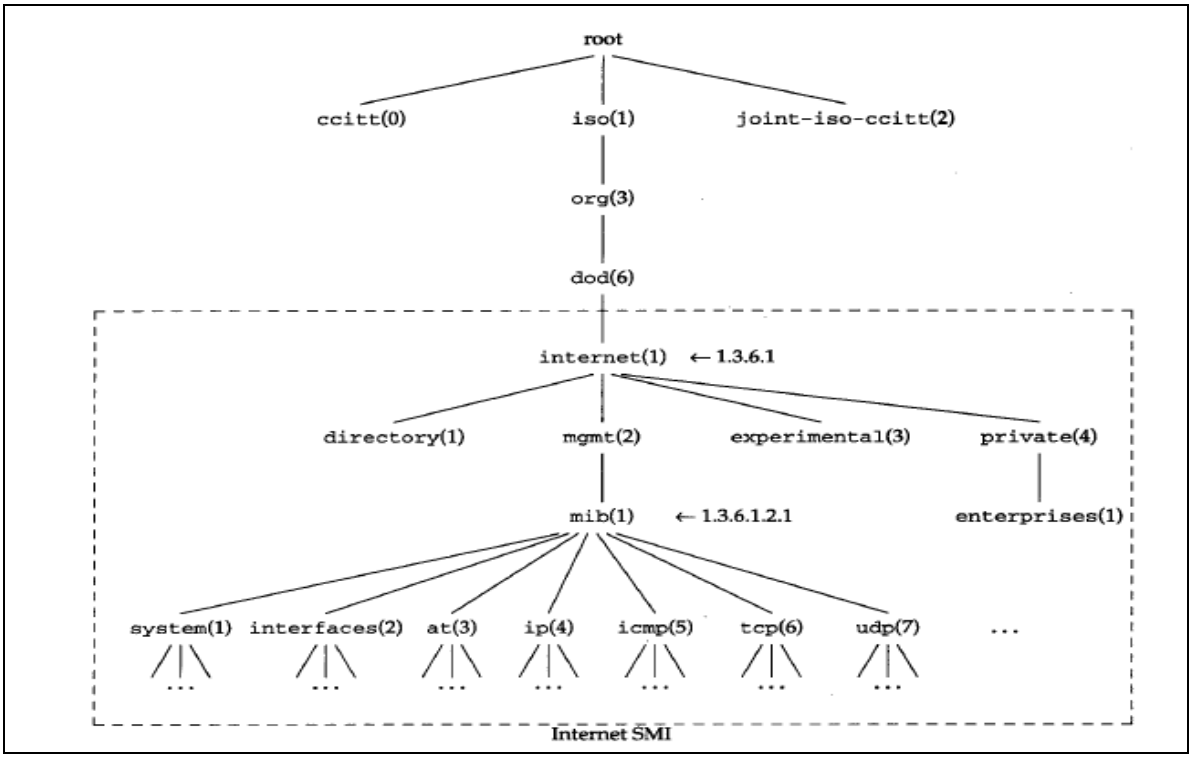


图 2-1 MIB 树结构

MIB 树的根节点无实际意义，它无名也无编号，但是下面有 3 棵子树：ccitt(0)、iso(1)和 joint-iso-ccitt，由 ISO 和 CCITT 共同管理。org(3)是 iso 节点下的子树，它是 ISO 为其他组织定义的子树 [15]。在 org 子树下，节点 dod(6)是被美国国防部(Department of Defense)使用的 [16]。目前，dod 子树下的对象资源是应用最广泛的。很大一部分的通信设备都遵守 DOD 协议如 TCP/IP 设备，能够提供的信息都位于这个子树下，完整的对象标识为 1.3.6.1，称为“internet”。internet 的 4 个子树中，重点关注 mgmt(2)和 private(4)。mgmt 子树中的 MIB-I(RFC1156)已经被 MIB-II(RFC1213)所取代。MIB-II 保留了同一对象标识符[17]。常见的系统组、接口组、地址转换组、IP 组、ICMP 组、TCP 组、UDP 组、SNMP 组等都是定义在这个子树下[18]。

private 子树用来指定单方面定义的对象。该子树中网络管理系统访问最多的部分是 enterprise(1)或{private 1}节点。企业产商通过分配到的节点，在该子树下创建它的产品特有的属性。厂商自定义的 MIB 都处于树型结构的这个位置上。例如{enterprise 9}分配给了 Cisco 公司，{enterprise 25506}分配给了 H3C 公司等。

2.1.2 网元相关 Mib

这里所说的网元相关 Mib 指的是与网元实体相关的 Mib，即与本系统构建网元逻辑模型所需的相关 Mib 信息。与网元实体信息相关的 Mib 一共是三张表，分别是 entPhysicalTalbe、ifTable 和 entAliasMappingTable。

entPhysicalTable 是实体表，里面包含当前网元所有实体的信息[19]。如表 2-1 所示，是实体表所包含的几个关键节点，表中列举了各节点的类型、访问权限及其代表的含义。

表 2-1 entPhysicalTable

MIB	节点类型	访问权限	描述
entPhysicalIndex	Integer	Not-accessible	实体索引
entPhysicalVendorType	Object Identifier	read-only	唯一标识该实体
entPhysicalContainedIn	Integer	read-only	该实体的父实体索引
entPhysicalClass	Integer	read-only	该值是个枚举值，表示实体的类别
entPhysicalParentRelPos	Integer	read-only	表示该实体在兄弟实体中的相对位置

ifTable 是端口表，里面包含当前网元设备所包含的所有端口信息[20]。如表 2-2 所示，是端口表所包含的几个关键节点，表中列举了各节点的类型、访问权限及其代表的含义。

表 2-2 ifTable

MIB	节点类型	访问权限	描述
ifIndex	Integer	Read-only	端口索引
ifDescr	Octets	Read-only	端口描述
ifSpeed	Gauge32	Read-only	端口速率
ifAdminStatus	Integer	Read-write	端口管理状态
ifOperStatus	Integer	Read-write	端口操作状态

entAliasMappingTable 是网元实体与端口的映射表，端口作为网元实体既存在与 entPhysicalTable 也存在与 ifTable，entAliasMappingTable 的作用就是将端口的实体索引 entIndex 与端口 ifIndex 联系起来[21]。如表 2-3 所示，是实体端口映射表所包含的几个关键节点，表中列举了各节点的类型、访问权限及其代表的含义。

表 2-3 entAliasMappingTable

MIB	节点类型	访问权限	描述
entPhysicalIndex	Integer	not-accessible	实体索引
entAliasLogicalIndexOrZero	Integer	not-accessible	类似命名空间的概念
entAliasMappingIdentifier	Object Identifier	read-only	对应的 ifIndex 值

2.2 WebGL 三维绘图

WebGL 标准是由科纳斯组织开发和维护的，该组织还是管理这 OpenGL、COLLADA 等其他规范的标准组织[22]。WebGL 将 3D 内容带入到浏览器中，通过 JavaScript 与电脑中的图形硬件进行交互。基于 OpenGL ES（同样的图形内容可以运行在智能手机与平板）的特性，使得 WebGL 得到了发展并且得到了主流桌面和移动浏览器厂商的广泛支持[23]。

2.2.1 3D 图形学

（1）网格、多边形和顶点

有很多方法来绘制 3D 图形，最常用的一种方法就是使用网格（Mesh）[24]。网格上由一个或多个多边形组成的物体，各个顶点的坐标（x,y,z）定义了多边形在 3D 空间中的位置。网格中的多边形通常都是三角形和四边形。3D 网格通常也被叫做模型（Model）。

（2）材质、纹理和光源

网格表面的其他特性是由除了顶点位置（x,y,z）坐标之外的属性来定义的。网格表面可以简单到只有一种颜色，也可以复杂到由多种不同的数据来共同决定的，比如物体如何反射照射到表面的光，物体的光泽度如何[25]。网格表面同样也可以由一个或多个位图来决定，这就是我们通常所说的纹理映射（texture map），或者简称纹理[26]。纹理可以定义一个线性表面的外观，另外，通过组合多张不同纹理可以实现更加复杂的效果，例如凹凸效果和辉光效果。在大多数的图形系统中，网格表面的特性被统称为材质（material）。材质通常依赖于一个或多个光源（light）来呈现出外观效果，这就是我们如何让场景被照亮的[27]。

（3）变换和矩阵

3D 网格的形状是由顶点的位置决定的。如果每次你想要移动模型都必须重新设定顶点位置，那将会是一件非常可怕和沉闷的事情，特别是当模型会持续穿过整个场景或者做其他动作的时候。因此，大部分 3D 系统都支持变换（transform），这是一种不需要遍历每个顶点就可以移动模型的操作，不需要明确的逐个改变每个顶点的位置。变换包括对渲染模型的缩放、旋转、位移操作，而不必动手设定任何顶点的值。变换通常由矩阵来操作，这是一个包含一组数值的数学模型，用于计算顶点的变换位置[28]。

（4）相机、透视、视口和投影

每个场景在渲染时都需要一个视点，用户正是从视点的位置去观察场景的。3D 图形系统通常使用相机（camera）来定义用户与场景的相对位置和朝向，和我们真实世界的相机一样，3D 世界的相机也有类似于视野尺寸等属性。视野尺寸决定了透视关系（如近大远小）[29]。相机的各个属性组合在一起将最终的 3D 场景渲染给 3D 视口（viewport），视口是由浏览器窗口或 Canvas 元素决定的。

相机永远是由两个矩阵来控制的。第一个矩阵第一了相机的位置和朝向，类似于矩阵变换操作的矩阵。第二个矩阵比较特殊，它用于将相机空间的 3D 坐标转换为视口中的 2D 坐标，这就是所谓的投影矩阵[30]。

（5）着色器

为了在最终的图像中渲染模型，开发者必须精确地定义顶点、变换、材质、光源和相机枝江的关系和交互，这就是由着色器来完成的[31]。着色器是一段程序代码，其中包含了将模型投射到屏幕上的像素算法。着色器通常由高等级的类 C 言编写的，编译并运行在图形处理单元 GPU 中[32]。大部分现代电脑都配置了 GPU，它是独立于 CPU 而专门用于渲染 3D 图形的处理器。

在许多图形系统中，着色器是可选的或比较高级的特性。但与之不同的是，WebGL 必须配备着色器。当你编写 WebGL 程序时，必须定义着色器，否则图形不会显示在屏幕上[33]。WebGL 的实现假定客户端电脑上配置有 GPU。GPU 可以处理顶点、纹理和其他一点东西；但是它对材质、光源和变换却一无所知。把后者这些高级的信息传递给 GPU 并最终绘制到屏幕上的转换过程由着色器来完成[34]。

2.2.2 WebGL 应用结构剖析

从本质上来将，WebGL 只是一个绘制库——一个增强型的绘制库，使用 WebGL 你可以绘制出惊人的图形，并可以在当前大部分机器上充分利用强大的 GPU 硬件能力。但也可以把 WebGL 理解成另一种画布，类似于 HTML5 浏览器中的 2D Canvas。实际上，WebGL 也正是使用了 HTML5 中的 Canvas 元素来在浏览器页面中显示 3D 图形。想要使用 WebGL 把图形渲染到页面中，一个应用至少应该执行如下几个步骤[35]。

（1）创建一个画布元素 Canvas。

（2）获取画布元素的上下文。

（3）初始化视口。

（4）创建一个或多个包含渲染数据的数组。（通常是顶点数组）

（5）创建一个或多个矩阵，将顶点数组变换到屏幕空间中。

（6）创建一个或多个着色器来实现绘制算法。

（7）使用参数初始化着色器。

（8）绘制。

2.2.3 开源 3D 引擎

需求是创新之母。每个开发者在项目之初都必须不厌其烦的使用 WebGL 原声 API 来编写代码，直到可以建立自己的代码库，用于随后的通用 3D 编程。实际上，很多人已经完成了这项工作。现在有很多不错的 WebGL 开源框架。如 GLGE、SceneJS、CubicVR 等。每个框架都有各自不同的地方，但它们的共同目的都是为了构建一个高等级的、对开发者相对友好的 WebGL 开发环境。

本系统中使用的 WebGL 开源框架叫做 Three.js，它是由居住在西班牙巴塞罗那的程序员 Richardo Cabello Miguel 开发的。Three.js 以简单、直观的方式封装了 3D 图形编程中常用的对象。Three.js 在开发中使用了很多图形引擎的高级技巧，极大地提高了性能。另外，由于内置了许多常用对象和极易上手的工具，Three.js 的功能也非常强大。

与 WebGL 应用结构相对应，使用 Three.js 三维引擎库渲染物体时必须用到的三大组件分别是：场景、相机和渲染器，有了这三个组件才能将需要渲染的三维物体渲染到页面上去[36]。

（1）场景

在 Three.js 中，渲染是针对一个场景去渲染的。场景就是你要渲染的三维世界，世界可以是复杂到涵盖很多的布景、物体、动画，也可以简单到只有一个静止的形状。可以把场景想象成一个容器，需要渲染的对象都需要添加进该容器中去。在 Three.js 中，场景就只有一种，通过语句 var scene = new THREE.Scene()来创建。

（2）相机

相机是 Three.js 的另一个重要组件，相机的位置和朝向决定了场景中哪个视角的景色会被渲染出来。可以把相机形象的想象成人的眼睛，人通过变化自己所占的位置和朝向来从不同的角度看场景中的景色。这也是本系统的模型提供的模型旋转功能的原理所在。相机有正交相机和透视相机等多种。

（3）渲染器

渲染器是 3D 引擎的核心组件。可以把渲染想象成作画，渲染器就是作画的那只画笔。在 Three.js 中，渲染器有很多种，最常用到的是 WebGRenderer，渲染器通过语句：var renderer = new THREE.WebGLRenderer()来创建。

如图 2-2 所示，形象地说明了三大组件之间的联系。一句话可以概括，相机将场景中的景色拍下来交给渲染器在浏览器中渲染出来。

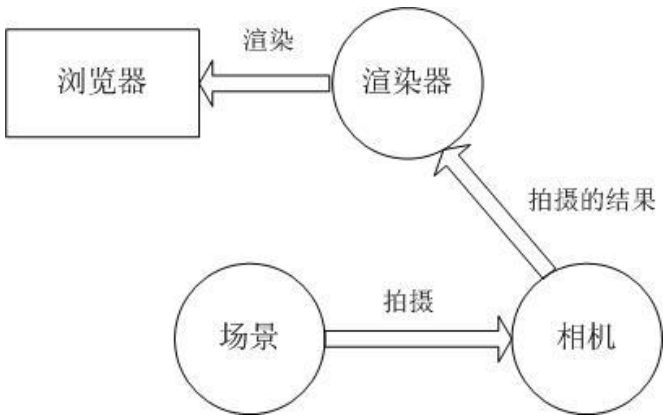


图 2-2　Three.js 三大组件

2.3 本章小结

本章主要对三维模型管理系统实现过程中用到的知识给出简要的说明，介绍了介绍了设备 Mib 信息库，这是我们获得设备信息的来源。介绍了 WebGL 这一 Web 端的三维渲染技术，涉及计算机图形学及线性代数的内容。

3 系统需求分析

3.1 需求概述

从宏观和物理上来看，网元主要包括路由器、交换机和应用服务器等主要网络设备。要真正有效显示和控制这些网络设备构成的网络实体，不仅要考虑各个生产厂家生产的不同类型的产品，还要细分到各网络设备包含的子网元对象。如常见的网络设备交换机，通常包含多个不同类型的端口，有些还包含多个插槽，插槽中插入的板卡可能也包含多个端口。网元管理到端口级才有实际意义[37]。

网管软件作为一款软件产品，其最终目的是服务用户，其生命力依赖于用户的满意度。因此，用户体验是决定产品成败的关键。网元面板管理是网元管理系统的子系统，它向用户提供网元设备的仿真面板，用户通过它可以对设备有了整体的直观把握，是与用户体验息息相关的一个重要子系统。然而，传统的网元面板仍然局限于二维空间，通常是将网元的前面板抽象出来预先制作成图片，通过获取的网元实体 Mib 信息构造网元逻辑模型，结合板卡和端口的图片，叠加完成网元面板的绘制，如图 3-1 所示。

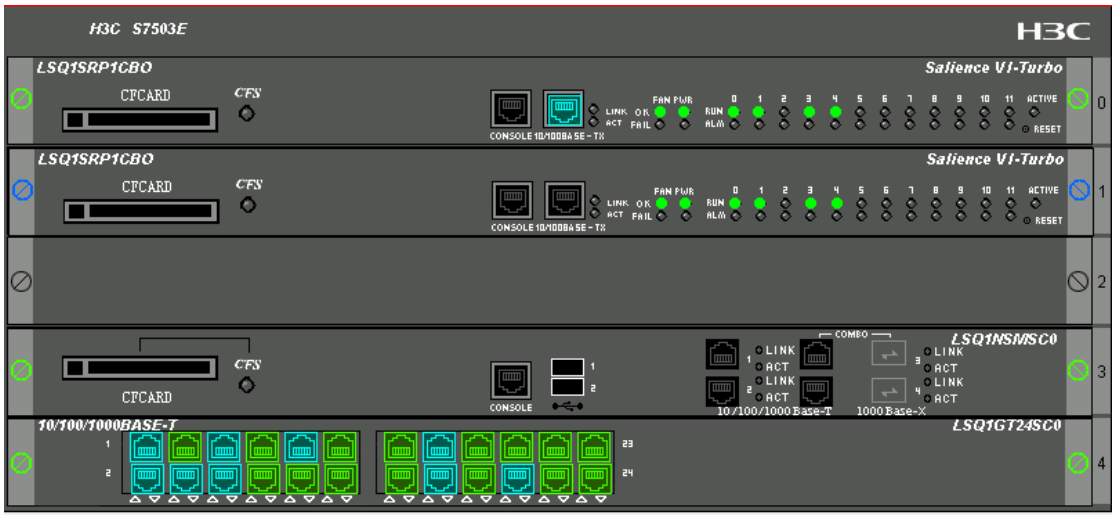


图 3-1 网元面板

图中是 H3C 公司的一款型号为 S7503E 的路由器设备。从图可以看出，该设备面板对于用户来说仍然是较抽象和单薄的，用户体验较差。随着面向 Web 的三维渲

染技术 WebGL 大行其道，利用 GPU 可以渲染出复杂逼真的三维世界。将 WebGL 引入网管系统，为网元面板系统设计实现一款高仿真度的网元三维模型系统，从而提升用户体验的，这个需求越来越迫切。

3.2 功能需求

网元设备面板管理是对网络设备的一种图形化管理。首先，网络设备以图形化的方式展现给用户。图形化管理是为管理员提供友好的操作界面，图片所涵盖的信息量是文字所无法媲美的。要实现对网元的面板管理，需要对网元提供的信息进行合理的描述[38]。而网元的面板相关信息都存在网元 Mib 表 entPhysicalTable 和 ifTable 中，因此需要对 entPhysicalTable 和 ifTable 表中的信息进行合理描述，即构建合理的数据结构，交给前端页面，完成面板构图。

从功能角度看，网元三维模型管理系统用于对网络设备的三维模型进行管理。功能需求包括以下几点：

1）模型的管理

（1）网元的添加：网元模型管理系统并不是对真实网络环境中的所有物理设备都能打开设备面板试图。只有添加进该网元管理系统中的网元设备，用户才能打开其设备面板视图。添加网元的过程就是设备联调的过程，即网元设备与网管系统适配的过程。

（2）网元的删除：可以添加就可以删除，管理员可以手动删除网元管理系统对一款网元设备的支持。对于删除后的设备，用户无法打开其网元面板试图。

2）模型的展示

（1）主体机框的显示：在网管系统中，通过设备的 IP 地址，定位到需要打开设备面板的网元设备。打开设备面板视图时，网元的主体机框可以正确渲染，网元的主体机框一般为立方体，网元主体机框上的丝印与真实设备吻合。同时，主体机框的渲染支持堆叠设备。当打开堆叠设备时，可以看到多台主体机框堆叠效果。

（2）板卡的显示：打开设备面板视图时，除了网元主体机框正确渲染，对于真

实设备上所插的所有板卡也应该在模型中正确显示出来，包括板卡的坐标位置、板卡上的丝印都应该与真实设备吻合。

（3）端口的显示：打开设备面板视图时，真实环境中的网元设备上所有的端口都在模型中正确显示，包括端口的坐标位置、型号都与真实设备吻合。

（4）与用户的交互：渲染出来的网元三维模型提供与用户的简单交互，这些交互包括：用户可以通过拖拽鼠标来旋转三维模型；通过左键点击网元上的实体（板卡、端口）来产生选中效果；通过鼠标右键点击端口，可以读取端口的信息。

（5）模型状态的刷新：网元模型反应的是真实环境中的物理设备的真实状态，状态是有可能发生变化的，因此需要提供固定时间间隔自动刷新的功能，刷新后重新渲染整个网元模型，从而达到更新网元状态的目的。

3）端口信息的管理

（1）端口信息的浏览：用户可以浏览端口的信息列表，包含：设备名称、端口索引、端口描述、端口别名、端口类型、管理状态、运行状态、接口速率。

（2）端口信息的配置：可以配置的端口的基本信息，包含：端口别名、管理状态。

4）性能监视

（1）端口监视：实时地监视指定端口的流量，包括：接口使用率(%)，输出包丢弃率(%)，输入包未知协议率(%)，输入包误码率(%)，输出包误码率(%)，输入包丢弃率(%)。可以同时监视多个指标。

（2）设备监视：实时监视制定设备的流量，包括：

IP 组：输入 IP 报文，缓冲溢出输入 IP 报文，转发的 IP 报文，表头错误的输入 IP 报文，地址错误 IP 报文，未知协议数据报，无路由的输出 IP 报文，成功重组的 IP 报文，缓冲溢出的输出 IP 报文。

TCP 组：输入 TCP 段、输出 TCP 段。

UDP 组：输入 UDP 数据报、输出 UDP 数据报、未发送到有效端口的数据报、接收到的错误数据报。

ICMP 组：输入 ICMP 消息、输出 ICMP 消息、输入错误 ICMP 消息、输出错误 ICMP 消息。

SNMP 组：接收 SNMP 报文、发送 SNMP 报文。

SNMP（操作）组：全部 Get/Get-Next 请求、全部 Set 请求、Get-Request 报文、Get–Next-Request 报文、Set-Request 报文、发送 Trap 报文。

SNMP（错误）组：无效版本错误、无效验证名错误、非法 SNMP 操作、报文 ASN1 解码错误、报文过大、报文名称错误、报文无效错误、操作只读对象错误、一般错误。

5）中英文切换：

提供进行中英文界面语言切换的快捷方式。

3.3 性能需求

从性能角度来看，网元模型管理系统的性能需求从以下几点去考虑：

（1）网元模型管理系统的性能与所安装平台和网络环境有关。网元模型管理系统作为网管系统的子系统，其性能与安装网管系统的平台有密切联系，服务器的硬件配置、运行程序以及网络状态都直接关系到系统的响应。

（2）运行时，应该在 10 秒内正常启动。当打开设备面板视图时，其速度与所处的网络环境有关，在网络运行正常的情况下，且设备能够管理的情况下，应该最慢 30 秒内打开设备。

（3）在设备无法管理的情况下，应该在重试次数×超时时长的时间内显示提示信息。

（4）由于三维渲染对 GPU 负荷较重，要求渲染的三维模型在运动的时候流畅没停顿。在三维的世界里，我们通常用帧数来衡量。所谓帧数，就是图形处理器每秒刷新的次数，通常用 FPS（Frames Per Second）来表示。动画都是由一帧一帧的图像连起来的，刷新越快，FPS 越高，动画越流畅平滑。当 FPS 低于 20 时，可以明显感觉到画面卡滞。因此，可以通过监视 FPS 来监视系统的性能表现。

3.4 本章小结

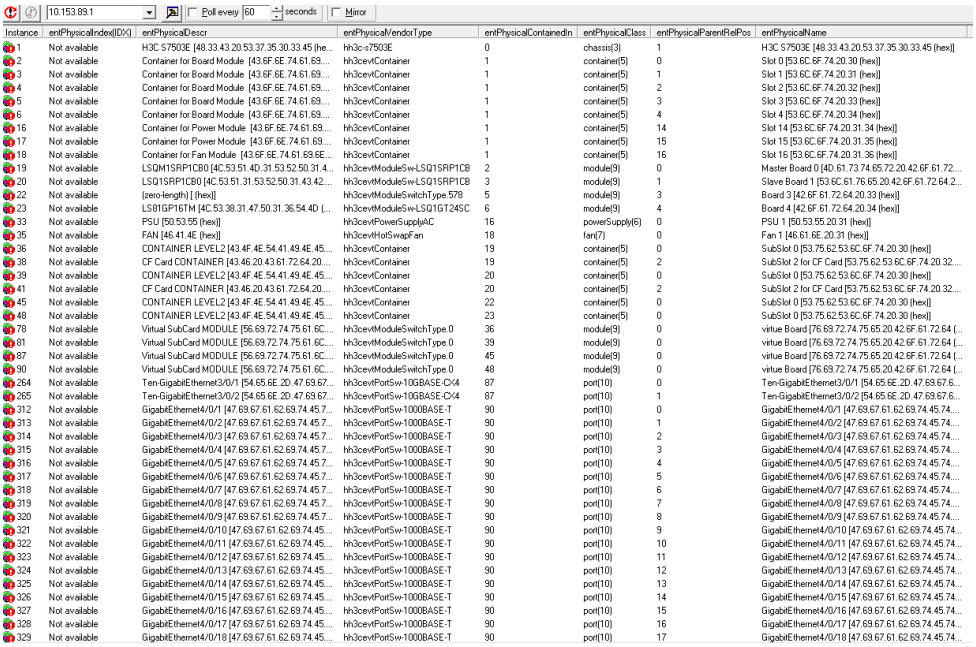
本章主要介绍网元模型管理系统的需求，从需求概述、功能需求和性能需求三个角度来阐释。需求概述简要介绍了将 WebGL 三维渲染技术引入网元管理系统的必要性；功能需求从模型管理、模型展示等 5 个方面阐释系统的功能概况；性能需求解释了系统的运行环境对性能的影响以及性能的衡量指标 FPS。

4 系统总体设计

4.1 系统设计约束与规范

本系统是基于 SNMP 通信协议从设备 Mib 中获取网元信息，通过 WebGL 以三维图形化方式实现网元设备图形化管理。因此，被管设备之间的通讯必须遵守 SNMP 协议，设备提供相应的 MIB 数据信息；从设备上读取 MIB 信息的功能由 SNMP4J 开源类库提供；编程语言采用 JAVA，开发 IDK 是 Eclipse，JDK 要求 1.5 及以上。

事实上，对于网元三维模型构建功能模块来说，系统所需获取的设备 Mib 信息主要来源于设备的实体 Mib 表即 entPhysicalTable，entPhysicalTable 将网元设备分解成不同的组成部件，一种类型的组成部件对应 entPhysicalTable 中的一种类型的物理实体[39]。将整个设备看成一个组合体，以树型结构分层管理设备中各种类型的物理实体，以及各种物理实体之间的关系。通过使用 Mib browser 软件远程登录设备观察 entPhysicalTable 的结果如图 4-1 所示。



10.153.89.1 | Poll every 60 seconds | Mirror

Instance	entPhysicalIndex[IDX]	entPhysicalDescr	entPhysicalVendorType	entPhysicalContainedIn	entPhysicalClass	entPhysicalParentRelPos	entPhysicalName
1	Not available	H3C S7503E [48.33.43.20.53.37.35.30.33.45 (he...	hh3c-s7503E	0	chassis(3)	1	H3C S7503E [48.33.43.20.53.37.35.30.33.45 (hex)]
2	Not available	Container for Board Module [43.6F.6E.74.61.69....	hh3cevtContainer	1	container(5)	0	Slot 0 [53.6C.6F.74.20.30 (hex)]
3	Not available	Container for Board Module [43.6F.6E.74.61.69....	hh3cevtContainer	1	container(5)	1	Slot 1 [53.6C.6F.74.20.31 (hex)]
4	Not available	Container for Board Module [43.6F.6E.74.61.69....	hh3cevtContainer	1	container(5)	2	Slot 2 [53.6C.6F.74.20.32 (hex)]
5	Not available	Container for Board Module [43.6F.6E.74.61.69....	hh3cevtContainer	1	container(5)	3	Slot 3 [53.6C.6F.74.20.33 (hex)]
6	Not available	Container for Board Module [43.6F.6E.74.61.69....	hh3cevtContainer	1	container(5)	4	Slot 4 [53.6C.6F.74.20.34 (hex)]
16	Not available	Container for Power Module [43.6F.6E.74.61.69....	hh3cevtContainer	1	container(5)	14	Slot 14 [53.6C.6F.74.20.31.34 (hex)]
17	Not available	Container for Power Module [43.6F.6E.74.61.69....	hh3cevtContainer	1	container(5)	15	Slot 15 [53.6C.6F.74.20.31.35 (hex)]
18	Not available	Container for Fan Module [43.6F.6E.74.61.69.6E....	hh3cevtContainer	1	container(5)	16	Slot 16 [53.6C.6F.74.20.31.36 (hex)]
19	Not available	LSQM1SRP1CB0 [4C.53.51.4D.31.53.52.50.31.4...	hh3cevtModuleSw-LSQ1SRP1CB	2	module(9)	0	Master Board 0 [4D.61.73.74.65.72.20.42.6F.61.72....
20	Not available	LSQ1SRP1CB0 [4C.53.51.31.53.52.50.31.43.42....	hh3cevtModuleSw-LSQ1SRP1CB	3	module(9)	1	Slave Board 1 [53.6C.61.76.65.20.42.6F.61.72.64.2...
22	Not available	(zero-length) [(hex)]	hh3cevtModuleSwitchType.578	5	module(9)	3	Board 3 [42.6F.61.72.64.20.33 (hex)]
23	Not available	LS81GP16TM [4C.53.38.31.47.50.31.36.54.4D (...	hh3cevtModuleSw-LSQ1GT24SC	6	module(9)	4	Board 4 [42.6F.61.72.64.20.34 (hex)]
33	Not available	PSU [50.53.55 (hex)]	hh3cevtPowerSupplyAC	16	powerSupply(6)	0	PSU 1 [50.53.55.20.31 (hex)]
35	Not available	FAN [46.41.4E (hex)]	hh3cevtHotSwapFan	18	fan(7)	0	Fan 1 [46.61.6E.20.31 (hex)]
36	Not available	CONTAINER LEVEL2 [43.4F.4E.54.41.49.4E.45....	hh3cevtContainer	19	container(5)	0	SubSlot 0 [53.75.62.53.6C.6F.74.20.30 (hex)]
38	Not available	CF Card CONTAINER [43.46.20.43.61.72.64.20....	hh3cevtContainer	19	container(5)	2	SubSlot 2 for CF Card [53.75.62.53.6C.6F.74.20.32....
39	Not available	CONTAINER LEVEL2 [43.4F.4E.54.41.49.4E.45....	hh3cevtContainer	20	container(5)	0	SubSlot 0 [53.75.62.53.6C.6F.74.20.30 (hex)]
41	Not available	CF Card CONTAINER [43.46.20.43.61.72.64.20....	hh3cevtContainer	20	container(5)	2	SubSlot 2 for CF Card [53.75.62.53.6C.6F.74.20.32....
45	Not available	CONTAINER LEVEL2 [43.4F.4E.54.41.49.4E.45....	hh3cevtContainer	22	container(5)	0	SubSlot 0 [53.75.62.53.6C.6F.74.20.30 (hex)]
48	Not available	CONTAINER LEVEL2 [43.4F.4E.54.41.49.4E.45....	hh3cevtContainer	23	container(5)	0	SubSlot 0 [53.75.62.53.6C.6F.74.20.30 (hex)]
78	Not available	Virtual SubCard MODULE [56.69.72.74.75.61.6C....	hh3cevtModuleSwitchType.0	36	module(9)	0	virtue Board [76.69.72.74.75.65.20.42.6F.61.72.64 (...
81	Not available	Virtual SubCard MODULE [56.69.72.74.75.61.6C....	hh3cevtModuleSwitchType.0	39	module(9)	0	virtue Board [76.69.72.74.75.65.20.42.6F.61.72.64 (...
87	Not available	Virtual SubCard MODULE [56.69.72.74.75.61.6C....	hh3cevtModuleSwitchType.0	45	module(9)	0	virtue Board [76.69.72.74.75.65.20.42.6F.61.72.64 (...
90	Not available	Virtual SubCard MODULE [56.69.72.74.75.61.6C....	hh3cevtModuleSwitchType.0	48	module(9)	0	virtue Board [76.69.72.74.75.65.20.42.6F.61.72.64 (...
264	Not available	Ten-GigabitEthernet3/0/1 [54.65.6E.2D.47.69.67...	hh3cevtPortSw-10GBASE-CX4	87	port(10)	0	Ten-GigabitEthernet3/0/1 [54.65.6E.2D.47.69.67.6...
265	Not available	Ten-GigabitEthernet3/0/2 [54.65.6E.2D.47.69.67...	hh3cevtPortSw-10GBASE-CX4	87	port(10)	1	Ten-GigabitEthernet3/0/2 [54.65.6E.2D.47.69.67.6...
312	Not available	GigabitEthernet4/0/1 [47.69.67.61.62.69.74.45.7...	hh3cevtPortSw-1000BASE-T	90	port(10)	0	GigabitEthernet4/0/1 [47.69.67.61.62.69.74.45.74....
313	Not available	GigabitEthernet4/0/2 [47.69.67.61.62.69.74.45.7...	hh3cevtPortSw-1000BASE-T	90	port(10)	1	GigabitEthernet4/0/2 [47.69.67.61.62.69.74.45.74....
314	Not available	GigabitEthernet4/0/3 [47.69.67.61.62.69.74.45.7...	hh3cevtPortSw-1000BASE-T	90	port(10)	2	GigabitEthernet4/0/3 [47.69.67.61.62.69.74.45.74....
315	Not available	GigabitEthernet4/0/4 [47.69.67.61.62.69.74.45.7...	hh3cevtPortSw-1000BASE-T	90	port(10)	3	GigabitEthernet4/0/4 [47.69.67.61.62.69.74.45.74....
316	Not available	GigabitEthernet4/0/5 [47.69.67.61.62.69.74.45.7...	hh3cevtPortSw-1000BASE-T	90	port(10)	4	GigabitEthernet4/0/5 [47.69.67.61.62.69.74.45.74....
317	Not available	GigabitEthernet4/0/6 [47.69.67.61.62.69.74.45.7...	hh3cevtPortSw-1000BASE-T	90	port(10)	5	GigabitEthernet4/0/6 [47.69.67.61.62.69.74.45.74....
318	Not available	GigabitEthernet4/0/7 [47.69.67.61.62.69.74.45.7...	hh3cevtPortSw-1000BASE-T	90	port(10)	6	GigabitEthernet4/0/7 [47.69.67.61.62.69.74.45.74....
319	Not available	GigabitEthernet4/0/8 [47.69.67.61.62.69.74.45.7...	hh3cevtPortSw-1000BASE-T	90	port(10)	7	GigabitEthernet4/0/8 [47.69.67.61.62.69.74.45.74....
320	Not available	GigabitEthernet4/0/9 [47.69.67.61.62.69.74.45.7...	hh3cevtPortSw-1000BASE-T	90	port(10)	8	GigabitEthernet4/0/9 [47.69.67.61.62.69.74.45.74....
321	Not available	GigabitEthernet4/0/10 [47.69.67.61.62.69.74.45....	hh3cevtPortSw-1000BASE-T	90	port(10)	9	GigabitEthernet4/0/10 [47.69.67.61.62.69.74.45.74....
322	Not available	GigabitEthernet4/0/11 [47.69.67.61.62.69.74.45....	hh3cevtPortSw-1000BASE-T	90	port(10)	10	GigabitEthernet4/0/11 [47.69.67.61.62.69.74.45.74....
323	Not available	GigabitEthernet4/0/12 [47.69.67.61.62.69.74.45....	hh3cevtPortSw-1000BASE-T	90	port(10)	11	GigabitEthernet4/0/12 [47.69.67.61.62.69.74.45.74....
324	Not available	GigabitEthernet4/0/13 [47.69.67.61.62.69.74.45....	hh3cevtPortSw-1000BASE-T	90	port(10)	12	GigabitEthernet4/0/13 [47.69.67.61.62.69.74.45.74....
325	Not available	GigabitEthernet4/0/14 [47.69.67.61.62.69.74.45....	hh3cevtPortSw-1000BASE-T	90	port(10)	13	GigabitEthernet4/0/14 [47.69.67.61.62.69.74.45.74....
326	Not available	GigabitEthernet4/0/15 [47.69.67.61.62.69.74.45....	hh3cevtPortSw-1000BASE-T	90	port(10)	14	GigabitEthernet4/0/15 [47.69.67.61.62.69.74.45.74....
327	Not available	GigabitEthernet4/0/16 [47.69.67.61.62.69.74.45....	hh3cevtPortSw-1000BASE-T	90	port(10)	15	GigabitEthernet4/0/16 [47.69.67.61.62.69.74.45.74....
328	Not available	GigabitEthernet4/0/17 [47.69.67.61.62.69.74.45....	hh3cevtPortSw-1000BASE-T	90	port(10)	16	GigabitEthernet4/0/17 [47.69.67.61.62.69.74.45.74....
329	Not available	GigabitEthernet4/0/18 [47.69.67.61.62.69.74.45....	hh3cevtPortSw-1000BASE-T	90	port(10)	17	GigabitEthernet4/0/18 [47.69.67.61.62.69.74.45.74....

图 4-1 entPhysicalTable 示例

从图 4-1 的表中可以看出 entPhysicalClass 节点代表了实体类型，如实体是一个板卡、风扇、端口等；rfc2737 中一共规定了十二类 chasis, port, other, powersupply,

unknown, fan, backplane, sensor,container, module, stack，rfc4133 增加了一类 cpu。stack 是堆叠，是最顶层容器，它可以放多个 chasis；chasis 代表了网元机框，是所有 container 的最上层，是网元面板的主体部分；Module 是独立的子系统，如交换机的板卡，如果可以一般也放在容器里；container 是支持可动态移动实体的容器，如 module,fan；Port 是物理端口。

如上所示，网元实体根据 entPhysicalClass 分为 12 种类，各厂商可根据这个分类构建自己的组件树[40]。节点 entPhysicalDescr 实体描述和 entPhysicalName 实体名称分别代表了厂商所生产实体的具体描述和名称。Rfc2737 在不同的 entPhysicalClass 之间规定了几种简单的约束关系，各厂商遵守该约束关系并结合自己的产品特点设计实现自己的 MIB 管理关系[41]。网元实体必须满足一下几条简单的约束条件：（1）端口 port 类型实体必须是 Module 类型的子实体。（2）module 类型实体只允许包含 port 和 container 类型实体。（3）unkown, powersupply, module, fan, backplane,other, cpu, sensor 等物理实体必须被 container 类型实体包含，即它们父实体为 container 类型实体。（4）container 实体可以被 chasis, module, backplane 类型实体包含。（5）对于堆叠设备，其树根必须为堆叠 stack，其下只能直接包含 chasis 类型的物理实体。（6）每个机框类型物理实体 Chasis 只有一种子实体类型即容器 Container。如图 4-2 所示，形象表示了实体 Mib 树形层次关系[42]。

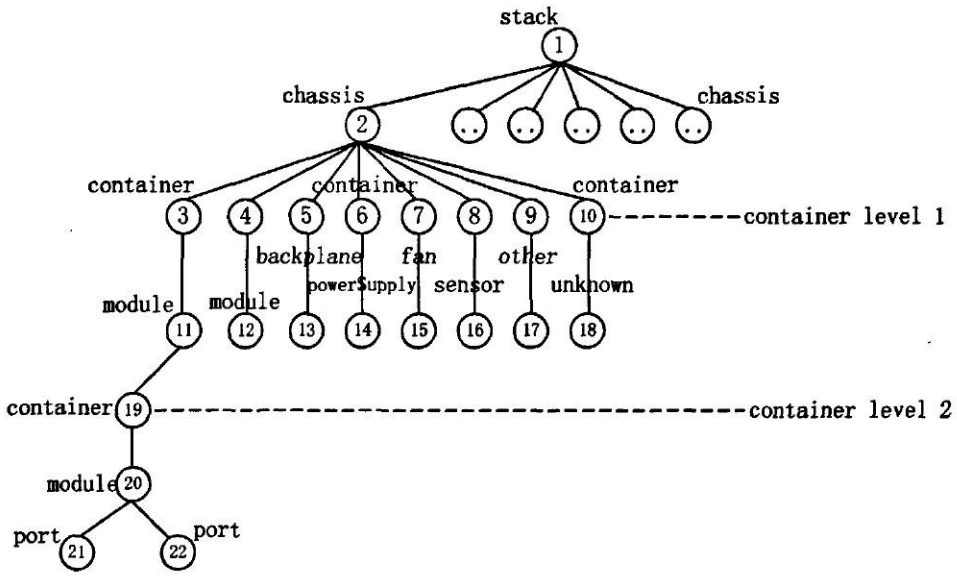


图 4-2　实体约束关系图

4.2 系统层次结构

网元三维模型管理系统采用 MVC（Model-View-Controller）的设计模式，即把一个应用的输入、处理、输出流程按照模型、视图、控制的方式进行分离，将系统分为模型层 Model、视图层 View、业务逻辑层 Controller，再加上一个数据访问层 DataAccess。每层完成相对独立的功能，层与层之间互相通信，实现系统的软件功能。

系统的层次结构图，如图 4-3 所示。

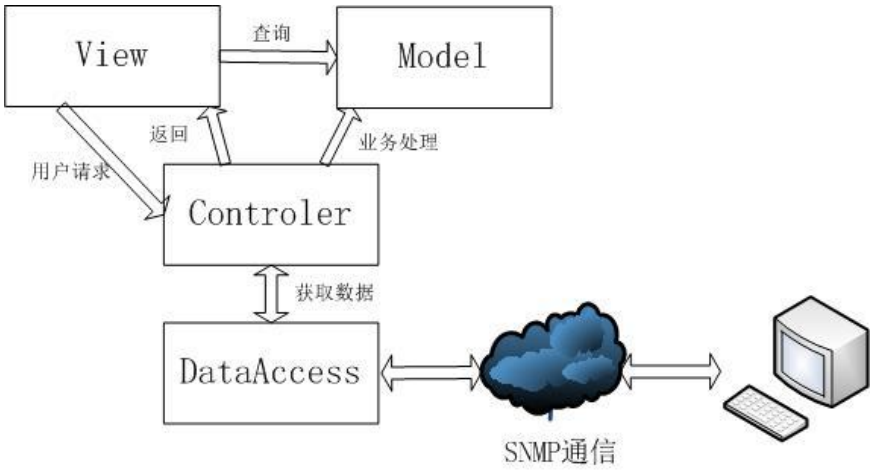


图 4-3 系统层次结构图

下面分别详细介绍各层的含义与作用。

数据访问层：提供与实际网元设备 SNMP 通信的能力，本系统主要利用 SNMP4J 开源类库来实现。该层获取网元 Mib 原始数据并提供给业务逻辑层。

模型层：该层为构建网元逻辑模型所需要的实体类。对于网元逻辑模型来说，其实质就是从堆叠到机框到板卡到端口的树状结构，每个节点都是一个网元实体对象，实体对象的创建就依赖于在模型层预先设计好的各种网元实体类，包括堆叠实体类、机框实体类、板卡实体类和端口实体类等。

业务逻辑层：业务逻辑层主要做两方面的工作，一是构建网元逻辑模型；二是将构建好的网元逻辑模型转换成 Json 格式的数据。构建网元逻辑模型利用模型层的实体类，结合数据访问层获取到的数据以及从 xml 配置文件获取的实体静态信息，最终构建出网元逻辑模型。

表现层：根据业务逻辑层提供的逻辑设备数据结构，渲染出表示当前网元实际

状态的三维模型。并且提供用户交互的功能，包括响应鼠标拖拽旋转三维模型，点击网元实体产生选中效果等。

4.3 系统模块设计

系统采用模块化设计，根据模块化编程的思想，应当尽量保持模块的独立性，减少模块之间的耦合性，从而实现系统的高内聚低耦合，使得系统具备健壮性和扩展性。系统大致可以分为如下几个模块，分别是 Page 页面模块、XML 网元实体静态信息模块、XMLParse 解析 XML 模块、DeviceController 设备控制器模块、DeviceModel 网元实体类模块、JsonBuilder 构造 Json 数据模块、Refresher 刷新器模块和 DataAcces 数据访问模块，下面分模块详细介绍各个模块的功能。

如图 4-4 所示，为系统的模块设计图。

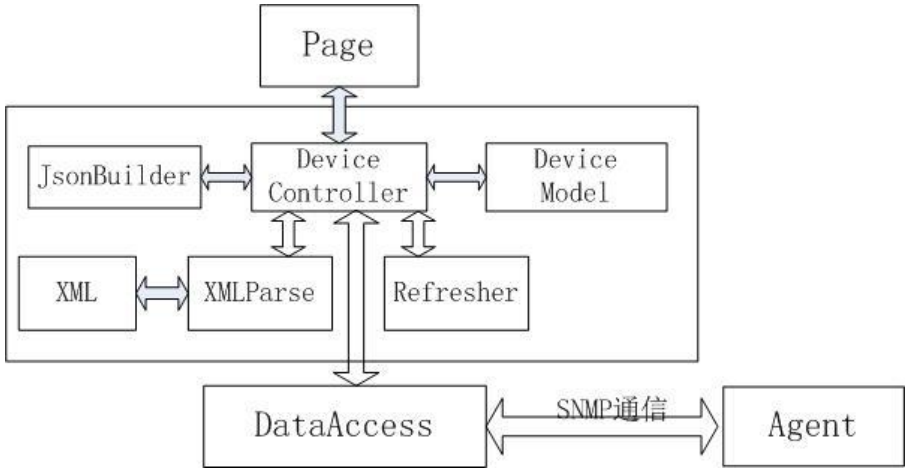


图 4-4 系统模块图

页面模块：该模块主要负责三维模型的渲染，处理与用户的互动。页面模块收到用户打开网元三维模型的请求后，向后台业务逻辑层请求该网元的实体逻辑结构。收到后台反馈后，解析反馈回来的 json 结构渲染出网元实体三维模型，这里使用到的技术是 WebGL。页面模块还监控用户的鼠标事件，根据用户点击的动作，做出相应的反应。当用户点击并拖抓鼠标时，三维模型跟随鼠标的拖抓进行旋转。当用户点击网元实体时，对应的网元三维模型（机框、板卡、端口）产生选中的效果。当用户鼠标右键网元端口实体时，弹出端口详细信息。

网元实体静态信息模块：该模块主要存储网元设备的静态结构，通过 XML 文件实现。通过 XML 文件的树状结构，将网元的实体约束关系构造出来。如最顶层是堆叠层，其下的子实体层是机框层，机框层下是槽位层等等，XML 只需要按树状结构一层层实现这些实体即可。每层又包含各层的实体信息，如实体的坐标位置、实体的图片，实体的相对位置等等。通过 XML 文件构造出网元实体的静态结构。通过这种方式管理网元，可以大大节省设备联通的工作量。每当有新的设备加入到网管系统来时，对于网元管理组件来说，只需要维护这个 XML 文件即可。

解析 XML 模块：该模块提供对网元静态信息模块即 XML 文件进行解析的功能。XML 解析技术有许多，本系统用到的是 JAXB（Java Architecture for XML Binding)。

设备控制器模块：设备控制器这里的设备指的内存中的逻辑设备。设备控制器完成的主要工作是接收用户的请求，完成网元逻辑模型的构建，并转换成 Json 格式的数据返回给前台页面模块。具体的过程是：当设备控制器收到来自前台的用户打开设备面板的请求时，调用 XMLParse 模块解析遍历 XML 文件，找到相应的网元静态信息，同时向数据访问层发出请求，获取真实网络环境中该网元的 Mib 信息，结合 Mib 信息和 XML 的静态信息，利用 Model 层设计好的实体类，构建网元实体对象树，即网元逻辑模型，最后调用 JsonBuilder 模块将该网元逻辑模型转换成 Json 格式的数据。

构造 Json 数据模块：构造了动态网元模型后，无法直接将该模型传递给前台页面，而且前台页面也无法解析这个模型，仍然需要将网元模型转换成 Json 结构。Json 是一种轻量级的数据交换格式，它是 JavaScript 的一个子集，因此浏览器可以轻松的解析它。因此动态构造 Json 模块的主要工作就是，重新解析动态网元模型，然后构造出符合该模型的 Json 结构，并传递给前台页面模块。

刷新器模块：由于网元三维模型展示的是网元当前的实时状态模型，因此它是一种对实时性能有一定要求的动态模型。为了保证一定的实时特性，设计了刷新器模块，该模块分为手动刷新和定时刷新。定时刷新是每隔一定时间，自动刷新网元三维模型，整个过程与第一次打开网元三维模型类似，重新获取网元 Mib 信息，构造网元模型，重新在页面渲染。除了定时刷新，还允许用户自己手动刷新，用户对

准网元三维模型右键鼠标，弹出的功能菜单上选择刷新按钮，网元模型执行刷新操作，过程与定时刷新一样。

数据访问模块：这里的数据指的是网元的 Mib 信息库，在网元三维模型管理系统中，特指实体表 entPhysicalTable、接口表 ifTable 以及实体接口映射表 entAliasMappingTable。数据访问模块的功能就是通过 SNMP 通信，获取网元实体表和接口表信息，并存储到适当的数据结构中。该模块主要通过 SNMP4J 开源类库来实现。

4.4 系统流程

对于网元三维模型管理系统，对其进行系统流程分析，系统流程图如图 4-5 所示。

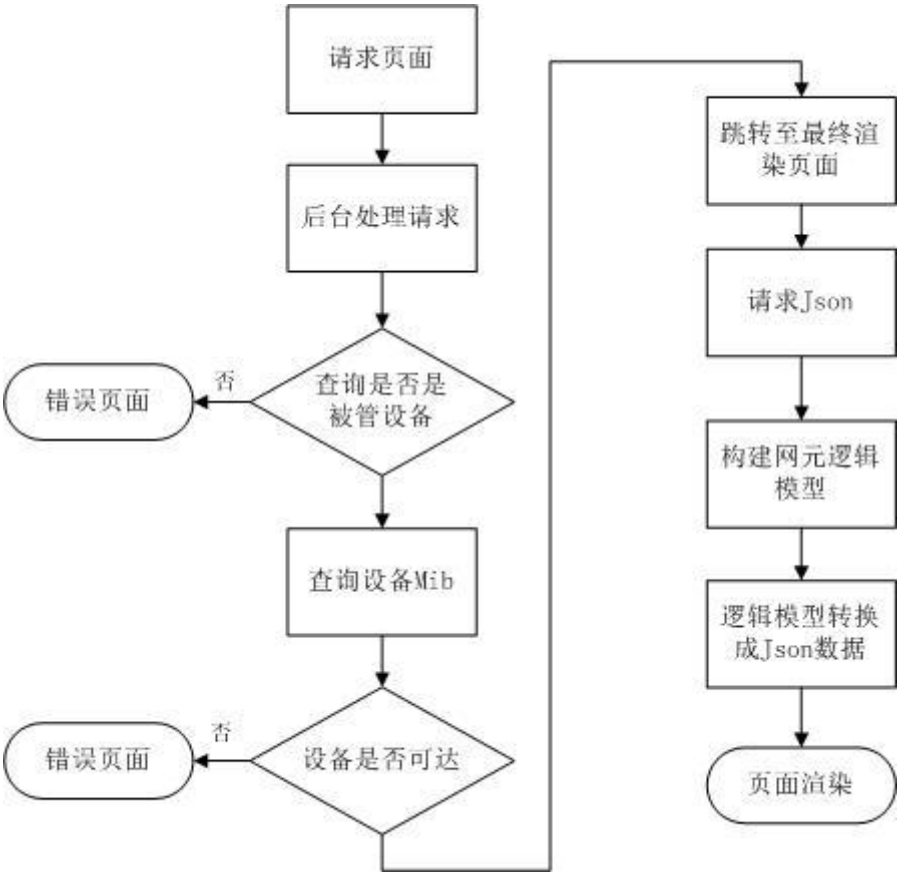


图 4-5 系统流程图

（1）用户发出对网元设备面板视图的页面请求，该请求携带两个重要参数即设备 Sysoid 和 IP。设备 Sysoid 是该设备的唯一标识，任何一个有效的 Sysoid 都唯一确

定一款型号的网元设备。IP 地址用于在真实网络环境中定位该物理设备。

（2）Web 服务器后台过滤到该请求，根据请求中的参数 sysoid，在预先定义好的存储网元静态信息的 XML 配置文件的 Stack 层里查询该 sysoid。

（3）若查询不到该 sysoid，则跳转至一个预先制作好的错误页面，向用户提示该网元设备未添加进网管系统，保证系统的容错性。

（4）若遍历查询到该 sysoid，则根据查询到的 stack 实体信息，创建网元逻辑模型的根节点 StackDataElement。并利用另一个请求参数设备的 IP 地址，调用数据访问层模块，去真实网络环境中的物理设备取设备 Mib 信息。

（5）若设备不可达，则跳转至一个预先制作好的错误页面，向用户提示该网元设备不可达，保证系统的容错性。

（6）若设备可达，并返回了网元 Mib 信息，一般取得的是所需的 Mib 表整张表信息，对于网元逻辑模型构建来说，需要取的是 entPhysicalTable、ifTable 和 entAliasMappingTable，将这些表信息各用一个数组 Vector 存放。然后系统跳转至最终渲染的页面，开始做渲染准备。

（7）页面通过 Ajax 请求，向后台请求渲染所需要的网元逻辑结构的 Json 数据。

（8）设备控制器模块开始构建网元逻辑模型，通过调用 XMLParse 模块解析 XML 配置文件里的网元静态信息和已经获取到的网元 Mib 信息，创建网元实体对象树，构造网元逻辑模型。

（9）调用 JsonBuilder 模块，将创建好的网元逻辑模型转换成前台页面可解析的 Json 数据结构。

（10） 页面在 Ajax 请求的回调函数里，获取到了网元实体模型的 Json 数据。解析该 Json，根据里面的信息逐层渲染模型，最终完成网元三维模型的渲染。最后，添加进用户交互功能。至此，整个系统的流程分析到此结束。

4.5 本章小结

本章主要介绍了网元三维模型系统的总体设计。首先给出了系统的分层结构，系统采用 MVC 的分层方式，外加数据访问层将系统分为四层。从模块化的角度为系

统设计了主要的功能模块，并讲解各个功能模块的作用。做系统流程分析时，从用户发出打开网元面板视图请求起，分析了整个系统的运作流程。最后讲解了本方案的关键技术与解决方案。

5 系统详细设计与实现

5.1 网元逻辑模型

所谓网元逻辑模型，即在内存中，通过构造合适的数据结构来描述获取到的网元 Mib 实体表中的信息。逻辑模型包含了网元三维模型构造所需要的所有信息，包括网元上个实体（堆叠、机框、板卡等）的信息及其约束关系[43]。构造网元逻辑模型，是网元管理的核心功能模块，因为构造的网元逻辑模型是构造网元三维模型的数据源，其正确性直接关系到能否渲染出正确的三维模型。构造网元逻辑模型的基本思想是：使用一个 xml 配置文件，里面记录了需要支持的网元设备的框架信息，所有支持的板卡、端口信息；当需要渲染出一个网元三维模型时，读取解析 xml 文件，找到对应的网元信息，根据该信息构建一个网元基础框架；然后读取实际设备的实体 mib 表，根据该表的网元状态信息（板卡的类型、端口的状态）去匹配充实已经构建好的网元基础框架，构成最终的网元逻辑模型。下面详细介绍构建网元逻辑模型的几个重要步骤。

5.1.1 网元实体信息存储文件

网元实体信息存储文件（以下简称 XML 配置文件）用于保存网元实体结构信息，它是一个 XML 格式的配置文件。该文件的内容是一个分层次的结构，每一层的内容都对应一种网元实体的信息，大致有 Stack 层、Chasis 层、Module 层和 Port 层等几个层次的内容。Stack 指的是堆叠，当设备支持堆叠时，为设备添加此层；Chasis 层是机框，该层包含了所有网管所支持的网元设备的机框信息；Module 层是板卡层，所有网管软件支持的板卡（包括虚拟板卡）信息都在此层；Port 层是端口层，描述的是各种类型的端口。一个网络实际设备生产出来的时候，其实体结构就已经固定下来。一般来说，无状态的网元实体信息包括：堆叠、机框、机框上的槽位、槽位上的电源、风扇、虚拟板卡及虚拟板卡上的固定端口。这些结构信息（包括图片信息、位置信息）都存储在 deviceConfig 文件中。例如 Chassis 实体需将上面将所有的 Module、PowerSupply、Fan 甚至 Port 实体都列出来，同时标明它在父实体上面的 xyz

坐标和相对位置（相对位置也就是排行第几位，对于单板来说就是槽位，对于端口来说就是第几个端口等）。下面针对 XML 配置文件各层内容做举例说明。

（1）XML 配置文件中的 Stack 层

```
<Datalist name="Stack" >
<Data name="Msr930" vendortType ="1.3.6.1.4.1.25506.1.371" >
<element entClass="Chassis" vendortType="1.3.6.1.4.1.25506.1.371"
    x="0" y="0" z="800" index="0"/>
  <element entClass="Chassis" vendortType ="" x="0" y="0" z="0" index="1"/>
  </Data>
<Data name="Msr931G" vendortType ="1.3.6.1.4.1.25506.1.374">
<element entClass="Chassis" vendorType ="1.3.6.1.4.1.25506.1.374"
x="0" y="0" z="0" index="0"/>
  </Data>
  ……
</Datalist>
```

<Datalist>节点是该层的根节点，name 属性值为 Stack，表明该层对应存储的是网元堆叠实体信息。

<Data>节点是<Datalist>的子节点，每一个<Data>节点都代表了一款堆叠设备。为了统一处理，对于不支持堆叠的网元设备，仍然需要在 Stack 层添加实体信息，当作只能堆叠一台设备的堆叠设备，如上 Msr931G 就是一台不支持堆叠路由器，而 Msr930 是一台支持两台设备堆叠的路由器。vendorType 属性是实体的唯一标识，该 vendorType 属性值和实体 Mib 表里的 vendorType 一致，通过 vendorType 来找到网元的 stack 根节点。

<element>节点是<Data>节点的子节点，<Data>节点是堆叠，根据网元实体约束关系，每个<element>子节点都代表一台网元机框。entClass 属性表明实体的类型，此处是 Chasis。vendorType 是实体唯一标识，该 vendorType 属性值和实体 Mib 表里的 vendorType 一致，每一个 vendorType 值都代表了一种网元实体，此处代表了一种网元机型。所谓堆叠，就是若干台网络设备堆叠在一起使用的设备，它有且只有一个主设备。所以我们注意到，每台堆叠其下的子实体 element 节点必有一个的 vendorType 与堆叠<Data>节点的 vendorType 一致，这个节点所代表的就是主设备。

从设备<element>节点的 vendorType 皆为空，因为堆叠设备可以堆叠各种不同的设备，在实际使用之前并不知道堆叠了那种型号的设备。Index 属性表明机框的顺序，一般主设备 index 为 0，表示第一顺位。Xyz 三维坐标，表示该机框在三维空间中的位置，是以三维机框的中心点为基准的。

（2）XML 配置文件中的 Chasis 层

```
<Datalist name="Chassis">
<Data name="Msr930" vendortype="1.3.6.1.4.1.25506.1.371"
img="msr930_front.jpg, msr930_back.jpg, msr930_top.jpg,
msr930_bottom.jpg, msr930_left.jgp, msr930_right.jpg">
<Element entClass="Module" vendortype="" x="-240" y="150" z="200" rotate ="0" index="0"/>
<Element entClass="Module" vendortype=""    x="0" y="150" z="200"
rotate ="0" index="1"/>
<Element entClass="Module" vendortype="" x="140" y="400" z="200"
rotate ="180" index="2"/>
<Element entClass="Module" vendortype="" x="210" y="400" z="200"
rotate ="180" index="3"/>
<Element entClass="PowerSupply" vendortype="" x="280" y="420"
z="-200" index="0"/>
<Element entClass="PowerSupply" vendortype="" x="350" y="420"
z="-200" index="1"/>
</Data>
<Data name="Msr931G" vendortype="1.3.6.1.4.1.25506.1.374"
img="msr931g_front.jpg, msr931g_back.jpg, msr931g_top.jpg
msr931g_bottom.jpg, msr931g_left.jpg, msr931g_right.jpg">
<Element entClass="Module" vendortype="" x="-240" y="150" z="200" rotate ="0" index="0"/>
<Element entClass="Module" vendortype=""    x="0" y="150" z="200"
rotate ="0" index="1"/>
<Element entClass="Module" vendortype="" x="140" y="400" z="200"
rotate ="180" index="2"/>
<Element entClass="Module" vendortype="" x="210" y="400" z="200"
rotate ="180" index="3"/>
<Element entClass="PowerSupply" vendortype="" x="280" y="420"
z="-200" index="0"/>
```

```
<Element entClass="PowerSupply" vendortype="" x="350" y="420"
z="-200" index="1"/>
</Data>

……
</Datalist>
```

<Datalist>节点是该层的根节点，name 属性值为 Chasis，表明该层对应存储的是网元机框实体信息。

<Data>节点是<Datalist>的子节点，每一个<Data>节点都代表了一款网元机框，表示实际的单个设备。Name 属性值是该型号设备的名称。VendortType 属性同 Mib 实体表的值，为该实体唯一标识，此处即为该机框的 sysoid。Img 属性值存放预先制作好的设备面板图片，由于是三维立体模型，预先须制作 6 张图片，分别代表六面体的六个面，图片命名规则是“设备名_方向”，如”msr930_front.jpg”。程序会根据不同的图片名称处理图片。

<Element>节点是<Data>节点的子节点，<Data>节点是网元机框，根据网元实体约束关系，可以为 Module 实体、PowerSupply 实体、Fan 实体，甚至为 Port 实体。entClass 属性表明实体的类型，此处是 Module。vendorType 是实体唯一标识，该 vendorType 属性值和实体 Mib 表里的 vendorType 一致。从上面的 xml 片段可以看出，此处的 vendorType 都为空。这是程序动态构造网元逻辑模型时需要匹配的部分，因为网元插了哪些板卡、端口需要动态获取网元实体 Mib 表才能获知。Rotate 属性表示设备上单板槽位的方向，取值为“0”表示设备上单板槽位是横向的，“90”表示设备上单板槽位为纵向的，默认为“0”， “180”表示上下颠倒。

（3）XML 配置文件中的 Module 层

```
<Datalist name="Module" >
<Data name="SQH104SG2" vendortype="1.3.6.1.4.1.25506.3. 18.3.21"
descr="" img="slot_sqh104sg2.gif">
<Element entClass="Port" vendortype="" x="130" y="9"
index="0" rotate="180"/>
<Element entClass="Port" vendortype="" x="130" y="33"
index="1" rotate="180"/>
```

```
</Data>
……
</Datalist>
```

<Datalist>节点是该层的根节点，name 属性值为 Module，表明该层对应存储的是 Module 实体信息。

<Data>节点是<Datalist>的子节点，每一个<Data>节点都代表了一款特定的 Module，通过 vendorType 值来区别。Descr 属性值是该 Module 的描述，而 img 值为预先制作好的 Module 面板图。

<Element>节点是<Data>节点的子节点， entClass 值一般为 port，代表了端口实体。vendorType 为空表明端口为可插拔，将根据实际情况渲染对应型号的端口。因为端口都是存在与板卡面板上的，只需要在板卡上定位其二维 xy 坐标。Rotate 表示旋转角度，index 表示端口序号。

（4）XML 配置文件中的 Port 层

```
<Datalist name="Port" >
<Data name="hh3cevtPortSw-1000BASE-T"
vendortype="1.3.6.1.4.1.25506.3.1.18.9.45"
descr="" status="Port1" img="1000base-t.jpg">
</Data>
……
</Datalist>
```

<Datalist>节点是该层的根节点，name 属性值为 Port，表明该层对应存储的是端口实体信息。

<Data>节点：与 Port 实体对应，表示实际的端口。vendorType 值与实际的网元实体 mib 表的值一致。Img 是预先制作好的端口图片。

（5）Status 元素对应的 XML 片段

```
<Datalist name="Status">
<StatusDatalist name="Port1" mib="ifTable.xml">
<Mibs name="ifAdminStatus;ifOperStatus"/>
<Statement name="green" value="1;1">
<Statement name="blue" value="2;1">
<Statement name="cynn" value="2;2">
```

</StatusDatalist>

……

</Datalist>

节点说明如下：

<StatusDatalist>节点：name 属性表示状态模板名称，mib 属性表示存储实体状态信息对应的 snmpConfig 文件名称。

<Mibs>节点：name 属性表明 snmpConfig 文件中存储实体状态信息的 MIB 字段名称，本例表明系统根据 ifTable.xml 文件从 ifAdminStatus 和 ifOperStatus 两字段对应的 MIB 节点中获取端口的管理态和操作态信息。

<Statusment>节点：表示实体状态信息和实体在面板上的显示颜色之间的映射关系，本例表明当 ifAdminStatus 字段对应的 MIB 节点值为 1，同时 ifOperStatus 字段对应的 MIB 节点值为 1 时，端口应显示为 green。

5.1.2 实体类设计

为与不同逻辑层的实体相对应，面板管理模块设计了五个子类，分别用来保存五种面板实体的信息，它们都继承自 DataElement 类。其中，PaneDataElement 用于保存面板数据，与 Stack 实体对应；FrameDataElement 用于保存机框数据，与 Chassis 实体对应；SlotDataElement 用于保存单板数据，与 Module 实体对应（单板对应的 Model 实体）；SubSlotDataElement 用于保存子卡数据，与 Module 实体对应（子卡对应的 Model 实体）；PortDataElement 用于保存端口数据，与 Port 实体对应。同时设计了 PowerDataElement、FanDataElement 及 OtherDataElement 类用来保存电源、风扇及 Other 实体信息，它们也继承自 DataElement 类。

为方便下文的阐述，此处简要介绍 DataElement 数据元类包括主要的数据成员及成员方法的简要说明。

1）DataElement 数据元

主要数据成员包括：

```
protected int    entityIndex;//记录实体的索引
protected int    p_entityIndex;//记录父实体的索引
```

protected int pr_entityIndex;// 此数据元素相对父元素的索引，记录相对位置

protected String vendorType;//此数据元素的类型，就是设备返回的实体的 vendortype 值

protected int m_iAdminStatus;// 此数据元素的管理状态

protected int m_iOperStatus;// 此数据元素的操作状态

protected DataElement m_deParent;// 此数据元素的父对象

protected Vector m_vSubList;// 此数据元素的子列表

protected DeviceController m_dmDevice;// 此数据元素对应的设备管理类

public boolean m_ifChildNeedDraw;// 子图元是否需要绘制

protected String m_strVendorType;//设备的生产厂商

主要的成员方法：

（1）abstract protected Folder getFolder(String strFolderName);

/**从 deviceConfig 配置文件中取到对应节点的 folder 对象

*@param strFolderName 具体的 folder name

@return 对应名称的 Folder/

（2）protected Hashtable getFolderHash(Folder folder);

/**把一个 Folder 对象转化为一个 Hashtable，这里的 Hashtable 对象以实体

*vendortype 值为键值

*存储自定义的一个数据结构 TypeSubInfo（包括实体的角度、位置信息）

*@param folder 具体的 folder

@return 传入 folder 转化的 Hashtable/

（3）public int[] getStackHightWidth();

/**获取堆叠设备面板的宽度及高度。注意以所有堆叠设备高度的加和作为

*堆叠设备面板的高度

*以堆叠设备中最宽设备的宽度作为堆叠面板的宽度。

@return 堆叠面板的宽度和高度/

（4）public Icon getBGImage();

/**获取当前元素的背景图片

@return 实体图片信息/

（5）public Point[] getPositions();

```
/**获得当前元素的位置 m_iIndex
*由父数据元类型、本数据元的 m_iIndex 决定。
*@return 位置*/
```

当网元业务逻辑层收到前台页面的用户请求时，根据用户请求的参数 sysoid，遍历存储网元静态模型结构的 xml 文件，找到与之相匹配的设备。构造好的网元静态模型是一个包含实体约束关系的模型，它是一个包含父子实体关系的树状结构。在构建动态模型时，从根节点开始遍历整个树结构。对于每个实体节点，遍历实体 Mib 表，看是否能匹配到与其相对应的记录。该匹配的规则包含 VendorType 和 entContainIn 两方面。

5.1.3 构建逻辑模型

逻辑模型包含真实环境中网元的实时状态，有了前面介绍的 xml 静态配置文件和设计好的网元实体类，就可以动态构造网元逻辑模型了。本模块使用了 MVC 三层结构，DeviceController 是控制层的核心，它完成了动态构造模型的主要工作以及视图层和模型层之间的数据交换。

根据前面的系统流程分析，当用户发出请求打开一个网元三维模型时，该请求携带一个唯一标识该网元的 sysoid。Web 服务器过滤到该请求，转交给对应的模块 DeviceContoller 面板控制器去处理。此时，DeviceController 按如下步骤动态构造网元逻辑模型，以 Msr931G 设备为例：

（1）根据请求中的 sysoid 参数（以 Msr931G 为例为 1.3.6.1.4.1.25506.1.374），遍历配置文件 Stack 层，找到对应的实体配置信息如下：

```
<Data name="Msr931G" vendortType ="1.3.6.1.4.1.25506.1.374">
<element entClass="Chassis" vendorType ="1.3.6.1.4.1.25506.1.374"
x="0" y="0" z="0" index="0"/>
</Data>
```

同时，DeviceContoller 向数据访问层发出访问网元实体 Mib 的请求，数据访问层根据请求的 ip 及设备 sysoid，获取设备的实体 mib 表，返回给 DeviceController，返回的实体表数据存储在一个向量 Vecotor 里。系统构建的网元逻辑模型所需的信

息，静态结构信息存在 xml 配置文件里，实时状态信息从实体表 Vector 取。

由于 Msr931G 设备不支持堆叠，DeviceController 无需找第二台设备，直接根据配置文件构建节点 StackDataElement。网元逻辑模型是一个树结构，StackDataElement 是其根节点。对于不支持堆叠的单设备来说，实体表是没有堆叠这一层次数据的。因此，设定其实体索引 entityIndex，父实体索引 f_entityIndex，父实体相对索引 rf_entityIndex 皆为-1。根据子节点 elment，构建 chasis 实体 ChasisDataElement，将 vendortType 信息、坐标信息填入到相应的 ChasisDataElement 属性中去。最后将构造好的 ChasisDataElement 添加至 StackDataElement 的子元素列表。

（2）步骤 1 完成了根节点 Stack 层的构建，但其子实体 Chasis 仍未充实。构建网元逻辑模型的过程，就是不断充实网元对象树的过程。根据 ChasisDataElement 中的 vendorType 属性，遍历 xml 配置文件的 Chasis 层，找到对应的实体配置。Chasis 层有个重要属性是 img，它包含预先制作好的 6 张设备面板图片的名称，分别代表网元三维模型的六个面。将 6 张图片名称存入到 ChasisDataElement 的 imgArr 属性中，该属性是一个 String 类型的数组。从配置文件看出该 Chasis 实体下可以插 4 个 Module，2 个 PowSupply。因此，新建 4 个 ModuleDataElement 和两个 PowerDataElement 添加进 Chasis 子实体列表，将其代表实体顺序的 index 值，填入到对应的类属性 rf_entityIndex 中。Chasis 层静态信息构建完毕，还需要进一步遍历步骤一取到的 Mib 实体表 Vector，找到实体类型 entClass 为“Chasis”的一行 Mib 数据，表明找到一个机框实体。该行的 entIndex、vendorType、relPos、containedIn 等属性值保存到 ChasisDataElement 中的对应属性 entityIndex、vendorType、rf_entityIndex 和 f_entityIndex。

（3）步骤 2 建立的 Chasis 只包含 Chasis 自身的信息，其下 6 个子实体都只是“空壳”，相当于 6 个待槽位（槽位是有顺序的，顺序关系体现在步骤二时填充的属性 rf_entityIndex），系统要根据网元实际的情况，填充相应款型的板卡或其他实体。遍历实体表 Vector，寻找匹配的实体。匹配的条件主要有两个，一个是实体类型是 Module，一个是相对父实体的相对位置（即 rf_entityIndex）。对于匹配上的实体，将该行的 entIndex、vendorType、containedIn 等属性值保存到 ModuleDataElement 中的

对应属性 entityIndex、vendorType、f_entityIndex。对于未匹配上的，其索引值 entityIndex 置为-1。所有 entityIndex 不为-1 的 ModuleDataElement，表示已经匹配上了网元实体 mib 表里 Module 实体。遍历 xml 配置文件，根据每个匹配上的 Module 的 vendorType 值去 xml 配置文件的 Module 层找到对应的 Module 信息，用这些 Module 信息去填充 ModuleDataElement 的对应属性。主要包括图片属性 img 和子实体列表属性。此处填充的子实体列表，与步骤二一样，每个子实体都是含有父索引值 f_entityIndex 和父实体相对索引信息 rf_entityIndex 的“空壳”。至此，网元模型的 Module 层构建完毕。

（4）开始构建 Module 层下的子实体，主要是 port 实体。步骤三已经创建好 PortDataElement，并含有部分实体信息，主要有父实体索引 f_entityIndex 和父实体相对索引 rf_entityIndex。有了这两个信息，就可以在实体 mib 里定位到该 port 记录。具体匹配方法是，遍历所有 entClass 为 Port 的记录，找到 relPos 与 rf_entityIndex 一致，并且 containIn 与 f_entityIndex 一致的记录。将该记录的 vendorType 和 entityIndex 填充到对应的 PortDataElement 中的属性中去。再根据该 port 的 vendorType 遍历 xml 配置文件，在 Port 层找到与其 vendorType 一致的 port 实体信息。将 xml 文件中的该 port 信息填充到 PortDataElement 中的属性中去，主要有该 Port 的图片属性。

（5）对于 Port 实体，系统还需要知道实体当前的管理状态和操作状态，以确定此端口在面板上的显示颜色。Port 的状态信息是存在网元 mib 中的 ifTable 里，有两个节点 ifAdminStatus 和 ifOperStatus，它们的取值只能是 0、1。通过这两个节点，可以表示端口的三个状态，1/1、1/0 和 0/0，设备端口没有 0/1 这个状态。系统通过不同颜色的端口来区分这三个状态。具体的做法是，每种端口都预先制作好三种颜色的图片代表三种不同的状态，并且每个图片的命名都是端口名_颜色的格式。构建逻辑模型致 port 层时，对于每个 port 实体，遍历 ifTable 找到对应的 port，取出其 ifAdminStatus 和 ifOperStatus，根据这两个属性值，手动拼接图片名称。将该图片名称填入到 PortDataElement 中的 img 属性中。至此，整个网元逻辑模型构建完毕。

5.2 网元 Json 结构

构造好的网元静态模型是一个包含父子实体关系的树状结构，每一层都涵盖了

该层实体的所有实体信息。在系统后台，该网元逻辑模型实际就是一个对象树。前台页面其实是无法解析该对象树的，因此，还需要将该对象树转换成为前端可以解析的 json 结构。

Json(Javascript Object notation)即 javascript 对象表示方法，它是存储和交换文本信息的语法，类似于 XML，但比 XML 更小、更快、更易解析。Json 的语法规则很简单，四点可以概括：（1）数据是在名称/值对中，如”firstname”:”John”，值和名称都须添加引号，之间用冒号分隔（2）数据由逗号分隔（3）花括号保存对象（4）方括号保存数组。

将一个对象转换成 Json 格式的数据，有很多开源类库做了这件事情，如 Jackson、json-lib、fastjson 等。转换的基本思想都是根据对象的类型，通过 StringBuild 拼接 Json 对象。对于网络来说，其实 Json 对象实质就是一个特定格式的字符串。由于网元逻辑模型的层次结构、每层的对象属性都较为固定，为系统设计了自己的转换工具 JsonUtil。

以 Msr931G 单设备插一块板卡型号 SQH104SG2 为例，以下如图 5-1 所示是经过转换后的 Json 格式数据。

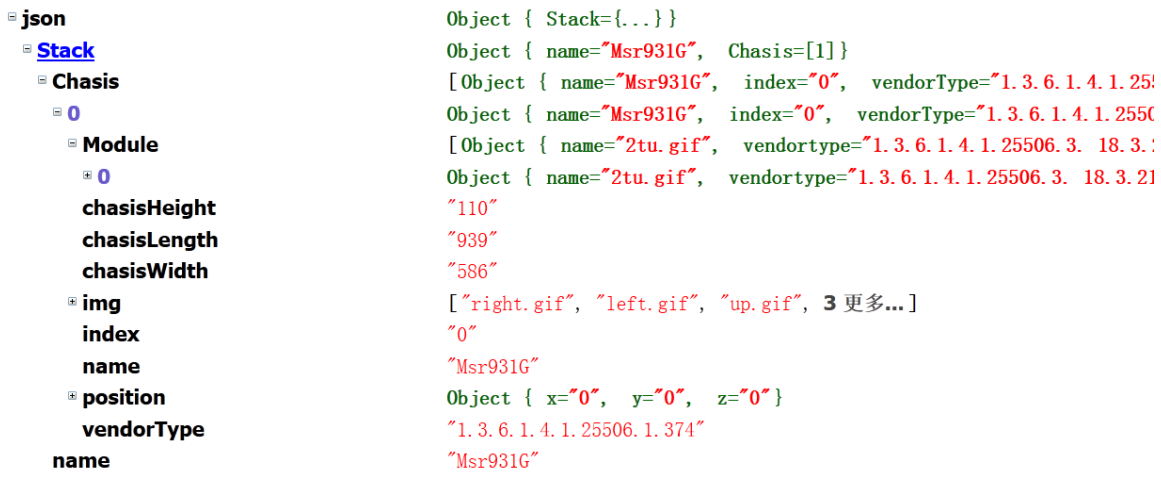


图 5-1　网元结构 Json 格式

5.3　三维渲染

5.3.1　JavaScript 解析 Json

前台页面收到 Ajax 请求返回的 Json 数据后，开始解析 Json，根据 Json 里的模

型信息渲染网元三维模型。在数据传输流程中，Json 是以文本，即字符串的形式传递的，而 JS 操作的是 Json 对象，所以，首先需要将 Json 字符串转换成 Json 对象，使用语句 var model = eval('('+json+')')。

从网元的逻辑结构来看，除了 Stack 层，其余 Chasis 层、Module 层和 Port 层都是对象数组。Json 对象属性的访问方式很简单，通过对象+"."+属性的方式。因此很容易获得堆叠对象：var stack = model.Stack。

对于 chasis 对象，定义成一维数组的形式：var chasis =stack.Chasis。

对于 Module 对象，定义成二维数组的形式，其定义和初始化如下：

```
var modules=[];
for(var i=0;i<chasis.length;i++){
    modules[i] = new Array();
    for(var j =0;j<chasis[i].Module.length;j++){
        modules[i][j] = chasis[i].Module[j];
    }
}
```

使用一个嵌套循环，外层遍历 chasis 对象数组，在外层循环里创建二维数组，内层循环遍历 Chasis[i]的所有 Module 对象，进行初始化。module[i][j]中，第一个下标 i 表示 chasis 的序号，第二个下标 j 表示 module 的序号。

对于 port 对象，定义成三维数组的形式，其定义和初始化如下：

```
var ports = [];
for(var i=0;i<chasis.length;i++){
ports[i] = new Array();
  for(var j =0;j<chasis[i].Module.length;j++){
    ports[i][j] =  new Array();
      for(var k=0;k<chasis[i].Module[j].port.length;k++){
          ports[i][j][k] = chasis[i].Module[j].port[k];
      }
    }
}
```

使用一个三层循环，第一层循环遍历 chasis 对象数组，在第一层循环里创建二维数组，遍历 Chasis[i]的所有 Module 对象，在第二层循环里创建三维数组，第三层

循环遍历所有的 Port 对象，进行初始化。ports[i][j][k]中，i 表示 chasis 的序号，j 表示 module 的序号，k 表示 port 的序号。

5.3.2 渲染准备

至此，我们已经可以从 Json 数据中解析出所有渲染网元三维模型所需要的网元实体信息了。接下来就要利用这些信息利用 WebGL 技术渲染出网元三维模型。

WebGL 的原生 API 是相当低级的，即使一个最简单的渲染仍然需要做大量的工作，需要有丰富的 3D 编程经验的程序员才能驾驭。现在有许多不错的 WebGL 开源 3D 引擎，它们构建了一个高等级的、对开发者相对友好的 WebGL 开发环境。本系统在众引擎中选择了最受欢迎的 Three.js。Three.js 封装了 3D 渲染的细节，利用其丰富的内置对象，可以快速上手构建需要的 3D 世界。

在利用 Three.js 做渲染前，有些通用的渲染准备的步骤需要做。这些通用的步骤是每个三维渲染页面都必须做的，因此，称之为渲染准备。

（1）容器准备

要利用 Three.js 引擎库，需要在页面中引用它。使用<Script>标签来嵌入 Three.js 的库文件：<Script source=”../libs/Three.js”>。然后就能通过全局变量 THREE 访问到所有属性和方法了。

渲染是在一个页面容器 container 中进行的，该容器可以是一个 div 块，也可以直接是 body 主体。需要注意的是，我们并不需要写一个<canvas>标签，我们只需要定义好盛放 canvas 的 div 就可以，canvas 是 three.js 动态生成的。

（2）设置渲染器 Renderer

三维空间里的物体映射到二维平面的过程被称为三维渲染。 一般来说我们都把进行渲染操作的对象叫做渲染器。

（3）设置相机 Camera

使用 Three.js 创建的场景是三维的，而通常情况下显示屏是二维的，那么三维的场景如何显示到二维的显示屏上呢？照相机就是这样一个抽象，它定义了三维空间到二维屏幕的投影方式，用“照相机”这样一个类比，可以使我们直观地理解这一投影方式。在图形学里，投影方式分为正交投影和透视投影两种。正交投影是垂直的

投影，大小和平行性不会改变，透视头像类似人眼成像有立体感，平行性会有改变。与此相对应，在 Three.js 里，相机也有正交投影相机和透视投影相机两种。一般说来，对于制图、建模软件通常使用正交投影，这样不会因为投影而改变物体比例；而对于其他大多数应用，通常使用透视投影，因为这更接近人眼的观察效果[44]。当然，照相机的选择并没有对错之分，你可以更具应用的特性，选择一个效果更佳的照相机。

本系统采用透视投影相机（Perspective Camera），透视投影相机的构造函数如下：THREE.PerspectiveCamera(fov, aspect, near, far)。如图 5-2 所示，透视图中，灰色的部分是视景体，即渲染器可以渲染的区域。fov 是视景体竖直方向上的张角（是角度制而非弧度制），如侧视图所示。aspect 等于 width / height，是照相机水平方向和竖直方向长度的比值，通常设为 Canvas 的横纵比例。near 和 far 分别是照相机到视景体最近、最远的距离，均为正值，且 far 应大于 near。

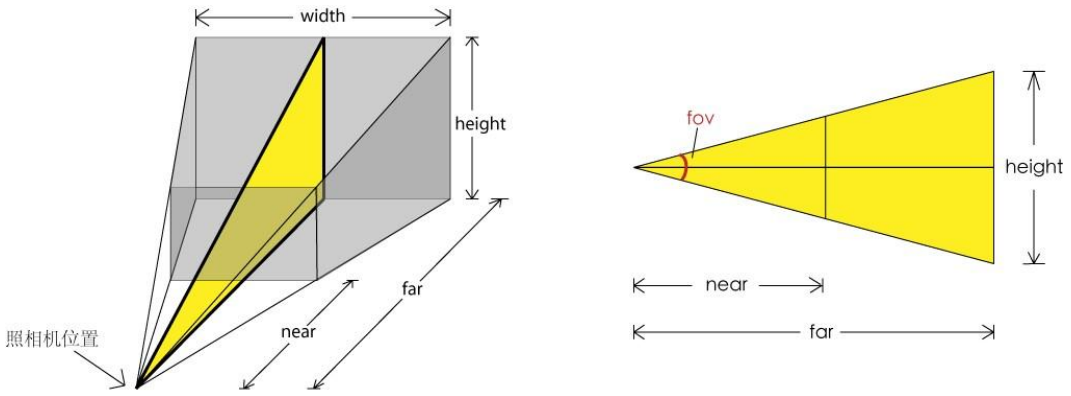


图 5-2　透视图与侧视图

除了构造函数，设置相机往往还需要设置相机的位置。通过 camera.position.set（x,y,z）来实现。通常，习惯设置相机位置位于 Z 轴正半轴上。因为默认情况下，相机都是沿 Z 轴负方向观察到。

（4）设置场景 Scene

在 Three.js 中添加的物体都是添加到场景中的，因此它相当于一个大容器。一般说，场景来没有很复杂的操作，在程序最开始的时候进行实例化，然后将物体添加

到场景中即可。场景的定义如下：var scene = new THREE.Scene()。需要注意的是，照相机 Camera 也需要添加到场景中去。

5.3.3 主体机框渲染

在 WebGL 开发中最常用到的渲染对象大概就是网格了。网格上由一系列的多边形（通常是三角形或四边形），每个多边形都是由三个或者更多的 3D 空间的顶点来描述的。一个网格既可以是一个简单的三角形，也可以是极其复杂的真实世界中的物体，如汽车、飞机或人。

在 Three.js 里，网格 Mesh 的唯一构造函数是：Mesh(geometry , material)。其两个参数 geometry 表示几何形状，而 material 表示材质。很好理解，创建任何物体需要指定几何形状和材质，其中，几何形状决定了物体的顶点位置等信息，材质决定了物体的颜色、纹理等信息[45]。

有了网格、材质、纹理，再加上前面通过解析 Json 获得的网元机框 Chasis 对象就可以构建供页面渲染的机框 Mesh 对象了。下面详细介绍机框 Mesh 构建的详细步骤。

（1）外层循环

根据网元实体结构知道，一个网元模型，其机框对象是可以多个的，定义时被定义成了对象数组。因此取得的 chasis 对象是个对象数组。需要在一个循环里循环构建所有的机框， for(var i =0;i<chasis.length;i++)。

（2）构建纹理

本系统的网元模型，其材质都是由纹理贴图来填充的。对于每一个添加进网元管理系统的网元设备，都需要预先根据其实际纹理来绘制好贴在立方体六面的纹理图片。不止是网元设备，包括板卡和端口等可插拔的动态网元实体，在添加进管理系统时都需要预先制作好其纹理图片。

网元机框比较特殊，它是个有六面的立方体。因此需要为这六个面都创建纹理对象。通过第 i 个 chasis 对象获取到所有图片路径：imgs = chasis[i].img。imgs 是一个包含 6 张纹理图片路径的字符串数组。在一个 for 循环里为每张图片创建一个纹理对象，代码如下。

```
for(var j=0;j<imgs.length;j++){
        var img = new Image();
        img.src =imgs[j];
        var tex = new THREE.Texture(img);
        img.tex = tex;
        img.onload = function() {
            this.tex.needsUpdate = true;
    };
```

对每一个图片路径，创建一个图片对象：new Image()。将图片的路径赋给图片对象的 src 属性中。创建纹理对象，将图片对象作为构造器的参数传递。创建纹理对象：var texture = new THREE.Texture(img)。需要注意的是，在调用图片加载函数时，在 onload 里有这么句：this.tex.needsUpdate = true。这句话的作用就是告诉 renderer 到这一帧应该更新缓存了。如果在创建好 img 后直接写 texture.needsUpdate=true 的话，three.js 的 renderer 中会这一帧中就使用_gl.texImage2D 将空的纹理数据传输到显存中，然后就将这个标志位设成 false, 之后真正等到图片加载完成的时候确不再更新显存数据了，所以必须要在 onload 事件中等整张图片加载完成后再写 texture.needsUpdate = true，

（3）构建材质

在 Three.js 中，提供了很多种材质可供选择。最基本的材质 MeshBasicMaterial，渲染后物体的颜色始终为该材质的颜色，而不会由于光照产生明暗、阴影效果。如果没有指定材质的颜色，则颜色是随机的。Lambert 材质是符合 Lambert 光照模型的材质。Lambert 光照模型的主要特点是只考虑漫反射而不考虑镜面反射的效果，因而对于金属、镜子等需要镜面反射效果的物体就不适应，对于其他大部分物体的漫反射效果都是适用的。Phong 材质（MeshPhongMaterial）是符合 Phong 光照模型的材质。和 Lambert 不同的是，Phong 模型考虑了镜面反射的效果，因此对于金属、镜面的表现尤为适合。

本系统中用到了两种材质，一种是最基本的材质 MeshBasicMaterial，一种是多面材质 MeshFaceMaterial。所谓多面材质，其实可以理解成一个材质数组，是针对多面体可以对多面体每个面赋上一种材质，材质的顺序与多面体的每个面对 face index

有关。对于本系统，上一步骤中，在循环体中，每次创建一个纹理对象，就 var mat = new THREE.MeshBasicMaterial({color: 0xffffff, map: tex})创建一个基本材质。定义一个数组，把创建的基本材质都放进去，materialArray.push(mat)。最后利 materialArray 创建多面材质 new THREE.MeshFaceMaterial(materialArray)。

（4）构建几何框架

对于本系统来说，网元机框的几何形状是一个立方体。在 Three.js 中提供了许多基础几何形状，其中立方体为CubeGeometry。所有基础几何形状都是继承自 Geometry 类。创建一个立方体很容易，其构造函数如下：

THREE.CubeGeometry(width, height, depth, widthSegments,heightSegments, depthSegments)。这里，width 是 x 方向上的长度；height 是 y 方向上的长度；depth 是 z 方向上的长度；后三个参数分别是在三个方向上的分段数，如 widthSegments 为 2 的话，代表 x 方向上水平分为三份,那么每个面将被表现成 2×2=4 个面，整个立方体由 24 个表面组成，正如同网格一样。一般情况下不需要分段的话，可以不设置后三个参数，后三个参数的缺省值为 1。构造时，只需将从 Json 文件中解析出的 Chasis[i]的长宽高赋给 CubeGeometry 构造函数的参数即可构造出需要的立方体。最后将 chasis[i]的坐标属性赋给立方体的三维坐标。

（5）构建网格

网格是由几何形状和材质共同构成的，是渲染器最终渲染的对象。构建网格很简单，利用其构造函数将前面构造好的材质和形状传入即可，如下：new THREE.Mesh(geometry, DiceBlueMaterial)。至此，循环体内的一次循环构造出一个 chasis[i]。将 chasis[i]放进预先定义好的数组 chasisArray。

最后，当所有的机框都构建好了，还需要将所有机框加入到场景中，就可以被渲染器渲染出来了。如图 5-3 所示，是非堆叠设备的一个机框渲染的实例。

图 5-3　网元机框

5.3.4 板卡和端口的渲染

有了主体机框，还需要对板卡和端口进行渲染。虽然，对于网元逻辑结构来说，板卡和端口存在一个父子关系，即板卡通常是端口的父实体。但对于页面渲染来说，机框、板卡、端口等都是对等的需要渲染的实体对象。板卡和端口的渲染过程几乎完全一样。与机框不同，板卡和端口是感官上二维的。所谓感官上二维，从真实网元物理设备来看，板卡和端口都是插在机框里的，只在机框面板上显示出插槽，视觉上就是贴在网元机框面板上的二维实体。

板卡和端口的渲染过程与机框的渲染过程基本一致。

（1）外层循环

Json 结构中，板卡是一个二维数组，端口是一个三维数组。所以外层循环，板卡是一个二层嵌套循环，modules[i][j]表示第 i 个机框的第 j 块板卡。端口是一个三层嵌套循环，ports[i][j][k]表示第 i 个机框的第 j 块板卡的第 k 个端口。

（2）构建纹理

分别取得板卡和端口的图片，modules[i][j].img 和 ports[i][j][k].img，与机框不同的是，该图片路径只有一个，无需创建多个纹理。对于端口来说，还需要注意的是，端口是有不同的状态的，不同的状态是用不同颜色的端口来表示的，不同颜色端口的图片是预先制作好的。因此，在创建端口纹理时，还需判断端口是处于何种状态。判断的依据是端口对象下的 Status 属性中的 ifAdminStatus 和 ifOperStatus。若两者值皆为 1，则表示端口正常工作，贴绿颜色端口；若为 1：0，则端口异常，贴红色端口；若为 0：0，则端口不工作，贴原始图片端口。不同颜色端口是通过拼接端口图片的名称来实现的。

（3）构建材质

因此也只需用到一个最基本的 MeshBasicMaterial 创建材质对象。每次创建一个纹理对象，就 var mat = new THREE.MeshBasicMaterial({color: 0xffffff, map: tex})创建一个基本材质。

（4）构建几何体

与机框不同的是，板卡和端口构建的几何体是一个三维世界中的平面。利用 new

THREE.PlaneGeometry(length,width)来创建。将 Json 中的板卡对象和端口的对象的长宽属性传进去。

（5）构建网格

构建网格使用 new THREE.Mesh(geometry,mat)来创建，将前面构造的几何对象 gemetry 和材质对象 mat 传进去。

至此，板卡和端口的渲染对象网格就创建完毕了，只需将所有的板卡和端口加入到场景，渲染器就能渲染出来。如图 5-4 所示。

图 5-4　网元机框、板卡与端口

5.4　交互功能

图形、动画和交互是任何在线媒介必要的三要素。到目前为止，我们已经利用图形完成了网元三维模型的构建，但还没有任何的交互，对用户的鼠标事件不会做出任何反应。　WebGL 只是一个图形绘制系统，并没有内置任何点击检测。Three.js 内置了此项支持，可以告诉我们鼠标位于哪个物体之上。借助于此，我们可以实现滚动、点击、拖拽及相机漫游等交互行为。本方案里，我们需要实现拖过鼠标的拖拽，网元三维模型跟随鼠标旋转。

5.4.1　循环渲染

到目前为止构造的模型仍然是静止不动的，物体运动还有一个关键点，就是要渲染物体运动的每一个过程，让它显示给用户。渲染的时候，我们调用的是渲染器的 render() 函数。代码如下：

```
renderer.render( scene, camera );
```

如果我们改变了物体的位置或者颜色之类的属性，就必须重新调用 render()函数，才能够将新的场景绘制到浏览器中去。不然浏览器是不会自动刷新场景的。

如果不断的改变物体的颜色，那么就需要不断的绘制新的场景，所以我们最好的方式，是让画面执行一个循环，不断的调用 render 来重绘，这个循环就是渲染循

环。为了实现循环，我们需要 javascript 的一个特殊函数，这个函数是 requestAnimationFrame。调用 requestAnimationFrame 函数，传递一个 callback 参数，则在下一个动画帧时，会调用 callback 这个函数。于是，我们的循环渲染会这样写：

```
function animate() {
render();
requestAnimationFrame( animate );
}
```

这样就会不断的执行 animate 这个函数。也就是不断的执行 render()函数。在 render()函数中不断的改变物体或者摄像机的位置，并渲染它们，就能够实现动画了。

5.4.2 模型的缩放

模型缩放的原理非常简单，从透视关系就能知道，在三维场景中，物体总是近大远小的。如图 5-5 所示，要放大物体，只需拉近相机与物体的距离，缩小物体，只需拉远相机与物体的距离。

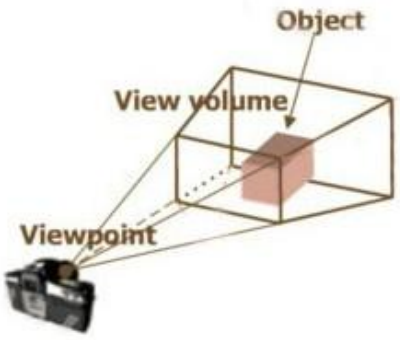


图 5-5 相机与渲染区域

下面详细介绍模型缩放的实现过程。

（1）鼠标事件的监听

模型的缩放事件起因是鼠标滑轮的滚动，因此，需要对鼠标事件进行监听，如下 this.domElement.addEventListener('mousewheel', mousewheel, false)。

在处理鼠标滑轮事件 function mousewheel(event)中，使用一个变量 delta 来表示滚轮滚动的幅度。需要注意的是，对于不同的浏览器滚轮事件的表示是不一样的，在 Webkit/IE 9/Opera 中，由 event.wheelDelta 表示；而在 FireFox 中，由 event.detail 来表示。因此需要同时考虑多种浏览器兼容的情况。

对于 event.wheelDelta 的返回值，如果是正值说明滚轮式向上滚动，如果是负值说明滚轮向下滚动。返回值均是 40 的倍数，即幅度大小=返回值/40.而对于 event.detail 的返回值，如果是负值说明滚轮式向上滚动，如果是正值说明滚轮是向下滚动。返回值均是 3 的倍数，即幅度大小=返回值/3。

在滚轮事件处理的最后，使用_zoomstart 变量来保存放大缩小的距离。zoomstart 和 delta 之间的换算关系是_zoomStart.y += (1 / delta) * 0.05;

（2）模型的放大缩小

定义了一个 ZoomCamera 函数来实现模型的放大与缩小。放大缩小的原理已经清楚，只需要改变相机的 eye 向量的长度即可。相机的 eye 向量是从相机出发指向所观察目标点的一个向量。改变向量的长度是通过计算定义的一个缩放因子 factor 来实现。计算 factor 有 var factor = 1.0 + (_zoomEnd.y - _zoomStart.y) * _this.zoomSpeed。其中 zoomstart 通过 delta 计算出来。而 zoomend 是通过监控鼠标事件 mousemove 得到，zoomEnd = _this.getMouseOnScreen(event.clientX, event.clientY)。getMouseOnScreen 是得到鼠标在屏幕上坐标的方法。

最后通过计算出的放大缩小因子，调用向量的缩放函数 multiplyScalar，代码如下 eye.multiplyScalar(factor)。至此，就完成了模型的缩放功能。

5.4.3 模型的旋转

作为一个三维模型，通过用户鼠标拖拽来控制模型旋转是与用户的最基本的交互功能了。一般来说，一个模型要旋转，首先想到的是模型自身的旋转。一种方案是，通过监控鼠标的点击拖拽事件，根据拖抓的距离与方向，通过旋转矩阵等手段来调整模型。这种方案并不是最优的，我们知道，物体的运动是相对的，除了旋转物体本身，也可以通过旋转视点即照相机来完成视觉上的模型旋转效果。本系统采用的就是后一种旋转方案，通过监控鼠标在屏幕上移动的向量来计算转换成照相机所需要旋转的方向与角度，从而达到模型旋转的效果算法。通常为了得到平滑的旋转效果，把鼠标在二维平面的坐标移动映射到三维球面，根据鼠标在球面旋转的角度来完成相机的旋转，因此该算法通常称为球面相机旋转算法。其原理图如图 5-6 所示，左边是沿着 X 轴旋转的情景，右边是沿着 Y 轴旋转的情景。

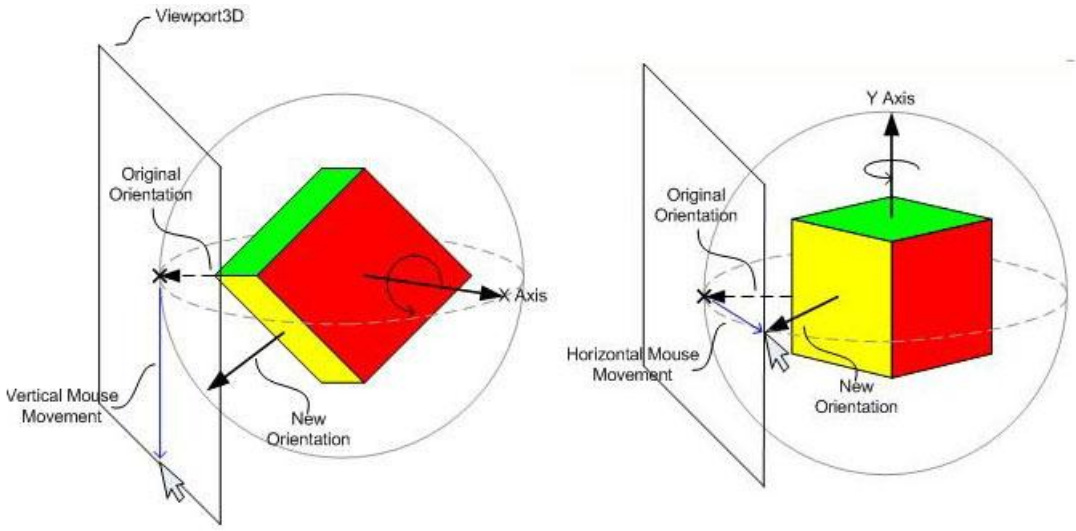


图 5-6　球面相机旋转算法

其实该算法不仅可以沿着坐标轴旋转，可以任意角度的旋转。下面详细介绍球面相机旋转算法的实现步骤。

（1）鼠标事件监听

模型旋转的动因是来自于鼠标事件，鼠标的拖拽导致了模型的旋转，因此需要对鼠标事件进行监听。

在 WebGL 里有三个事件特别有用：mousemove、mouseup 和 mousedown，本系统的拖拽事件涉及到这三个鼠标事件。当鼠标移动到某个对象时就会引发 mousemove 事件，在某个元素之上按下鼠标按键就会促发 mousedown 事件，在某个对象上释放鼠标按键引发 mouseup 事件。这三个事件都包含 clientX 和 clientY 属性，注意的是，这两个属性是指鼠标指针相对于浏览器视口的左上角的位置信息，而不是相对于三维坐标原点的位置信息。

mousedown 和 mouseup 也包含一个 button 属性，它决定了鼠标上的哪一个按键被按下或释放。如果 button 为 0，则表示此事件与鼠标的左按键有关；如果 button 为 1，则此事件与鼠标的中间按脚有关；如果 button 值为 2，则与鼠标的右按键有关。

鼠标拖拽动作是开始于用户按下鼠标左键。需要给页面添加事件监听器，同时给定事件处理程序。

代码如下：

```
document.addEventListener( 'mousedown', onDocumentMouseDown, false );
```

其中，onDocumentMouseDown 是定义的鼠标按下处理事件。它做了两件事，一是监听鼠标移动事件，二是将鼠标屏幕坐标转换成球面的坐标。鼠标移动和鼠标弹起的监听事件，处理的都是将鼠标事件的屏幕坐标映射成球面坐标。下面详细讲解坐标是如何映射的

（2）坐标的映射

这里说的坐标映射指的是，鼠标的二维屏幕坐标转换成三维球面的球面坐标，如图 5-7 所示。

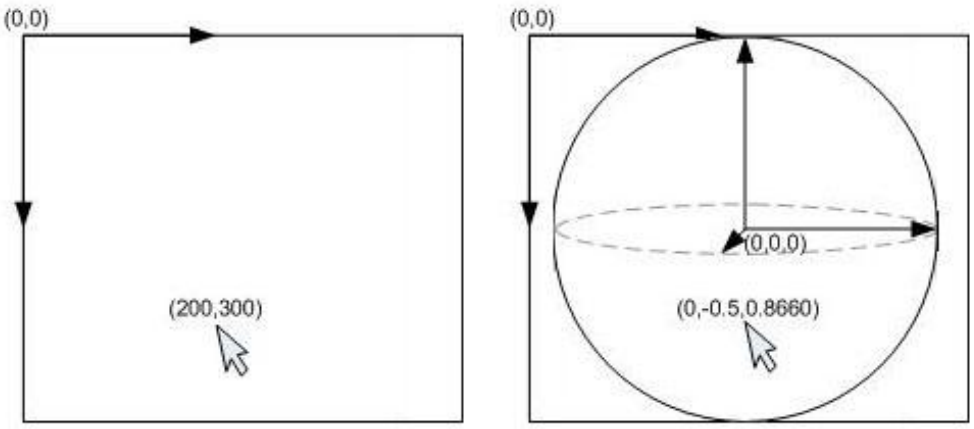


图 5-7　坐标映射指

一般平面坐标是一个矩形，因为屏幕一般为矩形。有图中的圆的半径我们令 _this.radius = (this.screen.width+this.screen.length)/4。需要映射的坐标就是相对于这个圆的坐标，准确说是以这个圆为中截面的球体。相对于球体的三维坐标的 X 坐标是 clentX-_this.Screen.width*0.5-this.screen.offsetleft。因为点击事件产生的横坐标是相对于图 5-5 左边的坐标系，要转换成有图的坐标系，需要减去屏幕宽度的一半，又由于渲染容器并不一定是整个屏幕，还需要减去容器相对于屏幕左边的距离。三维坐标的 Y 坐标类似，只不过 Y 轴沿向下的方向是递增的，所以应该做加法。Z 坐标始终为 0。为了计算方便，归一化将坐标各自除以球体半径，因此，映射第一步，完成球体坐标的映射如下：

```
var mouseOnBall = new THREE.Vector3(
( clientX - _this.screen.width * 0.5 - _this.screen.left ) / (_this.radius),
( _this.screen.height * 0.5 + _this.screen.top - clientY ) / (_this.radius),
```

```
0.0
);
```

从图中可以看出，这些坐标点有些落在球体外面，有些落在球体里面。为了将坐标映射到球体表面。对于落在球体外面的点，使用向量的归一化方法，将其归一到球面，mouseOnBall.normalize()。对于落在球面里面的点，通过改变其 Z 坐标，来将其归一到球面， mouseOnBall.z = Math.sqrt(1.0 - length * length)。

到这一步，完成了鼠标二维平面坐标到归一化的球面坐标的转换。但坐标映射仍未做完。球面坐标仍然只是一个中间变量。还需将相机坐标与该中间变量对应起来，最终得到相机坐标在该球面上的相对坐标。坐标映射的最终目的就是技术相机相对于旋转球面的相对坐标。

当前的相机是有状态的，它的状态由两个向量确定，分别是相机指向目标点的向量 eye、相机向上的向量 up。将相机的坐标与归一化球面求的的坐标对应起来，只需要求得归一化球面的相机本地坐标系坐标。所谓相机本地坐标系，即 eye 向量、up 向量及两向量叉乘所得的的向量，这三个向量构成的坐标系，计算过程如下。

```
var projection = _this.object.up.clone().setLength( mouseOnBall.y );
projection.add(this.object.up.clone().cross(eye ).setLength( mouseOnBall.x ) );
        projection.add( _eye.setLength( mouseOnBall.z ) );
        return projection;
```

第一句获得归一化球面坐标在相机本地坐标系 up 方向上的分量，第二句获得 up 与 eye 叉乘方向的分量，第三句获得 up 方向的分量。这样就能得到相机相对于球面的最终相对坐标，之所以说相对坐标，是因为这个坐标并不是相机真正的所在坐标。而是通过鼠标移动的前后两次相对坐标来计算旋转角度的。将上面的计算步骤封装在一个方法里 getMouseProjectionOnBall = function (clientX, clientY)。调用这个方法就能得到鼠标按下时和弹起是的两个相机球面相对坐标，我们定义为_rorateStart 和 _rorateEnd。

（3）旋转角的计算

在第二步已经完成了最主要的步骤，得到了与鼠标动作相对应的两个相机相对

于球面的相对坐标_rorateStart 和_rorateEnd。这是两个球面上的坐标，计算他们的夹角即是相机的旋转角度。

计算的原理是利用了向量的点积，两向量的点积等于两向量的模乘以两向量夹角的余弦，即 A Ƀ =|A| ¦B| cosө。因此 ө=arccos（A Ƀ/|A| ¦B|）。

因此夹角的计算，使用下面的语句：

```
var angle = Math.acos( _rotateStart.dot( _rotateEnd ) / _rotateStart.length() / _rotateEnd.length() );
```

其中，_rotateStart.dot(_rotateEnd)计算的移动前的向量和移动后的向量的点积。

（4）利用四元组旋转相机

在 3D 世界中，常见的旋转有三种工具表示，一是旋转矩阵，二是欧拉角，第三就是四元组。这三者之间可以相互转换，本系统使用四元组，因为 Three.js 已经封装好了四元组的运算，只需调用即可。而且四元组相对来说很好理解，它只需要一个旋转轴和绕着这个轴旋转的旋转角度。四元组在 Three.js 中，用 THREE.Quaternion 来表示。因为是本方案主要讨论网元渲染的原理，这里不打算讨论四元组的数学原理。

要使用四元组，只需要 quaternion = new THREE.Quaternion()。然后利用 quaternion 调用封装好的四元组方法 quaternion.setFromAxisAngle(axis, -angle)。

其中，第一个参数是围绕旋转的轴向量，第二个参数就是步骤三所计算出来的旋转的夹角。因此，还需要计算出轴向量来。由空间几何可知，旋转围绕的轴向量必然是旋转前后两个向量的叉乘。因此有 var axis = (new THREE.Vector3()).

crossVectors(_rotateStart, _rotateEnd).normalize()。crossVector 是计算叉乘，normalize()是将计算结果单位化。需要注意的是第二个参数 angle 是负的，这是因为鼠标移动的方向与模型旋转的方向是相反的。

有了四元组，就可以开始旋转相机了。相机上有个本地坐标系，也就是 up、eye 和两者叉乘 up.cross(eye)。其中 up 是相机沿球面切线向上的方向，eye 是相机指向目

标点的方向。这两个向量决定了相机的朝向与位置。在旋转时只需要旋转这两个向量即可。因此有_eye.applyQuaternion(quaternion)以及_this.object. up.apply Quaternion (quaternion)。

至此，已经完成了相机旋转的整个过程。事实上，在 Three.js 中，已经有对该动作脚本的封装，称为 TrackBallControls.js 轨迹球控制。该动作脚本也是通过上面所介绍的相机旋转的原理来完成 360 度视角观察模型的。除了提供的相机旋转功能，还包括对相机的其他控制，如平移等。使用 TrackBallControls.js 可以及其容易完成对场景的放大缩小及旋转功能。TrackBallControls 其构造函数如下：function (object, domElement)。第一个参数

一般传入相机变量，即被控制的对象。第二个参数 domeElement 表示要在哪一个 div 中监听，可以为空，为空时表示在整个 document 中监听鼠标事件。

在 TrackBallControls.js 时，只需像引用 Three.js 一样先引用该脚本。然后在通过 controls=new THREE.TrackBallControls(camera) 来创建同时把相机传进去。TrackBallControls 内置许多变量来控制相机的移动属性，如下：

```
controls.rorateSpeed=5.0; //表示鼠标左右旋转场景的速度
controls.zoomSpeed=5.0;//表示鼠标滚轮放大缩小的速度
controls.panSpeed=2.0;//表示摇晃镜头的速度，就是左右移动的速度
controls.noZoom=false;//表示场景是否允许放大
controls.noPan =false;//表示镜头是否允许摇动
controls.staticMoving=true;//是否是静态移动，就是移动是否有摇晃或者弹性；
若有摇晃，到了目标位置，相机会轻微摇摆
controls.dynamicDampingFactor=0.3//阻力系数
```

通过这些内置参数的设定，就可以很好的控制相机的移动来达到自己想要的效果了。最后还需要在循环渲染中添加 controls.update()这个函数。这个函数完成了相机属性的更新工作，它监听鼠标动作，当鼠标有所动作时，第一时间监听到并相应的变换相机。如图 5-8 所示，是旋转后的网元三维模型截图，可以很方便地从各个角度查看网元三维模型。

图 5-8　3D 网元最终效果

5.5　本章小结

本章介绍系统的详细设计与实现，重点关注两方面内容。一是如何根据 xml 配置文件里的网元实体静态信息和获取到的网元 mib 信息这两个网元实体信息源来构造一个代表网元实时状态的逻辑模型；二是如何根据构造的逻辑模型在前端页面渲染出一个仿真的三维模型，并添加用户互动功能。

6 总结与展望

6.1 全文总结

本文论述了网元三维模型展示系统的构建。本系统采用分层模块化设计，大体分成三层：表现层、业务逻辑层和数据访问层。数据访问层完成系统与实际网络设备间的数据通信功能；业务逻辑层主要完成网元静态信息的管理和动态构建网元逻辑模型；表现层完成三维网元模型的渲染。

本文侧重于讨论业务逻辑层与表现层，省略了数据访问层的介绍。对于业务逻辑层，从系统模块和系统流程这两个角度分析了整个系统的架构。在系统详细设计与实现部分，主要讨论了三大块内容。一是重点讨论了网元逻辑模型的构建，包括网元实体信息存储文件即 XML 配置文件、网元实体类的设计、构建逻辑模型的动态过程以及将逻辑模型转换成 Json 数据结构。二是重点讨论如何利用 Three.js 这一三维引擎库来完成三维模型的渲染，包括用 Js 解析 Json 的过程、渲染前的必要准备、主体机框的渲染、板卡和端口的渲染。三是讨论了模型与用户的交互功能的实现，主要是用户鼠标事件触发的模型旋转和模型缩放的功能。

本文不是一个软件工程文档，只是为网元面板系统提供一种三维的解决方案和思路。系统设计部分只是粗略介绍了系统的架构设计，没有详细讨论各个模块的技术细节以及 Web 服务器架设的通用技术等。

本文的主要成果有以下几点：

（1）网元实体静态信息的管理，主要通过设计 xml 配置文件来实现，该功能模块的实现，使得网元管理变得简单，每次新设备增加的时候，往往只需要在 xml 配置文件里添加即可获得网元管理系统的支持。

（2）通过 xml 配置文件取得的网元静态结构信息和从实际设备取得的 Mib 信息，动态构造一个代表了当前网元实际状态的逻辑模型。

（3）通过代表网元实际状态的 Json 格式数据，前端页面渲染出了一个仿真的网元三维模型。主要包括了：机框、板卡和带有状态的端口。

（4）完成了三维模型与用户简单的交互，用户可以通过拖拽鼠标来旋转设备，可以 360 度观察三维模型。

6.2 展望

本课题为网元三维面板系统提供了一种解决思路，旨在提高网元面板系统的用户体验。作为一种解决方案，本课题仍然存在许多不足的地方需要改进。

最主要的问题，就是三维模型的仿真度仍然做得不够，虽然较二维的图片面板已经有较大改进。提高模型仿真度，我认为应该从两方面入。一方面主要是网元 6 张纹理贴图需要改进，本系统设计时使用的是最基本的纹理贴图，其效果完全等价于将 6 张制作好的纹理图片贴在立方体的六面。其实除了最基本的纹理贴图，还有许多其他仿真度更高的纹理，比如考虑感光、漫反射、金属质感、颗粒等纹理效果。另一方面是板卡和端口的渲染问题，虽然板卡和端口是视觉上的二维网元实体，但并不真的是二维的，仍然是三维立体的，大致上也能看成是立方体，只是插进了机框的槽位。观察实际设备可以发现，其实板卡和端口并不是完全契合进机框面板的，就是说在机框面板表面板卡和端口部分仍然有部分是突出的。因此，在后续的改进版本中，应该考虑板卡和端口的三维效果。

第二个需要改进的是，与用户的交互功能仍然做的不够。当前方案的设计只是为系统添加了鼠标事件触发模型旋转和缩放的功能。后续的改进版本中，应该考虑添加更加丰富的互动效果。如对于板卡和端口，不再是贴在面板上不能动的二维图片。它们应该是三维立体的，并且用户通过鼠标事件可以将可插拔的板卡与端口从槽位上拔下来或者插进去。

第三点是模型的界面显得单薄。当前的方案只是在页面渲染出了网元三维模型并添加了交互功能。在界面上一个孤零零的模型显得单薄。其实，还有许多其他的功能没有实现。如在需求中提到的，网元设备信息和端口信息的管理、性能的监控等，这些功能需求都没有在本系统的设计中出现。这些信息的管理和监控也都是通过监控网元 Mib 库来完成的，可以取出 Mib 信息展示在本页面中，来丰富页面元素。

致 谢

在杭州华三通信技术有限公司实习的一年里，专业技能以及职场沟通能力方面都得到了极大的提高。在实习近一年的时间里，能取得如此大的进步并顺利完成了论文课题的研究工作，与学习老师同学和企业同事的帮助是分不开的。随着论文的完成，意味着我在华中科技大学软件学院工程硕士阶段的学习也将画上一个句号。在此，向那些给予我帮助老师和同学表示感谢。

首先，要感谢我的导师黄立群老师。黄老师是一位学识渊博又平易近人的好导师，黄老师的好脾气拉近了师生间的距离。在研究生学习期间，我不断得到他在科研和学习的严格指导、无微不至的关怀和孜孜不倦的教诲。他渊博的学识、严谨的治学作风、一丝不苟的学术态度、平易的待人之道和勤于钻研的科研作风，都使我受益匪浅，感受颇深。

还要感谢实习期间项目组同事，很庆幸自己可以与这样一个开发团队共事。项目经理李方宁是一个和蔼可亲的老大，给了我很多专业方向的建议，让我对未来规划更加清晰明确。我的企业导师刘浩，是个作风严谨、工作负责的项目负责人，从他身上我学到了领导一个项目团队的那种责任感。还要感谢所有项目组的同事，他们都很随和，对于我这个初入职场的新人很照顾，让我很快适应新环境、快速上手新项目。

最后，感谢我的父母，无论顺境逆境永远无理由地支持我、关心我，他们的支持让我敢于面对一切困难与挑战。

参考文献

[1] 陶洋, 胡敏. 网络管理原理与实践, 北京: 科学出版社, 2000: 211-214

[2] 张乾. 基于 B-S 的网元管理系统的研究与实现: [硕士学位论文]: 天津: 天津大学图书馆, 2007

[3] 谭文文, 丁世勇, 李桂英. 基于 WebGL 和 HTML5 的网页 3D 动画的设计与实现: [硕士学位论文]. 天津: 天津大学图书馆, 2011

[4] 殷周平, 吴勇. 基于 WebGL 和 AJAX 的 Web3D 应用研究—以在线 3D 协作交互式设计为例: [硕士学位论文]. 天津: 天津大学图书馆, 2013

[5] Tony Parisi. WebGL 入门指南. 北京: 人民邮电出版社, 2013: 5-8

[6] W. Richard Stevens. TCP/IP 详解卷一. 协议. 北京: 机械工业出版社, 2012: 120-123

[7] 王焕然, 徐明伟. SNMP 网络管理综述. 小型微型计算机系统, 2004, 25(3): 68-70

[8] K. McCloghrie and A. Bierman. "Entity MIB using SMIv2", RFC2037, October 1996

[9] K. McCloghrie, M. Rose. Management Information Base for network management of TCP/IP-based internets: MIB-II, RFC1213[S], 1991

[10] Sean Harnedy 著. SNMP 网络管理. 胡谷雨译. 北京: 中国电力出版社, 2001

[11] Blumenthal U, Wijnen B. "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD: 62, RFC3414, December 2002

[12] 夏海涛, 詹志强. 新一代网络管理技术: 北京: 北京邮电大学出版社, 2003

[13] Stallings W. SNMP and SNMPv2: the infrastructure for network Management: IEEE Communications Magazine, March 1998, 36(3): 37-43

[14] Amirthalingam, K. Moorhead RJ SNMP-an overview of its merits and Proceedings of the Twenty-Seventh Southeastern Symposium on System Theory [J]: IEEE Compute Soc Press, 1995: 180-183

[15] 应伟锋, 段小东, 沈金龙. SNMPv1、SNMPv2 和 SNMPv3 的安全性协议分析与比较. 计算机工程, 2002(10): 16-18

[16] Blumenthal U, Wijnen B. Security features of SNMPv3: The Simple Times, December 1997, 5(1): 8-13

[17] J. Case, D. Harrington. Message Processing and Dispatching for the Simple Network Management Protocol (SNMP), RFC2272, 1998

[18] Zeltserman, D. A Practical Guide to SNMPv3 and Network Management: Prentice Hall, 2000

[19] K. de Graaf, D. McMaster and K. McCloghrie. "Definitions of Managed Objects for IEEE802. 3 Repeater Devices using SMIv2", RFC2108, February 1997

[20] 孟洛明. 现代网络管理技术. 北京: 北京邮电大学出版社, 1999

[21] A. Bierman, K. McCloghrie. Entity MIB (Version 3), RFC4133, 2005

[22] Erik M. BUCK. Learning OpenGL ES for IOS. Crawfordsvill, Indiana: R. R Donnelley Press, 2013, 7

[23] Andreas Anyuru. WebGL 高级编程 开发 Web 3D 图形. 北京: 清华大学出版社, 2013

[24] Feng Yuan. Windows 图形编程. 北京: 机械工业出版社. 美国: 培生教育出版集团, 2002

[25] 杨柏林. OpenGL 编程精粹. 北京: 机械工业出版社, 2010

[26] 和克智. OpenGL 编程技术详解. 北京: 化学工业出版社, 2010

[27] Dave Shreiner, Graham Sellers, John Kessenich et al. OpenGL Programming Guide. Addison-Wesley Professional, 2009

[28] 张雯丽. Three. js 指南. 北京: 图灵书社, 2013, 10

[29] 徐文鹏, 徐跃通. 基于 WebGL 纹理映射技术的水立方贴图的设计与实现. 电脑知识与技术, 2013(16): 139-143

[30] 胡振中, 张建平. 面向 Web 的 BIM 三维浏览与信息管理. 土木建筑工程信息技术, 2013(3): 219-222

[31] Yuhui Yang, Jianping Zhang. Web3D technology and its application in the cultural relic protection, ICCSIT2009, 2009

[32] 韩毅. Web3D 及 Web 三维可视化新发展—以 WebGL 和 03D 为例. 科技广场, 2010(5): 30-33

[33] Ping Sun, ChangJun Jiang, and MengChu Zhou. Interactive Web service composition based on Petri net, 2011, 33: 116-132

[34] 韩毅. Web3D 及 Web 三维可视化新发展—以 WebGL 和 03D 为例. 科技广场, 2010(5): 30-33

[35] 南楠. 基于 Web-3D 的 VR 虚拟社区的交互研究与设计: [硕士学位论文]. 成都: 西南交通大学图书馆, 2011

[36] 杨文清, 懂飞. 基于 VRML 的 Web 3D 环境交互方法. 计算机系统应用, 2010(19): 45-48

[37] 罗成. 一种设备面板的生成方法和装置. 中国专利: 200910131571. 8, 2009: 8-19

[38] 李天剑, 曾文方. 基于 java 和 SNMP 的网络管理平台的研究与实现. 微电子学与计算机, 2000(1): 128-130

[39] 孙利辉, 于红, 王利彬等. 网络设备面板管理系统: 中国专利: 03157064. X, 2005: 3-16

[40] 何春叶. 基于 XML 的网元管理系统研究与实现:[硕士学位论文]. 上海: 上海交通大学图书馆, 2006

[41] 张先锋, 邱劲松, 金连甫. 基于 SNMP 代理技术的网络设备仿真. 计算机应用, Apr 2002, 22(4)

[42] 崔晓乾. 基于 SNMP 的网管系统的设计与实现: [硕士学位论文]. 成都: 电子科技大学图书馆, 2005

[43] 庞云光. 一种生成仿真设备面板的方法及系统. 中国专利: 200610057565. 9, 2006: 3-15

[44] 张雯丽. Three. js 指南. 北京: 图灵书社, 2013: 50-60

[45] Donald Hearn, M. Pauline Baker. 计算机图形学, 北京: 人民邮电出版社, 2005: 145-147