# 3022207128-杨宇鑫-实验报告4-1

## 1. 引言

绘制WebKit 模块关系依赖图  
使用d3力导向绘制WebKit 模块依赖图。节点为WebKit的模块，链接为模块间的依赖关系。数据：webkit-dep.json。  
要求：节点根据category染色。鼠标移到节点上高亮节点并显示出其名字。

## 2. 代码结构

代码主要分为以下几个部分：

* 引入 D3.js 库
* 创建 SVG 元素
* 定义缩放行为
* 加载数据并创建力导向图
* 处理窗口大小变化

## 3. 核心代码分析

### 3.1 引入 D3.js

// 引入 D3.js  
const width = window.innerWidth; // 获取浏览器窗口的宽度  
const height = window.innerHeight; // 获取浏览器窗口的高度

在这部分代码中，首先获取浏览器窗口的宽度和高度，以便为 SVG 元素设置合适的尺寸。

### 3.2 创建 SVG 元素

// 创建 SVG 元素  
const svg = d3.select("body").append("svg")  
 .attr("width", width)  
 .attr("height", height);

使用 D3.js 创建一个 SVG 元素，并将其添加到 HTML 的 body 中。SVG 元素的宽度和高度设置为浏览器窗口的宽度和高度。

### 3.3 定义缩放行为

// 创建缩放行为  
const zoom = d3.zoom()  
 .scaleExtent([0.1, 10]) // 设置缩放范围  
 .on("zoom", (event) => {  
 g.attr("transform", event.transform); // 应用缩放变换  
 });  
  
// 将缩放行为应用到 SVG  
svg.call(zoom);

这里定义了一个缩放行为，允许用户在 0.1 到 10 的范围内缩放图形。通过 on("zoom", ...) 方法，设置了缩放时的变换效果，将变换应用到组元素 g 上。

### 3.4 加载数据并创建力导向图

// 加载数据  
d3.json("webkit-dep.json").then(data => {  
 const color = d3.scaleOrdinal(d3.schemeCategory10);  
  
 // 创建力导向图  
 const simulation = d3.forceSimulation(data.nodes)  
 .force("link", d3.forceLink().id((d, i) => i).distance(50))  
 .force("charge", d3.forceManyBody().strength(-100))  
 .force("center", d3.forceCenter(width / 2, height / 2));

通过 d3.json 方法加载 JSON 格式的数据，数据包含节点和链接信息。使用 d3.forceSimulation 创建一个力导向图的模拟，设置了链接、排斥力和中心力。

### 3.5 绘制链接和节点

// 绘制链接  
 const link = g.append("g")  
 .attr("class", "links")  
 .selectAll("line")  
 .data(data.links)  
 .enter().append("line")  
 .attr("stroke-width", 1)  
 .attr("stroke", "#999");  
  
 // 绘制节点  
 const node = g.append("g")  
 .attr("class", "nodes")  
 .selectAll("circle")  
 .data(data.nodes)  
 .enter().append("circle")  
 .attr("r", 6)  
 .attr("fill", d => color(d.category))  
 .on("mouseover", function(event, d) {  
 d3.select(this).attr("r", 12); // 高亮节点  
 svg.append("text")  
 .attr("id", "tooltip")  
 .attr("x", event.pageX)  
 .attr("y", event.pageY)  
 .text(d.name);  
 })  
 .on("mouseout", function() {  
 d3.select(this).attr("r", 6); // 恢复节点大小  
 d3.select("#tooltip").remove(); // 移除提示  
 });

在这部分代码中，首先绘制了链接（线条），然后绘制了节点（圆形）。节点的颜色根据其类别进行设置，并添加了鼠标悬停事件，显示节点名称的提示信息。

### 3.6 更新节点和链接的位置

// 更新节点和链接的位置  
 simulation  
 .nodes(data.nodes)  
 .on("tick", () => {  
 link.attr("x1", d => d.source.x)  
 .attr("y1", d => d.source.y)  
 .attr("x2", d => d.target.x)  
 .attr("y2", d => d.target.y);  
  
 node.attr("cx", d => d.x)  
 .attr("cy", d => d.y);  
 });  
  
 simulation.force("link").links(data.links);

在每次模拟的“tick”事件中，更新链接和节点的位置，以确保它们根据力导向算法的计算结果进行动态调整。

### 3.7 设置初始缩放和中心位置

// 设置初始缩放和中心位置  
 const initialScale = 0.3; // 初始缩放比例  
 const initialTranslateX = width / 3; // 初始平移X  
 const initialTranslateY = height / 3; // 初始平移Y  
 svg.call(zoom.transform, d3.zoomIdentity.translate(initialTranslateX, initialTranslateY).scale(initialScale));  
});

设置初始的缩放比例和位置，使得图形在加载时能够以合适的视图展示。

### 3.8 监听窗口大小变化

// 监听窗口大小变化  
window.addEventListener('resize', () => {  
 const newWidth = window.innerWidth;  
 const newHeight = window.innerHeight;  
 svg.attr("width", newWidth).attr("height", newHeight);  
 simulation.force("center", d3.forceCenter(newWidth / 2, newHeight / 2)).alpha(1).restart();  
});

通过监听窗口的大小变化，动态调整 SVG 的宽度和高度，并重新计算力导向图的中心位置，确保图形在不同窗口尺寸下的适应性。

1. 成果展示



