## [Echarts通过Ajax实现动态数据加载](http://www.cnblogs.com/zhaoyingjie/p/5963056.html)

Echarts（3.x版）官网实例的数据都是静态的，实际使用中往往会要求从服务器端取数据进行动态显示，官网教程里给出的异步数据加载很粗略，下面就以官网最简单的实例为例子，详细演示如下过程：1.客户端通过ajax发送请求；2.服务器端Servlet接收请求；3.生成json数据并返回给客户端；4.客户端接收数据后显示。

**1.客户端通过ajax发送请求**

先绘制一个最简单的Echarts图表：

（这里就直接贴上代码了，直接用的是官网教程里异步数据加载和更新里的代码）

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html lang="zh-CN">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>ECharts</title>

<!-- 引入 echarts.js -->

<script type="text/javascript" src="echarts.min.js"></script>

<!-- 引入jquery.js -->

<script type="text/javascript" src="jquery-1.12.3.js"></script>

</head>

<body>

<!-- 为ECharts准备一个具备大小（宽高）的Dom -->

<div id="main" style="width: 600px;height:400px;"></div>

<script type="text/javascript">

var myChart = echarts.init(document.getElementById('main'));

// 显示标题，图例和空的坐标轴

myChart.setOption({

title: {

text: '异步数据加载示例'

},

tooltip: {},

legend: {

data:['销量']

},

xAxis: {

data: []

},

yAxis: {},

series: [{

name: '销量',

type: 'bar',

data: []

}]

});  
  
  
　　</script>

</body>

</html>

看到了吧，现在option中的xAxis和yAxis里的data都是空值。待会儿我们会用从服务器取回的数据去”填满“它。

下面贴上补充了ajax部分的完整前端代码：

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html lang="zh-CN">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>ECharts</title>

<!-- 引入 echarts.js -->

<script type="text/javascript" src="echarts.min.js"></script>

<!-- 引入jquery.js -->

<script type="text/javascript" src="jquery-1.12.3.js"></script>

</head>

<body>

<!-- 为ECharts准备一个具备大小（宽高）的Dom -->

<div id="main" style="width: 600px;height:400px;"></div>

<script type="text/javascript">

var myChart = echarts.init(document.getElementById('main'));

// 显示标题，图例和空的坐标轴

myChart.setOption({

title: {

text: '异步数据加载示例'

},

tooltip: {},

legend: {

data:['销量']

},

xAxis: {

data: []

},

yAxis: {},

series: [{

name: '销量',

type: 'bar',

data: []

}]

});

myChart.showLoading(); //数据加载完之前先显示一段简单的loading动画

var names=[]; //类别数组（实际用来盛放X轴坐标值）

var nums=[]; //销量数组（实际用来盛放Y坐标值）

$.ajax({

type : "post",

async : true, //异步请求（同步请求将会锁住浏览器，用户其他操作必须等待请求完成才可以执行）

url : "TestServlet", //请求发送到TestServlet处

data : {},

dataType : "json", //返回数据形式为json

success : function(result) {

//请求成功时执行该函数内容，result即为服务器返回的json对象

if (result) {

for(var i=0;i<result.length;i++){

names.push(result[i].name); //挨个取出类别并填入类别数组

}

for(var i=0;i<result.length;i++){

nums.push(result[i].num); //挨个取出销量并填入销量数组

}

myChart.hideLoading(); //隐藏加载动画

myChart.setOption({ //加载数据图表

xAxis: {

data: names

},

series: [{

// 根据名字对应到相应的系列

name: '销量',

data: nums

}]

});

}

},

error : function(errorMsg) {

//请求失败时执行该函数

alert("图表请求数据失败!");

myChart.hideLoading();

}

})

</script>

</body>

</html>

**2.服务器端Servlet接收请求**

客户端的请求url是’TestServlet‘，那我们得先在web.xml配置以下映射：

<servlet>

<servlet-name>TestServlet</servlet-name>

<servlet-class>test.TestServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>TestServlet</servlet-name>

<url-pattern>/TestServlet</url-pattern>

</servlet-mapping>

然后就来着手写处理客户端请求的TestServlet啦！

**3.生成json数据并返回给客户端**

生成Json数据要用到额外的jar包，这里我使用的jackson，json-lib 2010年就没有再更新了… （2016-5-3日更：发现Google出品的Gson也蛮好用的，重点是很小巧，只需引入一个jar就搞定，用法也很简单）

简单介绍一下jackson的用法：

①：先去<http://mvnrepository.com/>下载 jackson-core.jar，jackson-annotations.jar，jackson-databind.jar（都是2.x版本，jackson官网不知为何不提供服务了…这三个jar让我一阵好找）

②：在项目的项目WEB-INF/lib下引入这三个jar

然后就可以在TestServlet里使用jackson提供的工具类了。（关于jackson的详细用法，这里贴一下官网教程：<http://wiki.fasterxml.com/JacksonInFiveMinutes>）

TestServlet代码如下：

package test;

import java.io.IOException;

import java.util.\*;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.fasterxml.jackson.databind.ObjectMapper;

public class TestServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

doPost(req,resp);

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

List<Product> list = new ArrayList<Product>();

//这里把“类别名称”和“销量”作为两个属性封装在一个Product类里，每个Product类的对象都可以看作是一个类别（X轴坐标值）与销量（Y轴坐标值）的集合

list.add(new Product("衬衣", 10));

list.add(new Product("短袖", 20));

list.add(new Product("大衣", 30));

ObjectMapper mapper = new ObjectMapper(); //提供java-json相互转换功能的类

String json = mapper.writeValueAsString(list); //将list中的对象转换为Json格式的数组

//System.out.println(json);

//将json数据返回给客户端

response.setContentType("text/html; charset=utf-8");

response.getWriter().write(json);

}

TestServlet类中用到的自定义的Product类代码如下：

package test;

public class Product {

private String name; //类别名称

private int num; //销量

public Product(String name, int num) {

this.name = name;

this.num = num;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getNum() {

return num;

}

public void setNum(int num) {

this.num = num;

}

}

**4.客户端接收数据后显示**

客户端接受服务器数据并解析后，就可以正常显示图表数据了：