*JavaWeb*

目录

[一、 Web简介 2](#_Toc522627327)

[二、 Tomcat 3](#_Toc522627328)

[1. 安装 3](#_Toc522627329)

[2. Tomcat目录结构 3](#_Toc522627330)

[3. 端口 4](#_Toc522627331)

[三、 JavaWeb项目发布 5](#_Toc522627332)

[四、 其它 6](#_Toc522627333)

[五、 Maven 7](#_Toc522627334)

[1. 基本设置 7](#_Toc522627335)

[2. Import 7](#_Toc522627336)

[六、 JS 8](#_Toc522627337)

[1. JSON字符串 8](#_Toc522627338)

[2. 字符串转数字 9](#_Toc522627339)

[3. 判断非空 9](#_Toc522627340)

[4. 变量对象 9](#_Toc522627341)

[5. 页面混淆 10](#_Toc522627342)

[6. setTimeout()函数 10](#_Toc522627343)

[7. css设置 10](#_Toc522627344)

[8. js全局变量 10](#_Toc522627345)

[9. Value/Text 10](#_Toc522627346)

[七、 事件 11](#_Toc522627347)

[1. Focus、blur事件 11](#_Toc522627348)

[2. Document当前页面 11](#_Toc522627349)

[八、 CSS 13](#_Toc522627350)

[1. CSS文件格式 13](#_Toc522627351)

[2. 优先级 13](#_Toc522627352)

[九、 DataTables 14](#_Toc522627353)

[1. 获取data数据 14](#_Toc522627354)

[十、 TPSM/RMDS项目框架 15](#_Toc522627355)

# Web简介

Github网站（源码）

前端、后端 和 静态、动态网页

前端：浏览器、手机客户端、ATM

后端：Java服务器、Php服务器、Asp.net服务器

以Java服务器为例，使用任意一种前端工具（浏览器）访问Java程序，并且通过Java程序获取并处理数据，返回给前端，这一整套操作，都需要Java web应用服务器的支持。

静态 - 网页中的数据内容是固定的

动态 - 网页中的数据内容是会随着请求的不同发生变化的

前端(HCJ) - 后端(Java) - 数据库端(Oracle)

请求 - 响应

# Tomcat

## 安装

—添加环境变量：

变量名：CATALINA\_HOME

路径：D:\Program Files\Java\javaee\apache-tomcat-7.0.72

—运行路径（%CATALINA\_HOME%）bin下的startup.bat（运行的CMD不能关）(使用shutdown.bat关闭服务器)

—在浏览器中输入（http://localhost:8080/）【显示Tomcat网页则为配置正常】

Tomcat是一种当前比较流行的web应用服务器，属于开源的、轻量级服务器，在中小型系统和并发量不是很大的情况下使用，是调试Servlet&JSP首选的服务器。

（Java-Tomcat、JBoss；PHP-Apache；ASP-Win Server）

Tomcat-Servlet-JSP

Servlet本身是一种符合特定规则的Java类，一般是HttpServlet的子类，通过对Servlet的控制能实现前后台页面的交互操作（动态地去修改前台页面）。JSP是基于Servlet推出的一种能直接在类似HTML页面中使用Java代码进行混合开发的形式。实际上，JSP仍然是一个Servlet，而它们的运行、交互、维护都是通过Tomcat来完成的。

Tomcat本身是一系列Java程序的集合。包含了能对web应用系统的控制和转发、对网络层协议的解析、端口的监听等。同时它还可以帮助开发者去维护站点，如log功能，定时操作等。

Tomcat容器

Tomcat：Container->Engine->Host->Servlet（Context：Wrapper \* n）

在Eclipse配置Tomcat服务器

<http://jingyan.baidu.com/article/ca2d939dd90183eb6d31ce79.html>

<http://jingyan.baidu.com/article/3065b3b6efa9d7becff8a4c6.html>

## Tomcat目录结构

Bin——二进制目录，主要存放可执行文件

Conf——配置文件

Lib——库目录，包含Tomcat运行所必须的库。

Logs——运行时存放的一些log信息

Temp——运行时生成的一些临时文件

Webapps——网站/项目最终存放的路径（不是绝对的），以具体路径形式访问具体资源

Work——Tomcat将JSP转为Servlet的地方

## 端口

在conf文件下server.Xml修改端口号（默认是“8080”）

（...\apache-tomcat-7.0.72\conf\server.xml）

【Note http://默认端口是“80”；https://默认端口是“443”；】

<Connector

connectionTimeout="20000"

port="8080"

protocol="HTTP/1.1" redirectPort="8443"/>

端口

端口(port)在计算机中的定义：主要是描述设备和外界通讯的接口。分为物理端口和虚拟端口，物理端口主要指物理设备上的接口（网线RJ45），虚拟端口指计算机内部或之间连接的“桥梁”。虚拟端口一般指的是TCP/IP协议所指定的一些接口，如所有的电脑是办公楼，每一个业务是在楼里的办公室完成的，开门的可以处理业务，关门的不行。而IP指的就是办公楼的具体地址，端口指的就是每个办公室的具体门牌号。端口的最大数是65535个（IPv4中定义端口是双字节的，16位）。

监听指的就是办公室开门，针对这个办公室进行一个等待操作，类似于“传达室”，在数据包到达具体IP以后，首先由传达室进行检测（监听），如果监听正常（开启），数据包就会发送到具体的某个端口去。

NOTE: 传达室是一个传达室，而监听是每个端口都有自己的监听。

HTTP - 80, HTTPS – 443

Web程序目录

(Webapps)

/Java源代码

/其它网页、资源等

/Web-INF——web信息目录（web.xml 储存当前项目的配置信息(默认的打开页面设置等)；lib/当前web项目所引用的库；classes Java源代码编译后的字节码）

【Web-INF是一个私有目录(安全目录)，只有服务器端可以访问到，客户端不行】

3.Get和Post方法

3.1 HTTP的定义

# JavaWeb项目发布

（参考：http://blog.csdn.net/longyuhome/article/details/8362635/）

在eclipse里面的项目伤右键，然后Export，然后在弹出的框当中选择导出类型，这里选择web下面的WAR file，然后下一步，选择导出到哪里，然后把导出的war文件放到tomcat的webapp下面。启动tomcat就可以了。tomcat会自动把这个war文件解压为一个工程。

# 其它

Eclipse Servers的执行路径默认在\*\*\workspace\.metadata\.plugins\org.eclipse.wst.server.core

（也可放在tomcat系统下\*\*\apache-tomcat-7.0.72\work\Catalina\localhost）

2〉调试

浏览器-〉F12（控制台中可以看到输出也可以输入变量调试、查看源码）

输出到浏览器控制台console.log(“”);

警告框alert("jii");

Ctrl+Shift+R（浏览器刷新、eclipse查类）

# Maven

## 基本设置

Maven项目的配置文件（\*.xml）中设置database.properties的运行版本

<configurator:settings runningMode="release"/>

## Import

〉〉Maven更新失败（使用的IDEA）

〉〉将路径（~\m2\repository）下“\*lastupdate\*”文件全部删除；在能联外网的情况下再ReImport

（手动下载好了jar包，IDEA中还会显示错误提示，需要在能联外网的情况下再ReImport）

# JS

## JSON字符串

**Js对象转JSON字符串**

**var** temp ={  
 tempP:$.toJSON(param),  
 tempCharge:$.toJSON(paramMap(tempChargeListMap)),  
 tempContact:$.toJSON(paramMap(tempContactListMap)),  
 tempMember:$.toJSON(paramMap(tempMemberListMap))  
};//参数以JSON的形式传到后端，再通过JSON.parseObject(param, Object.class)转为对象、数组等；

JSON字符串转Js对象

使用js函数eval();

1. **function** ConvertToJsonForJs() {
2. //var testJson = "{ name: '小强', age: 16 }";(支持)
3. //var testJson = "{ 'name': '小强', 'age': 16 }";（支持）
4. **var** testJson = '{ "name": "小强", "age": 16 }';
5. //testJson=eval(testJson);//错误的转换方式
6. testJson = eval("(" + testJson + ")");
7. alert(testJson.name);
8. }

jquery.parseJSON()

1. function ConvertToJsonForJq() {
2. var testJson = '{ "name": "小强", "age": 16 }';
3. //不知道
4. //'{ name: "小强", age: 16 }' (name 没有使用双引号包裹)
5. //"{ 'name': "小强", 'age': 16 }"(name使用单引号)
6. testJson = $.parseJSON(testJson);
7. alert(testJson.name);
8. }

## 字符串转数字

parseInt(s);parseFloat(s); Number(s);

1. 当字符串中是纯数字

例如：var s = '234';

　　　/\*\*\*\*\*\*\*\*\*\* 字符串在运算操作中会被当做数字类型来处理 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

　　　/\*\*\*\*\*\*\* string的两个转换函数，只对string有效 \*\*\*\*\*\*\*\*/

　　　 parseInt(s); // 234

　　　 parseFloat(s); //234

　　　/\*\*\*\*\*\*\*\*\*\*\*\* 强制类型转换 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

　　　 Number(s); // 234

1. 当字符串是数字加字母等非数字

例如： var s = '234string';

　　　parseInt(s);　//234

　　　parseFloat(s);  //234.0

## 判断非空

var temp;

temp = ”、0、null、undefined;

(!temp);--当temp为””、0、null、undefined时为true。（可在浏览器控制台调试）

（注意：temp=0和temp=”0”不一样）

temp="123";->(!temp) 为false ->(!!temp) 为true;

## 变量对象

var temp = object1;//对象1

temp = object2;//对象2

（注意：会引起混淆，需要对temp再分配空间）

## 页面混淆

（限定范围：最好不要和$(“#id”)一同使用）

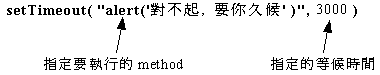
当不同JSP在同一个页面显示时；如果不同页面的标签id、name等属性相同时，在引用该标签会发生不希望看到的页面效果—〉可在引用该标签时限定范围

**var** root\_list = $('#TheProPack-Edit-'+id);//JSP根节点

$(＂#\*\*id＂，root\_list).\*\*;//\*\*id为标签的id

## setTimeout()函数

延迟执行方法：setTimeout( method/function , 时间)



## css设置

tempArr = $.find("input[type='text']");

tempArr.forEach(function(val,index){

$(val).css("background","rgba(0,0,0,0)");

console.log("--" + val + "--" + index);

})

## js全局变量

（对于多个标签页面在同一个页面中）

1〉使用window—例window.error

2〉在主页jsp中设置全局变量

3〉

## Value/Text

Button标签：$(‘…’).text();

Input标签：$(‘…’).val();

## jquery获取父级元素和子级元素

（<http://tianxingzhe.blog.51cto.com/3390077/1731667>）

**摘要：**       function show(x){     var y=$(x);//转换成jquery对象     //查找父元素    alert( y.parent().attr("name"));     alert( y.parent().attr("id"));     //查找祖先，是个集合 不包括自身

    <script type="text/javascript">

 function show(x){

    var y=$(x);//转换成jquery对象

    //查找父元素

   alert( y.parent().attr("name"));

    alert( y.parent().attr("id"));

    //查找祖先，是个集合 不包括自身

    alert(y.parents("tr").attr("name"))

    alert(y.parents("table").attr("name"))

    //closest()这个方法呢就是向上检查元素并逐级匹配。首先，会从自身来匹配，匹配成功就返回本身

    //查找第几个子元素，eq() 方法会用集合中的一个元素构造一个新的 jQuery 对象。所使用的 index 参数标示集合中元素的位置。

    alert(y.parent().children().eq(0).text());

    alert(y.parent().children().eq(1).text());

    alert(y.parent().children().eq(2).text());

    alert(y.parent().children().first().text());

    //children方法获得的仅仅是元素一下级的子元素

    //find方法获得所有下级元素

}

    </script>

</head>

<body>

<table name="表">

    <tr id="1"name="行1">

        <td>1</td>

        <td>袁海</td>

        <td onclick="show(this)">29</td>

    </tr>

    <tr id="2" name="行2">

        <td>2</td>

        <td>yuanhai</td>

        <td onclick="show(this)">29</td>

    </tr>

</table>

</body>

获取父级元素 parent parents() closest()

获取子级元素 children() find()

**jQuery.siblings(),返回兄弟姐妹节点，不分前后**

# 事件

## Focus、blur事件

$(window).focus(**function** () {  
 **if**(!document.hasFocus())//当前页面是否有焦点

{ }

**else** {  
 console.dir("-$(window).focus-----");  
 clearInterval(theTime);  
 theTime= setInterval(theTt,5000);  
 }  
});  
$(window).blur(**function** () {  
 console.dir("mmmm--");  
 clearInterval(theTime);  
});

## Document当前页面

**var** hiddenProperty = 'hidden' **in** document ? 'hidden' :  
 'webkitHidden' **in** document ? 'webkitHidden' :  
 'mozHidden' **in** document ? 'mozHidden' : **null**;  
**var** visibilityChangeEvent = hiddenProperty.replace(/hidden/i, 'visibilitychange');  
**var** onVisibilityChange = **function**(){  
 **if** (!document[hiddenProperty]) {  
 console.log('当前页面');  
 }**else**{  
 console.log('非当前页面');  
 }  
};  
document.addEventListener(visibilityChangeEvent, onVisibilityChange);//绑定事件

# CSS

## CSS文件格式

.class,

#id,

Select:focus{

---------

---------

}

可以将其写在JSP、html中标签的style中；也可在浏览器的调试中设置格式查看效果

## 优先级

对某个属性的后面加”!important”；例如：

.mySelect:focus,  
.portlet select.mySelect:focus {  
 background-color: rgba(15, 58, 102, 1)**!important**;  
 outline: none;  
}

# DataTables

（https://datatables.net/）

## 获取data数据

**var** data = $("#TheDicjsp").DataTable()//table的id  
 .rows()//cells( ".info" )  
 .data();  
 console.dir("------------ " );  
 console.dir(data);

console.dir(gridtable.getDataTable().api().rows().data());//gridtable为TableAdvanced()实例

# ECharts

图表绘制（API、文档说明）

（https://www.w3cschool.cn/echarts\_tutorial/）

（http://echarts.baidu.com）

（个性化图形扩展功能[未研究过]）

# TPSM/RMDS项目框架

主配置文件：rmds-web.Xml,其中的<configurator:settings runningMode="local"/>对应于properties文件夹下local文件夹（其中database.properties文件对应于项目所连接的数据库）

resources文件夹下spring-shiro.xml文件中设置不需要登入直接打开的文件（例：/theEnergy=anon）

# SpringBoot+IDEA+Maven快速入门

https://blog.csdn.net/qq\_28702545/article/details/53105232

## 首先创建一个Maven项目，比较简单

## 在pom文件中添加依赖

如下：

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.4.0.RELEASE</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>Springboot</artifactId>

<packaging>war</packaging>

<name>Springboot Maven Webapp</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

## 开始写Java代码

1、首先创建一个Application类，里面添加一个main()方法，

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

需要注意的是：这个类的上面需要添加SpringBoot的注解@SpringBootApplication，然后在主函数中开始启动。

2、然后创建一个HelloWord类，用来测试SpringBoot

如下：

@RestController

@EnableAutoConfiguration

public class HelloWord {

@RequestMapping("/hello")

public String hello() {

return "hello,Spring boot!";

}

//带参数

@RequestMapping("/word/{name}")

public String word(@PathVariable String name) {

return "word--spring boot:" + name;

}

}

需要注意的是这上面的几个注解：

* @RestController：表示这是个控制器，和Controller类似
* @EnableAutoConfiguration：springboot没有xml配置文件因为这个注解帮助我们干了这些事情，有了这个注解springboot启动的时候回自动猜测你的配置文件从而部署你的web服务器
* @RequestMapping(“/hello”)：这个和SpringMvc中的类似了。
* @PathVariable：参数

Ok，就这样，一个简单的SpringBoot小demo就写出来了，我们直接就可以运行了，然后输入：<http://localhost:8080/hello> 和 <http://localhost:8080/word/canshu> 浏览器上即可得到输出结果