## JAVA

1. String.format() 格式化字符串。
2. getBytes() 是java编程语言中将一个字符串转化为一个字节数组byte[]的方法。String的getBytes()方法是得到一个系统默认的编码格式的字节数组。可通过传入参数进行改变编码格式。

## 数据库

1. 查询名字中第二个字符为a，并且倒数第5个字符为a的员工编号：

select \* from student where last\_name like ‘\_a\_\_\_\_’ or last\_name like ‘\_a%a\_\_\_\_’ or last\_name like ‘aa\_\_\_’;

1. 分页查询

select \* from s\_emp limit m,n;（m代表从第几条数据开始查，n代表查几条数据）

【m = (第几页-1)\*n】

1. ifnull(列或表达式,值)函数：如果列或表达式结果为空，则取指定的值，否则取原来的值。

update student set studentName = ifnull(studentName, '成') where id=1;

1. 当为表名取别名后，原来的表名便不能再使用了。（取别名可以用as关键字或者空格）
2. 事物（transaction）

* 四大特性（ACID）：

原子性(Atomicity)：一个事务要么同时成功，要么同时失败

一致性(Consistency)：事务开始和结束后，数据库要保持完整性

隔离性(Isolation)：数据库可以并发多个事务，每个事务互不干涉

持久性(Durability)：事务处理结束后，对数据的修改是永久的，系统故障也不会丢失

* 开启一个事务：begin或者start transaction

提交事务：commit或者commit work

回滚事务：rollback或者rollback work

* mysql隔离性的级别是可重复读（repeatable read）

oracle隔离性的级别是读已提交的（read committed）

## JDBC

1. 初始化

|  |
| --- |
| **private** Connection **con**; **private** PreparedStatement **ps**; **private** ResultSet **set**;  **public void** testOne() **throws** ClassNotFoundException, SQLException {  *// 1. 注册驱动并实例化* Class.*forName*(**"com.mysql.jdbc.Driver"**);  *// 2. 创建与数据库之间的连接* **con** = DriverManager.*getConnection*(**"jdbc:mysql://localhost:3307/test"**, **"root"**, **"123456"**);  *// 3. 发送sql语句到数据库* **ps** = **con**.prepareStatement(**"SELECT** *\** **from student"**);  *// 4. 数据库执行sql语句* **set** = **ps**.executeQuery();  *// 5. 获取查询结果* **while** (**set**.next()) {  System.***out***.print(**set**.getString(1));  System.***out***.println(**set**.getString(2));  }  *// 6. 关闭资源* **set**.close();  **ps**.close();  **con**.close(); } |

### MyBatis（<http://www.mybatis.org/mybatis-3/zh/getting-started.html>）

1. 配置过程：<https://blog.csdn.net/nba_linshuhao/article/details/82734834>

配置文件

|  |
| --- |
| *// jdbc.properties*  **jdbc.driver**=**com.mysql.jdbc.Driver jdbc.url**=**jdbc:mysql://localhost:3307/test jdbc.username**=**root jdbc.password**=**123456** |
| *// log4j.properties*  *### 设置Logger输出级别和输出目的地 ###* **log4j.rootLogger**=**debug,stdout,logfile** *### 把日志信息输出到控制台 ###* **log4j.appender.stdout**=**org.apache.log4j.ConsoleAppender** *#log4j.appender.stdout.Target=System.err* **log4j.appender.stdout.layout**=**org.apache.log4j.SimpleLayout** *### 把日志信息输出到文件：jbit.log ###* **log4j.appender.logfile**=**org.apache.log4j.FileAppender log4j.appender.logfile.File**=**jbit.log log4j.appender.logfile.layout**=**org.apache.log4j.PatternLayout log4j.appender.logfile.layout.ConversionPattern**=**%d{yyyy-MM-dd HH:mm:ss} %F %p %m%n** *###显示SQL语句部分* **log4j.logger.com.ibatis**=**DEBUG log4j.logger.com.ibatis.common.jdbc.SimpleDataSource**=**DEBUG log4j.logger.com.ibatis.common.jdbc.ScriptRunner**=**DEBUG log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate**=**DEBUG log4j.logger.java.sql.Connection**=**DEBUG log4j.logger.java.sql.Statement**=**DEBUG log4j.logger.java.sql.PreparedStatement**=**DEBUG** |
| *// mybatis-cfg.xml*  *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd"*>*** <**configuration**>   *<!-- 引入外部配置文件 -->* <**properties resource="jdbc.properties"**/>   *<!-- 配置mybatis运行环境 -->* <**environments default="mysql"**>  <**environment id="mysql"**>  *<!-- type="JDBC" 代表使用JDBC的提交和回滚来管理事务 -->* <**transactionManager type="JDBC"** />   *<!-- mybatis提供了3种数据源类型，分别是：POOLED,UNPOOLED,JNDI -->  <!-- POOLED 表示支持JDBC数据源连接池 -->  <!-- UNPOOLED 表示不支持数据源连接池 -->  <!-- JNDI 表示支持外部数据源连接池 -->* <**dataSource type="POOLED"**>  <**property name="driver" value="${jdbc.driver}"** />  <**property name="url" value="${jdbc.url}"** />  <**property name="username" value="${jdbc.username}"** />  <**property name="password" value="${jdbc.password}"** />  </**dataSource**>  </**environment**>  </**environments**>   *<!--映射器-->* <**mappers**>  <**mapper resource="mapper/ClassRoomMapper.xml"**/>  </**mappers**> </**configuration**> |
| *// pom.xml*  *<?***xml version="1.0" encoding="UTF-8"***?>* <**project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**groupId**>com.oaec</**groupId**>  <**artifactId**>MyBatis</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>   <**dependencies**>  *<!--My SQL-->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.40</**version**>  </**dependency**>   *<!--mybatis-->* <**dependency**>  <**groupId**>org.mybatis</**groupId**>  <**artifactId**>mybatis</**artifactId**>  <**version**>3.4.6</**version**>  </**dependency**>   *<!--log4j-->* <**dependency**>  <**groupId**>log4j</**groupId**>  <**artifactId**>log4j</**artifactId**>  <**version**>1.2.12</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.hamcrest</**groupId**>  <**artifactId**>hamcrest-core</**artifactId**>  <**version**>1.3</**version**>  </**dependency**>   *<!--测试包junit-->* <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>  </**dependency**>  </**dependencies**> </**project**> |

1. 使用

|  |
| --- |
| **public class** DBUtils {  *// sql 会话工厂* **private static** SqlSessionFactory *sSqlSessionFactory*;   **static** {  **try** {  *// 读取配置文件* Reader reader = Resources.*getResourceAsReader*(**"mybatis-cfg.xml"**);  *// 建立会话工厂  sSqlSessionFactory* = **new** SqlSessionFactoryBuilder().build(reader);  } **catch** (IOException e) {  e.printStackTrace();  }  }  */\*\*  \* 回去会话  \** ***@return*** *\*/* **public static** SqlSession getSession() {  **return** *sSqlSessionFactory*.openSession();  } } |
| *// mapper/ClassRoomMapper.xml*  *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE mapper PUBLIC "-//mybatis.org/DTD Mapper 3.0" "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>*** <**mapper namespace="dao.IClassRoomDao"**>   *<!--查询所有班级-->  <!--id为Dao的方法名-->* <**select id="getAllClassRooms" resultType="pojo.ClassRoom"**>  SELECT *\** FROM classroom  </**select**>   *<!--useGeneratedKeys:主键回填属性-->* <**insert id="insertClassRoom" parameterType="pojo.ClassRoom" useGeneratedKeys="true" keyProperty="id"**>  INSERT INTO classroom VALUES (null, #{className}, null)  </**insert**>   *<!--插入多条信息-->  <!--separator隔断符-->  <!--collection标签：当参数为List时，必须为list；当参数为数组时，必须为array；-->  <!--当参数为map时，collection为map中自己的键值对-->  <!--parameterType参数不区分大小写，同时省略的方式见映射类型别名文件TypeAliasRegistry-->* <**insert id="addMoreClassRoom" parameterType="java.util.List"**>  INSERT INTO classroom VALUES  <**foreach collection="list" item="classroom" separator=","**>  (#{classroom.id}, #{classroom.className}, #{classroom.classNum})  </**foreach**>  </**insert**>   <**update id="updateClassNUm" parameterType="map"**>  UPDATE classroom SET classNum = classNum + #{money} WHERE id IN  <**foreach collection="classNumList" item="number" separator="," open="(" close=")"**>  #{number}  </**foreach**>  </**update**>   *<!--<![CDATA[]]> 处理特殊字符-->* <**update id="update" parameterType="map"**>  <![CDATA[  update s\_emp set salary = salary + #{salary} where salary < #{add\_salary} OR id in  ]]]>  <**foreach collection="array" item="number" open="(" close=")"**>  #{number}  </**foreach**>  </**update**>   *<!— 动态查询：根据条件查询员工信息 中间可跟多个if子标签-->  <!--where 标签：条件标签【如果后面没有条件，where标签会自动删除】-->* <**select id="selectEmp" parameterType="map" resultType="pojo.Emp"**>  SELECT *\** FROM s\_emp  <**where**>  <**if test="money > 0 and money != null"**>  salary > #{money}  </**if**>  </**where**>  </**select**>  *<!--choose标签类似switch-case-->* <**select id="" parameterType="map" resultType=""**>  SELECT *\** FROM s\_emp  <**where**>  <**choose**>  <**when test="state == 1"**>  <![CDATA[ salary < #{money} ]]>  </**when**>  <**when test="state == 2"**>  <![CDATA[ salary = #{money} ]]>  </**when**>  <**otherwise**>  <![CDATA[ salary > #{money} ]]>  </**otherwise**>  </**choose**>  </**where**> </**select**>  </**mapper**> |
| *// 测试方法*  @Test **public void** testClassRoom1() {  *// 获取数据库会话* SqlSession session = DBUtils.*getSession*();  *// 找到对应的接口* IClassRoomDao classRoomDao = session.getMapper(IClassRoomDao.**class**);  **int** count = classRoomDao.insertClassRoom(**new** ClassRoom(**null**, **"suiyi"**, 0));  **if** (count > 0) { *// MyBatis不会自动提交事务，需要手动提交*  session.commit();  }  *// 关闭会话* session.close(); } |

注意：动态修改标签（set-if）update中的set标签中最后要加逗号

|  |
| --- |
| ClassRoomMapper.xml  *<!--预期类型的自定义--> <!--如果查询的字段和接受的属性一样，可以省略不写-->* <**resultMap id="empMap" type="pojo.classRoom"**>  <**result column="id" property="id"**/> </**resultMap**>  *<!—当JavaBean中属性为对象时，需要按照如下格式，selectRegionById为方法名-->*  <**resultMap id="empMap" type="Dept"**>  <**association property="region" column="region\_id" javaType="Region" select="dao.IRegionDao.selectRegionById"**/> </**resultMap**> |
| mybatis-cfg.xml  *<!--给包路径起别名，两种方法选其一-->* <**typeAliases**>  *<!--typeAlias是一个一个起别名-->* <**typeAlias type="pojo.Emp" alias="Emp"**/>  *<!--package标签是让该包下面的类自动起别名，别名就是自己的类名-->* <**package name="pojo"**/> </**typeAliases**> |
| mybatis-cfg.xml  <**settings**>  *<!--延迟加载全局开关,当开启时，所有关联对象都会延迟加载-->* <**setting name="lazyLoadingEnabled" value="true"**/>  *<!--按需加载-->* <**setting name="aggressiveLazyLoading" value="true"**/> </**settings**> |
| *<!--集合嵌套查询-->* <**resultMap id="Region" type="list"**>  <**collection property="deptList" column="id" ofType="Dept" select="dao.IDept.selectByRegionById"**/> </**resultMap**> |

## Html

1. 标签

* 可见标签：行级标签、块级标签、行内块级标签
* 行级标签：前后不换行、不可以改变元素大小。宽度和高度都是文本的宽度和高度。

<span>

* 块级标签：浏览器会在其前后换行、可改变元素的大小。宽度默认是父级宽度，高度默认是自身的文本高度。

<div>、<p>

标题标签：h1-h6，字体加粗，字号依次变小

ol有序列表

ul无序列表

dl（dt、dd）定义描述标签

* 行内块级标签：前后不换行、可改变元素大小。默认是文本的宽度和高度。

可通过display属性改变行级标签和块级标签

* 不可见标签

表单标签form

action：表单提交地址

method：表单提交方式（get/post）

enctype：表单处理方式（multipart/form-data：二进制流处理，比如文件上传时需要这么写）

下拉选项select

分组 optgroup

## CSS

1. 外边距：上下外边距不相加，以最大的为准；左右相加
2. float浮动：清除浮动clear：left、right、both
3. position定位

相对定位的含义是根据原来的位置进行定位，位置保留。

绝对定位：位置不保留，根据父级元素含有position的定位。（父级一般设置为相对定位）

固定定位：position：fixed 相对于浏览器定位，不占据位置

1. text-shadow 文本阴影（参数：水平阴影的位置、垂直阴影的位置、模糊的距离、阴影的颜色）

box-shadow 边框阴影

## Javascript

1. js引入：

方式一：<a href=”javascript:alert(‘’)”></a>

方式二：<script>双标签

1. typeof + 变量：打印类型
2. parseInt(“32.563”) 类型转换
3. 输入框：window.prompt(“请输入你的年龄：”)
4. foreach循环

|  |
| --- |
| for(var x in array) { // x为当前循环的次数  console.log()；  } |

1. 前进到下一页的网页：

history.forward()

history.go(1)

1. DOM

|  |
| --- |
| var sun = document.getElementById("sun");  // 返回当前节点的父节点  var parent = sun.parentNode;  // 返回当前节点的上一个兄弟节点,包括空格(文本节点)  var previousSibling = sun.previousSibling;  // 返回当前节点的所有子节点,包括空格(文本节点)  var childNodes = parent.childNodes;  // 返回当前节点的第一个子节点,包括空格(文本节点)  var firstChild = parent.firstChild; |

## JQuery

1. 数据加载完成后执行function函数

jQuery(document).ready(function () {})

$(document).ready(function () {})

$(function(){})

1. DOM对象转JQuery：$(DOM)

JQuery对象转DOM：jqueryObject[0]或者jqueryObject.get(0)

1. 选择器

:first 第一个

:last 最后一个

:odd 奇数，从0开始算起

:even 偶数

:eq(2) 获取第几个，从0开始算起

:gt(index) 获取大于index

:lt(index) 获取小于index

:header 标题标签

:not() 去除

> 只筛选孩子；空格筛选子孙

$(“[title]”) 含有title标签

$(“[id=sun]”) 包含id，且值sun的

$(“[title^=s]”) 以s为开头的

$(“[title$=s]”) 以s为结尾的

$(“[title\*=value]”) 包含value的

jquery支持多重属性选择器

1. $("ul”).find(“.blue”) find是从子元素中找

$(“li”).filter(“.yellow”) filter是从自己和子元素中找

each(function(){}) each是循环遍历

end() 回到操作对象的上一个对象

endBack() 添加操作对象的上一个对象

$(“ul”).append(list) 在ul后边追加list；也可为list.appendTo(“ul”)

prepend() 前置,用法同上

after() 外部追加；相对应的为insertAfter()

before() 外部前置；insertBefore()

append() 添加元素节点

detach() remove() 删除元素节点

empty() 清空元素内容

clone() 克隆元素节点。在clone中传入true，即将绑定的时间也一起克隆。

wrap() 为每个元素外层包裹元素

wrapInner() 为每个元素内层包裹元素

attr({}) 设置属性值，当只有一个属性时，{}可省略。内部放的应该为JSON格式。

removeAttr(“title class”) 删除多个属性

addClass(“red back”) 追加样式

toggleClass(“red back”) 切换多个样式，有就删除，没有就追加。

hasClass(“red”) 是否有这个样式

.html()中添加的会被自动解析为html标签；.text()中添加的值为纯文本

1. .children() 获取孩子元素

.prev() 获取元素的同级元素的上一个元素

.next() 获取元素的同级元素的下一个元素

.parent() 获取元素的父级元素

.siblings() 获取同级元素的其他元素，不包括自己

1. 获取和设置元素节点的定位

offset() 获取和设置div与窗口之间的偏移量。

postion() 获取匹配元素和含有postion属性的父元素的值。

1. 事件

* .bind(“click”,function(){}) 绑定一个事件

绑定多个事件：

|  |
| --- |
| $(function () {  *// 不推荐使用* $(**"h3:eq(1)"**).bind(**"click dblclick"**, function () {  **if** (window.event.type == **"click"**) {  $(**this**).css(**"color"**, **"blue"**);  } **else** {  $(**this**).css(**"color"**, **"green"**);  }  });  *// 推荐使用* $(**"h3:eq(0)"**).bind({  click:function () {  $(**this**).css(**"color"**, **"blue"**);  },  dblclick:function () {  $(**this**).css(**"color"**, **"green"**);  } }); }) |

.unbind() 移除所有事件，括号里写入特定事件，则移除特定事件。

当为一个对象绑定多个同一事件时，只会最后一个有用。

* 合成事件：hover(function(){}, function(){}) 鼠标移入移出事件，若参数只有一个，则表示鼠标移入移出是同一个事件。
* 模拟事件（触发器）：

|  |
| --- |
| $(**"h3:eq(1)"**).mouseover(function () {  $(**"h3:first"**).trigger(**"click"**); }); |

1. 显示

show(time) 第一个参数为时间（改变的是宽度、高度和透明度）

hide() 隐藏，参数如上

toggle() 切换，参数如上

slideDown()、slideUp()、slideToggle() 类似投影仪效果，只改变高度

fadeIn()、fadeOut()、fadeToggle()

fadeTo() 淡到某种程度，第一个参数为时间，第二个参数为淡到的程度

1. animate() 自定义动画，第一个参数为JSON格式的参数，第二个为时间，第三个为执行结束所要执行的方法

$(“ul:animated”).stop() 停止当前动画，若有多个动画，剩下的动画继续执行。未传参数表示立即结束动画；若传入参数true，则暂停所有动画；若传入两个true，表示立即停止本次动画，清空动画序列。

## Servlet

1. 基本配置

web.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"  version="4.0"**>   *<!--配置servelt进入网址，或者使用@WebServlet-->* <**servlet**>  <**servlet-name**>c</**servlet-name**>  <**servlet-class**>com.oracle.servlet.Student</**servlet-class**>  <**init-param**>  <**param-name**>userName</**param-name**>  <**param-value**>hhh</**param-value**>  </**init-param**>  <**init-param**>  <**param-name**>password</**param-name**>  <**param-value**>123456</**param-value**>  </**init-param**>  </**servlet**>    <**servlet-mapping**>  <**servlet-name**>c</**servlet-name**>  <**url-pattern**>/student</**url-pattern**>  </**servlet-mapping**>    *<!--配置欢迎页面-->* <**welcome-file-list**>  *<!--<welcome-file>index.jsp</welcome-file>-->* <**welcome-file**>form.jsp</**welcome-file**>  </**welcome-file-list**>   *<!--配置错误展示页面-->* <**error-page**>  <**error-code**>500</**error-code**>  <**location**>/error.jsp</**location**>  </**error-page**>  </**web-app**> |

1. 过滤器

|  |
| --- |
| 方法一：  **public class** RequestFilter **implements** Filter {  **private** String **mCharset**;  *// 服务器启动时，初始化* @Override  **public void** init(FilterConfig filterConfig) **throws** ServletException {  **mCharset** = filterConfig.getInitParameter(**"charset"**);  }  *// 过滤* @Override  **public void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain) **throws** IOException, ServletException {  request.setCharacterEncoding(**mCharset**);  *// 继续下一个符合要求的过滤器* chain.doFilter(request, response);  } }  web.xml  <**filter**>  <**filter-name**>request</**filter-name**>  <**filter-class**>com.baidu.filter.RequestFilter</**filter-class**>  <**init-param**>  <**param-name**>charset</**param-name**>  <**param-value**>UTF-8</**param-value**>  </**init-param**> </**filter**> <**filter-mapping**>  <**filter-name**>request</**filter-name**>  <**url-pattern**>/\*</**url-pattern**> </**filter-mapping**> |
| 方法二：  @WebFilter(urlPatterns = **"/\*"**,  initParams = {@WebInitParam(name = **"contentType"**, value = **"text/html;charset=UTF-8"**)}) **public class** ResponseFilter **implements** Filter {  **private** String **mContentType**;   @Override  **public void** init(FilterConfig filterConfig) **throws** ServletException {  **mContentType** = filterConfig.getInitParameter(**"contentType"**);  }   @Override  **public void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain) **throws** IOException, ServletException {  response.setContentType(**mContentType**);  chain.doFilter(request, response);  } } |

1. 监听器

|  |
| --- |
| @WebListener **public class** onLineCount **implements** HttpSessionListener {  **private int count**; *// 在线人数  // 创建会话* @Override  **public void** sessionCreated(HttpSessionEvent se) {  **count**++;  System.***out***.println(**"发现新的会话："** + **count**);  *// 设置session会话的秒数* se.getSession().setMaxInactiveInterval(60);  }   *// 销毁* @Override  **public void** sessionDestroyed(HttpSessionEvent se) {  **count**--;  System.***out***.println(**"发现会话销毁："** + **count**);  } } |

1. 转发有两种形式（forward/include）

* 相同点：转发站内的资源【jsp，html，servlet，…】

转发会把请求和响应数据一起转发。

转发时客户的url地址不发生改变

* 不同点：forward全权由新的servlet实现，include是由现在的去实现

req.getRequestDispatcher(**"new"**).forward(req, resp);

1. 重定向：

既可以转发站内的资源，又可以转发站外的资源。

不能将请求和响应数据重定向到新的业务类中。

重定向URL地址会发生改变

resp.sendRedirect(**"http://www.baidu.com"**);

## Ajax

1. 使用方式

ajax.jsp

|  |
| --- |
| **function** *Javascript\_Ajax\_Get*() {  **var** userName = ***document***.getElementsByName(**"username"**)[0].**value**;  **var** password = ***document***.getElementsByName(**"password"**)[0].**value**;  ***console***.log(**"userName"** + userName);   *// 1. 创建 Ajax 引擎对象* **var** xmlHttp;  **if** (***window***.***XMLHttpRequest***) {  *// 支持 IE7+, firefox, chorme, Opera* xmlHttp = **new *XMLHttpRequest***();  } **else** {  xmlHttp = **new *ActiveXObject***(**"Microsoft.XMLHTTP"**);  }  *// 2. 绑定监听器* xmlHttp.**onreadystatechange** = **function** () {  *// 4代表请求完成，200代表请求正常* **if** (xmlHttp.**readyState** == 4 && xmlHttp.**status** == 200) {  *// 取出相应的字符串* **if** (xmlHttp.**responseText** == **"0"**) {  ***document***.getElementById(**"context"**).**innerText** = **"登陆成功"**;  } **else** {  ***document***.getElementById(**"context"**).**innerText** = **"登陆失败"**;  }  }  };  **var** url = **"http://localhost:8888/mavanajax/ajax?username="** + userName + **"&password="** + password;  *// 3. 绑定服务器地址  // 第一个参数代表请求方式，第二个参数表示url，第三个参数代表是否异步提交* xmlHttp.open(**"GET"**, url, **true**);  *// 4. 发送* xmlHttp.send(); }  **function** *Javascript\_Ajax\_Post*() {  **var** userName = ***document***.getElementsByName(**"username"**)[0].**value**;  **var** password = ***document***.getElementsByName(**"password"**)[0].**value**;  ***console***.log(**"userName:"** + userName);  ***console***.log(**"password:"** + password);   *// 1. 创建 Ajax 引擎对象* **var** xmlHttp;  **if** (***window***.***XMLHttpRequest***) {  *// 支持 IE7+, firefox, chorme, Opera* xmlHttp = **new *XMLHttpRequest***();  } **else** {  xmlHttp = **new *ActiveXObject***(**"Microsoft.XMLHTTP"**);  }  *// 2. 绑定监听器* xmlHttp.**onreadystatechange** = **function** () {  *// 4代表请求完成，200代表请求正常* **if** (xmlHttp.**readyState** == 4 && xmlHttp.**status** == 200) {  *// 取出相应的字符串*  *// 可使用JSON.parse()方法将text转为JSON格式* ***console***.log(xmlHttp.**responseText**);  **if** (xmlHttp.**responseText** == **"0"**) {  ***document***.getElementById(**"context"**).**innerText** = **"登陆成功"**;  } **else** {  ***document***.getElementById(**"context"**).**innerText** = **"登陆失败"**;  }  }  };  **var** url = **"http://localhost:8888/mavanajax/ajax"**;  *// 3. 绑定服务器地址  // 第一个参数代表请求方式，第二个参数表示url，第三个参数代表是否异步提交* xmlHttp.open(**"POST"**, url, **true**);  *// 4. 设置请求头 [可省略] 以下为正常请求头，二进制时使用multipart/form-data* xmlHttp.setRequestHeader(**"Content-type"**, **"application/x-www-form-urlencoded"**);  *// 5. 发送* ***console***.log(**"password:"** + password);  xmlHttp.send(**"password="** + password + **"&username="** + userName); }  **function** *JQuery\_Get*() {  **var** userName = **$**(**"[name=username]"**).val();  **var** password = **$**(**"[name=password]"**).val();  **$**.ajax({  **type**: **"GET"**,  **url**: **"http://localhost:8888/mavanajax/ajax?username="** + userName + **"&password="** + password,  **data**:{}, *// 传参，JSON格式* **async**: **true**, *// 是否异步提交* **timeout**: 10000, *// 请求处理的时间* **dataType**: **"json"**, *// text,html,xml,script,json* **context**: **$**(**"#context"**), *// 回调函数内this指向的对象* **contentType**: **"application/x-www-form-urlencoded"**,  success: **function** (object) {  **if** (object[**"state"**] == 0) {  ***console***.log(**"success"**);  **this**.**html**(object.stateInfo);  } **else** {  **this**.**html**(**"登录失败"**);  }  },  error: **function** () {  ***console***.log(**"服务器未找到！"**);  }  }) }  **function** *JQuery\_Post*() {  **var** userName = **$**(**"[name=username]"**).val();  **var** password = **$**(**"[name=password]"**).val();  **$**.ajax({  **type**: **"POST"**,  **url**: **"http://localhost:8888/mavanajax/ajax"**,  **data**:{**"username"**:userName, **"password"**:password}, *// 传参，JSON格式* **async**: **true**, *// 是否异步提交* **timeout**: 10000, *// 请求处理的时间* **context**: **$**(**"#context"**), *// 回调函数内this指向的对象* **dataType**: **"json"**, *// text,html,xml,script,json* **contentType**: **"application/x-www-form-urlencoded"**,  success: **function** (object) {  **if** (object[**"state"**] == 0) {  ***console***.log(**"success"**);  **this**.**html**(**"登录成功"**)  } **else** {  **this**.**html**(**"登录失败"**);  }  },  error: **function** () {  ***console***.log(**"服务器未找到！"**);  }  }) } |
| @WebServlet(**"/ajax"**) **public class** AjaxServlet **extends** HttpServlet {   @Override  **protected void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  Map<String, Object> map = **new** HashMap<>();   *// 设置请求和相应字符集* req.setCharacterEncoding(**"UTF-8"**);  resp.setCharacterEncoding(**"UTF-8"**);  resp.setContentType(**"text/html;charset=UTF-8"**);   String username = req.getParameter(**"username"**);  String password = req.getParameter(**"password"**);   PrintWriter writer = resp.getWriter();   *// 账号判断* **if** (**""**.equals(username) || username == **null**) {  map.put(**"state"**, 1);  map.put(**"stateInfo"**, **"账号不能为空"**);  String json = **new** GsonBuilder().serializeNulls().create().toJson(map);  writer.write(json);  writer.close();  **return**;  }  *// 密码判断* **if** (**""**.equals(password) || password == **null**) {  map.put(**"state"**, 2);  map.put(**"stateInfo"**, **"密码不能为空"**);  String json = **new** GsonBuilder().serializeNulls().create().toJson(map);  writer.write(json);  writer.close();  **return**;  }    **if** (**"hhh"**.equals(username) && **"123456"**.equals(password)) {  map.put(**"state"**, 0);  map.put(**"stateInfo"**, **"登陆成功"**);  } **else** {  map.put(**"state"**, 4);  map.put(**"stateInfo"**, **"登录失败"**);  }   String json = **new** GsonBuilder().serializeNulls().create().toJson(map);  writer.write(json);  writer.close();  }   @Override  **protected void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  doPost(req, resp);  }  } |

## JSON

1. 转化为JSON对象

|  |
| --- |
| *// 该方法无视属性为null值的属性 //Gson gson = new Gson(); // 将所有属性封装，包括属性为null值的* Gson gson = **new** GsonBuilder().serializeNulls().create(); String jsonStr = gson.toJson(user); System.***out***.println(jsonStr); |

1. JSON转化为实体类

|  |
| --- |
| *// 将 JSON 字符串转为对象* @Test **public void** testFour() {  String jsonStr = **"{\"userName\":"** + **"lijing"** + **",\"age\":"** + **"28}"**;  Gson gson = **new** GsonBuilder().serializeNulls().create();  User user = gson.fromJson(jsonStr, User.**class**);  System.***out***.println(user); }  *// 将 JSON 字符串转为对象数组* @Test **public void** testFive() {  String jsonStr = **"[{\"userName\":"** + **"lijing"** + **",\"age\":"** + **"28}]"**;  Gson gson = **new** GsonBuilder().serializeNulls().create();  User[] users = gson.fromJson(jsonStr, User[].**class**);  **for** (User user : users) {  System.***out***.println(user);  } }  *// 将 JSON 字符串转为集合* @Test **public void** testSix() {  String jsonStr = **"[{\"userName\":\"李静\",\"age\":28},{\"userName\":\"刘磊\",\"age\":26}]"**;  Gson gson = **new** GsonBuilder().serializeNulls().create();  List<User> users = gson.fromJson(jsonStr, **new** TypeToken<List<User>>(){}.getType());  **for** (User user : users) {  System.***out***.println(user);  } } |

## SpringMVC

1. 配置

web.xml

|  |
| --- |
| *<!--配置上下文信息-->* <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>WEB-INF/applicationContext.xml</**param-value**> </**context-param**>  *<!--监听器-->* <**listener**>  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**> </**listener**>  <**servlet**>  <**servlet-name**>dispatcher</**servlet-name**>  <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**> </**servlet**> <**servlet-mapping**>  <**servlet-name**>dispatcher</**servlet-name**>  <**url-pattern**>/</**url-pattern**> </**servlet-mapping**> |