JDBC

JDBC 基本操作

步骤

- 1.导包
- 2.注册驱动

  Class. forName ("com.mysql. jdbc.Driver") :

  类里存在静态代码块

```
static {
    try {
        DriverManager.registerDriver(new Driver());
    } catch (SQLException var1) {
        throw new RuntimeException("Can't register driver!");
    }
}
```

- 3.获取Connection对象

  Connection conn = DriverManager.getConnection("jdbc:mysql:///rainbow", "root", "12345678");

- 4.定义sql

  String sql = " "

- 5.执行,接收返回值

  resultSet = statement.executeQuery(sql);

- 6.处理结果

```
while (resultSet.next()) {

}
```

- 7.释放资源

  释放：ResultSet　Statement　Connection

```
if (resultSet != null) {
    resultSet.close();
}
if (statement != null) {
    statement.close();
}
if (root != null) {
    root.close();
}
```

对象

Connection　数据库连接对象

- 1.获取执行sql的对象

  Statement createstatement()
  Preparedstatement preparestatement （String sql）

- 2.管理事务

  开启事务：setAutocommit（boolean autocommit）:调用该方法设置参数为false，即开启事务
  提交事务：commit（）
  回滚事务：rollback（）

- Statement　　执行sql对象

  boolean execute（String sql）：可以执行任意的sql
  int executeupdate（string sql）；执行DML （insert、update. delete）语句、DDL （create, alter、drop）语句
  *返回值：影响的行数，可以通过这个影响的行数判断*DML*语句是否执行成功 返回值>0的则执行成功，反之，则失败。
  Resultset executequery（String sql）：执行DQL （select）语句

- ResultSet　　结果集对象(封装查询结果)

  next（）：游标向下移动一行
  getXxx（参数）：获取数据
  　　xxx：代表数据型 如：int getInt（），String getstring（）
  参数：1. int：代表列的编号，从1开始　如：getstring（1）
  　　2. String：代表列名称　　getDouble("balance")

- PreparedStatement　　执行sql对象

JDBC 工具类(练习)

分析

- 抽取方法 注册驱动
- 抽取方法 获取连接对象

  解决不想用传递参数,还要保证通用性的方法:
  配置文件/工具类的静态代码块内读取文件,获取值)

  - Properties

    new Properties() 创建对象
    load(FileReader)方法传入FileReader
    getProperty(键名) 获取数据

  - 获取src路径下的文件——> ClassLoader 类加载器

    (先获取字节码文件对象)
    　ClassLoader cl = 类名.class.getClassLoader()　　返回ClassLoader
    URL u= cl.getResource 获取资源(以src为跟目录)　　返回URL
    u.getpath()　　返回绝对路径

- 抽取方法 释放资源

PreparedStatement

是预编译SQL,继承自Statement,解决SQL注入

- SQL注入

  原静态sql 代码 :
  "select * from users where admin ='" + admin + "'and password ='" + password+"'"

  　　密码输入 a' or 'a'='a

  注入后SQL语句:
  select * from users where admin ='t'and password ='a' or 'a'='a'

  (false or true = true) ,where后条件为true,返回所有数据　登录成功

步骤

- 1.导包
- 2.注册驱动
- 3.获取Connection
- 4.**定义SQL**

  sql参数使用占位符？
  e.g:　select * from users where admin =? and password =?

- 5.获取 PreparedStatement
  connection.preparedStatement()
- 6.占位符赋值
  PreparedStatement.setxxx(参数1,参数2)
  参数1: ?位置,1开始
  参数2: 值
- 7.执行SQL(不需要传递参数)
- 处理结果 释放资源

JDBC事务

操作 (使用Connection对象管理事务)

- 开启事务
  setAutoCommit() 默认关闭事务,设置为false 开启事务
- 提交事务
  commit()
- 回滚事务
  rollback()

数据库连接池

实现

标准接口 DataSource (javax.sql) 数据库厂商实现

方法

- 获取连接 getConnection()
- 归还连接 Connection.close()
  若Connection是从连接池获取,调用close方法不会关闭而是归还

C3P0

步骤

- 1.导包
- 2.定义配置文件
  (不推荐硬编码) 定义
  c3p0.properties或c3p0-config.xml
  放在src下
- 3.创建核心对象
  数据库连接池对象 ComboPooledDataSource
- 4.获取连接 getConnection

Druid

步骤

- 1.导包
- 2.配置文件(properties)
  任意名称 任意位置
- 3.获取数据库连接对象:工厂获取(DruidDataSourceFactory)
- 4.获取连接 getConnection

Spring JDBC

Spring对JDBC的简单封装(提供JDBCTemplate对象简化)

步骤

- 1.导包
- 2.创建JDBCTemplate对象(依赖数据源DataSource)
- 3.调方法进行CRUD操作
  - update()
    执行DML语句 增删改
  - queryForMap()
    将结果集封装为Map集合 (结果集长度只能是1)
  - queryForList()
    将结果集封装为List集合 (每一条记录封装成Map集合,Map集合封装到List集合)
  - query()
    将结果封装成JavaBean对象
    传入参数RowMapper:
    一般使用BeanRowMapper实现类,可以自动封装
    List<users> query = jdbcTemplate.query("select * from users where salary = 1000", new BeanPropertyRowMapper<users>(users.class));

    (手动:new RowMapper接口,重写mapRow方法)
  - queryForObject()
    将结果封装 (常用来执行聚合函数)
    Long l = jdbcTemplate.queryForObject("select count(id) from users where salary = 1000", Long.class);

HTML

-

属性

color

- 1.单词 red
- 2.rgb rgb(0,0,255)
- 3.16进制 #3626F1

width

- 数值 width = '20' 单位 px像素
- 比例 width = "50%" 相对父元素比例

标签

-
 换行 自闭和
- <h1> - <h6>
- 段落标签
- <hr> 一条水平线 自闭和
  属性
  color = "red"

width 宽度
size 高度粗细
align 对齐方式 center left right

- <b> 加粗
- <i> 斜体
- ~~<font>~~

  属性
  color 颜色
  size 大小
  face 字体
- ~~ ~~ 文本居中
- <img> 图片 自闭和

  src属性 位置

列表标签

有序列表

属性:type 项目符号类型
disc square circle

-
-

无序列表

-
-

- 超链接标签 a

  href属性:访问资源的 URL
  邮件地址 mailto://
  target属性: self 本页 blank 新空白页

语意化标签

- 块级标签 独占一行
- 内联标签
- <header> 头
- <footer> 尾

表格标签

- <table>

  属性
  width 宽度
  boarder 边框
  cellpadding 内容和单元格距离
  cellspacing 单元格间距离 0:线合为一条
  bgcolor 背景色
  align 对齐方式
- <tr> 行
- <th> 表头单元格
- <td> 单元格

  colspan 合并列
  rowspan 合并行
- <caption>表格标题
- <thead>表格头
- <tbody>表格体
- <tfoot> 表格脚

表单 <form>

属性: action: url
method get post
表单数据想被提交,必须指定name属性

<input> 标签

属性:type name

type属性的值

- text 默认

  placeholder 属性 指定输入框提示信息 框内容变化会自动消失
- password
- radio 单选框

  checked属性指定默认值
- checkbox 复选框

  value属性指定提交时的值
  checked属性 默认选中(checked="checked" 或直接写 checked)
- file 文件选择器
- hidden 隐藏域
- color 取色器
- date 年月日
- datetime-local 年月日时分秒

按钮

- submit 提交按钮

  value属性 按钮字样
- button 普通按钮
- image 配合 src属性 图片按钮

- <label> 指定输入项的文字描述信息

  label的for属性一般会和 input 的 id属性值 对应。如果对应了，则点击label区域，会让input输入框获取焦点。

<select> 下拉列表

- <option> 列表中每个选项

- <textera> 文本域

  属性 cols 一行文字
  rows 行

CSS

定义方式

- 内联样式

  标签内使用style属性指定css
   ……

- 内部样式

  定义style标签
  <style>
  div{
  color:red;
  }

  </style>

- 外部样式

  定义CSS文件,head标签内定义link标签 引入
  <link rel="stylesheet" href="css/a.css">
  hello css
  还可以写:
  <style>
  @import"css/a.css";
  </style>

- 语法

  选择器{
  属性名1:属性值;
  属性名2:属性值;
  }
  用;隔开,最后一对属性可以不加

选择器

基础选择器

优先级 id>元素
类>元素

- id选择器

  建议一个html id值唯一

  #id属性值 {

  }

- 元素选择器

  标签名{

  }

- 类选择器

  .class属性值{

  }

  aaaaa

扩展选择器

- 选择所有元素

  *{
  }

- 并集选择器

  选择器1,选择器2{
  }

- 子选择器

  选择器1 选择器2{
  }

- 父选择器

  选择2的父元素1

  选择器1 > 选择器2{
  }

- 属性选择器

  选择元素名称,属性名=值的元素

  元素名称[属性名='值']{
  }

- 伪类选择器

  选择一些元素具有的状态

  e.g.

  <a>
  状态:
  link:初始化的状态
  visited:被访问过的状态
  active:正在访问状态
  hover:鼠标悬浮状态

属性

1.字体 文本

- font-size 字体大小
- text-align 对齐方式
- line-hight 行高

2.背景

- background

3.边框

- border

4.尺寸

- width
- height

5.盒子模型 (相对)

- margin 外边距

- padding 内边距
  默认内边距会影响整个盒子大小
  指定 box-sizing : border-box 设置盒子属性 让width和height就是最终盒子大小
- float : 浮动
  - left
  - right

JS

ECMAScript 加自己特有的东西(BOM DOM)

结合方式

- 1.内部JS
  <script> 标签
- 2.外部JS
  <script> 标签指定src属性
- (可定义多个 任意位置)

注释

- 单行
- 多行

数据类型

1.原始数据类型

- number 整数 小数 NaN
- string 字符串
  "a" 'abc'
- boolean
- null 对象为空占位符
- undefined 未定义,变量没给初始化值的默认值

2.引用数据类型: 对象

typeof 查看类型

typeof null : object 历史遗留问题

- 变量

运算符

- ==
  比较前类型转换
- === 全等于

- 流程控制语句

基本对象

1.function 函数对象

方式1: var 函数名 = new function(形参列表,"方法体") 不推荐
方式2 : function 方法名(形参){
}
方式3 : var 方法名 : function(){
}

特点

- 1.不写形参类型
- 2.方法是对象 同名会覆盖
- 3.方法调用只与方法名有关,和参数列表无关
- 4.其中隐藏内置对象 (数组) arguments 封装所有 形参
  arguments[0]

Array 数组

数组中 元素类型可变
长度可变

创建方式

- 1. var arr1 = new Array(1,2,3)
  指定元素列表
- 2.var arr2 = new Array(5)
  指定长度
- 3.var arr3 = new Array[1,2,3,4,5]
  指定元素列表 推荐

属性

- length 数组长度

方法

- join
  var str = a.join(' ')
  将指定数组元素用指定符号拼接,返回字符串
- push
  添加新元素
  e.g
  fruits.push("Lemon");

Date 日期对象

创建

- var date = new Date();

方法

- toLocalString(); 返回当前对象对应本地字符串格式
- getTime(); 自1970.1.1 零点到现在毫秒值

Math 对象

特点: 不用创建 直接使用 Math.方法名

方法

- random()
  返回 [0-1) 间随机数

- ceil()
  向上舍入
- floor()
  向下舍入
- round()
  四舍五入

▼ 属性

- PI

▼ RegExp 正则表达式对象

▼ 1.规则

- 单个字符
  \d ：[0-9] 单个数字字符
  \w : [a-zA-Z0-9_] 单个单词字符

▼ 量词符号

- ？ 出现0次或1次
- * 出现0次或多次
- + 出现1次或多次
- {m,n}
  n≥数量≥m
  若m缺省 {,n} 最多n次
  若n缺省 {m,} 最少m次

- ^ 匹配开头
- $ 匹配结尾

▼ 2.正则对象

▼ 创建

- var reg = new RegExp("正则表达式")
  内用转义字符 //w 表示
- var reg = /正则表达式/
  e.g
  var reg = /^\w{6,12}$/;

▼ 验证

```
var username = "zhangsan"
var flag = reg.test(username)
```

- test

```
var reg = /^\w{6,12}$/;
var username = 'zhangsan';
var flag = reg.test(username)
```

▼ global 全局对象

不需要对象可以直接调用 方法名();

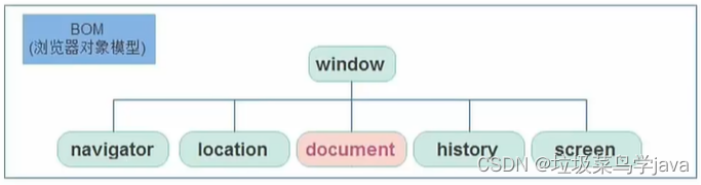
- encodeURI()
  URL 编码
- decodeURI()
  URL 解码
- encodeURIComponent( )
- decodeURIComponent( )
- parseInt( ) 字符串转数字
- isNaN( ) 判断是否NaN

▼ BOM

浏览器各个组成部分封装成对象

▼ window 对象

不需要创建可以直接使用， window.方法名( ) window引用可省略



▼ 弹窗

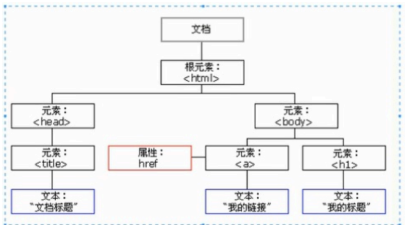
- alert()
- confirm
  点击取消 返回 false 确认 返回 true
- prompt
  prompt 输入框,输入内容做返回值

▼ 窗口相关

- close 关闭窗口
  谁调用关谁
- open 打开窗口
  返回新window对象

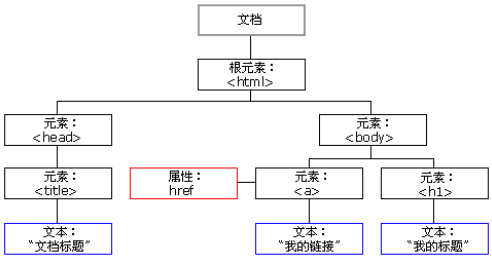
▼ 定时器相关

- setTimeout
  指定的毫秒数后调用函数或计算表达式。
- clearTimeout
  取消 setTimeout 设置的 timeout
- setInterval
  按照指定的周期（以毫秒计）来调用函数或计算表达式。
- clearInterval
  取消 setInterval 设置的 timeout

- 对象
  - location 地址栏对象
    window.location.方法，可省略window
    - 方法
      - reload 刷新
    - 属性
      - href 设置地址栏
  - history 历史记录对象
    - 方法
      - back 加载 history 列表中的前一个 URL
      - forward 加载 history 列表中的下一个 URL
      - go 加载 history 列表中的某个具体页面。
    - 属性
      - length 返回当前窗口历史列表中的 URL 数量
- 核心 DOM
  控制文档内容
  - 

  - 标准的三个部分
    - 核心 DOM
      针对任何结构化文档的标准模型
      - **document　文档对象**
        - 创建其它Dom对象
          - createAttribute (name)
          - createComment ()
          - createElement()
          - createTextNode()
        - 获取Element对象
          - getElementById（）：根据id属性值获取元素对象。id属性值一般唯一
          - getElementsByTagName（）：根据元素名称获取元素对象们。返回值是一个数组
          - getElementsByClassName（）：根据class属性值获取元素对象们。返回值是一个数到
          - getElementsByName（）：根据name属性值获取元素对象们。
      - **Element：元素对象**
        - 获取/创建:　通过document对象
        - 方法:
          - setAttribute()　设置属性
          - removeAttribute()　移除属性
      - **Attribute：属性对象**
      - **Text：文本对象**
      - **Comment　注释对象**
      - Node　节点对象，其他5个的父对象
        所有dom对象都可被认为是个节点
        节点可以是元素节点、属性节点、文本节点，或者也可以是"节点类型"那一节中所介绍的任何一种节点。

        所有的对象均能继承用于处理父节点和子节点的属性和方法，但是并不是所有的对象都拥有父节点或子节点。
        例如，文本节点不能拥有子节点，所以向类似的节点添加子节点就会导致DOM 错误。
        - 方法
          - appendchild（）：向节点的子节点列表的结尾添加新的子节点。
          - removechild 删除(并返回）当前节点的指定子节点。
          - replacechild（）：用新节点替换一个子节点
        - 属性
          - parentNode 返回当前节点的父节点
    - XML DOM
      针对 XML 文档的标准模型
    - HTML DOM
      针对 HTML 文档的标准模型
  - 方法
    - document.getElementByid(" ") 通过id获取元素对象
  - 事件
    - onclick 属性
      onclick = '函数名'
- HTML DOM



- HTML DOM

  获取 修改 添加 删除 HTML 元素

  - inner HTML 属性

    var div = document.getElementById(" ")

    - 替换

      div.innerHTML = " 元素 标签"

    - 添加

      div.innerHTML += " 元素 标签"

  - 修改样式

    - 1.使用元素style属性 设置
    - 2.提前定义好类选择器,通过元素className属性来设置class属性值

- JS补充

  - 超链接 <a> 标签

    功能: 1. 可被点击的样式
    2. 点后跳转
    href属性不设置: 标签不显示
    设置为空串:在本页跳转
    设置 href="javascript:void(0)" 不跳转

  - 定时器
  - 数组

    - splice 方法

      splice 方法 直接改变原数组
      arr.splice(起始索引,改变个数,元素(可选))
      arr.splice(index,many):index后删除many个
      arr.splice(5):删除之后所有元素
      e.g. arr.splice(1,0,"orange"):1后不删除 只插入元素 orange
      many>0:删除且添加元素
      arr.splice(1,0):1后删除0个,既 不删除

- Bootstrap

  - 响应式布局

    - 实现:依赖栅格系统 一行分成12个格,可指定元素占几格

  - 步骤

    - 1.定义容器

      - container 两边留白
      - container-fluid 所有设备均100%宽度

    - 2.定义行 row
    - 3.定义元素

      指定在不同设备上所占格子数 col-设备代号-格子数

      - 设备代号

        col-md-1

        - xs 超小 <768px
        - sm 小屏 ≥768px
        - md 中等屏 ≥992px
        - lg 大屏 ≥1200px

      - 注意

        - 1.一行中格子超过12个,超过部分自动换行
        - 2.栅格类可向上兼容,适用屏幕宽度大于等于分界点的设备
        - 3.真实设备宽度小于代号最小值,一个元素占满一整行

    - 全局CSS 样式

      - 按钮
      - 图片
      - 表格
      - 表单
      - 组件

        - 导航条
        - 分页条

      - 插件

        - 轮播图

- 事件监听机制

  某组件被执行操作后触发代码执行

  例: 一个方法function checkform(){
  return true;
  }
  一个表单
  <form action = "#" id = "form" onclick = "return checkform();" > 方法返回 true 表单提交

  - 组成

- 事件源
  按钮 文本输入匡
- 监听器
  代码
- 注册监听

常见事件

点击事件

- onclick 单击事件
- ondbclick 双击事件

焦点事件

- onblur 元素失去焦点
- onfocus元素获得焦点

加载事件

- onload 页面或图像完成加载

鼠标事件

- 1. onmousedown 鼠标按钮被按下。
- 2. onmouseup 鼠标按键被松开。
- 3. onmousemove 鼠标被移动。
- 4, onmouseover 鼠标移到某元素之上。
- 4.onmouseout 鼠标从某元素移开。

键盘事件

- onkeydown 某个键盘按键被按下。
- onkeyup 某个键盘按键被松开。
- onkeypress 某个键盘按键被按下并松开

选择改变事件

- onchange 域的内容被改变。
- onselect 文本被选中

表单事件

- onsubmit 确认按钮被点击
- onreset 重置按钮被点击

XML

组成

- 1.文档声名
  vension：版有号，必须的属性
  encoding：编码方式
  standalone：是否独立 [yes: 依赖其他文件 No:不依赖]
- 2.指令
  结合CSS
  <?xml-stylesheet type="text/css" href="a.css" ?>
- 3.标签
  自定义
  名称可以包含字母、数字以及其他的字符
  名称不能以数字或者标点符号开始
  名称不能以字母 xml（或者 XML、Xm1等等）开始
  名称不能包含空格
- 4.属性
  id 唯一

5.文本

- CDATA区
  在该区域中的数据会被原样展示
  格式：<！ [CDATA［ 数据门>

约束

DTD

引入

- 内部dtd：将约束规则定义在xml文栏中
- 外部dtd：将约束的规则定义在外部的dtd文件中
  本地：<！ DOCTYPE 根标签名 SYSTEM “dtd文件的位置”>
  网络：<!DOCTyPE 根标签名 PUBLIC “dtd文件名字”“dtd文件的位置URL>

Schema

- 引入：
  填写xml文档的根元素 引入xsi前缀 引入xsd文件命名空间。
  xmlns: xsi="http://www.w3.org/2001/XMLSchema-instance"
  <students
  xmlns: xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xmlns= "http://www.itcast.cn/xml"
  xsi:schemaLocation="http://www.itcast.cn/xml student.xsd>

解析

方式

- DOM
  一次性加载进内存 形成DOM树 CRUD
- SAX
  逐行读取 基于事件驱动 只能读取 不能CRUD

解析器

JAXP:sun公司提供的解析器，支持dom和sax两种思想
DOM4J ： 一款非常优秀的解析器
Jsoup:jsoup 是一款Java的HTML解析器，可直接解析某个URL地址、HTML文本内容。它提供了一套非常省力的API，可通过DOM，CSS以及类似于jquery的操作方法来取出和操作数据。
PULLJ Android操作系统内置的解析器，sax方式的

Jsoup

步骤

```
String path = main.class.getClassLoader().getResource("m.xml").getPath();
    System.out.println(path);
```

```
File file = new File(path);
Document parse = Jsoup.parse(file,"utf-8");
//      System.out.println(parse);//返回整个文档内容

Elements name = parse.getElementsByTag("age");
System.out.println(name.size());
for (int i = 0; i < name.size(); i++) {
    Element element = name.get(i);
    System.out.println(element.text());
}
```

- 1. 导包
- 2.获取Document对象
- 3.获取对应标签Element对象
- 4.获取数据

- 对象
- Jsoup：工具类，可以解析html或xml文档，返回Document
  - parse：解析html或xml文档，返回Document

    parse（File in, String charsetName）：解析xml或htmL文件

    parse(String html) : 解析xml html 字符串

    parse（URL url, int timeoutnillis）：通过网络路径获取指定的html或xml的文档对象
  - Document　文档对象。代表内存中的dom树
  - Elements　元素Element对象的集合

    可以当做 ArrayList<Element>来使用
  - Element　元素对象
    - 1. 获取子元素对象

      方法继承自父类Node
      - getElementById（String id）：根据id属性值获取唯一的element对象
      - getElementsByTag（String tagName）：根据标签名称获取元素对象集合
      - getElementsByAttribute（String key）：根据属性名称获取元素对象集合
      - getElementsByAttributevalue（String key, string value）：根据对应的属性名和属性值获取元素对象集合
    - 2. 获取属性值
      - string attr（String key）：根据属性名称获取属性值
    - 3.　获取文本内容
      - string text（）：获取文本内容
      - String html（）：获取标签体的所有内容（包括字标签的字符串内容）
  - Node　节点对

    是Document和Element的父类
- 快速查询方式
  - selector 选择器
    - 方法:　：Elements select （String　cssQuery） 传入css选择器

      参考Selector类中定义的语法

      Elements elements = document. select ("name") ;

      Elements elements1 = document. select ("#itcast") ;

      document. select ("student [numner='heima_00001']");
  - XPath

    4.1查询所有student标签

    jDocument. selN(xpath: "//student");

    4.2查询所有student标签下的name标签

    List<JXNode> jNodes2 = jDocument.selN(xpath: "//student/name") ;

    4.3查询student标签下带有id属性的name标签

    List<JXNode> jxNodes3 = jxDocument.selN(xpath: "//student/name[@id]");

- Tomcat
  - 网络通信三要素

    端口　协议 IP
  - 部署项目
    - 1. 直接放到webapps目录 或 把war包放到webapps目录
    - 2.server.xml 在host 标签体中 添加

      <Context docBase="D: \hello" path="/hehe" />
    - 3 .在conf/catalina/localhost 创建xml文件,文件名为虚拟路径
  - 动态项目
    - 项目名称
      - - WEB-INF
        - - web.xml：该项目的核心配置文件
          - - classes目录：放置字节码文件
          - - lib目录：放置项目依赖的jar包
- Servlet

  一个接口，定义servlet类被识别的规则

  自己类，实现servlet接口，重写方法
  - web.xml

```
<servlet>
    <servlet-name>demo1</servlet-name>
    < servlet-class>cn.itcast.web.servlet.ServletDemo1</servlet-class>

    <load-on-startup>0</load-on-startup>        可指定Servlet创建时机：0或正数:启动时创建        负数:第一次被访问时创建 默认第一次被访问时创建
</servlet>
<servlet-mapping>
    <servlet-name>demo1</servlet-name>
    <url-pattern>demo1</url-pattern>
</servlet-mapping>
```

  - 原理
    - 当服务器接受到客户端浏览器的请求后，会解析请求URL路径，获取访问的Servlet的资源路径
    - 查找web.xml文件，是否有对应的<url-pattern>标签体内容。
    - 如果有，则在找到对应的<servlet-class>全类名
    - tomcat会将字节码文件加载进内存，并且创建其对象

      将全类名加载进内存 Class.forName()

      创建对象 cls.newInstance();
    - 调用其方法

      service()

方法

- void init(Servlet Config)

  Servlet被创建时执行 只执行一次 ,Servlet 内存中只存在一个对象(Servlet是单例的)

  多个用户同时访问时，可能存在线程安全问题。

  解决：尽量不要在Servlet中定义成员变量。即使定义了成员变量，也不要对其赋值

- ServletConfig getServletConfig()

  获取ServletConfig:Servlet 的配置对象

- void service (ServiceRequest servicerequest, ServletResponse servletresponse)

  提供服务 每一次Servlet被访问时执行 执行多次

- String getServletInfo()

  获取Servlet信息 版本 作者

- destroy()

  服务器正常关闭时 执行一次

Servlet3.0 注解配置

类前注解 @WebServlet("/demo2")

value=路径,仅有一个时value可省略 直接 /路径

- IDEA&Tomcat

  工作空间项目 和 tomcat部署的web项目

  tomcat 访问"tomcat部署的web项目"，"tomcat部署的web项目"对应着"工作空间项目"的web目录下的所有资源

  Web-inf 不能被直接访问

  断点调试:debug模式

体系结构

Servlet 接口

|

GenericServlet 抽象类

|

HttpServlet 抽象类|

- GenericServlet

  将Servlet接口中其他的方法做了默认空实现，只将service（）方法抽象

  将来定义Servlet类时，可以继承GenericServlet，实现service（）方法

HttpServlet

对http协议的一种封装，简化操作

- 1.定义类继承
- 2.重写 doGet doPost 方法

相关配置

- urlpattern Servlet 访问路径

  可定义多个

  @webservlet（{"/d4"，"/dd4"，"/ddd4"}）

  规则:

  1. /xxx

  2./xxx/xxx 多层路径，目录结构

  3.*.do

HTTP

*基于TCP/IP的高级协议

*默认端口号：80

*基于请求/响应模型的：一次请求对应一次响应

*无状态的：每次请求之间相互独立，不能交互数据

历史版本：

1.0：每一次请求响应都会建立新的连接

1.1：复用连

格式

- 1.请求行

  请求方式 请求url 请求协议/版本

  GET /login.html HTTP/1.1

2.请求头

头名称 : 值

常见头

- User-Agent
- Refer

  请求从哪里来

- 3.请求空行

  空行

- 4.请求体

  (post)

Request

请求 http://localhost/day14/demo1 时:

1.tomcat 根据url路径 创建Servlet对象

2.**tomcat创建 request response 对象**,request 封装请求数据

3.tomcat将request response 传给 service方法 并调用

4.可通过request获取请求数据,通过response设置响应

5.服务器在给浏览器响应前会从response中拿设置的响应

- 体系结构

  ServletRequest 接口

  | 继承

  HttpServletRequest 接口

  | 实现

  org.apache.catalina.connector.RequestFacade (tomcat)

方法

获取请求行数据

e.g. GET /day14/demo1?name=zhangsan HTTP/1.1

获取请求方式：

- String getMethod ()

  GET

2. 获取虚拟目录：

- String getContextPath()

  /day14

3. 获取Servlet路径：

- String getServletPath()

  /demo1

- 4. 获取get方式请求参数：
  - String getQueryString()
    name=zhangsan
- **5. 获取请求URI** /day14/demo1
  - String getRequestURI()：
    /day14/demo1
  - StringBuffer getRequestURL( )
    :http://localhost/day14/demo1
- 6. 获取协议及版本：HTTP/1.1
  - String getProtocol（）
- 7. 获取客户机的IP地址：
  - String getRemoteAddr ()

- 获取请求头数据
  - 方法
    - string getHeader（String name）：通过请求头的名称获取请求头的值
    - Enumeration<string> getHeaderNames （）：获取所有的请求头名称
      可按迭代器方式遍历
- 获取请求体数据
  只有POST
  - 1.获取流对象
    - BufferedReader getReader() 获取字符输入流
    - ServletInputStream getInputStream() 获取字节输入流
  - 2.从流对象中拿数据
- 其他
  - 通用方式 : 获取请求参数
    中文乱码问题：
    get方式：tomcat 8 已经将get方式乱码问题解决了
    post方式：会乱码
    解决：在获取参数前，设置request的编码request.setCharacterEncoding（"utf-8"）；
    - 1.String getParameter（String name）：根据参数名称获取参数值
      username=zsan&password=123
    - 2. string[ ] getParametervalues（String name）：根据参数名称获取参数值的数组
      hobby=xx&hobby=game
    - 3. Enumeratlon<String> getparameterNames()：获取所有请求的参数名称
    - 4. Map<string, String []>getParameterMap（）：获取所有参数的map集合
  - 请求转发
    链式 request.getRequestDispatcher (s: "/requestDemo9"). forward (request, response) ;
    特点:地址栏不变,只能转发到当前服务器内部资源,转发是一次请求
    - 1.通过request对象获取请求转发器对象：RequestDispatcher getRequestDispatcher（String path）
    - 2.使用RequestDispatcher对象来进行转发：forward（ServletRequest request, ServletResponse Response）
  - 共享数据
    - 域
      - 域对象：一个有作用范围的对象，可以在范围内共享数据
      - requpest域：代表一次请求的疤围，一般用于请求转发的多个资源中共享数据
    - 方法
      - void setAttribute（String name,object obj）：存储数据
      - Object getAttitude（String name）：通过键获取值
      - void removeAttribute（String name）：通过键移除键值对
  - 获取ServletContext
    - ServletContext getServletContext()

- 案例 登录
  " 用户登录案例需求：
  编写1ogin.html登录页面
  username & password 两个输入框
  使用Druid数据库连接池技术，操作mysql，day14数据库中user表
  使用JdbcTemp-te技术封装JDBC
  登录成功跳转到SuccessServlet展示：登录成功！用户名，欢迎您
  登录失败跳转到FailServlet展示：登录失败，用户名或密码错误
  - Druid&SpringJDBC
    JDBCUtils类： 静态代码块 加载配置文件properties 初始化连接池对象

    ```
    static DataSource dataSource;

       static {
          Properties properties = new Properties();
          InputStream resourceAsStream = JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties");
           properties.load(resourceAsStream);
           dataSource=DruidDataSourceFactory.createDataSource(properties);
           }

    public static DataSource getdataSource(){return dataSource;}
    }
    ```

    UserDao类:定义SQL语句,用SpringJDBC 的JDBCTemplate从Druid获取连接,执行,返回结果

    ```
       public static user login(user u) {

          String sql = "select * from user where admin = ? and pwd = ?";
          JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.getdataSource());
          List<user> query = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(user.class), u.getName(), u.getPwd());
          if (query.isEmpty()){
             return null;
          } else return query.get(0);}
    ```

  - BeanUtils 简化数据封装
    Servlet 类:创建空对象,从request中获取请求参数Map集合.BeanUtils populate方法把参数封装进对象

    ```
    user user1 = new user();
    Map<String, String[]> parameterMap = req.getParameterMap();
       BeanUtils.populate(user1,req.getParameterMap());
    user login = userDao.login(user1);
    ```

- 封装
  必须提供公共setter getter,通过属性进行封装
  属性: 截取setter getter后产物(getName() ->Name -> name 属性)
- 方法
  - setProperty
  - getProperty
  - populate(Object obj,Map map) 将Map键值对封装进Bean对象

Response

数据格式

响应行

协议 版本 状态码 描述

- 状态码
  - 1xx
  - 2xx
  - 3xx
  - 4xx
  - 5xx

响应头

- Content-Type:响应体数据格式 编码格式
- Content-disposition:怎么打开
  - in-line (默认) 当前页面打开
  - attachment 作为附件下载

响应空行

响应体

内容

方法功能

设置响应行

- setStatus(int sc) 设置状态码

设置响应头

- setHeader(String name,String value)

设置响应体

- 获取输出流
  - 字符输出流 PrinterWriter getWriter()
  - 字节输出流 ServletOutputStream getOutputStream()

重定向

方法

- 详细
  - 设置状态码302
    response.setStatus(302)
  - 设置响应头location
    response. setHeader ("location", "/day15/responseDemo2");
- 简便
  response.sendRedirect ("/day15/responseDemo2");

对比转发

- 重定向
  不可用 request.setAttribute 存储信息
  - 地址栏变化
  - 两次请求
  - 可访问其它站点
- 转发
  可用 request.setAttribute 存储信息
  - 地址栏不变
  - 一次请求
  - 只能访问当前服务器资源

路径问题

- 绝对路径
  可确定唯一资源
  如：http://localhost/day15/responseDemo2 /day15/responseDemo2
  以/开头的路径
- 相对路径
  不可确定唯一资源
  规则：找到访问挡前资源和目标资源之间的相对位置关系
  以 ./ 开头 当前
  ../开头 后退一级

最后

- 若路径给客户端用,加虚拟目录
- 若路径给服务器用 不需加虚拟目录

- 动态获取虚拟路径
  方法:
  String contextpath = request.getcontextPath( );

  response.sendRedirect(s: contextPath+"/responseDemo2");

写出字符数据

PrintWriter pw = response getwriter()

pw.write() 和 pw.print()

详尽

通常:write需要手动flush print不需要
Servlet中 write和print都不需要手动flush

- 获取流前 先设置流默认编码
  response. setCharacterEncoding("utf-8");
- 再 告知浏览器服务器响应消息体的编码
  response setHeader ("content-type", "text/html; charset=utf-8");
- 简化写法 直接设置编码 在获取流之前
  response. setContentType("text/html;charset=utf-8");

▼ 写出字节数据

▼ 方法

- 1.获取字节输出流
  ServletOutputStream sos = response-getOutputStream();
- 2.输出
  sos write("hello" getBytes (utf-8));

- 案例 验证码

```
BufferedImage bufferedImage = new BufferedImage(200, 100, BufferedImage.TYPE_INT_RGB);
        Graphics graphics = bufferedImage.getGraphics();
        Color color = 颜色
        graphics.setColor(color);
        graphics.fillRect(0, 0, 200, 100);
         graphics.drawString(str,x,y);
         ImageIO.write(bufferedImage, "jpg", resp.getOutputStream();
```

▼ ServletContext

代表整个web应用可以各容器通信

▼ 获取

两个一模一样

- HttpServlet
  this.getServletContext();
- Request
  request.getServletContext();

▼ 功能

- 获取资源MIME类型
  String filename = "a.jpg";
  String mimeType = context.getMimeType(filename);

▼ 域对象 共享数据

Servletcontext树象范围：所有用户所有请求的数据

- setAttribute(String name, object value)
- getAttribute (String name)
- removeAttribute(String name)

▼ 获取文件真实路径

- String getRealPath（String path)
  web目录下: getRealPath (“a.txt”)
  web-inf 目录下: getRealPath (“/WEB-INF/a.txt”)
  src目录下: context.getRealPath(“/WEB- INF/classes/a.txt”);

▼ 案例 文件下载

- 1.获取请求参数，文件名称
- 2.加载文件进内存

▼ 3.设置response响应头

- 设置响应content-type头 MIME类型
- 设置content-desposition头 attachment;filename=“ ” 打开方式

- 4.输入流写到输出流

▼ 会话 Cookie

在一次会话的范围内的多次请求间，共享数据
原理: 基于响应头set-cookie和请求头cookie实现

- 会话
  一次会话中包含多次请求和响应。
  一次会话：浏器第一次給服多器資源送清求、会活建立、直有一方断开为止

▼ 步骤

- 创建Cookie对象，绑定数据
  new cookie（String name, String value）
- 2. 发送cookie对象
  response.addCookie(Cookie cookie)
  一次可发送多个Cookie,多次调用方法
- 3. 获取Cookie，拿到数据
  Cookie[] request. getcookies () ;

▼ 保存时间

- 默认:浏览器关闭后 数据销毁

▼ 持久化存储

- setMaxAge（int seconds）
  正数：将Cookie数据写到硬盘的文件中。持久化存储。并指定cookie存活时间，时间到后，cookie文件自动失效
  负数：默认值
  零：删除cookie信息

▼ 共享问题

- 同一服务器
  假设在一个tomcat服务器中，部署了多个web项目，那么在这些web项目中cookie能不能共享?
  默认情况下cookie不能共享

  setPath（String path）：设置cookie的获取范围。默认情况下，设置当前的虚拟目录
  如果要共享，则可以将path设置为“/“
- 不同服务器
  不同的tomcat服务器间cookie共享问题?
  setDomain（string path）：如果设置一级域名相同，那么多个服务器之间cookie可以共享 I
  setDomain（*.baidu.com*），那么tieba.baidu.com和news.baidu.com

▼ JSP

java服务器端页面 既可以指定定义html标签，又可以定义java代码

流程：请求jsp;服务器将jsp转换成java文件,java文件被编译成class字节码,字节码提供访问
jsp本质上是个Servlet

- JSP脚本
  定义java代码的方式
  - <%　　%>　在service方法中。service方法中可以定义什么，该脚本中就可以定义什么。
  - <%!　　%>　jsp转换后类的成员位置
  - <%=　　%>　会输出到页面上。输出语句中可以定义什么，该脚本中就可以定义什么。
- 内置对象
  在jsp页面中不需要获取和创建，可以直接使用的对象
  一共九个
  - request
    HttpServletRequest　一次请求访问的多个资源(转发)
  - pageContext
    PageContext　　当前页面共享数据 还可获取其他八个对象
  - Session
    HttpSession　　一次会话的多个请求间
  - application
    ServletContext　所有用户间共享数据
  - response
    HttpServletResponse 响应对象
  - page
    Object　当前页面Servlet的对象　this
  - config
    ServletConfig　Servlet配置对象
  - out
    JspWritter
    字符输出流对象。可以将数据输出到页面上。和response.getwriter（）类似
    区别:
    　　tomcat做出响应前,先找response缓冲区数据,再找out缓冲区数据
    　　所以 response.getWriter输出永远在out前
  - exception
    Throwable　异常对象
- JSP指令
  作用：用于配置JSP页面，导入资源文件
  格式：
  <%@ 指令名称 属性名1=属性值1属性名2=属性值2 ···%>
  - page 配置JSP页面
    content Type:　等同于 response. setContentType()
    　　1.设置响应体的mime类型 字符集
    　　2. 设置当前jsp页面的编码（只能是高级的IDE才能生效，如果使用低级工具，则需要设置pageEncoding属性设置当前页面的字符
    import : 导包
    errorPage：当前页面发生异常后，会自动跳转到指定的错误页面
    　　isErrorPage：标识当前也是是否是错误页面。
    　　　　true：是，可以使用内置对象exception
    　　　　false：否。默认值。不可以使用内置对象exception
  - include 页面包含的。导入页面的资源文件
    <%@ include file="top.jsp"%
  - taglib 导入资源
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jst1/core" %>
    prefix： 前缀，自定义的 c
- 注释
  - html注释 <!— —> 只能注释html
  - jsp注释 <%— —%> 注释所有

会话 Session

服务器端会话技术，在一次会话的多次请求间共享数据，将数据保存在服务器端的对象中。

- Httpsession 对象
  - 方法
    - 获取
      - HttpSession session = request getSession();
    - 使用
      - Object getAttribute(String name)
      - void setAttribute(String name, object value)
      - void removeAttribute (String name)
- 原理
  基于Cookie ,set-cookie:JSESSIONID= 742938a4289,　保证一次会话中多次获取的是同一个
- 问题
  - 当客户端关闭后，服务器不关闭，两次获取session是否为同一个?
    默认情况下。不是。
    如果需要相同，则可以创建Cookie，键为JSESSIONID，设置最大存活时间，让cookie持久化保存。
    Cookie c = new Cookie("JSESSIONID", session getid());
    C. SetMaxAge(60*60);
    response.addCookie(c)
  - 客户端不关闭，服务器关闭后，两次获取的session是同一个吗?
    不是同一个，但是要确保数据不丢失。tomcat自动完成以下工作
    - session的钝化
      在服务器正常关闭之前，将session对象系列化到硬盘上
    - session的活化：
      在服务器启动后，将session文件转化为内存中的session对象即可。
  - session默认失效时间 30分钟
    选择性配置修改
    <session-config>
    <session-timeout>30</session-timeout>
    </sesslon-config>
- 特点
  - session用于存储一次会话的多次请求的数据，存在服务器端
  - session可以存储任意类型，任意大小的数据
- 对比 Cookie

- session存储数据在服务器端，Cookie在客户端
- session没有数据大小限制，Cookie有
- session数据安全，Cookie相对于不安全

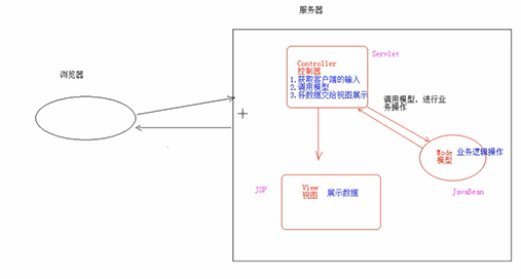
- MVC

  模型　视图　控制器
  Model View Controller

  M: 完成具体的业务操作，如：查询数据库，封装对象
  V: 展示数据
  C: 获取用户的输入　　调用模型　　将数据交给视图进行展示

  -

    jsp演变历史
    1、早期只有servlet，只能使用response输出标签数据，非常麻烦
    2、后来又jSP，简化了Servlet的开发，如果过度使用jSP，在jSp中即写大量的java代码，有写html表，造成难于维护，难于分工协作
    3. 再后来，java的web开发，借鉴mvc开发模式，使得程序的设计更加合理

  -

    优点 耦合性低，方便维护，可以利于分工协作　　重用性高

- El 表达式

  Expression Language 表达式语言
  作用：替换和简化jsp页面中java代码的编写

  - 语法：${表达式}
  - 忽略JSP表达式

    jsp页面默认支持jsp表达式

    - 1.设置page指令中 isELIgnored="true" 忽略当前页面所有表达式
    - 2. \${表达式} 忽略单个表达式
  - 使用
    - 运算
      - 算术运算符
        - +
        - —
        - *
        - /　div
        - %　mod
      - 比较运算符
        - >　<　==　>=　<=　!=
      - 逻辑运算符
        - &&　and
        - ||　or
        - !　not
      - 空运算符
        - empty

          功能：用于判断字符串、集合、数组对象是否为null并目长度是否为0
          ${empty list}
          ${not empty list} 判断字符串、集合、数组对象是否不为null并目长度>0
    - 获取值

      el表达式只能从域对象中获取值
      是通过对象属性获取的 (setName—>Name—>name)
      - 语法
        - 1.${域名称.键名}　从指定域中获取键的值
          - 域名称

            e.g　request域中存储了 name 张三
            ${requestScope.name}　获取
            - pageScope　->　pageContext
            - requestScope　->request
            - sessionScope　-> session
            - applicationScope　-> application (ServletContext)
        - 2.${键名}　依次从最小域中查找是否有该键对应值

          获取对象u的name属性的值
          ${requestScope.u.name}
        - 3.获取 对象、List集合、Map集合的值
          - 对象

            对象：$（域名称.键名.属性名）　本质上会去调用对象的getter方法
          - List

            ${list}
            ${list [1]}
            ${list [10]}
　越界显示空 不报错
          - Map

            域名称.键名.key名
            - ${map. gender}
            - ${map["gender"]}

- 逻辑视图

  User类:
  public class User {
  private int age;
  private Date birthday

  public String getBitStr(){ bean类添加getBitStr()方法,方便可直接从jsp用el表达式获取值
  if (birthday != null){

  SimpleDateFormat sdf = new SimpleDateFormat (pattern: "yYYy-MM-dd Him: s");
  return sdf. format (birthday);
  }
  else return " " ;
  }
  }

- El表达式的隐式对象

  11个

  - pageContext

    获取jsp其他八个内置对象

    - ${pagecontext. request.contextPath} ：动态获取虚拟目录

- JSTL

  Javaserver Pages Tag Library JSP标准标签库 用于简化和替换jsp页面上的java代码

  - 使用
    - 导入jstl相关jar包
    - 引入标签库：taglib指令
    - 使用标签
  - 标签
    - if

      类似java if

      - 属性： test 必须属性，接受boolean表达式

        如果表达式为true，则显示if标签体内容，如果为false，则不显示标签体内容
        一般情况下，test属性值会结合el表达式一起使用
        c:if标签没有else情况，想要else情况，则可以在定义一个c:if标签
    - choose

      类似java switch

      - when标签做判断 相当于case
      - otherwise标签做其他情况的声明相当于default
    - foreach

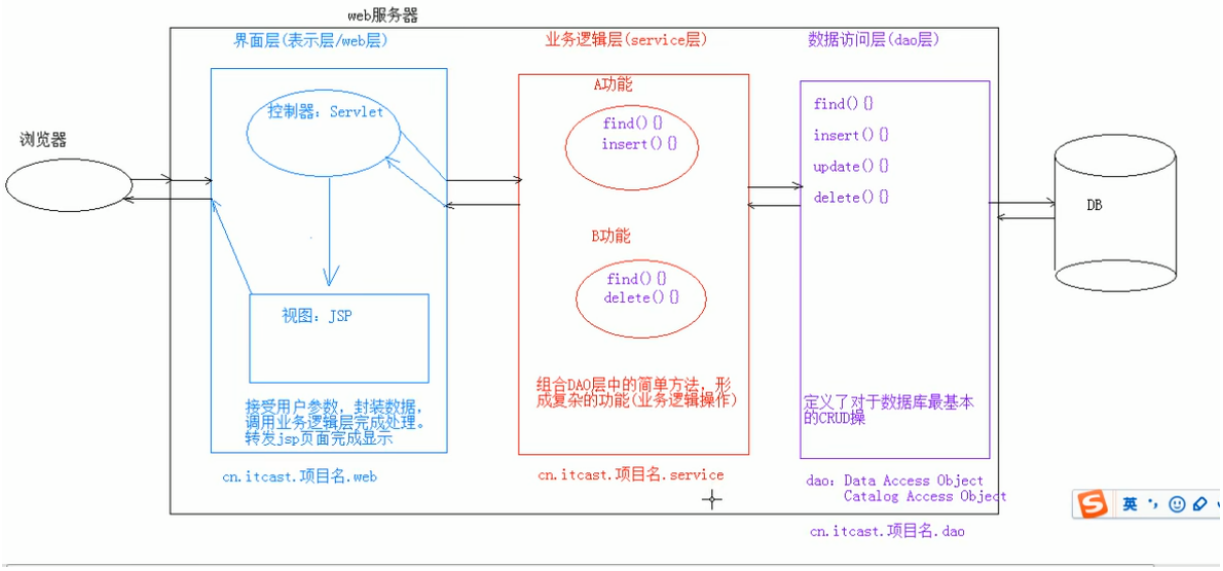
      类似java for

      - 属性：

        begin：开始值 end：结束值 step:步长 var 临时变量
        varstatus：循环状态对象
        index：容器中元素的索引，队开始
        count：循环次数从1开始

        遍历容器
        items: 容器对象
        var：容器中元素的临时变量
        varstatus：循环状态对象
        index：容器中元素的索引，从0开始
        count：循环次数，从1开始

- 三层架构
  - 图片



  - 浏览器
  - web服务器
    - 界面层

      用户看到,通过组件与服务器交互

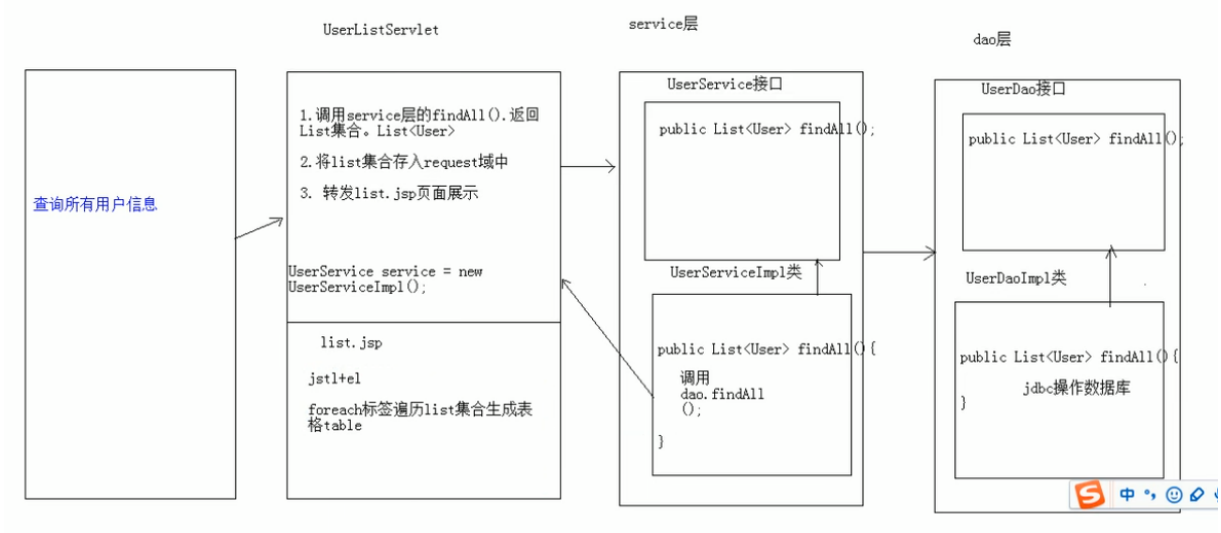
    - 表示层 service

      处理业务逻辑

    - 数据访问层 dao

      操作数据库
  - 数据库

- 案例
  - service层做接口,实现接口的实现类处理逻辑　　三个包：项目名 dao(接口和实现类) service



- Filter 过滤器

  doFilter:放行方法
  - 步骤
    - 1.定义类,实现Filter接口
    - 2.重写方法
    - 3.配置拦截路径
      - web.xml

        ```
        <filter>
        <filter-name>demo1</filter-name>
        <filter-class>den.itcast-er filter FilterDemol</filter-class>
        </filter>
         <filter-mapping>
        <filter-name>demo1</filter-name>
        <url-pattern>/*</url-pattern>
         </filter-mapping>
        ```

      - 注解

        @webFilter（"/*"）　　访问所有资源之前，都会执行该过滤器
  - 流程
    - 执行过滤器
    - 执行放行后资源
    - 执行过滤器放行代码下的代码
  - 生命周期方法
    - init

      服务器启动时创建Filter对象执行 执行一次用于加载资源
    - doFilter

      每次请求被拦截时执行 执行多次
    - destroy

      服务器正常关闭时Filter对象销毁,执行一次
  - 配置拦截路径
    - 路径
      - 1.具体资源路径：/index.jsp

        只有访问index.jsp资源时，过滤器才会被执行
      - 2. 拦截目录：/user/*

        访问/user下的所有资源时，过滤器都会被执行
      - 3.后缀名拦截 *.jsp
      - 拦截所有资源：/*

        访问所有资源时，过滤器都会被执行
    - 配置

      注解: 配置dispacherType 属性　　web.xml:　设置<dispatcher></dispatcher>标签即可
      - REQUEST ： 默认值。浏览器直接请求资源
      - FORWARD：转发访问资源
      - INCLUDE：包含访向贷源
      - ERROR：错误跳转资源
      - ASYNC：异步访问资源
- 过滤器链

  配置多个过滤器　器1->器2->资源执行->器2->器1

  先后顺序：注解:按字符串比较规则

  web.xml:<filter-mapping> 在上的先执行

- 代理模式

  代理对象代理真实对象,以增强真实对象功能
  - 实现方式
    - 静态代理

      有一个类文件描述代理模式

- 动态代理
  在内存中形成代理类
  - 步骤
    - 1.代理对象和真实对象实现相同接口
    - 2.**代理对象 = *proxy.newInstance();***
      - 方法的三个参数
        - 类加载器:真实对象.getClass().getClassLoader();
        - 接口数组:真实对象.getClass.Interfaces()
        - 处理器:new InvocationHandler()
          - new 接口匿名内部类, 实现方法 Object invoke()
            代理对象实现的所有方法都会触发方法执行 返回值就是方法增强后的返回值,不写逻辑增强返回原值
            - 方法三个参数
              - Object proxy : 代理对象
              - Method method　代理对象调用的方法被封装成的对象
                method.getName();　方法名　　method.invoke(原对象实例,参数)
              - Object[ ] arg　　代理对象调用方法时传递的参数
    - 3.代理对象调用方法
    - 4.增强方法
      - 增强方式
        - 增强参数列表
        - 增强返回值类型
        - 增强方法体执行逻辑
        - *Example*
          - 1.
            ```
            //判断是否是sale方法
            if(method.getName().equals("sale")){
                //1.增强参数
                double money = (double) args[0];
                money = money * 0.85;
                //使用真实对象调用该方法
                Object obj = method.invoke(lenovo, money);
                return obj;
            }else{
                Object obj = method.invoke(lenovo, args);
                return obj;
            }
            ```
          - 2.Filter 代理 实现过滤敏感词
            SeryletRequest prory reg = (ServletRequest) Proxy.newerorynstance(rea.setclass().getclassLoader(), rea.getclass().getInterface　获取代理对象
            判断方法名,method.invoke(原对象实例,参数),获取返回值,replaceall后return

- Listener 监听器
  - 事件监听机制
    - 事件
    - 事件源
      事件发生地方
    - 监听器
      一个对象
    - 注册监听
      将事件、事件源、监听器绑定在一起。当事件源上发生某个事件后，执行监听器代码
  - ServletContextListener
    监听ServletContext对象 创建 销毁
    - 方法
      void contextDestroyed（Servletcontextevent sce）： Servletcontext对象被销毁之前会调用该方法
      void contextInitialized（ServletContextEvent sce）： ServletContext对象创建后会调用该方法
    - 步骤
      - 定义一个类，实现ServletContextListener接口
      - 重写方法
      - 配置
        ```
        <context-param>
            <param-name>contextConfiguration</param-name>
            <param-value>/</param-value>
        </context-param>
        ```
- jQuery
  - jQuery对象 和 Js对象
    jQuery 对象　->js对象:　　jquery对象[索引]　或　jquery对象.get(索引
    js对象－>jquery对象:　　$(js对象)
  - 事件绑定　入口函数
    $('#b1').click(function(

    ));

    入口函数　: dom加载后再执行里面代码
    $(function(

    ))
    对比 window.onload:　　window.onload只能定义一次,定义多次,后面会覆盖前面
    $(function()
)　　可以定义多次
  - 选择器
    - 基本选择器
      - 标签选择器
        $("标签名")

- id选择器
  $("id")
- 属性选择器
  $(.class)
- 并集选择器
  $("选择器1","选择器2")

层级选择器

- 后代选择器
  $(A B) A内所有子孙 B 元素
- 子选择器
  $(A>B) A内所有直接儿子B元素

属性选择器

- 属性名称选择器
  $("A[属性名]") 选择包含指定属性的元素
- 属性值选择器
  $("A[属性名='值']") 选择指定属性=值的元素 $("A[属性名!='值']") 选择指定属性不等于值和不包含属性的元素
- 复合属性选择器
  $("A[属性名=值][ ]……") 选择包含多个属性的元素

过滤选择器

- 首元素选择器
  .first 获取选择的元素中第一个元素
- 尾元素选择器
  .last 获取选择的元素中最尾一个元素
- 非元素选择器
  :not(Selecter) 不包括指定内容的元素
- 偶数选择器
  :even 从0起计
- 奇数选择器
  :odd 从0起计
- 等于索引选择器
  :eq(index)
- 大于索引选择器
  :gt(index)
- 小于索引选择器
  :lt(index)
- 标题索引选择器
  :header 选择 h1-h6元素

表单过滤选择器

e.g.
$("input [type= 'text ']: enabled") .val("aaa");
$("input [type='text']: disabled"). val ("aaa");
$("input [type=' checkbox']: checked") .length);
$("#job: checked") .length);
$("#job › option:selected"). length);

- 可用元素选择器
  :enabled
- 不可用元素选择器
  :disabled
- 选中选择器 单复选
  :checked
- 选中选择器 下拉框
  :selected

DOM 操作

内容操作

- html()
  获取/设置元素的标签体内容 <a>PA --› Pa
  获取 : var a = $("#myinput"). val() 修改: $("#myinput"). val ("李四");
- text()
  获取/设置元素的标签体纯文本内容 <a>font>内容›/a> -→> 内容
  同上
- val()
  获取/设置元素的value属性值
  同上

属性操作

通用属性操作

操作元素固有属性: prop
操作自定义属性: attr
checked selected disabled只用 prop

- attr()
  获取/设置元素属性
- removeAttr()
  移除元素属性
- prop()
  获取/设置元素属性
- removeProp()
  移除元素属性

class属性操作

- addClass()
  添加class属性
- removeClass()
  删除class属性
- toggleClass()
  切换class属性
  e.g.

toggleclass("one"):
判断如果元素对象上存在class="one"，则将属性值one删除掉。 如果元素对象上不存在class="one"，则添加

- .css()
  $ ("#one"). css ("backgroundColor", "green");

CURD 操作

父子 追加

- append（）：父元素将子元素追加到末尾
  对象1.append（对象2）；将对象2添加到对象1元素内部，并且在末尾
- prepend（）：父元素将子元素追加到开头
  对象1.prepend（对象2）：将对象2添加到对象1元素内部、并目在形
- appendTo():
  对象1.appendTo（对象2）：将对象1添加到对象2内部，并且在末尾
- prependTo():
  对象1.prependTo（对象2）：将对象1添加到对象2内部，并且在开头

兄弟 追加

- after（）：添加元素到元素后边
  对象1.after（对象2）：将对象2添加到对象1后边。对象1和对象2是兄弟关系
- before（）：添加元素到元素前边
  对象1.before（对象2）：将对象2添加到对象1前边。对象1和对象2是兄弟关系
- insertAfter（）
  对象1.insertAfter（对象2）：将对象1添加到对2后边。对象1和对象2是兄弟关系
- insertBefore（）
  对象1.insertBefore（对象2）：将对象1添加到对象2前边。对象1和对象2是兄弟关

- remove（）
  对象.remove（）：将对象删除掉
- empty（）
  对象.empty（）　清空元素的所有后代元素。
- clone()
  克隆并选中元素
  $("word"). append ($(this).clone());

动画操作

显示 隐藏元素

参数:
speed:速度 slow normal fast 或表示动画时长毫秒值(e.g. 1000)
easing:切换效果
swing: 慢ー>快ー>慢
linear:匀速
fn:动画完成执行函数,每个元素执行一次

- 默认
  show([speed, [easing], [fn]])
  hide ([speed, [easing], [fn])
  toggle([speed], [easing], [fn])
- 滑动
  slideDown ([speed], [easing], [fn])
  slideUp ([speed, [easing], [fn])
  slideToggle([speed], [easing], [fn])
- 淡入淡出
  fadeln([speed], [easing], [fn])
  fadeOut ([speed], [easing], [fn])
  fadeToggle([speed, [easing], [fn])

遍历

- 1.js方式
  for(初始化值;条件;步长)
- 2.jQuery方式 jquery 对象.each(function(index索引,element元素对象)
  jquery 对象.each(function(index索引,element元素对象){

  if 条件{
  return false; }
  })
  返回false 跳出循环(break)
  返回true 跳出本次 进行下次(continue)
- 3.jQuery 方式 $.each(对象,函数)
  对象也可以是js数组
- 4.for 元素对象 of 容器对象
  jQuery3.0后支持
  e.g. for (li of citys){ }

事件绑定

- 1. 标准方式 jq对象.事件方法（回调函数）
  可以链式编程
  e.g. $("#1").mouseover(function).mouseout(function)
- 2. jq对象.on 绑定 / jq对象.off 解除绑定
  jq对象.on（"事件名称"，回调函数）
  jq对象.off（"事件名称"）
  jq对象.off（）解绑全部事件
- 3.事件切换
  jq对象.toggle（fn1,fn2...）
  高版本不可用

插件机制

- $.fn.extend(object)
  增强通过jquery获取的对象的功能

  e.g.
  $.fn.extend ({
  check:function(){
  方法体
  },
  })
- $.extend(object)
  增强jQuery自身功能

  e.g.
  $.extend({
  max:function(a,b){

```
            return……
                },
        min:function(a,b){

            return……
        }
    })
```

- Ajax

  异步的Javascript 和 XML

  - 实现方式

    - 原生JS

      - 先创建核心对象

        xmlhttp=new XMLHttpRequest();　　(IE6: xmlhttp=new ActiveXObject ("Microsoft. XMLHTTP"); 用if (window. XMLHttpRequest) 判断)

      - 建立连接

        xmlhttp. open ("GET", "test1.txt", true);　//请求方式　(get:send空参,参数在url后拼接　post:参数在send方法)　　url　　同步(false) 异步(true)

      - 发送请求

        xmlhttp.send()

      - 处理响应

        ```
        xmlhttp.onreadystatechange=function(){
            if (xmlhttp.readyState==4 && xmlhttp.status==200){
            document-getElementById…….}}

        readystate:XMLHttprequest 状态
            0：请求未初始化
            1. 服务器连接已建立
            2.请求已接收
            3.请求处理中
            4：请求已完成，且响应已就绪
        ```

    - JQuery

      - $.ajax( )

        ```
        $.ajax({
            url:"ajaxServlet"
            type:"post"
            data:{"name":"zs"},
        success:function(){                    //  成功回调函数

            },

        error:function(){                    //出错回调函数
        },
        dataType:"text"  }),                //设置接收到的响应数据格式
        ```

      - $.get(url ,[data] , [callback],[type])

        - url:请求url
        - data:请求参数
        - callback:回调函数
        - type : 响应类型

      - $.post()

- JSON

  var person = {"name": "张=", age: 23, 'gender': true};

  - 基本

    键值对构成的　　键用引号（单双都行）引起来，也可以不使用引号

    - 值 类型

      - 数字
      - 字符串
      - 逻辑

        true false

      - 数组

        [,]

      - 对象

        {"address": {"name":21,"age",22}}

      - null

    - 获取数据

      - json对象.键名
      - json对象["键名"]
      - 数组对象[索引]

  - JSON与Java对象互转

    - Java对象—> Json

      - Jackson包　方法

        - writeValue(参数1,obj)

          ```
          参数1:  obj转换为json字符串,保存到文件,写到字节(符)  流中
                  File
                  writer
                  OutputStream
          ```

        - writeValueAsString(obj)

          obj转换为json字符串

      - 注解:

        - @JsonIgnore 排除属性
        - @JsonFormart 属性值格式化

          e.g. JsonFormart(pattern = "yyyy-MM-dd")

- Redis

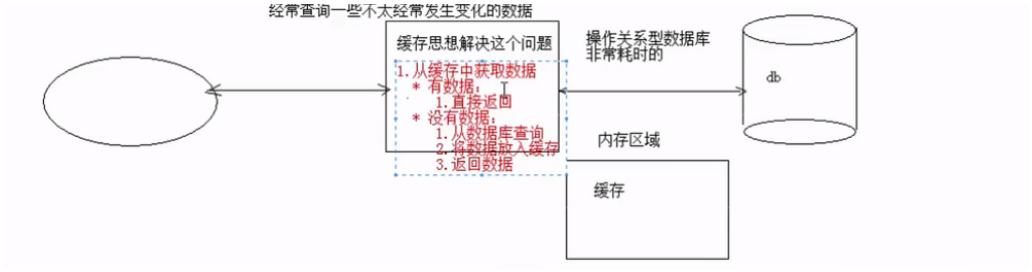
  非关系型数据库 NoSQL　　存储 key:value

  数据间没有关系　存储在内存中

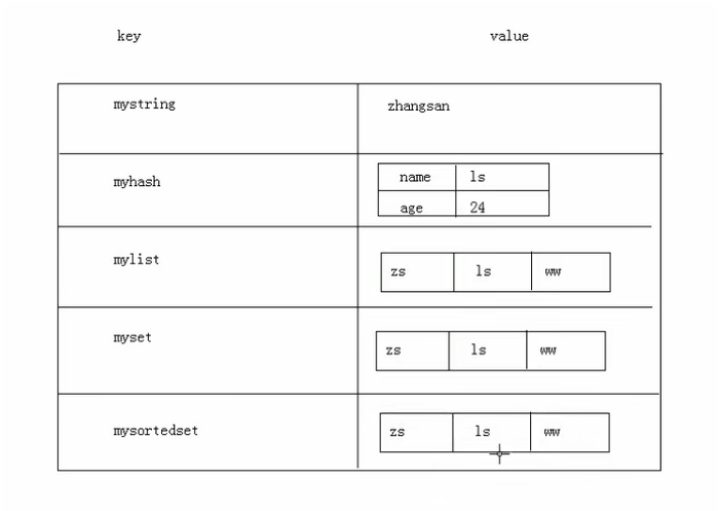
  - 缓存思想

    ```
    从缓存中获取数据
        有:返回
        没有:
          1.数据库查询
          2.放入缓存(redis)
          3.返回
    ```

    - 图



- Redis 的 数据结构&命令
  - key: 字符串
  - value:
    - 图片



    - 1、字符串类型 string
      - 存储
        set key value
      - 获取
        get key
      - 删除
        del key
    - 2、哈希类型 hash
      类似map
      - 存储
        hset *key field value*
      - 获取
        - 获取指定field对应值
          hget *key filed*
        - 获取所有 filed value
          hget all key
      - 删除
        hdel *key field*
    - 3、列表类型 list
      类似队列 ————按插入顺序 添加到头部或尾部
      - 添加
        lpush key value 从左边添加
        rpush key vaue 从右边添加
      - 获取
        lrange key *start end* 范围
      - 删除
        从左(右)删除并返回元素
        lpop key value
        rpop key value
    - 4、集合类型 set
      无序 不允许重复
      - 添加
        sadd key value
      - 获取
        smembers key 获取所有元素
      - 删除
        srem key value
    - 5、有序集合类型 sortedset
      不允许重复 有序(根据score 排序)
      - 添加
        zadd *key score value*

- 获取

  zrange *key start end*

  e.g.

  zrange mysort 0 -1 (索引0 -1 表示所有)

  zrange mysort 0 -1 withscores (带上scores)

- 删除

  zrem key value

- 通用命令
  - keys* 查询所有键
  - type *key* 查询键对应类型
  - del key 删除指定 key : value
- 持久化

  内存数据 持久保存到硬盘
  - RDB

    默认;

    一定 时间间隔,检测key变化情况,持久化写入

    conf文件

    save 900 1 900s若有1个key改变,写入一次

    save 300 10 300s若有10个key改变,写入一次

    save 60 10000 60s若有10000个key改变,写入一次
  - AOF

    日志式方式,可以每一次操作后 持久化一次

    conf文件

    appendonly no（关闭aof） --> appendonly yes (开启aof)

    # appendfsync always 每一次操作都进行持久化

    #appendfsync everysec 每隔一秒进行一次持久化

    # appendfsync no 不进行持久化
- Jedis
  - 基本
    - 1. new 对象

      空参:默认localhost 端口 6379
    - 2.操作
      - string
        - set (键,值)
        - setex(键,过期时间秒,值)

          存储指定过期时间的 键值 过期后自动删除
      - list
        - lpush rpush (key,string……)

          存储list
        - lrange(key,起,尾) 返回list

          范围 0,-1 表示所有
      - set
        - sadd(key,…….members)
        - smembers(key) 返回set
    - 3.close 关闭连接
  - 连接池
    - 1.创建配置对象

      new JedisPoolConfig( )

      可用config对象.set参数

      e.g. jedis.setMaxTotal(整数)

      若用properties配置文件: 创建Properties对象

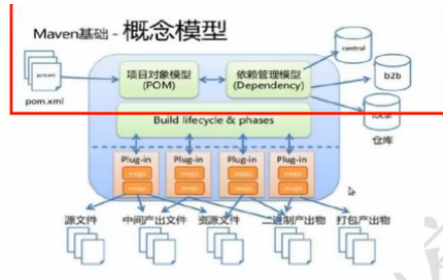
      获得输入流(可用类加载器,AsSream)

      properties load 输入流 properties对象.getProperties(key) 获得值 ，用config.set
    - 2.创建连接池对象

      new JedisPool(config对象,主机名,端口)
    - 3.获取连接

      jedispool对象.getDataResource() 返回jedis对象
    - 4.归还连接给连接池

      jedis.close()
- Maven
  - 概念模型
    - 图片

  - settings.xml 配置文件

    指定本地仓库: localRepository 标签内写路径
  - 项目目录结构
    - src
      - main
        - java
        - resources 放配置文件

- webapp (仅web项目)

- test
  - java
  - resources
- pom.xml

- 生命周期

  执行后面的操作会自动执行剪的所有操作

  - 编译
    - 添加插件

      build plugins plugin 下 配置坐标(groupid,artificialid,version,configuration(下指定 配置))

      tomcat插件:configuration下可指定 port 端口
      path 虚拟目录
    - 依赖
      - 作用域

        坐标下面指定 scope 标签

        - comple (默认)

          编译 测试 运行 均有效
        - test

          只有测试(编译、运行)有效   e.g. JUnit
        - provided

          编译 测试有效 运行无效   e.g.jsp-api   servlet-api ,防止和tomcat自带冲突
        - runtime

          编译无效 测试 运行 有效 e.g. JDBC 驱动
        - system
      - 图片

依赖范围	对于编译classpath有效	对于测试classpath有效	对于运行时classpath有效	例子
compile	Y	Y	Y	spring-core
test	-	Y	-	Junit
provided	Y	Y	-	servlet-api
runtime	-	Y	Y	JDBC驱动
system	Y	Y	-	本地的，Maven仓库之外的类库
    - 测试
    - 打包

      pom.xml 中 标签 （packaging）jar（/packaging） 放在上面
      - 方式
        - jar
        - war
        - pom
    - 安装