复习笔记

一,HTML

1.概念

超文本标记语言

2.常见标签

2.1表格

```
<table>
    <tr>
        <td>内容</td>
    </tr>
</table>
```

2.2表单

- 作用: 把用户输入的数据提交到服务器
- get/post区别(面试题)

  get请求请求参数在请求路径后面, post请求请求参数在请求体

  get请求对请求参数大小是有限制的, post请求对请求参数大小没有限制

  get请求相对不安全, post请求相对安全一些

```
<form  action="提交路径" method="post">
    //1. input: text, password, radio, checkbox, submit, hidden, file, button
    <input type="" name=""/>
    //2. select: 选择列表
    <select name="">
        <option value=""></option>
    </select>
    //3. textarea
    <textarea rows="" cols="" name=""></textarea>
    <inpu type="submit"/>
</form>
```

二,JS和JQ

1,概念

JS: 通过浏览器解释的一种脚本语言

JQ: 本质就是JS, 把常见的操作封装起来了, 方便开发

2,JS常用

2.1 事件

- 说白了就是设置标签的属性

```
onclick: 单击事件
onfocus: 获得焦点
onblur: 失去焦点
onchange: 内容改变
onkeydown/up

eg:
<input type="button" onclick="sayHello()" value="sayHello" />
<script>
        //函数语法: function 函数名(参数列表){//函数体}

        function sayHello(){
            alert("hello...");
        }

</script>
```

2.2 函数

- 语法

```
function 函数名(参数列表){
    //函数体
}
eg:
    <script>
        function getSum(i,j){//参数列表不需要加var
            return i+j;
        }

        var sum =  getSum(10,20);
        console.log("sum="+sum);
    </script>

    //1. 格式是固定的(不管有没有返回值)
    //2. 参数不需要写类型,直接写变量名
    //3. 没有重载
```

2.3 操作文档

- innerHTML: 1. 支持标签的(解析标签) 2. 会把之前的内容覆盖　实际开发用途: 添加标签 清空内容
- document.getElementById(): 获得标签元素

2.4 BOM

- setInterval("show()",3000); 周期执行 setTimeout("show()",3000); 执行一次

- location.href ="路径"; 在js里面请求服务器(get方式, 同步,写在script)
- alert(); 警告框
- confirm(); 确定框

2.5 JSON

- 概念: 一个特殊格式字符串,通常用json作为数据的交换. 本来是JS里面的, 其它的平台(Java,C++,Python...)发现JSON数据比较好用, 借鉴这个格式作为数据交换了
- JSON格式

```
//1. JSON对象 {key:value,key:value}
        以key:value形式存在的   多个之间用,隔开
        key通常是字符串
        value可以是任何合法的数据类型

//2. JSON数组 []
        [], 里面可以放不同类型的数据

//3. JSON和JSON数组的组合
```

3.JQ常用

3.1 选择器

- 获得标签对象

```
id选择器: $("#id名");  id名其实就是标签的id属性值
类选择器: $(".类名");  类名其实就是标签的class属性值
标签选择器: $("标签名");
```

3.2操作文档

```
append(标签字符串); 添加孩子, 只会添加,不会覆盖
html("标签字符串"); 会覆盖

remove(); 移除

after();
before();
```

3.3 事件

- 把事件封装成了方法

```
语法: jq对象.事件方法名(function(){

      });
     eg: click(); 点击
        focus(); 获得焦点
        blur(); 失去焦点
        change(); 内容改变
        ....
```

3.4 遍历对象

- 语法

```
jq对象.each(function(i,j){
  //第一个参数i就是下标，第二个参数j下标对应的值
});
```

三, mysql多表

0. 表关系

1. 一对多(类别和商品, 用户和银行卡, 用户和订单)

   在多方创建一个字段作为外键, 指向一方的主键.

2. 多对多,本质就是两个一对多(学生和课程,订单和商品, 用户和角色)

   创建一张中间表,这个表里面至少包含两个字段, 这两个字段都作为外键,分别指向各自的主键

3. 一对一

1.查询

```
select [*][列,列...][聚合函数] from 表名  [where 条件] [group by 列][having 条件][oder by 列 desc/asc][limit a ,b]
```

2, 连接

- 交叉查询(笛卡尔积)

```
select * from 表A, 表B
```

- 内连接查询

```
---隐式的
select * from 表A, 表B where A.主键 = B.外键
---显式的
select * from 表A inner join 表B on A.主键 = B.外键
```

- 外连接

```
-- 左外连接(以左边的表为主表，查询左边表的所有，通过on后面的条件匹配出右边表的数据，如果满足条件就展示；不满足,就通过null来代替)
select * from 表A left outer join 表B on A.主键 = B.外键
```

3. 子查询

- 在oracle接着讲

四, JDBC开发步骤

- 注册驱动
- 获得连接
- 创建预编译sql语句对象
- 设置参数, 执行sql语句
- 处理结果
- 释放资源

四, xml

1. xml定义

可扩展标记语言, 标签随便定义

2, 解析

- dom4J+xpath(下次课扩展IOC的时候复习)

五, 反射

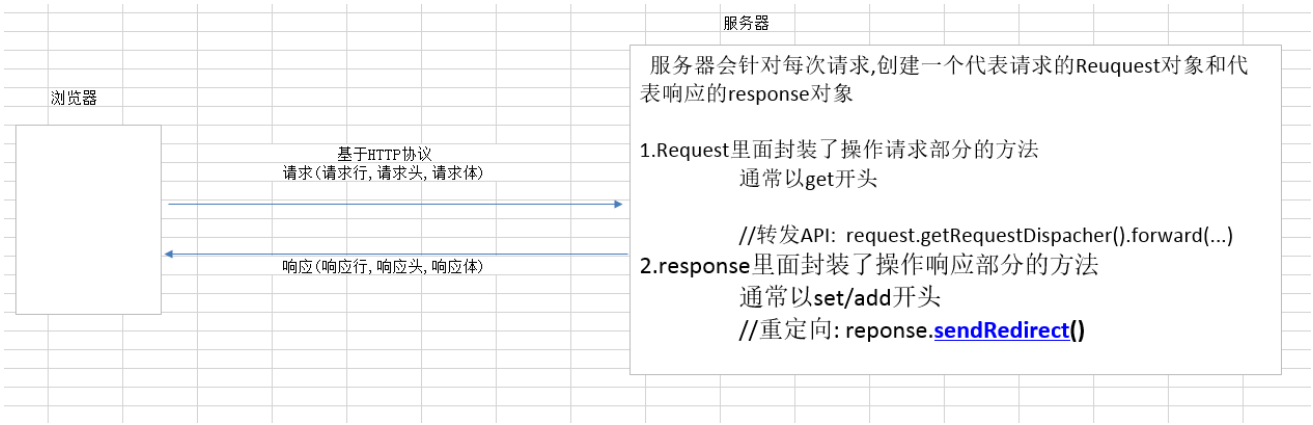
1. 反射原理

反射其实就是把java类的组成成分(字段, 方法) 映射成对应的java对象(File类型对象,Method方法对象)

2. 反射

- 获得字节码, 通过字节码创建对象(根据无参构造)
- 反射字段
- 反射成员方法

六, 请求响应机制



1.转发和重定向

1.1 区别

- 转发是一次请求, 重定向两次请求
- 转发地址栏不变, 重定向地址栏变化
- 转发只能转发内部的资源,重定向可以到外部(内部)资源
- 转发request域存的数据有效的, 重定向request域存的数据无效的,

1.2 选择

- 如果对应的操作是 select , 通常用转发
- 如果对应的操作是insert , delete, update , 通常用重定向

2. 三大域对象

域对象类型	创建时机	销毁时机	作用范围	应用场景
request	来了请求	请求结束	一次请求	转发里面存数据
session	调用getSession()	1.异常关闭服务器 2. 调用invalidate() 3.默认30分钟过期	会话(多次请求)	保存登录状态,保存用户各自数据(购物车)
Application/ServletContext	服务器启动	服务器正常关闭	整个应用	统计网站访问次数

- 访问Servlet, 不一定会创建session(只有当代码里面调用了getSession()方法时候); 访问JSP, 一定会创建session
- eg: 把我们班当做项目, 把每一位同学当前Servlet, 把班主任当做ServletContext

3. Cookie和Session

3.1 概述

会话技术. 保存用户各自(以浏览器为单位)的数据

3.2 选择

- 如果保存的数据很重要, 保存是对象,保存用户的登录状态, 一般用session(); 应用场景: 购物车, 保存对象重定向使用
- 如果保存的数据不大, 数据需要时长的支持, 一般用cookie; 应用场景: 保存用户名 ,购物车(cookie+redis)

4.EL和JSTL(用的不算特别多)

4.1 EL和JSTL概念

EL: 表达式 语法: ${} 作用: 1.获得域对象里面存的数据 2. 执行运算 3.获得web开发对象

JSTL: 标签库; 核心标签库

EL和JSTL有什么关系?

没有关系. 因为JSTL里面需要数据, 这些数据通常是从域里面获得, 因为从域里面或者数据我们通常用el

4.2 EL表达式

- 获得数据(域里面存的数据)

```
//一,获得简单数据(eg:字符串)
request.setAttribute("str","哈哈哈");
${str}

//二, 获得复杂数据
//2.1 数组
int[] array = {1,2,3,4,5};
session.setAttribute("a",array);
获得第二个数据2: ${a[1]}

//2.2 List
list.add("aaa");
list.add("bbb");
list.add("ccc");
session.setAttribute("l",list);
获得第三个数据ccc: ${l[2]}

//2.3 map
map.put("akey","aaa");
map.put("bkey","bbb");
map.put("ckey","ccc");
session.setAttribute("m",map);
获得ckey对应数据ccc: ${m.ckey} 或者 ${m['ckey']}

//2.4 获得javaBean数据
user.setUname("张三");  getUname()
user.setAge(18);
request.setAttribute("u",user);

获得用户名: ${u.uname}---> 依赖的javaBean属性, 依赖 getUname();
```

七, Ajax

1. Ajax概念

Ajax= 异步的xml+js

- 异步的请求, 局部刷新

2. 使用JQ的Ajax

```
$.get(请求路径, 请求参数, 回调函数(得到结果), 结果的类型(默认是字符串));
$.post(请求路径, 请求参数, 回调函数(得到结果), 结果的类型(默认是字符串));
----------------------------------------------------------------
$.ajax([setting]);
```

八, Listener

- ServletContextListener: 监听ServletContext的创建和销毁的; 换句话说:其实就是监听服务器启动(项目部署)和服务器关闭; Spring里面就有这个监听器

九,Filter过滤器

1. Filter概念和作用

Filter也是运行在服务器端的程序, 只不过在达到目标资源(html,jsp,Servlet)之前执行.

作用: 拦截或者放行 当做保安

2.Filter生命周期

- 创建--->init()方法: 服务器启动(项目部署)就会创建
- 拦截-->doFilter()方法 来了一次请求就会执行一次(前提是路径相吻合)
- 销毁-->destory()方法

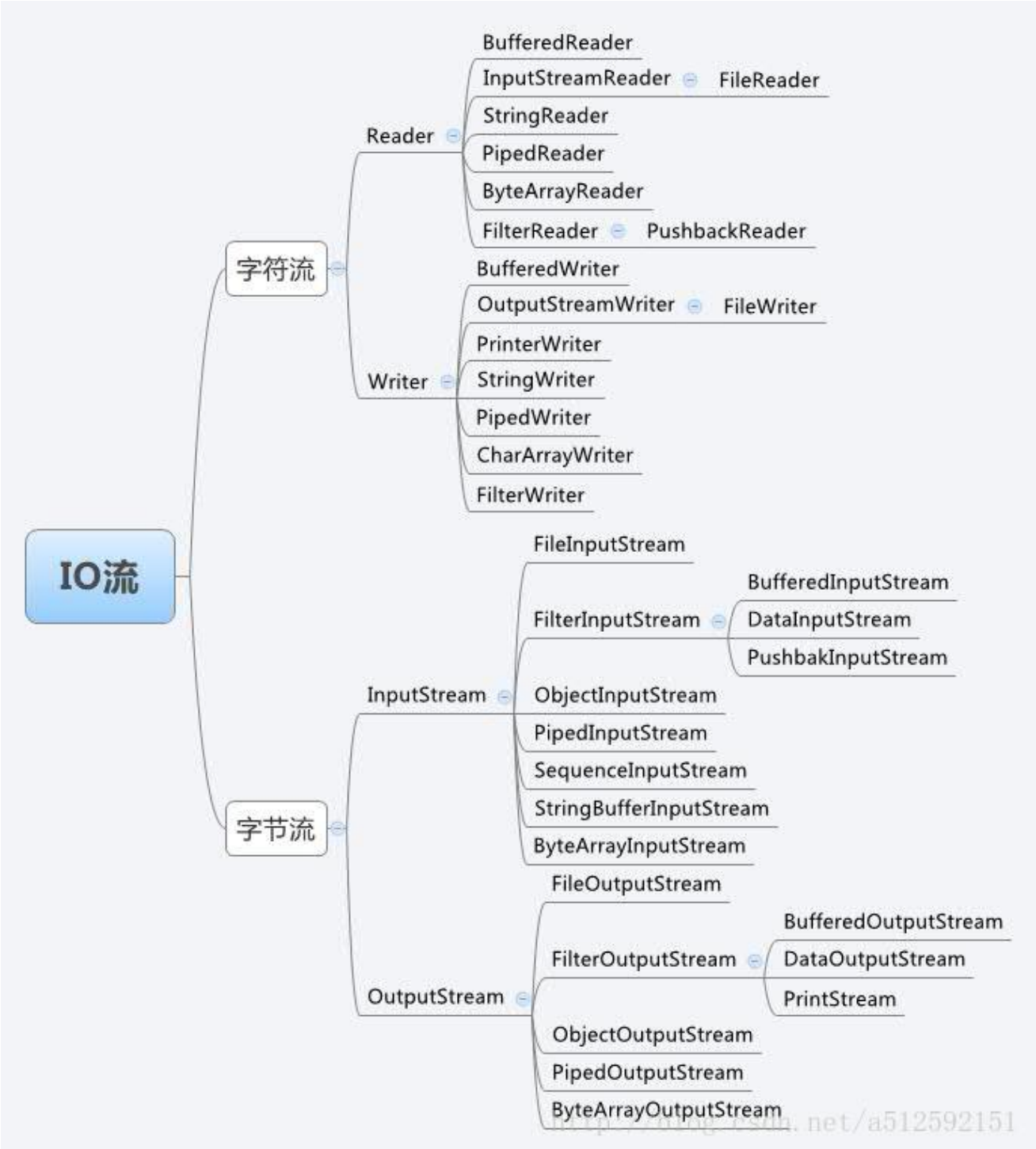
3.Servlet生命周期

默认情况下, 来了第一次请求就会创建Servlet, 每一次请求都会获得一个线程调用service()方法处理这个请求,

服务器正常关闭销毁

单例多线程的

十,IO流



0.相关的概念

- 字节（Byte）是计算机用于计量存储容量的一种计量单位

  1T =1024G 1G = 1024M 1M = 1024KB 1kb = 1024字节

- 字符(Char):字符是指计算机中使用的文字和符号

  字节和字符是两个完全不同的概念. 不同的编码下,同样的字符占的空间大小(字节)是不一样的

1.常见的输入流(读)

类型	特点	所属流
FileInputStream	读取文件的.	字节流
BufferedReader	读取一行一行的读取,效率高.	字符流

- 使用FileInputStream读取文件

```
public void fun01() throws Exception{
        //1.创建输入流
        InputStream is = new FileInputStream("E:/data/Desktop/a.txt");
        //2.开始读取
        byte[] b = new byte[1024];
        int len = 0;
        while( (len = is.read(b)) != -1){
            String str = new String(b, 0, len);
            System.out.println("str="+str);
        }

        //3.关流
        is.close();

    }
```

- 使用BufferedReader读取文件

```
    @Test
    public void fun02() throws Exception{
        //1.创建输入流
        InputStream is = new FileInputStream("E:/data/Desktop/a.txt");
        //2.开始读取
        BufferedReader reader = new BufferedReader(new InputStreamReader(is));
        String line = null;

        while((line = reader.readLine()) != null){
            System.out.println("line="+line);
        }


        //3.关流
        is.close();
        reader.close();

    }
```

2.常见的输出流(写)

类型	特点	所属流
FileOutputStream	写文件(byte)	字节流
BufferedWritter	写的是字符串	字符流

- 使用FileOutputStream写文件

```
    @Test
    public void fun01() throws Exception{
        OutputStream os = new FileOutputStream("E:/data/Desktop/a.txt");
        String str = "hello...";
        byte[] bytes = str.getBytes();
        os.write(bytes, 0, bytes.length);
        os.flush();
        os.close();

    }
```

- 使用BufferedWritter写文件

```
@Test
    public void fun02() throws Exception{
        OutputStream os = new FileOutputStream("E:/data/Desktop/a.txt");
        BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os));
        String str = "你好";
        writer.write(str);
        writer.close();

    }
```

3,文件的拷贝

- 文件的拷贝通常用字节流

```
    @Test
    public void fun01() throws Exception{
        InputStream is = new FileInputStream("E:/data/Desktop/a.jpg");
        OutputStream os = new FileOutputStream("G:/b.jpg");

        byte[] b = new byte[1024];
        int len = 0;
        while( (len = is.read(b)) != -1){
            os.write(b, 0, len);
        }


        is.close();
        os.close();
```

```
    }
```

十一,线程

1.概念

1.1进程和线程概述

进程：正在运行的应用程序 软件

单线程：程序的一条执行路径

多线程：程序的多条执行路径

1.2进程和线程关系

一个进程至少有一个线程 主线程

一个线程必须在一个进程之内

EG: 酷狗这个进程：

一条执行路径用来显示界面

还有一条执行路径用来播放音乐

还有一条执行路径用来显示歌词

2.创建线程的方式

2.1方式一:继承Thread 单继承

2.2方式二:实现runable接口 实现多个接口

3.多线程卖票例子

- EG

```
package com.lcuyp.thread;

public class WindowSale02 {

    public static void main(String[] args) {

        TickedRunable02 tickedRunable = new TickedRunable02();

        Thread t1 = new Thread(tickedRunable);
        Thread t2 = new Thread(tickedRunable);
        Thread t3 = new Thread(tickedRunable);
        Thread t4 = new Thread(tickedRunable);


        t1.setName("1号窗口");
```

```
        t2.setName("2号窗口");
        t3.setName("3号窗口");
        t4.setName("4号窗口");

        t1.start();
        t2.start();
        t3.start();
        t4.start();

    }

}

class TickedRunable02 implements Runnable {

    private int ticked = 200;

    @Override
    public void run() {
        while (true) {
            synchronized (this) {

                if (ticked > 0) {
                    try {
                        // t1睡了
                        // t2睡了
                        // t3睡了
                        // t4睡了
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    // t1醒了 t1卖1号票
                    // t2醒了 t2卖0号票
                    // t3醒了 t3卖-1号票
                    // t4醒了 t4卖-2号票

                    System.out.println(Thread.currentThread().getName() + "卖" + ticked-- +
"号票");

                    // 相同的票:
                    // ticked--:
                    // a.读取ticked的值 t1 80 t2 80
                    // b. 修改ticked的值
                    // c. 把修改后的赋值给ticked
                }
            }
        }

    }

}
```