# 第一天：字节流，字符流 ，高效流，序列化

字节流序列化注意事项：

序列化的对象要实现：Serializxable 标记接口

线程的注意事项：

https://blog.csdn.net/a303549861/article/details/93619042

# 第二天：线程同步

创建线程的四种方式 <1>继承Thread ，<2>实现 Runnable ，<3>实现Callable <v> , <4>创建线程池

线程休眠 解释： 即线程进入阻塞状态后并进入就绪状态 ，

进入休眠的方法：

join(); A线程调用B线程的join()方法，A线程等待B线程终止后再运行

sleep(mills);定义当前线程休眠多少毫秒

wait();永久性休眠，只能用notify();或notifyAll();方法唤醒

线程同步（锁对象）synchronized 主要用于修饰 代码块 ，修饰方法(锁对象是(this))，修饰类（锁对象是字节码（.class）文件）

锁对象特点：唯一性

死锁：两个线程互相等待，占用系统内存资源

解决方式：要尽量避免程序出现死锁

第三天：网络通信

#### Tcp通信协议：

Tcp属于服务端和客户端的关系

ServerSocket（服务端通信协议对象），Socket（客户端通信协议对象）

ServerSocket调用accept()方法获取字节流对象进行通信

Socket直接调用方法就可以获取字节流对象

#### Udp通信协议：

（服务端，客户端）和（服务端，客户端）的关系

DatagramSocket(通信协议对象) , DatagramPacket（数据对象）

DatagramSocket调用send()方法就可以发送通信信息

DatagramSocket 调用receive()方法就可以接收数据

注意：，要往发送和接收的方法中传入DatagramPacket（数据对象）进行传输数据，也就是传输数据要用数据对象进行传输

# 第三天：html,css,js,jquery

## Html:

可根据菜鸟教程或w3cl网站自行学习

## Css:

可根据菜鸟教程或w3cl网站自行学习

## Js类型转换：

<https://www.cnblogs.com/ConfidentLiu/p/7802939.html>

#### 1. 转换函数：

js提供了parseInt()和parseFloat()两个转换函数。前者把值转换成整数，后者把值转换成浮点数。只有对String类型调用这些方法，这两个函数才能正确运行；对其他类型返回的都是NaN(Not a Number)。

#### 2. 强制类型转换

还可使用强制类型转换（type casting）处理转换值的类型。使用强制类型转换可以访问特定的值，即使它是另一种类型的。

ECMAScript中可用的3种强制类型转换如下：

Boolean(value)——把给定的值转换成Boolean型；

Number(value)——把给定的值转换成数字（可以是整数或浮点数）；

String(value)——把给定的值转换成字符串。

用这三个函数之一转换值，将创建一个新值，存放由原始值直接转换成的值。这会造成意想不到的后果。

当要转换的值是至少有一个字符的字符串、非0数字或对象（下一节将讨论这一点）时，Boolean()函数将返回true。如果该值是空字符串、数字0、undefined或null，它将返回false。

## Js的内置对象: (DHTML)API

#### Window(窗口对象)

##### 方法：

提示框：alert();

输入提示框：返回值：String; 语法：window.prompt();

确认提示框：返回值：true||false; 语法： window.confirm();

关闭当前窗口：window.close();

createPopup() :创建自定义窗口

resizeTo () :更改窗口的宽度和高度

open() :打开新窗口并装入指定URL文档的路径

Print() :把内容打印到网页上

moveTo() :将窗口左上角的屏幕位置移动到指定的x , y 轴位置

setTimeout() :延迟执行，参数：（要延迟执行的内容（字符串类型），延迟的时间（number类型））

setInterval(); 延迟执行，参数：（要延迟执行的内容（字符串类型），延迟的时间（number类型））

##### Js创建一个窗口：

window.open('https://hao.360.com/', 'newwindow', 'height=100, width=400, top=0, left=0, toolbar=no, menubar=no, scrollbars=no, resizable=no, location=no, status=no')

//参数

alwaysLowered //始终在其它浏览器窗口的后面(NN)

alwaysRaised //始终在其它浏览器窗口的前面(NN)

channelMode //是否为导航模式(IE)

copyhistory //复制历史记录至新开的窗口(NN)

dependent //新窗口随打开它的主窗口关闭而关闭(NN)

fullscreen //全屏模式(所有相关的工具栏都没有)(IE)

location //是否显示地址栏(NN,IE)

menubar //是否显示菜单栏(NN,IE)

scrollbars //是否显示滚动条(NN,IE)

status //是否显示状态栏(NN,IE)

toolbar //是否显示工具栏(NN,IE)

directories //是否显示链接栏(NN,IE)

titlebar //是否显示标题栏(NN)

hotkeys //显示菜单快捷键(NN)

innerHeight //内容区域的高度(NN)

innerWidth //内容区域的宽度(NN)

resizable //是否可以调整大小(NN,IE)

top //窗口距离桌面上边界的大小(NN,IE)

left //窗口距离桌面左边界的大小(NN,IE)

height //窗口高度(NN,IE)

width //浏览器的宽度

##### window事件：

onload 页面加载完后执行的事件

#### Location（地址栏对象）

##### 方法：

属性：herf:获取地址栏的URL路径

#### Document(页面对象)

##### 对象解释：

把页面所有的标签都封装成对象

页面元素的：增删改查

Node接口：页面元素的所有父接口

Element接口：对标签的封装

Attr接口：对属性的封装

Text接口：对文本的封装

Comment接口：对注释的封装

增删改查：create delete update read---crud

##### Element中的标签属性（非函数，不带括号，直接赋值就能用）：

获取文本内容：

inenrHTML ： 获取标签中的所有html代码

Innertext： 获取标签中的文本内容

获取id和class：

id ： 获取标签id的属性值，也可以直接设置属性值

className：获取class标签的属性值，也可以直接设置属性值

获取父对象：

parentElement：获取 对象层次 的父标签对象

parentNode： 获取 文本对象层次 的父标签对象

获取子对象：

children： 获取所有 对象层次 的子标签

childNodes： 获取所有 文本对象层次 的子元素

获取节点：

nodeName: 获取节点名字

nodeType: 获取节点类型（Element（标签）是1 ， Text（文本）是3 ，document（文档）9）

##### Element中的方法（带括号）

获取标签元素（元素即标签本身）：

document.ElementById(); 通过id属性的value值获取标签对象

document.ElementByName(); 通过name属性的value值获取集合

document.ElementByTagName(); 通过标签名获取集合

设置标签属性（属性即标签内部的属性）：

setAttribute(); 参数（“标签名”，值），设置指定标签属性的值

移除标签属性：

removeAttribute(); 参数（“要移除的标签的属性名字符串”）移除标签属性

removeChild(); 参数（要移除的子元素对象）移除所有子标签的节点

添加标签：

document.createElement(); 参数（“标签名”） 创建标签

appendChild(); 参数(子元素对象) 添加子标签

##### Element中的删除等于把标签，元素，文本设置为空即可！

## Jquery:

可参考Jquery 帮助文档，自行学习

# 第四天：mysql

#### 远程连接：

. 授权法。例如，你想myuser使用mypassword从任何主机连接到mysql服务器的话。

GRANT ALL PRIVILEGES ON \*.\* TO 'myuser'@'%' IDENTIFIED BY 'mypassword' WITH GRANT OPTION;

如果你想允许用户myuser从ip为192.168.1.3的主机连接到mysql服务器，并使用mypassword作为密码

GRANT ALL PRIVILEGES ON \*.\* TO 'myuser'@'192.168.1.3' IDENTIFIED BY 'mypassword' WITH GRANT OPTION;

## 数据类型：

##### 整数：

bit(n) 1-64 二进制

tinyint 1个字节

int/integer 4个字节

bigint 8个字节

##### 浮点：

float(n,m) n是有效位数 m是小数位数 4个字节

float(5,3) 整数5-3位 小数3位

double(n,m) 8个字节

decimal 专门表示钱

##### 字符串：

char(n)固定长度 1-255 char(10)数据在数据库中固定占10个字符的空间

varchar(n)可变长度 1-65535

text 文本 64kb

longtext 大文本 4G

##### 二进制：

tinyblob 256b

blob 64kb

longblob 4G

##### 布尔：

bool/boolean 1b

##### 日期：

date xxxx-xx-xx

time xx:xx:xx

datetime xxxx-xx-xx xx:xx:xx

timestamp

year(n) 年

## 数据库语言种类：

DDL:操作表的结构

DML:表记录的增删改

DQL:表记录的查询

DCL:数据库的权限管理

TCL:数据库的事务管理

## 使用数据库：

use db\_26;

创建表：

create table tab\_2(

tid int,

tname varchar(4),

tsex char(1),

tisby boolean,

tscore float(4,1),

tshow text,

tbirthday datatime

);

## 数据增删改操作：

### 表的增删改：

查询表结构：

desc tab\_3;

查询创建表的sql语句：

show create table tab\_2;

修改表名：

Alter table tab\_2 rename to tab\_3;

删除表：

Drop table tab\_3;

### 数据库：对表的列的增删改(统一前缀：alter table 表名)

添加列：

语法： alter table 表名 add 字段名 字段类型(长度);

示例： alter table tab\_2 add tesex char(1);

删除列：

语法：alter table 表名 drop 列名；

示例：alter table tab\_2 drop tsex;

修改列名：

语法：alter table 表名 change 旧列名 新列名 列类型

示例：alter table tab\_2 change tsex sex varchar(3);

修改列类型：

语法：1. alter table 表名 change 列名 列名 新列类型

示例：alter table tab\_2 change sex sex char(1);

修改列类型：

语法：2.alter table 表名 modify 列名 新列类型

示例：alter table tab\_2 modify sex char(2);

### DML:数据库：对表记录的行的增删改

添加记录：

insert into tab\_2 values(1,"李华"，‘男’，true，90.5,“我是一只小小鸟”，“2010-12-13”); //添加全部，必须要和数据库表的字段顺序对应

inset into tab\_2(tname,tsex,tscore,tshow) values(“李心怡”，‘女’，34.2，“温柔的我”); //添加自己定义的字段，要与自己定义的字段的顺序对应

删除记录：

delete from tab\_2 ; // 删除全表

delete from tab\_2 where id = 1; //删除自己过滤后的一条数据

delete和truncate的区别

相同之处：都可以用于删除记录

不同之处：1.delete 是逐行删除 效率比较低

Truncate 是删除表的数据文件 效率高

2.delete是dml 需要事务 可以回滚

Truncate 是ddl 没有事务 不能回滚

3.delete 后面可以加where 条件 删除指定行

Truncate不能加where条件 只能删除所有行

修改数据：

updata tab\_2 set tid=1,tname="王方”tsex='男' where id = 2;

uptata tab\_2 set 1,"王明"，‘男’，true，13.5，“喜欢蓝天！”，“2014-14-56”where id = 2；

## mysql的函数

：

### 计算类的函数：

1.求绝对值：

abs(n)；

示例：select abs(-1),abs(1);

2.近似值：

ceil(n):大于等于参数的最小整数

floor(n):小于等于参数的最大整数

round(n):四舍五入

round(n,m):四舍五入保留m位小数

truncate(n,m):保留m位小数

3.开方：

sqrt(n):求n的平方

4.随机：

rand();随机[0,1]

5.求余：

mod(n,m);求n余m的值

6.进制转换 ：

Bin(n) 获取二进制的n

Hex(n)获取16进制的n

### 操作字符串函数：

1.length(str);获取字符个数

char\_length(str) 获取字符串个数

2.subString(str,startIndex,length) 截取从startIndex开始 截取length个字符

Substring（str,startIndex）截取从startIndx开始到末尾

和substr完全等价

注意：字符串索引从1开始

3. ascii(str) 获取str第一个字符在编码表中对应的整数

4.concat(str1,str2,str3) 字符串拼接

5.trim(str) 去除两边的空格

ltrim(str) 去除左边的空格

rtrim(str) 去除右边的空格

6.reverse(str)字符串反转

7.upper（str）把所有字母大写

Lower(str)把所有字母小写

### 操作日期时间的函数

1 获取当前时间：current\_date()::获取当前年月日

current\_time()::获取当前时分秒

current\_timestamp()::获取当前时间戳

now()::获取年月日时分秒

curdate();等价于current\_date()

curtime();等价于current\_time()

2 获取时间参数： year(date) 获取年

month(date) 获取月

dayofmonth(date) 获取号

dayofweek(date) 获取星期 1-7对应的是日-六

hour(date) 获取时

minute(date)获取分

second(date)获取秒

3 时间参数加指定值：

date\_add(date, intaval value field) 把date时间的field字段加value

4 时间和字符串之间的转换

str\_to\_date(str,strMS) 把strMS指定格式的字符串str转换为日期

date\_format(date,strMs) 把date转换为strMS指定的格式的字符串

strMS来指定字符串的模式如：%Y/%m/%d %H-%i-%s

\*/

## 数据库查询操作：

### DQL:单表查询：

1. 查询所有行，所有列：select \* from 表名

Select \* from tab\_2;

2.查询，所有行, 指定列：select 列名 ，列名，列名 from 表名

Select tid , tname,sex from tab\_2;

3.数字类型列运算：+ - \* / % （等价于mod方法）

Select tage+1,tage-1,tage\*2,tage/2,tage%3 from tab\_2;

4.Concat(str,str,..)字符串拼接运算

Select concat(“我叫”,tname,”今年”,tage,”岁！”)from tab\_2;

5.distinct(字段) 去重：重复的记录显示一次，可以在括号里面添加字段当函数使用，也可以不添加字段直接当关键字使用

Select distinct tname ,tage from tab\_2;

6.聚合函数：sum() :求和 max():最大值 min():最小值 avg():平均值

Count（tage）tage列不为null的记录数

7.as起别名 可以省略

8.like:模糊查询 %表示0到多个任意字符 \_表示1个任意字符

9.limit a , b ; 分页查询：a为第一行的索引（从0开始）b是获取的记录数（limit是mysql的方言只能在mysql中使用）

10.条件查询

比较运算符： > ,>=,<,<=,!=,=,<>(注意比较时不算值为null的记录)

判断是否为null ::: is null ,is not null

与和或 ：：：or ,and

范围：：：x between a and b 等价于 x>=a and x<=b

In(x1,x2,x3) 等价于 x=x1 or x=x2 or x= x3

注意： 在where 后面不能加 聚合函数

order by(排序 desc降序，asc升序，默认是asc升序) , groud by(分组) ，having在order by 和 groud by 后做判断使用

### 约束：

#### 添加约束：

unique:设置该字段下的值不能重复，唯一约束

not null:设置该列字段下的值不能为空

default :设置该字段下的默认值

primary key设置主键 在主键的后面添加 auto\_increment 设置主键自增功能

##### 创建表后添加约束：

1. 添加非空约束

alter table 表名 change 旧列名 字段 类型 not null;

alter table 表名 modify 旧列名 字段 类型 not null;

1. 添加默认约束

alter table 表名 change 旧列名 字段 类型 default 默认值；

alter table 表名 modify 旧列名 字段 类型 default 默认值；

1. 添加主键约束

alter table 表名 change 旧列名 字段 类型 primary key;

alter table 表名 modify 旧列名 字段 类型 primary key;

1. 添加唯一约束

alter table 表名 change 旧列名 字段 类型 unique;

alter table 表名 modify 旧列名 字段 类型 unique;

1. 添加自增约束

alter table 表名 change 旧列名 字段 类型 auto\_increment;

alter table 表名 modify 旧列名 字段 类型 auto\_increment;

#### 删除约束：

1. 删除非空约束

alter table 表名 change 旧列名 字段 类型 null;

alter table 表名 modify 旧列名 字段 类型 null;

1. 删除默认约束

alter table 表名 change 旧列名 字段 类型；

alter table 表名 modify 旧列名 字段 类型；

1. 删除自增约束

alter table 表名 change 旧列名 字段 类型;

alter table 表名 modify 旧列名 字段 类型;

1. 删除主键约束

alter table 表名 drop primary key;

1. 删除唯一约束

alter table 表名 drop index 字段；

#### 修改约束：

先删除约束再添加新的约束

### 外键约束：

外键约束的特点：在从表中定义外键列来引用主表的主键

1 一个从表中可以定义多个外键

2 可以为null

3 可以重复

4 如果外键列有值 此值必须在主表的主键列中存在

5 外键列只能引用主表的主键

6 删除主表/主表记录前 需要先删除从表/从表记录/从表中的外键约束

#### 添加外键约束：

主表（即就是老师表），从表（即就是学生表）

给学生表添加一个外键：

示例：

alter table student add constraint stid foreign key (sid) references teacher(tid)

语法：alter table 从表表名 add constraint 自定义外键名 foreign key(从表的外键字段) references 主表表名(主表主键字段);

#### 删除外键约束：

alter table 从表表名 drop foreign key 从表的外键字段名

### check：检查约束：

为表添加额外的约束：每次对表进行操作时此约束就会检查一次sql语句是否满足要求，满足操作成功，不满足操作失败！

**仅在Oracle数据库中可使用，mysql中不支持！**

#### 创建表时添加check约束语法：

CONSTRAINT 自定义此约束名 CHECK (在括号中添加约束判断表达式)

#### 创建表后添加check约束语法：

alter table 表名 add constraint 自定义此约束名 check(在括号中添加约束判断表达式);

### 多表查询：

#### 多表上下连接：

union all -- 多表上下连接(加 all 显示全部，不加all 等于把两张表相同的数据进行去重)

语法： select \* from 表名1 union select \* from 表名2；

select \* from 表名1 union all select \* from 表名2；

示例： select \* from tab\_1 union select \* from tab\_2;

select \* from tab\_1 union all tab\_2 ;

#### 内连接：

inner join -- 内连接

语法：select \* from 表名1 inner join 表名2 on 表1的字段 = 表2的字段；

示例：select \* from tab\_1 inner join tab\_2 on 表1的字段 = 表2的字段；

#### 外连接：

left outer join -- 左外连接

语法：select \* from 表名1 left outer join 表名2 on 表1的字段 = 表2的字段；

示例：select \* from tab\_1 left outer join tab\_2 on tid = sid；

right outer join -- 右外连接

语法：select \* from 表名1 right outer join 表名2 on 表1的字段 = 表2的字段；

示例：select \* from tab\_1 right outer join tab\_1 on tid = sid；

#### 特殊查询字符函数：

Any同等于|| ， All同等于&& ，in范围内为true ，not in不在范围内为true

### Mysql : JDBC:

jdbc查询主要步骤:

1. 加载驱动

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

1. 注册连接

Connection connection = DriverManager.getConnection(“数据库路径地址” , ”用户名 ” , ”密码 ” );

解释：

jdbc协议 + mysql协议 + IP协议 + mysql端口 + 数据库

jdbc:mysql//localhost:3306/user

##### 形式一：

1. 获得执行语句对象

Statement statement = connection.createStatement();

1. 执行sql,获得结果集

ResultSet resultSet = statement.executeQuery("select \* from student");

##### 形式二：

3 获得执行语句对象进行预处理

Statement statement = connection.prepareStatement("select \* from student where id = ?");

preparedStatement.setString(1,"");

解释：

Sql语句有几个问号就添加几个参数，

添加什么类型就set什么类型（第几个问号（int类型），添加的值）

4 执行sql，获得结果集

1. 遍历结果集

while (resultSet.next()){

resultSet.getString("");

resultSet.getFloat("");

resultSet.getXxx("");

什么类型就 .get什么类型名（“列名”）

}

6.关流

resultSet.close();

statement.close();

connection.close();

从内向外关流

### Oracle : jdbc：

Class.*forName*("oracle.jdbc.driver.OracleDriver");  
// oracle中层级关系是 Oracle服务-->用户-->表 服务可以有多个  
// 注意事项：Oracle和mysql中都可以调用存储过程和函数  
// orcl是代表你要连接的Oracle中的具体服务  
 String url= "jdbc:oracle:thin:@192.168.217.12:1521:ORCL";  
 String user="RT";  
 String password ="1234567";  
 Connection connection = DriverManager.*getConnection*(url,user,password);  
 PreparedStatement preparedStatement = connection.prepareStatement("select \* from score");  
 ResultSet resultSet = preparedStatement.executeQuery();  
 while (resultSet.next()){  
 String sno = resultSet.getString("sno");  
 System.*out*.println(sno);  
 }  
 resultSet.close();  
 preparedStatement.close();  
 connection.close();

Oracle 调用 构造过程和函数

<https://www.cnblogs.com/lz2017/p/7500411.html>

# 第五天：web

### 问题：Tomcat初始页面可加载：页面发出请求服务器无响应，无报错

检查项目使用的servlet引入的依赖是不是当前使用的Tomcat服务器lib下的jar包

#### 原因：default servlet不可用

#### 解决：检查Tomcat变量：

cata\_base： Tomcat的工作目录；

cata\_home： Tomcat的安装目录；

out目录作用：编译后的Java文件

写项目时实时查看out目录有没有更新

#### 不同版本的servlet不一样

Tomcat8.5是Javax包下的servlet

Tomcat10.0是jakat包下的servlet

高版本的Tomcat有点不稳定

### 问题：导入过MySQL的依赖后，工具类启动失败！

原因：部署在tomcat上的war包内部没有MySQL的依赖，仅仅只是在项目中有

解决：检查项目中Artifacts中的lib目录下有没有少依赖，补充完整的依赖，问题解决！

## 请求转发：

请求转发：req.getRequestDispatcher(“发出新的请求，通过配置好的映射路径进行转发”).forward(req , resp);

请求包含：req.getRequestDispatcher(“”).include(req , resp);

响应头：

请求转发：a可以设置，b也可以设置

请求包含：a可以设置，b不能设置

响应体：

请求转发：a不能设置，b可以设置

## 重定向：

resp.sendRedirect();

### 重定向：错误日志：

错误日志：重定向发出第二次get请求路径中文乱码

解决方案：对要发送的数据进行编码，不能对整个路径进行编码，不然路径编码后会请求错误，只能对要发送的数据进行编码；

注意：只能是数据，除数据外的 （ ？键 = ）都不能进行编码 ，不然取不出值

示例：

第一次重定向：

@Override  
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
  
 String name = "张亮";  
 String newName = URLEncoder.*encode*(name, "UTF-8");  
 System.*out*.println(newName);  
 resp.sendRedirect(req.getContextPath()+"/Test?data="+newName);  
}

第二次发出get请求；req接收数据不用进行解码，直接设置请求的编码格式为UTF-8即可接收成功！

@Override  
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 req.setCharacterEncoding("UTF-8");  
 resp.setCharacterEncoding("UTF-8");  
  
 String data = req.getParameter("data");  
 System.*out*.println(data);  
 resp.getWriter().write(data);  
}

## 原生态Ajax的get请求：

        var b = new XMLHttpRequest();//创建Ajax传输数据的对象

        b.open("post", "url", true);//设置  （请求方式，请求路径，是否异步）

        b.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");//设置  请求内容的类型

        b.send("name='李四'&sex='男'&age=12");//发送给服务器的参数 post请求时写在send方法里面，get请求时直接写在url路径后面即可

        b.onreadystatechange = function () {//存有处理服务器响应的函数，每当 readyState 改变时，onreadystatechange 函数就会被执行

            if (b.readyState == 4) {//readyState响应的状态码 0: 请求未初始化（代理被创建，但尚未调用 open() 方法）

              alert(b.responseText);// 1: 服务器连接已建立（open方法已经被调用）

            }                       // 2: 请求已接收（send方法已经被调用，并且头部和状态已经可获得）

        }                           // 3: 请求处理中（下载中，responseText 属性已经包含部分数据）

                                    // 4: 请求已完成，且响应已就绪（下载操作已完成）

## Json:

页面中 ：eval(String string) 函数 把json字符串转换成json对象，可以直接使用

Java类中：JsonObject.fromObject(Object object); 把Java对象变成，json对象

Java类中：JsonArray(Object object);把集合变成json集合对象

# 第六天：框架底层

## Java反射：

可查看 Class 类的Api帮助文档

## Xml解析：

### 创建工厂对象，获取Document对象：

public static Document getXmlDocument(String path) throws ParserConfigurationException, IOException, SAXException {  
 return DocumentBuilderFactory.*newInstance*().newDocumentBuilder().parse(new File(path));  
}

### 操作Document对象，相当于操作js中的Document对象一样：

Attribute 解释单词意思：属性(字段)

Node 解释单词意思：节点(元素(标签))

Document 对像获取Element元素对象，进而操作标签

Document 对象创建节点或属性； 借用Element中的方法可以直接在xml文件的标签下追加或添加节点或属性；

### 获取xml文件刷新对象，传入Document对象和xml的文件路径，把修改或添加过的Document对象刷新到xml文件中去：

**public static void refresh(Document doc, String path) throws TransformerException {  
 TransformerFactory.newInstance().newTransformer().transform(new DOMSource(doc), new StreamResult(new File(path)));  
}**

## 注解：

### 两个元注解：

**@Target({ElementType.TYPE, ElementType.METHOD, ElementType.FIELD,ElementType.ANNOTATION\_TYPE})  
@Retention(RetentionPolicy.RUNTIME) //注解保留在运行时**

**@Retention(RetentionPolicy.CLASS) //默认值,注解保留在编译时**

**@Retention(RetentionPolicy.SOURCE) //注解保留在源代码时（即没有编译前）  
@interface Annotations {  
 /\*\*  
 \* 注解可以添加属性  
 \* 属性格式：属性类型 属性名（） default 默认值;  
 \* 属性类型： 8种基本数据类型 String、Class、Annotation、Enum、Array  
 \*/  
 String value() default "无名类";  
  
 int age();  
  
 Class cla();  
  
 int[] arr();  
}**

### 注解用反射取值：

public static void main(String[] args) throws IllegalAccessException, InstantiationException {  
 Class<? extends Test> aClass = new Test().getClass();  
 Annotation[] declaredAnnotations = aClass.getDeclaredAnnotations();  
 for (int i = 0; i < declaredAnnotations.length; i++) {  
 if (declaredAnnotations[i] instanceof Annotations) {  
 Annotations declaredAnnotation = aClass.getDeclaredAnnotation(Annotations.class);  
 String value = declaredAnnotation.value();  
 System.out.println(value);  
 }  
 }  
  
}

## 代理：注意：动态代理会给类的每个方法都加一个增强的方法

### 静态代理:

实现同一个接口，在代理类上多态引用另一个实现类中的所有方法（代理类也要实现相同的接口，方法内部调用接口的其他实现类中的方法进行代理）

### 动态代理：代理接口，实现动态代理 ：代理工厂实现接口 ：InvocationHandler

**package day02;  
  
import java.lang.reflect.InvocationHandler;  
import java.lang.reflect.Method;  
import java.lang.reflect.Proxy;  
  
/\*\*  
 \*  
 \* 实现接口形式的 动态代理  
 \*  
 \* @author RT  
 \*/  
public class AgeEncyTest02 {  
 public static void main(String[] args) {  
 TestDome testDome = new TestDome(1,"小米");  
 ProxyFactroy proxyFactroy = new ProxyFactroy(testDome);  
 TestDomes proxyFactroy1 = (TestDomes) proxyFactroy.getProxyFactroy();  
 proxyFactroy1.one();  
  
 }  
}  
/\*\*  
 \* 创建代理工厂类  
 \* \*/  
class ProxyFactroy implements InvocationHandler{  
 Object target;  
  
 public ProxyFactroy(Object target) {  
 this.target = target;  
 }  
  
 @Override  
 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
 System.out.println(method.getName()+"方法前执行的扩展===");  
 Object invoke = method.invoke(target, args);  
 System.out.println(method.getName()+"方法后执行的扩展===");  
 return invoke;  
 }  
 public Object getProxyFactroy(){  
 return Proxy.newProxyInstance(target.getClass().getClassLoader(),target.getClass().getInterfaces(),this);  
 }  
  
}  
/\*\*  
 \* 被代理的类必须 有接口  
 \* \*/  
interface TestDomes{  
 void one();  
 void two();  
}  
class TestDome implements TestDomes{  
 int age ;  
 String name ;  
  
 public TestDome(int age, String name) {  
 this.age = age;  
 this.name = name;  
 }  
  
 public TestDome() {  
 }  
  
 @Override  
 public void one() {  
 System.out.println("我叫："+name+"今年"+age+"岁了！");  
 }  
  
 @Override  
 public void two() {  
 System.out.println("你是谁？");  
 }  
}**

### Cglib 动态代理：继承目标类，子类实现代理 ：代理工厂实现接口：MethodInterceptor

**package day02;  
  
import net.sf.cglib.proxy.Enhancer;  
import net.sf.cglib.proxy.MethodInterceptor;  
import net.sf.cglib.proxy.MethodProxy;  
  
import java.lang.reflect.Method;  
  
/\*\*  
 \* Cglib 动态代理实现过程  
 \*  
 \* @author RT  
 \*/  
  
/\*\*  
 \* 测试方法  
 \* \*/  
public class AagEncyTest {  
 public static void main(String[] args) {  
 TeacherTest teacher = new TeacherTest(1, "张老师");  
// 创建工厂对象调用getInstance方法  
 ZiProFactroy ziProFactroy = new ZiProFactroy(teacher);  
 TeacherTest instance = (TeacherTest) ziProFactroy.getInstance();  
// 调用对象中的方法，工厂类会自动给方法加代理的内容  
 instance.hehe();  
 }  
}  
  
/\*\*  
 \* 创建代理工厂类  
 \*/  
class ZiProFactroy implements MethodInterceptor {  
 Object target;  
  
 public ZiProFactroy(Object target) {  
 this.target = target;  
 }  
  
 @Override //方法 参数数组  
 public Object intercept(Object o, Method method, Object[] objects, MethodProxy methodProxy) throws Throwable {  
 System.out.println(method.getName() + "方法执行前执行的代码！");  
 //执行方法并返回  
 Object invoke = method.invoke(target, objects);  
 System.out.println(method.getName() + "方法执行后执行的代码!");  
 //返回执行过的方法  
 return invoke;  
 }  
  
 /\*\*  
 \* 生成对象，调用此方法返回的对象中的任何方法，可对其中的方法进行代理  
 \*/  
 public Object getInstance() {  
 Enhancer enhancer = new Enhancer();  
 enhancer.setSuperclass(target.getClass());  
 enhancer.setCallback(this);  
 return enhancer.create();  
 }  
}  
  
/\*\*  
 \* 测试类  
 \*/  
class TeacherTest {  
 int age;  
 String name;  
  
 public TeacherTest() {  
 }  
  
 public TeacherTest(int age, String name) {  
 this.age = age;  
 this.name = name;  
 }  
  
 public void hehe() {  
 System.out.println("老师" + age + "名字" + name);  
 }  
  
}**

## 日志框架:log4j

1.导入日志jar包

2.在java文件src路径下创建日志的配置文件 配置文件的后缀为: .properties

有四个级别:logger级别: DEBUG、INFO、WRNING、ERROR 小写: debug, info, wrning, error

### 输出DEBUG 级别以上的日志到=E://logs/error.log ###

#打印的日志的驱动文件路径

log4j.appender.B = org.apache.log4j.DailyRollingFileAppender

#打印的位置

log4j.appender.B.File = E://log4j\_logs/log.log

#是否追加打印的日志

log4j.appender.B.Append = true

#打印的级别

log4j.appender.B.Threshold = DEBUG

#打印的模板对象

log4j.appender.B.layout = org.apache.log4j.PatternLayout

#打印的模板

log4j.appender.B.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n

3.在java代码中创建对象

static Logger logger = Logger.getLogger(当前类名.class.getName());

4.有四种打印日志的方法:logger.debug(object);

logger.info(object);

logger.warn(object);

logger.error(object);

## c3p0的数据库连接池:

<c3p0-config>

<!--默认配置-->

<default-config>

<!--initialPoolSize：初始化时获取5个连接，取值应在minPoolSize与maxPoolSize之间。 -->

<property name="initialPoolSize">50</property>

<!--maxIdleTime：最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。-->

<property name="maxIdleTime">60</property>

<!--最大连接100个 -->

<property name="maxPoolSize">100</property>

<!--最少连接10个 -->

<property name="minPoolSize">10</property>

<!--maxStatements：最大链接数。-->

<property name="maxStatements">200</property>

</default-config>

<!--配置连接池mysql-->

<named-config name="mysql">

<property name="driverClass">com.mysql.jdbc.Driver</property>

<property name="jdbcUrl">jdbc:mysql://localhost:3306/day27</property>

<property name="user">root</property>

<property name="password">123456</property>

<property name="initialPoolSize">5</property>

<property name="maxIdleTime">30</property>

<property name="maxPoolSize">100</property>

<property name="minPoolSize">10</property>

<property name="maxStatements">200</property>

</named-config>

</c3p0-config>

//通过标识名来创建相应连接池

static ComboPooledDataSource dataSource = new ComboPooledDataSource("mysql");

dataSource.getConnection();

关流时正常关流包括(connection)都是一样正常关流连接会自动返还给数据库连接池

## Mybatis :

1. 导入jar包：
2. 配置全局配置的xml文件：
3. 配置mapper的xml文件，一个mapper.xml 文件代表一个接口

### 动态sql：

if标签：<if test=”判断表达式”></if>

choose/wnen/otherwise 标签：<choose>

<when test=”判断语句表达式”></when>

<otherwise></otherwise>//上方标签都没有生效时，执行此标签

</choose>

trim标签： prefix: 在整体前面加的片段

suffix： 在整体后面加的判断

prefixoverrides: 智能判断每个判断标签（if）成功输出sql语句片段时，是否保留前缀 语法格式 and | or 语

句中有and 就会自动往语句的前缀添加or 有 or 就会自动往语句中添加 and

suffixoverrides: 智能判断后缀，和上方的智能判断前缀差不多

where标签：智能添加where标签

set标签：会智能的去除set标签中语句中多余的逗号

foreach标签：collection：创建容器的类型：list map array

close:整体结束需要加的片段

open：整体开始需要加的片段

index：索引

item：元素

separator：分隔符

### mybatis一级，二级缓存：

一级缓存：默认打开

二级缓存：需要手动打开

session.clearCache(); //可以清空 一级，二级 ，缓存

### Mybatis逆向工程：

可自行上网查询，进行操作

### 注解版mybatis : 使用注解版就不用再写mapper的映射文件

注解自动返回数据，不用写返回的数据类型，如果要指定返回值类型可以写下方的@Results注解

@Select(“写sql语句”)

@Delete(“写sql语句”)

@Updata(“写sql语句”)

@Insert(“写sql语句”)

@Results(id=”给这个注解代码设置的名字，别的代码可以通过id名直接调用提高复用性”,

value={ @Result(column=”列名”,property=”属性名”)， //相当于Mapper中的ResultMap属性

@Result(column=”列名”,property=”属性名”)，

@Result(column=”列名”,property=”属性名”)})

注解版动态sql: 需要在大括号中写标签，sql语句要写在<script>标签中，然后用各种各样的判断标签进行判断即可

注解版动态sql示例：@Select({"<script> " +

"select \* from t\_user " +

"where 1=1 " +

"<if test='userId!=null'> and id = #{userId}</if> " +

"</script>"})

## Spring：

Spring的xml配置文件的命名空间的网址：https://blog.csdn.net/xhaimail/article/details/79886478

### IOC:控制反转：

IOC:方法必须有构造方法

1.导入5个jar包

2.写xml配置文件

3.ClassPathXmlApplicationContext app = new ClassPathXmlApplicationContext(path);

HouseDaoimpl bean = (HouseDaoimpl) app.getBean("houseDaoimpl");

还可以不用强转直接获取:HouseDaoimpl bean =app.getBean("houseDaoimpl",HouseDaoimpl.class);

bean.setA(2);//利用获取到的对象输出

作用域: bean标签的 scope 作用范围的属性

四个作用域: 1.singleton//单列 xml文件默认单列

2.prototype//多列

3.request //请求域

4.session //会话域

4.FileSystemXmlApplicationContext fileSystemXmlApplicationContext = new FileSystemXmlApplicationContext(path);

此方法和上面的方法一样只不过是一个在src路径，一个是以项目为当前路径

## Dl注入:

<bean>

#### set注入: 这个标签有三个属性 name名字 value值 ref引用的值

<property name="属性名" value="要赋的值"></property>

#### 构造方法注入: 这个标签有五个属性 type参数类型 ref引用类型

<constructor-arg index="第几个参数" name="构造参数名" value="传入构造参数的值"></constructor>

### 集合注入:

<property>

<Array>

<value><value> 或 <ref bean=""/>

</Array>

<list></list>

<set></set>

<map>

<entry key="" value=""></entry> 基本数据类型 或<entry key-ref="" value-ref=""></entry>引用数据类型

</map>

</property>

</bean>

### 注解版IOC控制反转：

被spring容器托管的类的注解：此注解只能在类上使用

@Controller

@Service

@Repository

@Component

DI属性注入的注解：此注解只能在属性上使用

@Value 相当于bean 标签中的Value 属性

@Resource 相当于bean 标签中的ref 属性

关于bean 生命周期的注解：此注解只能在方法上使用

@PostConstruct //初始化方法

@PreDestroy //销毁方法

注意：配置文件中的bean可以和 注解形式创建出来的bean可以共存

作用域：@Scope

懒加载：@lazy

## Aop代理：

### 切面类：

Spring的aop的切面类需要使用以下几种接口

Spring-AOP切面类实现的接口

? 前置通知 org.springframework.aop.MethodBeforeAdvice

在目标方法执行前实施增强

? 后置通知 org.springframework.aop.AfterReturningAdvice

在目标方法执行后实施增强

? 环绕通知 org.aopalliance.intercept.MethodInterceptor

在目标方法执行前后实施增强

? 异常抛出通知 org.springframework.aop.ThrowsAdvice

在方法抛出异常后实施增强

? 引介通知 org.springframework.aop.IntroductionInterceptor

在目标类中添加一些新的方法和属性

### 目标类：

#### 目标类必须有接口：

### Spring配置：

创建bean org.springframework.aop.framework.ProxyFactoryBean

传入两个属性：target : 目标类

InterceptorNames 代理通知的实现类，内部写 <list>

<value>代理通知bean的id名字<value>

<value>代理通知bean的id名字<value>

</list>

可以对“方法名“进行模糊调用代理通知方法：

org.springframework.aop.support.RegexpMethodPointcutAdvisor

此类有两个属性：advice：代理的类

pattern：模糊匹配相当于正则匹配

使用：把此类bean的id名字写进list的value的标签中的即可，代替代理通知的id

#### 测试：

通过ClasspathxmlApplicationContext类执行目标类，即可，后台会自动通过配置的。。。通知。。。进行代理执行

注意：执行时要通过目标类的接口来接收数据

### SpringMVC：

Web.xml的dispatservlet会自动加载springmvc.xml文件，所以要想加载applicationContext.xml要在web.xml中配置监视器。详情见，java总结笔记

<!-- 此标签能代表默认配置的映射器和适配器-->

<mvc:annotation-driven/>

<!-- 映射器有两种 -->

<!-- 通过此类bean的name属性来指定URL路径-->

<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<!-- 通过此类的bean的id属性来指定URL路径-->

<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"/>

<!-- 适配器有两种 -->

<!-- (默认) 此类需要类实现Controller-->

<bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/>

<!-- 此类需要类实现HttpRequestHandler-->

<bean class="org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter"/>

<!-- 视图解析器-->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/"/>

<property name="suffix" value=".jsp"/>

</bean>

<bean name="/yu.do" class="servlet.Dome01"/>