1. 基础加强

## 反射：框架设计的灵魂

\* 框架：半成品软件。可以在框架的基础上进行软件开发，简化编码

\* 反射：将类的各个组成部分封装为其他对象，这就是反射机制

\* 好处：

1. 可以在程序运行过程中，操作这些对象。

2. 可以解耦，提高程序的可扩展性。

\* 获取Class对象的方式：

1. Class.forName("全类名")：将字节码文件加载进内存，返回Class对象

\* 多用于配置文件，将类名定义在配置文件中。读取文件，加载类

2. 类名.class：通过类名的属性class获取

\* 多用于参数的传递

3. 对象.getClass()：getClass()方法在Object类中定义着。

\* 多用于对象的获取字节码的方式

\* 结论：

同一个字节码文件(\*.class)在一次程序运行过程中，只会被加载一次，不论通过哪一种方式获取的Class对象都是同一个。

\* Class对象功能：

\* 获取功能：

1. 获取成员变量们

\* Field[] getFields() ：获取所有public修饰的成员变量

\* Field getField(String name) 获取指定名称的 public修饰的成员变量

\* Field[] getDeclaredFields() 获取所有的成员变量，不考虑修饰符

\* Field getDeclaredField(String name)

2. 获取构造方法们

\* Constructor<?>[] getConstructors()

\* Constructor<T> getConstructor(类<?>... parameterTypes)

\* Constructor<T> getDeclaredConstructor(类<?>... parameterTypes)

\* Constructor<?>[] getDeclaredConstructors()

3. 获取成员方法们：

\* Method[] getMethods()

\* Method getMethod(String name, 类<?>... parameterTypes)

\* Method[] getDeclaredMethods()

\* Method getDeclaredMethod(String name, 类<?>... parameterTypes)

4. 获取全类名

\* String getName()

\* Field：成员变量

\* 操作：

1. 设置值

\* void set(Object obj, Object value)

2. 获取值

\* get(Object obj)

3. 忽略访问权限修饰符的安全检查

\* setAccessible(true):暴力反射

\* Constructor:构造方法

\* 创建对象：

\* T newInstance(Object... initargs)

\* 如果使用空参数构造方法创建对象，操作可以简化：Class对象的newInstance方法

\* Method：方法对象

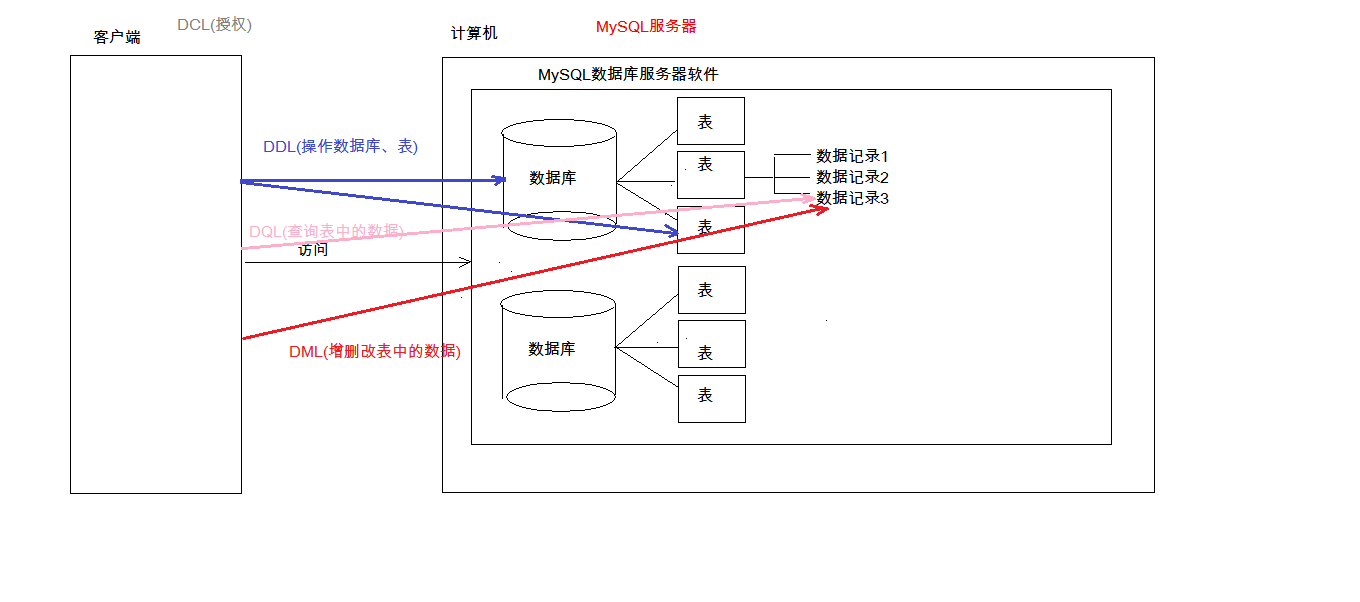
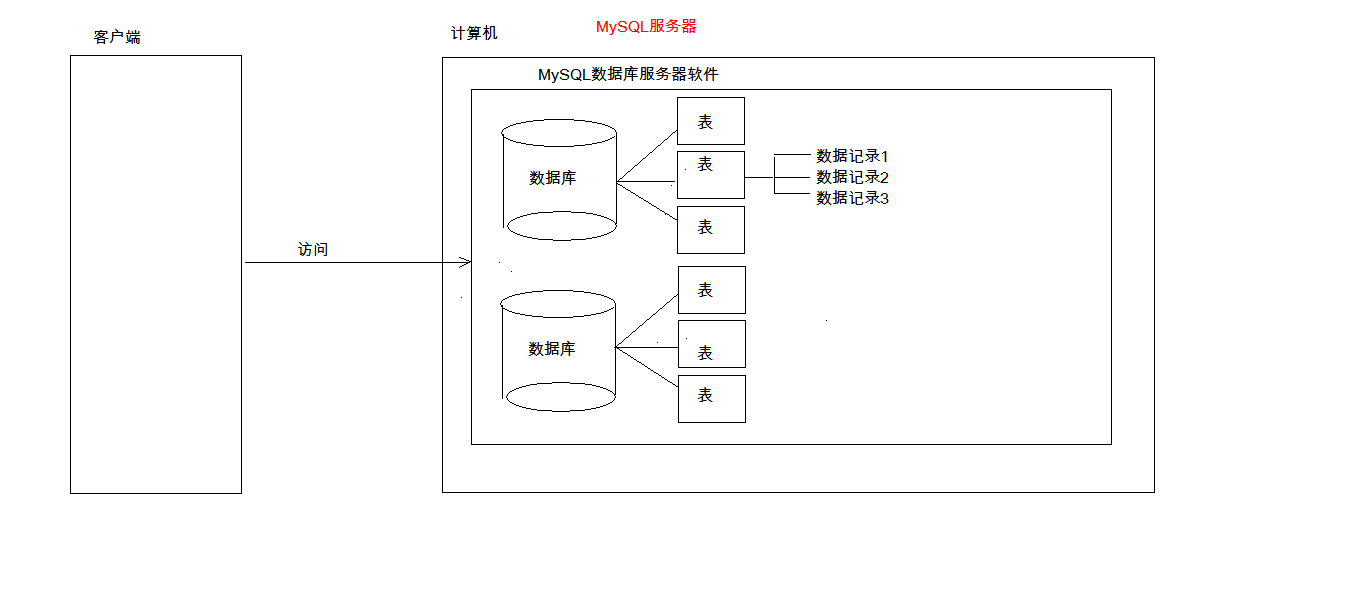
\* 执行方法：

\* Object invoke(Object obj, Object... args)

\* 获取方法名称：

\* String getName:获取方法名

1. MySQL



2. 数据库设计的范式

\* 概念：设计数据库时，需要遵循的一些规范。要遵循后边的范式要求，必须先遵循前边的所有范式要求

设计关系数据库时，遵从不同的规范要求，设计出合理的关系型数据库，这些不同的规范要求被称为不同的范式，各种范式呈递次规范，越高的范式数据库冗余越小。

目前关系数据库有六种范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、巴斯-科德范式（BCNF）、第四范式(4NF）和第五范式（5NF，又称完美范式）。

\* 分类：

1. 第一范式（1NF）：每一列都是不可分割的原子数据项

2. 第二范式（2NF）：在1NF的基础上，非码属性必须完全依赖于码（在1NF基础上消除非主属性对主码的部分函数依赖）

\* 几个概念：

1. 函数依赖：A-->B,如果通过A属性(属性组)的值，可以确定唯一B属性的值。则称B依赖于A

例如：学号-->姓名。 （学号，课程名称） --> 分数

2. 完全函数依赖：A-->B， 如果A是一个属性组，则B属性值得确定需要依赖于A属性组中所有的属性值。

例如：（学号，课程名称） --> 分数

3. 部分函数依赖：A-->B， 如果A是一个属性组，则B属性值得确定只需要依赖于A属性组中某一些值即可。

例如：（学号，课程名称） -- > 姓名

4. 传递函数依赖：A-->B, B -- >C . 如果通过A属性(属性组)的值，可以确定唯一B属性的值，在通过B属性（属性组）的值可以确定唯一C属性的值，则称 C 传递函数依赖于A

例如：学号-->系名，系名-->系主任

5. 码：如果在一张表中，一个属性或属性组，被其他所有属性所完全依赖，则称这个属性(属性组)为该表的码

例如：该表中码为：（学号，课程名称）

\* 主属性：码属性组中的所有属性

\* 非主属性：除过码属性组的属性

1. 第三范式（3NF）：在2NF基础上，任何非主属性不依赖于其它非主属性（在2NF基础上消除传递依赖）

## 事务

1. 事务的基本介绍

1. 概念：

\* 如果一个包含多个步骤的业务操作，被事务管理，那么这些操作要么同时成功，要么同时失败。

2. 操作：

1. 开启事务： start transaction;

2. 回滚：rollback;

3. 提交：commit;

3. 例子：

CREATE TABLE account (

id INT PRIMARY KEY AUTO\_INCREMENT,

NAME VARCHAR(10),

balance DOUBLE

);

-- 添加数据

INSERT INTO account (NAME, balance) VALUES ('zhangsan', 1000), ('lisi', 1000);

SELECT \* FROM account;

UPDATE account SET balance = 1000;

-- 张三给李四转账 500 元

-- 0. 开启事务

START TRANSACTION;

-- 1. 张三账户 -500

UPDATE account SET balance = balance - 500 WHERE NAME = 'zhangsan';

-- 2. 李四账户 +500

-- 出错了...

UPDATE account SET balance = balance + 500 WHERE NAME = 'lisi';

-- 发现执行没有问题，提交事务

COMMIT;

-- 发现出问题了，回滚事务

ROLLBACK;

4. MySQL数据库中事务默认自动提交

\* 事务提交的两种方式：

\* 自动提交：

\* mysql就是自动提交的

\* 一条DML(增删改)语句会自动提交一次事务。

\* 手动提交：

\* Oracle 数据库默认是手动提交事务

\* 需要先开启事务，再提交

\* 修改事务的默认提交方式：

\* 查看事务的默认提交方式：SELECT @@autocommit; -- 1 代表自动提交 0 代表手动提交

\* 修改默认提交方式： set @@autocommit = 0;

2. 事务的四大特征：

1. 原子性：是不可分割的最小操作单位，要么同时成功，要么同时失败。

2. 持久性：当事务提交或回滚后，数据库会持久化的保存数据。

3. 隔离性：多个事务之间。相互独立。

4. 一致性：事务操作前后，数据总量不变

3. 事务的隔离级别（了解）

\* 概念：多个事务之间隔离的，相互独立的。但是如果多个事务操作同一批数据，则会引发一些问题，设置不同的隔离级别就可以解决这些问题。

\* 存在问题：

1. 脏读：一个事务，读取到另一个事务中没有提交的数据

2. 不可重复读(虚读)：在同一个事务中，两次读取到的数据不一样。

3. 幻读：一个事务操作(DML)数据表中所有记录，另一个事务添加了一条数据，则第一个事务查询不到自己的修改。

\* 隔离级别：

1. read uncommitted：读未提交

\* 产生的问题：脏读、不可重复读、幻读

2. read committed：读已提交 （Oracle）

\* 产生的问题：不可重复读、幻读

3. repeatable read：可重复读 （MySQL默认）

\* 产生的问题：幻读

4. serializable：串行化

\* 可以解决所有的问题

\* 注意：隔离级别从小到大安全性越来越高，但是效率越来越低

\* 数据库查询隔离级别：

\* select @@tx\_isolation;

\* 数据库设置隔离级别：

\* set global transaction isolation level 级别字符串;

\* 演示：

set global transaction isolation level read uncommitted;

start transaction;

-- 转账操作

update account set balance = balance - 500 where id = 1;

update account set balance = balance + 500 where id = 2;

## DCL：

\* SQL分类：

1. DDL：操作数据库和表

2. DML：增删改表中数据

3. DQL：查询表中数据

4. DCL：管理用户，授权

\* DBA：数据库管理员

\* DCL：管理用户，授权

1. 管理用户

1. 添加用户：

\* 语法：CREATE USER '用户名'@'主机名' IDENTIFIED BY '密码';

2. 删除用户：

\* 语法：DROP USER '用户名'@'主机名';

3. 修改用户密码：

UPDATE USER SET PASSWORD = PASSWORD('新密码') WHERE USER = '用户名';

UPDATE USER SET PASSWORD = PASSWORD('abc') WHERE USER = 'lisi';

SET PASSWORD FOR '用户名'@'主机名' = PASSWORD('新密码');

SET PASSWORD FOR 'root'@'localhost' = PASSWORD('123');

\* mysql中忘记了root用户的密码？

1. cmd -- > net stop mysql 停止mysql服务

\* 需要管理员运行该cmd

2. 使用无验证方式启动mysql服务： mysqld --skip-grant-tables

3. 打开新的cmd窗口,直接输入mysql命令，敲回车。就可以登录成功

4. use mysql;

5. update user set password = password('你的新密码') where user = 'root';

6. 关闭两个窗口

7. 打开任务管理器，手动结束mysqld.exe 的进程

8. 启动mysql服务

9. 使用新密码登录。

4. 查询用户：

-- 1. 切换到mysql数据库

USE myql;

-- 2. 查询user表

SELECT \* FROM USER;

\* 通配符： % 表示可以在任意主机使用用户登录数据库

2. 权限管理：

1. 查询权限：

-- 查询权限

SHOW GRANTS FOR '用户名'@'主机名';

SHOW GRANTS FOR 'lisi'@'%';

2. 授予权限：

-- 授予权限

grant 权限列表 on 数据库名.表名 to '用户名'@'主机名';

-- 给张三用户授予所有权限，在任意数据库任意表上

GRANT ALL ON \*.\* TO 'zhangsan'@'localhost';

3. 撤销权限：

-- 撤销权限：

revoke 权限列表 on 数据库名.表名 from '用户名'@'主机名';

REVOKE UPDATE ON db3.`account` FROM 'lisi'@'%';

JDBC

## JDBC：

1. 概念：Java DataBase Connectivity Java 数据库连接， Java语言操作数据库

\* JDBC本质：其实是官方（sun公司）定义的一套操作所有关系型数据库的规则，即接口。各个数据库厂商去实现这套接口，提供数据库驱动jar包。我们可以使用这套接口（JDBC）编程，真正执行的代码是驱动jar包中的实现类。

2. 快速入门：

\* 步骤：

1. 导入驱动jar包 mysql-connector-java-5.1.37-bin.jar

1.复制mysql-connector-java-5.1.37-bin.jar到项目的libs目录下

2.右键-->Add As Library

2. 注册驱动

3. 获取数据库连接对象 Connection

4. 定义sql

5. 获取执行sql语句的对象 PreparedStatement

给？赋值

6. 执行sql，接受返回结果

7. 处理结果

8. 释放资源

数据库连接池：

. Druid：数据库连接池实现技术，由阿里巴巴提供的

1. 步骤：

1. 导入jar包 druid-1.0.9.jar

2. 定义配置文件：

\* 是properties形式的

\* 可以叫任意名称，可以放在任意目录下

3. 加载配置文件。Properties

4. 获取数据库连接池对象：通过工厂来来获取 DruidDataSourceFactory

5. 获取连接：getConnection

\* 代码：

//3.加载配置文件

Properties pro = new Properties();

InputStream is = DruidDemo.class.getClassLoader().getResourceAsStream("druid.properties");

pro.load(is);

//4.获取连接池对象

DataSource ds = DruidDataSourceFactory.createDataSource(pro);

//5.获取连接

Connection conn = ds.getConnection();

2. 定义工具类

1. 定义一个类 JDBCUtils

2. 提供静态代码块加载配置文件，初始化连接池对象

3. 提供方法

1. 获取连接方法：通过数据库连接池获取连接

2. 释放资源

3. 获取连接池的方法

\* 代码：

public class JDBCUtils {

//1.定义成员变量 DataSource

private static DataSource ds ;

static{

try {

//1.加载配置文件

Properties pro = new Properties();

pro.load(JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties"));

//2.获取DataSource

ds = DruidDataSourceFactory.createDataSource(pro);

} catch (IOException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 获取连接

\*/

public static Connection getConnection() throws SQLException {

return ds.getConnection();

}

/\*\*

\* 释放资源

\*/

public static void close(Statement stmt,Connection conn){

close(null,stmt,conn);

}

public static void close(ResultSet rs , Statement stmt, Connection conn){

if(rs != null){

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if(stmt != null){

try {

stmt.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

if(conn != null){

try {

conn.close();//归还连接

} catch (SQLException e) {

e.printStackTrace();

}

}

}

/\*\*

\* 获取连接池方法

\*/

public static DataSource getDataSource(){

return ds;

}

}

## Spring JDBC

\* Spring框架对JDBC的简单封装。提供了一个JDBCTemplate对象简化JDBC的开发

\* 步骤：

1. 导入jar包

2. 创建JdbcTemplate对象。依赖于数据源DataSource

\* JdbcTemplate template = new JdbcTemplate(ds);

3. 调用JdbcTemplate的方法来完成CRUD的操作

\* update():执行DML语句。增、删、改语句

\* queryForMap():查询结果将结果集封装为map集合，将列名作为key，将值作为value 将这条记录封装为一个map集合

\* 注意：这个方法查询的结果集长度只能是1

\* queryForList():查询结果将结果集封装为list集合

\* 注意：将每一条记录封装为一个Map集合，再将Map集合装载到List集合中

\* query():查询结果，将结果封装为JavaBean对象

\* query的参数：RowMapper

\* 一般我们使用BeanPropertyRowMapper实现类。可以完成数据到JavaBean的自动封装

\* new BeanPropertyRowMapper<类型>(类型.class)

\* queryForObject：查询结果，将结果封装为对象

\* 一般用于聚合函数的查询

WEB

## HTML标签：表单标签

\* 表单：

\* 概念：用于采集用户输入的数据的。用于和服务器进行交互。

\* form：用于定义表单的。可以定义一个范围，范围代表采集用户数据的范围

\* 属性：

\* action：指定提交数据的URL

\* method:指定提交方式

\* 分类：一共7种，2种比较常用

\* get：

1. 请求参数会在地址栏中显示。会封装到请求行中(HTTP协议后讲解)。

2. 请求参数大小是有限制的。

3. 不太安全。

\* post：

2. 请求参数不会再地址栏中显示。会封装在请求体中(HTTP协议后讲解)

2. 请求参数的大小没有限制。

3. 较为安全。

JAVAScript

2. 基本对象：

1. Function：函数(方法)对象

1. 创建：

1. var fun = new Function(形式参数列表,方法体); //忘掉吧

2.

function 方法名称(形式参数列表){

方法体

}

3.

var 方法名 = function(形式参数列表){

方法体

}

2. 方法：

3. 属性：

length:代表形参的个数

4. 特点：

1. 方法定义是，形参的类型不用写,返回值类型也不写。

2. 方法是一个对象，如果定义名称相同的方法，会覆盖

3. 在JS中，方法的调用只与方法的名称有关，和参数列表无关

4. 在方法声明中有一个隐藏的内置对象（数组），arguments,封装所有的实际参数

5. 调用：

方法名称(实际参数列表);

2. Array:数组对象

1. 创建：

1. var arr = new Array(元素列表);

2. var arr = new Array(默认长度);

3. var arr = [元素列表];

2. 方法

join(参数):将数组中的元素按照指定的分隔符拼接为字符串

push() 向数组的末尾添加一个或更多元素，并返回新的长度。

3. 属性

length:数组的长度

4. 特点：

1. JS中，数组元素的类型可变的。

2. JS中，数组长度可变的。

3. Boolean

4. Date：日期对象

1. 创建：

var date = new Date();

2. 方法：

toLocaleString()：返回当前date对象对应的时间本地字符串格式

getTime():获取毫秒值。返回当前如期对象描述的时间到1970年1月1日零点的毫秒值差

5. Math：数学对象

1. 创建：

\* 特点：Math对象不用创建，直接使用。 Math.方法名();

2. 方法：

random():返回 0 ~ 1 之间的随机数。 含0不含1

ceil(x)：对数进行上舍入。

floor(x)：对数进行下舍入。

round(x)：把数四舍五入为最接近的整数。

3. 属性：

PI

6. Number

7. String

8. RegExp：正则表达式对象

1. 正则表达式：定义字符串的组成规则。

1. 单个字符:[]

如： [a] [ab] [a-zA-Z0-9\_]

\* 特殊符号代表特殊含义的单个字符:

\d:单个数字字符 [0-9]

\w:单个单词字符[a-zA-Z0-9\_]

2. 量词符号：

?：表示出现0次或1次

\*：表示出现0次或多次

+：出现1次或多次

{m,n}:表示 m<= 数量 <= n

\* m如果缺省： {,n}:最多n次

\* n如果缺省：{m,} 最少m次

3. 开始结束符号

\* ^:开始

\* $:结束

2. 正则对象：

1. 创建

1. var reg = new RegExp("正则表达式");

2. var reg = /正则表达式/;

2. 方法

1. test(参数):验证指定的字符串是否符合正则定义的规范

9. Global

1. 特点：全局对象，这个Global中封装的方法不需要对象就可以直接调用。 方法名();

2. 方法：

encodeURI():url编码

decodeURI():url解码

encodeURIComponent():url编码,编码的字符更多

decodeURIComponent():url解码

parseInt():将字符串转为数字

\* 逐一判断每一个字符是否是数字，直到不是数字为止，将前边数字部分转为number

isNaN():判断一个值是否是NaN

\* NaN六亲不认，连自己都不认。NaN参与的==比较全部问false

eval():讲 JavaScript 字符串，并把它作为脚本代码来执行。

3. URL编码

传智播客 = %E4%BC%A0%E6%99%BA%E6%92%AD%E5%AE%A2

# BOM:

1. 概念：Browser Object Model 浏览器对象模型

\* 将浏览器的各个组成部分封装成对象。

2. 组成：

\* Window：窗口对象

\* Navigator：浏览器对象

\* Screen：显示器屏幕对象

\* History：历史记录对象

\* Location：地址栏对象

3. Window：窗口对象

1. 创建

2. 方法

1. 与弹出框有关的方法：

alert() 显示带有一段消息和一个确认按钮的警告框。

confirm() 显示带有一段消息以及确认按钮和取消按钮的对话框。

\* 如果用户点击确定按钮，则方法返回true

\* 如果用户点击取消按钮，则方法返回false

prompt() 显示可提示用户输入的对话框。

\* 返回值：获取用户输入的值

2. 与打开关闭有关的方法：

close() 关闭浏览器窗口。

\* 谁调用我 ，我关谁

open() 打开一个新的浏览器窗口

\* 返回新的Window对象

3. 与定时器有关的方式

setTimeout() 在指定的毫秒数后调用函数或计算表达式。

\* 参数：

1. js代码或者方法对象

2. 毫秒值

\* 返回值：唯一标识，用于取消定时器

clearTimeout() 取消由 setTimeout() 方法设置的 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() 取消由 setInterval() 设置的 timeout。

3. 属性：

1. 获取其他BOM对象：

history

location

Navigator

Screen:

2. 获取DOM对象

document

4. 特点

\* Window对象不需要创建可以直接使用 window使用。 window.方法名();

\* window引用可以省略。 方法名();

4. Location：地址栏对象

1. 创建(获取)：

1. window.location

2. location

2. 方法：

\* reload() 重新加载当前文档。刷新

3. 属性

\* href 设置或返回完整的 URL。

5. History：历史记录对象

1. 创建(获取)：

1. window.history

2. history

2. 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(参数) 加载 history 列表中的某个具体页面。

\* 参数：

\* 正数：前进几个历史记录

\* 负数：后退几个历史记录

3. 属性：

\* length 返回当前窗口历史列表中的 URL 数量。

## DOM：

\* 概念： Document Object Model 文档对象模型

\* 将标记语言文档的各个组成部分，封装为对象。可以使用这些对象，对标记语言文档进行CRUD的动态操作

\* W3C DOM 标准被分为 3 个不同的部分：

\* 核心 DOM - 针对任何结构化文档的标准模型

\* Document：文档对象

\* Element：元素对象

\* 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

\* XML DOM - 针对 XML 文档的标准模型

\* HTML DOM - 针对 HTML 文档的标准模型

\* 核心DOM模型：

\* Document：文档对象

1. 创建(获取)：在html dom模型中可以使用window对象来获取

1. window.document

2. document

2. 方法：

1. 获取Element对象：

1. getElementById() ： 根据id属性值获取元素对象。id属性值一般唯一

2. getElementsByTagName()：根据元素名称获取元素对象们。返回值是一个数组

3. getElementsByClassName():根据Class属性值获取元素对象们。返回值是一个数组

4. getElementsByName(): 根据name属性值获取元素对象们。返回值是一个数组

2. 创建其他DOM对象：

createAttribute(name)

createComment()

createElement()

createTextNode()

3. 属性

\* Element：元素对象

1. 获取/创建：通过document来获取和创建

2. 方法：

1. removeAttribute()：删除属性

2. setAttribute()：设置属性

\* Node：节点对象，其他5个的父对象

\* 特点：所有dom对象都可以被认为是一个节点

\* 方法：

\* CRUD dom树：

\* 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 返回节点的父节点。

\* HTML DOM

1. 标签体的设置和获取：innerHTML

2. 使用html元素对象的属性

3. 控制元素样式

1. 使用元素的style属性来设置

如：

//修改样式方式1

div1.style.border = "1px solid red";

div1.style.width = "200px";

//font-size--> fontSize

div1.style.fontSize = "20px";

2. 提前定义好类选择器的样式，通过元素的className属性来设置其class属性值。

## 事件监听机制：

\* 概念：某些组件被执行了某些操作后，触发某些代码的执行。

\* 事件：某些操作。如： 单击，双击，键盘按下了，鼠标移动了

\* 事件源：组件。如： 按钮 文本输入框...

\* 监听器：代码。

\* 注册监听：将事件，事件源，监听器结合在一起。 当事件源上发生了某个事件，则触发执行某个监听器代码。

\* 常见的事件：

1. 点击事件：

1. onclick：单击事件

2. ondblclick：双击事件

2. 焦点事件

1. onblur：失去焦点

2. onfocus:元素获得焦点。

3. 加载事件：

1. onload：一张页面或一幅图像完成加载。

4. 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。

5. onmouseout 鼠标从某元素移开。

5. 键盘事件：

1. onkeydown 某个键盘按键被按下。

2. onkeyup 某个键盘按键被松开。

3. onkeypress 某个键盘按键被按下并松开。

6. 选择和改变

1. onchange 域的内容被改变。

2. onselect 文本被选中。

7. 表单事件：

1. onsubmit 确认按钮被点击。

2. onreset 重置按钮被点击。

XML：

## XML：

1. 概念：Extensible Markup Language 可扩展标记语言

\* 可扩展：标签都是自定义的。 <user> <student>

\* 功能

\* 存储数据

1. 配置文件

2. 在网络中传输

\* xml与html的区别

1. xml标签都是自定义的，html标签是预定义。

2. xml的语法严格，html语法松散

3. xml是存储数据的，html是展示数据

\* 组成部分：

1. 文档声明

1. 格式：<?xml 属性列表 ?>

2. 属性列表：

\* version：版本号，必须的属性

\* encoding：编码方式。告知解析引擎当前文档使用的字符集，默认值：ISO-8859-1

\* standalone：是否独立

\* 取值：

\* yes：不依赖其他文件

\* no：依赖其他文件

2. 指令(了解)：结合css的

\* <?xml-stylesheet type="text/css" href="a.css" ?>

3. 标签：标签名称自定义的

\* 规则：

\* 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、Xml 等等）开始

\* 名称不能包含空格

\* xml常见的解析器：

1. JAXP：sun公司提供的解析器，支持dom和sax两种思想

2. DOM4J：一款非常优秀的解析器

3. 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。

4. PULL：Android操作系统内置的解析器，sax方式的。

\* 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。

\* 快速入门：

\* 步骤：

1. 导入jar包

2. 获取Document对象

3. 获取对应的标签Element对象

4. 获取数据

\* 代码：

//2.1获取student.xml的path

String path = JsoupDemo1.class.getClassLoader().getResource("student.xml").getPath();

//2.2解析xml文档，加载文档进内存，获取dom树--->Document

Document document = Jsoup.parse(new File(path), "utf-8");

//3.获取元素对象 Element

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

System.out.println(elements.size());

//3.1获取第一个name的Element对象

Element element = elements.get(0);

//3.2获取数据

String name = element.text();

Tomcat/servlet

3. 网络通信三要素

1. IP：电子设备(计算机)在网络中的唯一标识。

2. 端口：应用程序在计算机中的唯一标识。 0~65536

3. 传输协议：规定了数据传输的规则

1. 基础协议：

1. tcp:安全协议，三次握手。 速度稍慢

2. udp：不安全协议。 速度快

6. 配置:

\* 部署项目的方式：

1. 直接将项目放到webapps目录下即可。

\* /hello：项目的访问路径-->虚拟目录

\* 简化部署：将项目打成一个war包，再将war包放置到webapps目录下。

\* war包会自动解压缩

2. 配置conf/server.xml文件

在<Host>标签体中配置

<Context docBase="D:\hello" path="/hehe" />

\* docBase:项目存放的路径

\* path：虚拟目录

3. 在conf\Catalina\localhost创建任意名称的xml文件。在文件中编写

<Context docBase="D:\hello" />

\* 虚拟目录：xml文件的名称

\* 静态项目和动态项目：

\* 目录结构

\* java动态项目的目录结构：

-- 项目的根目录

-- WEB-INF目录：

-- web.xml：web项目的核心配置文件

-- classes目录：放置字节码文件的目录

-- lib目录：放置依赖的jar包

## Servlet： server applet

\* 概念：运行在服务器端的小程序

\* Servlet就是一个接口，定义了Java类被浏览器访问到(tomcat识别)的规则。

\* 将来我们自定义一个类，实现Servlet接口，复写方法。

\* 快速入门：

1. 创建JavaEE项目

2. 定义一个类，实现Servlet接口

\* public class ServletDemo1 implements Servlet

3. 实现接口中的抽象方法

4. 配置Servlet

在web.xml中配置：

<!--配置Servlet -->

<servlet>

<servlet-name>demo1</servlet-name>

<servlet-class>cn.itcast.web.servlet.ServletDemo1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>demo1</servlet-name>

<url-pattern>/demo1</url-pattern>

</servlet-mapping>

\* 执行原理：

1. 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

2. 查找web.xml文件，是否有对应的<url-pattern>标签体内容。

3. 如果有，则在找到对应的<servlet-class>全类名

4. tomcat会将字节码文件加载进内存，并且创建其对象

5. 调用其方法

\* Servlet中的生命周期方法：

1. 被创建：执行init方法，只执行一次

\* Servlet什么时候被创建？

\* 默认情况下，第一次被访问时，Servlet被创建

\* 可以配置执行Servlet的创建时机。

\* 在<servlet>标签下配置

1. 第一次被访问时，创建

\* <load-on-startup>的值为负数

2. 在服务器启动时，创建

\* <load-on-startup>的值为0或正整数

\* Servlet的init方法，只执行一次，说明一个Servlet在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，也不要对修改值

2. 提供服务：执行service方法，执行多次

\* 每次访问Servlet时，Service方法都会被调用一次。

3. 被销毁：执行destroy方法，只执行一次

\* Servlet被销毁时执行。服务器关闭时，Servlet被销毁

\* 只有服务器正常关闭时，才会执行destroy方法。

\* destroy方法在Servlet被销毁之前执行，一般用于释放资源

\* Servlet3.0：

\* 好处：

\* 支持注解配置。可以不需要web.xml了。

\* 步骤：

1. 创建JavaEE项目，选择Servlet的版本3.0以上，可以不创建web.xml

2. 定义一个类，实现Servlet接口

3. 复写方法

4. 在类上使用@WebServlet注解，进行配置

\* @WebServlet("资源路径")

@Target({ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface WebServlet {

String name() default "";//相当于<Servlet-name>

String[] value() default {};//代表urlPatterns()属性配置

String[] urlPatterns() default {};//相当于<url-pattern>

int loadOnStartup() default -1;//相当于<load-on-startup>

WebInitParam[] initParams() default {};

boolean asyncSupported() default false;

String smallIcon() default "";

String largeIcon() default "";

String description() default "";

String displayName() default "";

}

Servlet

## Servlet：

1. 概念

2. 步骤

3. 执行原理

4. 生命周期

5. Servlet3.0 注解配置

6. Servlet的体系结构

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()方法作为抽象

\* 将来定义Servlet类时，可以继承GenericServlet，实现service()方法即可

\* HttpServlet：对http协议的一种封装，简化操作

1. 定义类继承HttpServlet

2. 复写doGet/doPost方法

7. Servlet相关配置

1. urlpartten:Servlet访问路径

1. 一个Servlet可以定义多个访问路径 ： @WebServlet({"/d4","/dd4","/ddd4"})

2. 路径定义规则：

1. /xxx：路径匹配

2. /xxx/xxx:多层路径，目录结构

3. \*.do：扩展名匹配

## HTTP：

\* 概念：Hyper Text Transfer Protocol 超文本传输协议

\* 传输协议：定义了，客户端和服务器端通信时，发送数据的格式

\* 特点：

1. 基于TCP/IP的高级协议

2. 默认端口号:80

3. 基于请求/响应模型的:一次请求对应一次响应

4. 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：复用连接

\* 请求消息数据格式

1. 请求行

请求方式 请求url 请求协议/版本

GET /login.html HTTP/1.1

\* 请求方式：

\* HTTP协议有7中请求方式，常用的有2种

\* GET：

1. 请求参数在请求行中，在url后。

2. 请求的url长度有限制的

3. 不太安全

\* POST：

1. 请求参数在请求体中

2. 请求的url长度没有限制的

3. 相对安全

2. 请求头：客户端浏览器告诉服务器一些信息

请求头名称: 请求头值

\* 常见的请求头：

1. User-Agent：浏览器告诉服务器，我访问你使用的浏览器版本信息

\* 可以在服务器端获取该头的信息，解决浏览器的兼容性问题

2. Referer：http://localhost/login.html

\* 告诉服务器，我(当前请求)从哪里来？

\* 作用：

1. 防盗链：

2. 统计工作：

3. 请求空行

空行，就是用于分割POST请求的请求头，和请求体的。

4. 请求体(正文)：

\* 封装POST请求消息的请求参数的

\* 字符串格式：

POST /login.html HTTP/1.1

Host: localhost

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

Referer: http://localhost/login.html

Connection: keep-alive

Upgrade-Insecure-Requests: 1

username=zhangsan

\* 响应消息数据格式

## Request：

1. request对象和response对象的原理

1. request和response对象是由服务器创建的。我们来使用它们

2. request对象是来获取请求消息，response对象是来设置响应消息

2. request对象继承体系结构：

ServletRequest -- 接口

| 继承

HttpServletRequest -- 接口

| 实现

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

3. request功能：

1. 获取请求消息数据

1. 获取请求行数据

\* GET /day14/demo1?name=zhangsan HTTP/1.1

\* 方法：

1. 获取请求方式 ：GET

\* String getMethod()

2. (\*)获取虚拟目录：/day14

\* String getContextPath()

3. 获取Servlet路径: /demo1

\* String getServletPath()

4. 获取get方式请求参数：name=zhangsan

\* String getQueryString()

5. (\*)获取请求URI：/day14/demo1

\* String getRequestURI(): /day14/demo1

\* StringBuffer getRequestURL() :http://localhost/day14/demo1

\* URL:统一资源定位符 ： http://localhost/day14/demo1 中华人民共和国

\* URI：统一资源标识符 : /day14/demo1 共和国

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()

2. 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

3. 获取请求体数据:

\* 请求体：只有POST请求方式，才有请求体，在请求体中封装了POST请求的请求参数

\* 步骤：

1. 获取流对象

\* BufferedReader getReader()：获取字符输入流，只能操作字符数据

\* ServletInputStream getInputStream()：获取字节输入流，可以操作所有类型数据

\* 在文件上传知识点后讲解

2. 再从流对象中拿数据

2. 其他功能：

1. 获取请求参数通用方式：不论get还是post请求方式都可以使用下列方法来获取请求参数

1. String getParameter(String name):根据参数名称获取参数值 username=zs&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. Enumeration<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

\* 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");

2. 请求转发：一种在服务器内部的资源跳转方式

1. 步骤：

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)

2. 特点：

1. 浏览器地址栏路径不发生变化

2. 只能转发到当前服务器内部资源中。

3. 转发是一次请求

3. 共享数据：

\* 域对象：一个有作用范围的对象，可以在范围内共享数据

\* request域：代表一次请求的范围，一般用于请求转发的多个资源中共享数据

\* 方法：

1. void setAttribute(String name,Object obj):存储数据

2. Object getAttitude(String name):通过键获取值

3. void removeAttribute(String name):通过键移除键值对

4. 获取ServletContext：

\* ServletContext getServletContext()

# 今日内容

1. HTTP协议：响应消息

2. Response对象

3. ServletContext对象

## HTTP协议：

1. 请求消息：客户端发送给服务器端的数据

\* 数据格式：

1. 请求行

2. 请求头

3. 请求空行

4. 请求体

2. 响应消息：服务器端发送给客户端的数据

\* 数据格式：

1. 响应行

1. 组成：协议/版本 响应状态码 状态码描述

2. 响应状态码：服务器告诉客户端浏览器本次请求和响应的一个状态。

1. 状态码都是3位数字

2. 分类：

1. 1xx：服务器就收客户端消息，但没有接受完成，等待一段时间后，发送1xx多状态码

2. 2xx：成功。代表：200

3. 3xx：重定向。代表：302(重定向)，304(访问缓存)

4. 4xx：客户端错误。

\* 代表：

\* 404（请求路径没有对应的资源）

\* 405：请求方式没有对应的doXxx方法

5. 5xx：服务器端错误。代表：500(服务器内部出现异常)

2. 响应头：

1. 格式：头名称： 值

2. 常见的响应头：

1. Content-Type：服务器告诉客户端本次响应体数据格式以及编码格式

2. Content-disposition：服务器告诉客户端以什么格式打开响应体数据

\* 值：

\* in-line:默认值,在当前页面内打开

\* attachment;filename=xxx：以附件形式打开响应体。文件下载

3. 响应空行

4. 响应体:传输的数据

\* 响应字符串格式

HTTP/1.1 200 OK

Content-Type: text/html;charset=UTF-8

Content-Length: 101

Date: Wed, 06 Jun 2018 07:08:42 GMT

<html>

<head>

<title>$Title$</title>

</head>

<body>

hello , response

</body>

</html>

## Response对象

\* 功能：设置响应消息

1. 设置响应行

1. 格式：HTTP/1.1 200 ok

2. 设置状态码：setStatus(int sc)

2. 设置响应头：setHeader(String name, String value)

3. 设置响应体：

\* 使用步骤：

1. 获取输出流

\* 字符输出流：PrintWriter getWriter()

\* 字节输出流：ServletOutputStream getOutputStream()

2. 使用输出流，将数据输出到客户端浏览器

\* 案例：

1. 完成重定向

\* 重定向：资源跳转的方式

\* 代码实现：

//1. 设置状态码为302

response.setStatus(302);

//2.设置响应头location

response.setHeader("location","/day15/responseDemo2");

//简单的重定向方法

response.sendRedirect("/day15/responseDemo2");

\* 重定向的特点:redirect

1. 地址栏发生变化

2. 重定向可以访问其他站点(服务器)的资源

3. 重定向是两次请求。不能使用request对象来共享数据

\* 转发的特点：forward

1. 转发地址栏路径不变

2. 转发只能访问当前服务器下的资源

3. 转发是一次请求，可以使用request对象来共享数据

\* forward 和 redirect 区别

\* 路径写法：

1. 路径分类

1. 相对路径：通过相对路径不可以确定唯一资源

\* 如：./index.html

\* 不以/开头，以.开头路径

\* 规则：找到当前资源和目标资源之间的相对位置关系

\* ./：当前目录

\* ../:后退一级目录

2. 绝对路径：通过绝对路径可以确定唯一资源

\* 如：http://localhost/day15/responseDemo2 /day15/responseDemo2

\* 以/开头的路径

\* 规则：判断定义的路径是给谁用的？判断请求将来从哪儿发出

\* 给客户端浏览器使用：需要加虚拟目录(项目的访问路径)

\* 建议虚拟目录动态获取：request.getContextPath()

\* <a> , <form> 重定向...

\* 给服务器使用：不需要加虚拟目录

\* 转发路径

2. 服务器输出字符数据到浏览器

\* 步骤：

1. 获取字符输出流

2. 输出数据

\* 注意：

\* 乱码问题：

1. PrintWriter pw = response.getWriter();获取的流的默认编码是ISO-8859-1

2. 设置该流的默认编码

3. 告诉浏览器响应体使用的编码

//简单的形式，设置编码，是在获取流之前设置

response.setContentType("text/html;charset=utf-8");

3. 服务器输出字节数据到浏览器

\* 步骤：

1. 获取字节输出流

2. 输出数据

4. 验证码

1. 本质：图片

2. 目的：防止恶意表单注册

## ServletContext对象：

1. 概念：代表整个web应用，可以和程序的容器(服务器)来通信

2. 获取：

1. 通过request对象获取

request.getServletContext();

2. 通过HttpServlet获取

this.getServletContext();

3. 功能：

1. 获取MIME类型：

\* MIME类型:在互联网通信过程中定义的一种文件数据类型

\* 格式： 大类型/小类型 text/html image/jpeg

\* 获取：String getMimeType(String file)

2. 域对象：共享数据

1. setAttribute(String name,Object value)

2. getAttribute(String name)

3. removeAttribute(String name)

\* ServletContext对象范围：所有用户所有请求的数据

3. 获取文件的真实(服务器)路径

1. 方法：String getRealPath(String path)

String b = context.getRealPath("/b.txt");//web目录下资源访问

System.out.println(b);

String c = context.getRealPath("/WEB-INF/c.txt");//WEB-INF目录下的资源访问

System.out.println(c);

String a = context.getRealPath("/WEB-INF/classes/a.txt");//src目录下的资源访问

System.out.println(a);

Cookie和Session

# 今日内容

1. 会话技术

1. Cookie

2. Session

2. JSP：入门学习

## 会话技术

1. 会话：一次会话中包含多次请求和响应。

\* 一次会话：浏览器第一次给服务器资源发送请求，会话建立，直到有一方断开为止

2. 功能：在一次会话的范围内的多次请求间，共享数据

3. 方式：

1. 客户端会话技术：Cookie

2. 服务器端会话技术：Session

## Cookie：

1. 概念：客户端会话技术，将数据保存到客户端

2. 快速入门：

\* 使用步骤：

1. 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)

3. 获取Cookie，拿到数据

\* Cookie[] request.getCookies()

3. 实现原理

\* 基于响应头set-cookie和请求头cookie实现

4. cookie的细节

1. 一次可不可以发送多个cookie?

\* 可以

\* 可以创建多个Cookie对象，使用response调用多次addCookie方法发送cookie即可。

2. cookie在浏览器中保存多长时间？

1. 默认情况下，当浏览器关闭后，Cookie数据被销毁

2. 持久化存储：

\* setMaxAge(int seconds)

1. 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

2. 负数：默认值

3. 零：删除cookie信息

3. cookie能不能存中文？

\* 在tomcat 8 之前 cookie中不能直接存储中文数据。

\* 需要将中文数据转码---一般采用URL编码(%E3)

\* 在tomcat 8 之后，cookie支持中文数据。特殊字符还是不支持，建议使用URL编码存储，URL解码解析

4. cookie共享问题？

1. 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"/"

2. 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

\* setDomain(".baidu.com"),那么tieba.baidu.com和news.baidu.com中cookie可以共享

5. Cookie的特点和作用

1. cookie存储数据在客户端浏览器

2. 浏览器对于单个cookie 的大小有限制(4kb) 以及 对同一个域名下的总cookie数量也有限制(20个)

\* 作用：

1. cookie一般用于存出少量的不太敏感的数据

2. 在不登录的情况下，完成服务器对客户端的身份识别

## JSP：入门学习

1. 概念：

\* Java Server Pages： java服务器端页面

\* 可以理解为：一个特殊的页面，其中既可以指定定义html标签，又可以定义java代码

\* 用于简化书写！！！

2. 原理

\* JSP本质上就是一个Servlet

3. JSP的脚本：JSP定义Java代码的方式

1. <% 代码 %>：定义的java代码，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。

2. <%! 代码 %>：定义的java代码，在jsp转换后的java类的成员位置。

3. <%= 代码 %>：定义的java代码，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。

4. JSP的内置对象：

\* 在jsp页面中不需要获取和创建，可以直接使用的对象

\* jsp一共有9个内置对象。

\* 今天学习3个：

\* request

\* response

\* out：字符输出流对象。可以将数据输出到页面上。和response.getWriter()类似

\* response.getWriter()和out.write()的区别：

\* 在tomcat服务器真正给客户端做出响应之前，会先找response缓冲区数据，再找out缓冲区数据。

\* response.getWriter()数据输出永远在out.write()之前

5. 案例:改造Cookie案例

## Session：主菜

1. 概念：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

2. 快速入门：

1. 获取HttpSession对象：

HttpSession session = request.getSession();

2. 使用HttpSession对象：

Object getAttribute(String name)

void setAttribute(String name, Object value)

void removeAttribute(String name)

3. 原理

\* Session的实现是依赖于Cookie的。

4. 细节：

1. 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);

2. 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。

3. session什么时候被销毁？

1. 服务器关闭

2. session对象调用invalidate() 。

3. session默认失效时间 30分钟

选择性配置修改

<session-config>

<session-timeout>30</session-timeout>

</session-config>

5. session的特点

1. session用于存储一次会话的多次请求的数据，存在服务器端

2. session可以存储任意类型，任意大小的数据

\* session与Cookie的区别：

1. session存储数据在服务器端，Cookie在客户端

2. session没有数据大小限制，Cookie有

3. session数据安全，Cookie相对于不安全

AJAX和JSON

# AJAX：

1. 概念： ASynchronous JavaScript And XML 异步的JavaScript 和 XML

1. 异步和同步：客户端和服务器端相互通信的基础上

\* 客户端必须等待服务器端的响应。在等待的期间客户端不能做其他操作。

\* 客户端不需要等待服务器端的响应。在服务器处理请求的过程中，客户端可以进行其他的操作。

Ajax 是一种在无需重新加载整个网页的情况下，能够更新部分网页的技术。 [1]

通过在后台与服务器进行少量数据交换，Ajax 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

传统的网页（不使用 Ajax）如果需要更新内容，必须重载整个网页页面。

提升用户的体验

JQeury实现方式

1. $.ajax()

\* 语法：$.ajax({键值对});

//使用$.ajax()发送异步请求

$.ajax({

url:"ajaxServlet1111" , // 请求路径

type:"POST" , //请求方式

//data: "username=jack&age=23",//请求参数

data:{"username":"jack","age":23},

success:function (data) {

alert(data);

},//响应成功后的回调函数

error:function () {

alert("出错啦...")

},//表示如果请求响应出现错误，会执行的回调函数

dataType:"text"//设置接受到的响应数据的格式

});

2. $.get()：发送get请求

\* 语法：$.get(url, [data], [callback], [type])

\* 参数：

\* url：请求路径

\* data：请求参数

\* callback：回调函数

\* type：响应结果的类型

3. $.post()：发送post请求

\* 语法：$.post(url, [data], [callback], [type])

\* 参数：

\* url：请求路径

\* data：请求参数

\* callback：回调函数

\* type：响应结果的类型

# JSON：

1. 概念： JavaScript Object Notation JavaScript对象表示法

Person p = new Person();

p.setName("张三");

p.setAge(23);

p.setGender("男");

var p = {"name":"张三","age":23,"gender":"男"};

\* json现在多用于存储和交换文本信息的语法

\* 进行数据的传输

\* JSON 比 XML 更小、更快，更易解析。

2. 语法：

1. 基本规则

\* 数据在名称/值对中：json数据是由键值对构成的

\* 键用引号(单双都行)引起来，也可以不使用引号

\* 值得取值类型：

1. 数字（整数或浮点数）

2. 字符串（在双引号中）

3. 逻辑值（true 或 false）

4. 数组（在方括号中） {"persons":[{},{}]}

5. 对象（在花括号中） {"address":{"province"："陕西"....}}

6. null

\* 数据由逗号分隔：多个键值对由逗号分隔

\* 花括号保存对象：使用{}定义json 格式

\* 方括号保存数组：[]

2. 获取数据:

1. json对象.键名

2. json对象["键名"]

3. 数组对象[索引]

4. 遍历

//1.定义基本格式

var person = {"name": "张三", age: 23, 'gender': true};

var ps = [{"name": "张三", "age": 23, "gender": true},

{"name": "李四", "age": 24, "gender": true},

{"name": "王五", "age": 25, "gender": false}];

//获取person对象中所有的键和值

//for in 循环

/\* for(var key in person){

//这样的方式获取不行。因为相当于 person."name"

//alert(key + ":" + person.key);

alert(key+":"+person[key]);

}\*/

//获取ps中的所有值

for (var i = 0; i < ps.length; i++) {

var p = ps[i];

for(var key in p){

alert(key+":"+p[key]);

}

}

3. JSON数据和Java对象的相互转换

\* JSON解析器：

\* 常见的解析器：Jsonlib，Gson，fastjson，jackson

1. JSON转为Java对象

1. 导入jackson的相关jar包

2. 创建Jackson核心对象 ObjectMapper

3. 调用ObjectMapper的相关方法进行转换

1. readValue(json字符串数据,Class)

2. Java对象转换JSON

1. 使用步骤：

1. 导入jackson的相关jar包

2. 创建Jackson核心对象 ObjectMapper

ObjectMapper mapper=new ObjectMapper();

3. 调用ObjectMapper的相关方法进行转换

1. 转换方法：

\* writeValue(参数1，obj):

参数1：

File：将obj对象转换为JSON字符串，并保存到指定的文件中

Writer：将obj对象转换为JSON字符串，并将json数据填充到字符输出流中

OutputStream：将obj对象转换为JSON字符串，并将json数据填充到字节输出流中

\* writeValueAsString(obj):将对象转为json字符串

String json=Mapper.writeValueAsString(p);

2. 注解：

1. @JsonIgnore：排除属性。

2. @JsonFormat：属性值得格式化

\* @JsonFormat(pattern = "yyyy-MM-dd")

3. 复杂java对象转换

1. List：数组

2. Map：对象格式一致

Redis

什么是Redis

Redis是用C语言开发的一个开源的高性能键值对（key-value）数据库，官方提供测试数据，50个并发执行100000个请求,读的速度是110000次/s,写的速度是81000次/s ，且Redis通过提供多种键值数据类型来适应不同场景下的存储需求，目前为止Redis支持的键值数据类型如下：

1) 字符串类型 string

2) 哈希类型 hash

3) 列表类型 list

4) 集合类型 set

5) 有序集合类型 sortedset

1.3.1 redis的应用场景

• 缓存（数据查询、短连接、新闻内容、商品内容等等）

• 聊天室的在线好友列表

• 任务队列。（秒杀、抢购、12306等等）

• 应用排行榜

• 网站访问统计

• 数据过期处理（可以精确到毫秒

• 分布式集群架构中的session分离

## 今日内容

1. redis

1. 概念

2. 下载安装

3. 命令操作

1. 数据结构

4. 持久化操作

5. 使用Java客户端操作redis

# Redis

1. 概念： redis是一款高性能的NOSQL系列的非关系型数据库

1.1.什么是NOSQL

NoSQL(NoSQL = Not Only SQL)，意即“不仅仅是SQL”，是一项全新的数据库理念，泛指非关系型的数据库。

随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。

1.1.1. NOSQL和关系型数据库比较

优点：

1）成本：nosql数据库简单易部署，基本都是开源软件，不需要像使用oracle那样花费大量成本购买使用，相比关系型数据库价格便宜。

2）查询速度：nosql数据库将数据存储于缓存之中，关系型数据库将数据存储在硬盘中，自然查询速度远不及nosql数据库。

3）存储数据的格式：nosql的存储格式是key,value形式、文档形式、图片形式等等，所以可以存储基础类型以及对象或者是集合等各种格式，而数据库则只支持基础类型。

4）扩展性：关系型数据库有类似join这样的多表查询机制的限制导致扩展很艰难。

缺点：

1）维护的工具和资料有限，因为nosql是属于新的技术，不能和关系型数据库10几年的技术同日而语。

2）不提供对sql的支持，如果不支持sql这样的工业标准，将产生一定用户的学习和使用成本。

3）不提供关系型数据库对事务的处理。

1.1.2. 非关系型数据库的优势：

1）性能NOSQL是基于键值对的，可以想象成表中的主键和值的对应关系，而且不需要经过SQL层的解析，所以性能非常高。

2）可扩展性同样也是因为基于键值对，数据之间没有耦合性，所以非常容易水平扩展。

1.1.3. 关系型数据库的优势：

1）复杂查询可以用SQL语句方便的在一个表以及多个表之间做非常复杂的数据查询。

2）事务支持使得对于安全性能很高的数据访问要求得以实现。对于这两类数据库，对方的优势就是自己的弱势，反之亦然。

1.1.4. 总结

关系型数据库与NoSQL数据库并非对立而是互补的关系，即通常情况下使用关系型数据库，在适合使用NoSQL的时候使用NoSQL数据库，

让NoSQL数据库对关系型数据库的不足进行弥补。

一般会将数据存储在关系型数据库中，在nosql数据库中备份存储关系型数据库的数据

1.2.主流的NOSQL产品

• 键值(Key-Value)存储数据库

相关产品： Tokyo Cabinet/Tyrant、Redis、Voldemort、Berkeley DB

典型应用： 内容缓存，主要用于处理大量数据的高访问负载。

数据模型： 一系列键值对

优势： 快速查询

劣势： 存储的数据缺少结构化

• 列存储数据库

相关产品：Cassandra, HBase, Riak

典型应用：分布式的文件系统

数据模型：以列簇式存储，将同一列数据存在一起

优势：查找速度快，可扩展性强，更容易进行分布式扩展

劣势：功能相对局限

• 文档型数据库

相关产品：CouchDB、MongoDB

典型应用：Web应用（与Key-Value类似，Value是结构化的）

数据模型： 一系列键值对

优势：数据结构要求不严格

劣势： 查询性能不高，而且缺乏统一的查询语法

• 图形(Graph)数据库

相关数据库：Neo4J、InfoGrid、Infinite Graph

典型应用：社交网络

数据模型：图结构

优势：利用图结构相关算法。

劣势：需要对整个图做计算才能得出结果，不容易做分布式的集群方案。

1.3 什么是Redis

Redis是用C语言开发的一个开源的高性能键值对（key-value）数据库，官方提供测试数据，50个并发执行100000个请求,读的速度是110000次/s,写的速度是81000次/s ，且Redis通过提供多种键值数据类型来适应不同场景下的存储需求，目前为止Redis支持的键值数据类型如下：

1) 字符串类型 string

2) 哈希类型 hash

3) 列表类型 list

4) 集合类型 set

5) 有序集合类型 sortedset

1.3.1 redis的应用场景

• 缓存（数据查询、短连接、新闻内容、商品内容等等）

• 聊天室的在线好友列表

• 任务队列。（秒杀、抢购、12306等等）

• 应用排行榜

• 网站访问统计

• 数据过期处理（可以精确到毫秒

• 分布式集群架构中的session分离

2. 下载安装

1. 官网：https://redis.io

2. 中文网：http://www.redis.net.cn/

3. 解压直接可以使用：

\* redis.windows.conf：配置文件

\* redis-cli.exe：redis的客户端

\* redis-server.exe：redis服务器端

3. 命令操作

1. redis的数据结构：

\* redis存储的是：key,value格式的数据，其中key都是字符串，value有5种不同的数据结构

\* value的数据结构：

1) 字符串类型 string

2) 哈希类型 hash ： map格式

3) 列表类型 list ： linkedlist格式。支持重复元素

4) 集合类型 set ： 不允许重复元素

5) 有序集合类型 sortedset：不允许重复元素，且元素有顺序

2. 字符串类型 string

1. 存储： set key value

127.0.0.1:6379> set username zhangsan

OK

2. 获取： get key

127.0.0.1:6379> get username

"zhangsan"

3. 删除： del key

127.0.0.1:6379> del age

(integer) 1

3. 哈希类型 hash

1. 存储： hset key field value

127.0.0.1:6379> hset myhash username lisi

(integer) 1

127.0.0.1:6379> hset myhash password 123

(integer) 1

2. 获取：

\* hget key field: 获取指定的field对应的值

127.0.0.1:6379> hget myhash username

"lisi"

\* hgetall key：获取所有的field和value

127.0.0.1:6379> hgetall myhash

1) "username"

2) "lisi"

3) "password"

4) "123"

3. 删除： hdel key field

127.0.0.1:6379> hdel myhash username

(integer) 1

4. 列表类型 list:可以添加一个元素到列表的头部（左边）或者尾部（右边）

1. 添加：

1. lpush key value: 将元素加入列表左表

2. rpush key value：将元素加入列表右边

127.0.0.1:6379> lpush myList a

(integer) 1

127.0.0.1:6379> lpush myList b

(integer) 2

127.0.0.1:6379> rpush myList c

(integer) 3

2. 获取：

\* lrange key start end ：范围获取

127.0.0.1:6379> lrange myList 0 -1

1) "b"

2) "a"

3) "c"

3. 删除：

\* lpop key： 删除列表最左边的元素，并将元素返回

\* rpop key： 删除列表最右边的元素，并将元素返回

5. 集合类型 set ： 不允许重复元素

1. 存储：sadd key value

127.0.0.1:6379> sadd myset a

(integer) 1

127.0.0.1:6379> sadd myset a

(integer) 0

2. 获取：smembers key:获取set集合中所有元素

127.0.0.1:6379> smembers myset

1) "a"

3. 删除：srem key value:删除set集合中的某个元素

127.0.0.1:6379> srem myset a

(integer) 1

6. 有序集合类型 sortedset：不允许重复元素，且元素有顺序.每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

1. 存储：zadd key score value

127.0.0.1:6379> zadd mysort 60 zhangsan

(integer) 1

127.0.0.1:6379> zadd mysort 50 lisi

(integer) 1

127.0.0.1:6379> zadd mysort 80 wangwu

(integer) 1

2. 获取：zrange key start end [withscores]

127.0.0.1:6379> zrange mysort 0 -1

1) "lisi"

2) "zhangsan"

3) "wangwu"

127.0.0.1:6379> zrange mysort 0 -1 withscores

1) "zhangsan"

2) "60"

3) "wangwu"

4) "80"

5) "lisi"

6) "500"

3. 删除：zrem key value

127.0.0.1:6379> zrem mysort lisi

(integer) 1

7. 通用命令

1. keys \* : 查询所有的键

2. type key ： 获取键对应的value的类型

3. del key：删除指定的key value

4. 持久化

1. redis是一个内存数据库，当redis服务器重启，获取电脑重启，数据会丢失，我们可以将redis内存中的数据持久化保存到硬盘的文件中。

2. redis持久化机制：

1. RDB：默认方式，不需要进行配置，默认就使用这种机制

\* 在一定的间隔时间中，检测key的变化情况，然后持久化数据

1. 编辑redis.windwos.conf文件

# after 900 sec (15 min) if at least 1 key changed

save 900 1

# after 300 sec (5 min) if at least 10 keys changed

save 300 10

# after 60 sec if at least 10000 keys changed

save 60 10000

2. 重新启动redis服务器，并指定配置文件名称

D:\JavaWeb2018\day23\_redis\资料\redis\windows-64\redis-2.8.9>redis-server.exe redis.windows.conf

2. AOF：日志记录的方式，可以记录每一条命令的操作。可以每一次命令操作后，持久化数据

1. 编辑redis.windwos.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

5. Java客户端 Jedis

\* Jedis: 一款java操作redis数据库的工具.

\* 使用步骤：

1. 下载jedis的jar包

2. 使用

//1. 获取连接

Jedis jedis = new Jedis("localhost",6379);

//2. 操作

jedis.set("username","zhangsan");

//3. 关闭连接

jedis.close();

\* Jedis操作各种redis中的数据结构

1) 字符串类型 string

set

get

//1. 获取连接

Jedis jedis = new Jedis();//如果使用空参构造，默认值 "localhost",6379端口

//2. 操作

//存储

jedis.set("username","zhangsan");

//获取

String username = jedis.get("username");

System.out.println(username);

//可以使用setex()方法存储可以指定过期时间的 key value

jedis.setex("activecode",20,"hehe");//将activecode：hehe键值对存入redis，并且20秒后自动删除该键值对

//3. 关闭连接

jedis.close();

2) 哈希类型 hash ： map格式

hset

hget

hgetAll

//1. 获取连接

Jedis jedis = new Jedis();//如果使用空参构造，默认值 "localhost",6379端口

//2. 操作

// 存储hash

jedis.hset("user","name","lisi");

jedis.hset("user","age","23");

jedis.hset("user","gender","female");

// 获取hash

String name = jedis.hget("user", "name");

System.out.println(name);

// 获取hash的所有map中的数据

Map<String, String> user = jedis.hgetAll("user");

// keyset

Set<String> keySet = user.keySet();

for (String key : keySet) {

//获取value

String value = user.get(key);

System.out.println(key + ":" + value);

}

//3. 关闭连接

jedis.close();

3) 列表类型 list ： linkedlist格式。支持重复元素

lpush / rpush

lpop / rpop

lrange start end : 范围获取

//1. 获取连接

Jedis jedis = new Jedis();//如果使用空参构造，默认值 "localhost",6379端口

//2. 操作

// list 存储

jedis.lpush("mylist","a","b","c");//从左边存

jedis.rpush("mylist","a","b","c");//从右边存

// list 范围获取

List<String> mylist = jedis.lrange("mylist", 0, -1);

System.out.println(mylist);

// list 弹出

String element1 = jedis.lpop("mylist");//c

System.out.println(element1);

String element2 = jedis.rpop("mylist");//c

System.out.println(element2);

// list 范围获取

List<String> mylist2 = jedis.lrange("mylist", 0, -1);

System.out.println(mylist2);

//3. 关闭连接

jedis.close();

4) 集合类型 set ： 不允许重复元素

sadd

smembers:获取所有元素

//1. 获取连接

Jedis jedis = new Jedis();//如果使用空参构造，默认值 "localhost",6379端口

//2. 操作

// set 存储

jedis.sadd("myset","java","php","c++");

// set 获取

Set<String> myset = jedis.smembers("myset");

System.out.println(myset);

//3. 关闭连接

jedis.close();

5) 有序集合类型 sortedset：不允许重复元素，且元素有顺序

zadd

zrange

//1. 获取连接

Jedis jedis = new Jedis();//如果使用空参构造，默认值 "localhost",6379端口

//2. 操作

// sortedset 存储

jedis.zadd("mysortedset",3,"亚瑟");

jedis.zadd("mysortedset",30,"后裔");

jedis.zadd("mysortedset",55,"孙悟空");

// sortedset 获取

Set<String> mysortedset = jedis.zrange("mysortedset", 0, -1);

System.out.println(mysortedset);

//3. 关闭连接

jedis.close();

\* jedis连接池： JedisPool

\* 使用：

1. 创建JedisPool连接池对象

2. 调用方法 getResource()方法获取Jedis连接

//0.创建一个配置对象

JedisPoolConfig config = new JedisPoolConfig();

config.setMaxTotal(50);

config.setMaxIdle(10);

//1.创建Jedis连接池对象

JedisPool jedisPool = new JedisPool(config,"localhost",6379);

//2.获取连接

Jedis jedis = jedisPool.getResource();

//3. 使用

jedis.set("hehe","heihei");

//4. 关闭 归还到连接池中

jedis.close();

\* 连接池工具类

public class JedisPoolUtils {

private static JedisPool jedisPool;

static{

//读取配置文件

InputStream is = JedisPoolUtils.class.getClassLoader().getResourceAsStream("jedis.properties");

//创建Properties对象

Properties pro = new Properties();

//关联文件

try {

pro.load(is);

} catch (IOException e) {

e.printStackTrace();

}

//获取数据，设置到JedisPoolConfig中

JedisPoolConfig config = new JedisPoolConfig();

config.setMaxTotal(Integer.parseInt(pro.getProperty("maxTotal")));

config.setMaxIdle(Integer.parseInt(pro.getProperty("maxIdle")));

//初始化JedisPool

jedisPool = new JedisPool(config,pro.getProperty("host"),Integer.parseInt(pro.getProperty("port")));

}

/\*\*

\* 获取连接方法

\*/

public static Jedis getJedis(){

return jedisPool.getResource();

}

}

Maven

**Maven 指令的生命周期**

maven 对项目构建过程分为三套相互独立的生命周期，请注意这里说的是“三套”，而且“相互独立”，

这三套生命周期分别是：

Clean Lifecycle 在进行真正的构建之前进行一些清理工作。

Default Lifecycle 构建的核心部分，编译，测试，打包，部署等等。

Site Lifecycle 生成项目报告，站点，发布站点。

使用 maven 完成项目的构建，项目构建包括：清理、编译、测试、部署等过程，maven 将这些过程规范为一个生命周期.