第一章、FUNDAMENTAL

## JAVA开发环境

#### java编译运行过程

-------------常见面试题

1)编译期:.java源文件，经过编译，生成.class字节码文件

2)运行期:JVM加载.class并运行.class

特点:跨平台、一次编程到处使用

因为字节码文件是在虚拟机（JVM）上运行的，而不是编译器。JVM能跨平台安装，所以相应的java字节码文件便可以跟着在不同平台上运行。

#### 名词解释(JVM、JRE、JDK)

-----------------常见面试题

1)JVM:java虚拟机

加载.class并运行.class

2)JRE:java运行环境

除了包含JVM以外还包含了运行java程序所必须的环境

JRE=JVM+java系统类库(小零件)

3)JDK:java开发工具包

除了包含JRE以外还包含了开发java程序所必须的命令工具

JDK=JRE+编译、运行等命令工具

说明:

2.1)运行java程序的最小环境为JRE

2.2)开发java程序的最小环境为JDK

3、配置环境变量

-----------------了解

1)JAVA\_HOME:指向jdk的安装目录

2)CLASSPATH:表示类的搜索路径，一般简写为.

3)PATH:指向jdk下的bin目录

## 二．Eclipse

1、IBM、开源的、免费的、不需要安装仅需要解压即可

2、开发步骤:

1)新建Java项目/工程-------------------小区

2)新建Java包--------------------------楼号+单元号

3)新建Java类--------------------------房子

3、注释:解释性文本

1)单行注释://

2)多行注释:/\* \*/

3)文档注释:/\*\* \*/-----------------4月份

## 三、变量（存数的）

1、变量声明

int a; //声明一个整型的变量，名为a

int b,c,d; //声明三个整型的变量，名为b,c,d

2、初始化:第一次赋值

1)声明的同时初始化

int a = 250; //声明整型变量a并赋值为250

2)先声明后初始化

int a; //声明整型变量a

a = 250; //给变量a赋值为250

3、使用:

1)对变量的使用就是对它所存的那个数的操作

int a = 5;

int b = a+10; //取出a的值5，加10后，再赋值给整型变量b

System.out.println(b); //输出变量b的值15

System.out.println("b"); //输出b，双引号中的原样输出

a = a+10; //取出a的值5，加10后，再赋值给a

//在a本身基础之上增10

System.out.println(a); //15

2)变量的操作必须与数据类型匹配

int a = 3.14; //编译错误，数据类型不匹配

3)变量在用之前必须声明并初始化

System.out.println(m); //编译错误，m未声明

int m;

System.out.println(m); //编译错误，m未初始化

4、命名:

1)只能包含字母、数字、\_和$符，并且不能以数字开头

2)严格区分大小写

3)不能使用关键字

4)可以中文命名，但不建议

建议:"英文的见名知意"、"驼峰命名法"

## 四、基本数据类型

1、int:整型，4个字节，-21个多亿到21个多亿

1)整数直接量默认为int型，不能超出范围，超出范围则编译错误

直接量超范围为编译错误

运算时超范围为溢出

2)两个整数相除，结果还是整数，小数位无条件舍弃(不会四舍五入)

3)整数运算时超出范围，则发生溢出(需要避免的)

2、long:长整型，8个字节，很大很大很大

1)长整型直接量需在数字后加L或l

2)运算时若有可能溢出，建议在第1个数字后加L

3)System.currentTimeMillis()用于获取自

1970.1.1零时到此时此刻的毫秒数-------用途下周四讲

3、double:浮点型，8个字节，很大很大很大

1)浮点数直接量默认为double型，表示float需在数字后加F或f

2)浮点型数据在运算时，有可能会出现舍入误差

4、boolean:布尔型，1个字节

1)只能取值为true和false

5、char:字符型，2个字节

1)采用的是Unicode字符集编码格式，

每个字符都有一个对应的码(0到65535之间)

表现的形式的是字符char，实质上是码int

ASCII码('a'--97 'A'--65 '0'--48)

2)字符直接量必须放在单引号中，只能有一个

3)特殊符号需通过\来转义

## 五、基本数据类型间的转换:

数据类型从小到大依次为:

byte,short,int,long,float,double

char

1、两种方式:

1)自动类型转换:从小类型到大类型

2)强制类型转换:从大类型到小类型

语法: (要转换成为的数据类型)变量

byte b1 = 5;

byte b2 = 6;

byte b3 = (byte)(b1+b2);

强转有可能会溢出或丢失精度

2、两点规则:

1)整数直接量可以直接赋值给byte,short,char，但不能超出范围

2)byte,short,char型数据参与运算时，先一律转换为int再运算

4.Scanner接收用户的输入:

1)在package下:

import java.util.Scanner;

2)在main方法中:

Scanner scan = new Scanner(System.in);

3)在第2步之下:

System.out.println("请输入年龄:");

int age = scan.nextInt();

System.out.println("请输入价格:");

double price = scan.nextDouble()

## 六、运算符

1)算术:+,-,\*,/,%,++,--

注： 1/2=0 ———— 1.0/2=0.5

2)关系:>,<,>=,<=,==,!= boolean

3)逻辑:&&,||,! （boolean类型）

短路逻辑：&&的第一个操作数为false时不再判断第二个操作数

||的第一个操作数为true时不再判断第二个操作数

4)赋值:=, 扩展赋值：+=,-=,\*=,/=,%=

5)字符串连接:+

字符串的连接，字符串与其它数据类型的连接

6)条件/三目运算:boolean?数1:数2 （数1数2可嵌套三目运算）

## 七、分支结构

任何复杂的程序逻辑都可以通过“顺序”、“分支”、“循环”三种基本的程序结构实现

1. if结构:1条路 if(逻辑表达式){}

条件满足时运行某些语句，条件不满足时不运行这些语句

1. if...else结构:2条路

条件满足时运行某些语句，条件不满足时运行另一些语句

3)if...else if结构:多条路

4)switch...case结构:多条路

优点:效率高、结构清晰

缺点:整数、相等

break:跳出switch

## 八、循环结构

反复执行一段相同或相似的代码

1、循环三要素:

1)循环变量的初始化

2)循环的条件(以循环变量为基础)

3)循环变量的改变(向着循环的结束变)

循环变量:在整个循环过程中所反复改变的那个数

2、循环结构:

1)while:先判断后执行，有可能一次都不执行

2)do...while:先执行后判断，至少执行一次

要素1与要素3相同时，首选do...while

3)for循环：应用率高，语法：for(要素1；要素2；要素3){}

3、break:跳出循环或switch，常与条件语句一起用（if）

Continue:只能用于循环中，跳出循环体中剩余语句而进入到下一次循环

4、三种循环结构得更佳适用情况

1）while：（当……）不固定次数

要素1与要素3不同时首选

2）do…while：（直到……）不固定次数

要素1与要素3相同时首选

3）for:应用率最高，固定次数循环

5、嵌套循环

1）循环中套循环，一般多行多列时使用，外层控制行，内层控制列

2）执行过程成：外层走一次，内层走所有词

3）建议：循环层数越少越好，能用一层就不用两层，最多三层，若业务必须通过三层以上的循环来解决，说明设计有问题

4）break只能跳出一层循环。

6、程序=算法+数据结构

1）算法：解决问题的流程/步骤（顺序、分支、循环）

2）数据结构：将数据按照某种特定的结构来保存

数怎么存

设计良好的/合理的数据结构会导致好的算法

7、数组

1）是一种数据类型（引用类型）

2）相同数据类型元素的集合

3）声明：

int[] a=new int[3];//声明整型数组变量a,包含3个元素

//每个元素都是int型，默认值为0（boolean型默认为false）

4)数组的初始化：

Int[] arr=new int[3];//0,0,0

Int[] arr={1,4,7};//1,4,7

Int[] arr=new int[]{1,4,7};//1,4,7

// int[] arr;

arr={1,4,7};//编译错误，此方式只能声明同时初始化

arr=new int[]{1,4,7};//

1. 数组的访问：
   1. 通过(数组名.length)来获取数组的长度（元素的个数）System.out.println(arr.length);

5.2）通过下标/索引来访问元素，

下标从0开始，最大到（数组长度-1）arr[0]=1,arr[1]=4,arr[2]=7

6）积累：Arrays.toString(数组名);将数组转变成字符串，便于快速输出

6)数组的遍历：

int[] arr=new int[10];

for(int i=0;i<arr.length;i++){

arr[i]=100;

}

找最大数的算法：

1. 假设第1个元素为最大值int max=arr[0];
2. 遍历剩余元素，将剩余元素与max比较

若剩余元素大于max，则修改max为较大的

For(int i=1;i<arr.length;i++){

If(arr[i]>max){

max=arr[i];

}

}

1. 数组的复制

数组一旦创建，长度不可改变。

方法1：System.*arraycopy*(a, 1, a1, 0, 4);

a:源数组，1：源数组中起始下标，a1:目标数组

0:目标数组的起始下标，4：要复制的元素个数

注意：下标不能越界。

特点：灵活性好，效率高

方法2：整个数组复制，Arrays.copyOf(a,num);

int[] a={...};

int[] arr=Arrays.copyOf(a,num); 从第一位开始

num:新数组的长度

num大于源数组长度，则扩容；

num小于源数组长度，则截取。

特点： 灵活性差，效率低。作用：数组扩容

数组扩容：创建了一个新的数组并将源数组数据复制进去

Int[] a={12,23,2,4,2};

a=Arrays.copyOf(a,6);

1. 数组的排序

常用排序算法：插入排序，冒泡排序，快速排序。

7.1）不同的数据类型用不同的排序算法，优劣不同

7.2）有一种排序方式一直是最快的：Arrays.sort(arr);//升序

7.3）冒泡排序：

7.3.1）5个数冒4轮

7.3.2）每一轮都从第一个元素开始冒，每一次都和他下一个元素比（大的换）

7.3.3）冒出来就不在比

**int[] arr = {67,23,45,1};** //升序(1,23,45,67)

**for(int i=0;i<arr.length-1;i++){** //控制轮数

**for(int j=0;j<arr.length-1-i;j++){** //控制次数

**if(arr[j]>arr[j+1]){**

**int t = arr[j];**

**arr[j] = arr[j+1];**

**arr[j+1] = t;**

**}**

**}**

**}**

检测效率：

**Long a=System.currentTimeMillis();**//毫秒数

**Arrays.sort(arr);**

**Long a=System.currentTimeMillis();**

（B/S:浏览器/服务器）

## 九、方法（函数、过程）

1、用于封装一段特定的业务逻辑功能。

方法定义：**public static int sum(int num1,int num2){**

**//方法体**

**}**

定义方法的五要素：修饰词、返回值类型、方法名，参数列表，方法体

作用：

1）方法尽可能的独立，只干一件事

2）方法可以被反复多次的调用

3）减少代码重复，有利于代码的维护,有利于团队的协作开发

2、方法可以有返回值也可以没有返回值：

1）无返回值时，返回值类型设计为**void**

2）有返回值时，返回值类型设计为具体的数据类型即可

方法体中必须有**return**语句。

3、 设计方法有无返回值：

方法执行完后，

1）若后期要用方法中的某个数据，设计为有返回值

2）若后期不再需要用方法中的某个数据，设计为无返回值

4、方法可以有参，可以无参：有参可以使方法更灵活

5、方法的调用：

1）无返回值：**方法名（有参传参）**；

2）有返回值：2.1）**数据类型 变量 = 方法名（有参传参）**；

2.2）**方法名（有参传参）**；给了没要，不建议

传参时参数类型要匹配

6、**return**的用法：

1)**return 值;** //1.1)结束方法的执行 1.2)返回结果给调用方

2)**return;** //2.1)结束方法的执行

7、形参（形式参数）：定义方法时参数列表（括号中）

实参（实际参数）：调用方法时传参（在括号中）

8、方法可以嵌套调用：定义方法时可以嵌套调用方法，非嵌套定义

猜字符小游戏：

步骤（一）

1. 设计数据结构：
2. **char[] chs;**//随机字符数组
3. **char[] input;**//用户输入字符数组
4. **int letterRight;**//字符对的个数

**int positionRight;**//位置对的个数

**int[]result;**//对比结果

1. **int score;**//得分
2. 设计程序结构：方法
3. 主方法：

**Public static void main(String[] args){**

**}**

1. 生成随机字符数组

**Public static char[] generate(){**

**Char[] chs=new char[5];**

**//…**

**Return chs;**

**}**

1. 对比随机字符数组**chs**与用户输入字符数组**input**

返回字符对的个数，位置对的个数

(位置对是在字符对的基础上)

**public static int[] check(char[] chs,char[] input){**

**Int[] result=new int[2];**

**//…**

**Return result;**

**}**

步骤（二）

1. 设计算法：方法体
2. 对比最简单，check();
3. 生成随机字符数组

小算法：随机并保证不重复,创建数组开关（boolean[]）

1. 主方法：

3.1)藏起来数组chs

3.2)

While(true){}//自造死循环

猜吧,接收input

对比：

猜对了，算分结束

猜错了，提示并继续第二步

扩展：

（1） **String str=scan.next();**

**Char[] input=str.toCharArray();**//将字符串转换成数组

**String str=scan.next().toUpperCase();**//接受并转换成大写字符串

**String str=scan.next().toLowerCase();**//接受并转换成小写字符串

（2）基本类型数据判断相等--------用==

引用类型（String）判断内容相等--------不能用==，用.equals(…)

# 面向对象

面向对象分析：OOA

面向对象设计：OOD

面向对象分析与设计：OOAD

面向对象编程：OOP

高质量的代码：

**复用性好**、扩展性好、维护性好

可移植性好、健壮性好。可读性好。效率好’

面向过程的结构化程序设计弊端：

1. 缺乏对数据的封装
2. 数据与方法（操作数据）的分离

基本类型---------------------只能装一个数据

数组类型---------------------必须数据类型相同

自己造数据类型---------------想装什么就装什么

Class Emp{//Emp就是自己造的数据类型，名为Emp

String name;

Int age;

double salary;

}

## 对象和类

### 1、什么是类：什么是对象？

1. 现实世界是由很多对象组成的

基于对象抽出了类

1. 对象：真实存在的单个个体

类：类型/类别，代表结构相同，具有相同属性和行为的一类个体

1. 类中可以包含：

3.1）所有对象所共有的属性/特征---------变量（成员变量）

3.2）所有对象所共有的行为--------------方法

4）一个类可以创建多个对象：

同一个类型所创建出来的对象，结构相同，数据（值）不同

Student zs=new Student();

Student ls = new Student();

类（数据类型） 引用类型变量 指向 对象

a、创建了一个对象b、给成员变量赋默认值c、调用了Student类的构造方法

5）类是对象的模板，对象是类具体的实例

2、如何创建类，如何创建对象，如何访问成员？

### 3、方法的重载

方法的签名：包含方法名和参数列表。

一个类中不能有多个方法签名完全相同，即方法名和参数列表都相同

可以方法名相同，参数列表不同。

方法重载（over-load）：

1、发生在同一个类中，多个方法名相同，但参数列表不同（方法体不同）

2、编译器在编译时会根据方法的签名自动绑定调用的方法

### 4、构造方法

(构造函数、构造器、构建器）：

1）给(对象）成员变量赋初值

2）构造方法必须与类同名，没有返回值类型，但也不能写void

3）在创建（new）对象时被自动调用

4）若自己不写构造方法，则编译器默认提供一个无参构造方法

若自己写了构造方法，则不再默认提供

5)构造方法可以重载

### 5、this作用

（1）在方法体中，指代当前对象，哪个对象调用该方法指的就是哪个对象

只能在方法中，方法中访问成员变量之前默认有个this

（2）在构造方法中区别成员变量和参数

（3）this的用法：

1. **this.成员变量名**:访问成员变量
2. **This.方法名()**:调用方法（一般不用）
3. **This()**:调用构造方法

### null和NullPointerException

null:空，没有指向任何对象。

若引用的值为null,则该引用不能再进行任何操作了

若操作则发生NullPointerException空指针异常

Student zs=new Student();

zs=null;//空，没有指向任何对象

zs.name=”zhangsan”;

内存管理：由JVM管理的

1. 堆：存所有new出来的对象（包括成员变量）
2. 栈：存所有的局部变量（引用----对象的地址，方法内的基本类型变量）
3. 方法区：存（**.class**）字节码文件（包括方法）

### 7、引用类型变量之间画等号

1）指向同一个对象

2）通过一个引用对数据的修改会影响另一个引用对数据的访问

基本类型之间画等号：

1）赋值

2）对一个变量的修改不会影响另一个变量的值

## 二、数组、继承

#### 1、引用类型数组：

数组是引用数据类型，也是new出来的，也是对象。

基本类型数组：**int[] arr=new int[3];**

**Int a=5;**

栈中存的是局部变量：

局部变量是基本类型，存值(a=5)

局部变量是引用类型，存地址(arr)

1. **Student[] stus=new Student[3];**

**stus[0]=new Student(“zhangsan”,25,”LF”);**

**stus[1]=new Student(“lisi”,26,”JMS”);**

**stus[2]=new Student(“wangwu”,45,”LF”);**

1. **Student[] stus=new Student{**

**(“zhangsan”,25,”LF”),**

**(“lisi”,26,”JMS”),**

**(“wangwu”,45,”LF”)**

**}**

**stus[0].age=28;**

1. **Int[] [] as=new int[3][];------**数组的数组

声明int[]型数组，包含3个元素，每个元素都是int[]型，默认为null

**as[0]=new int[2];**

**Int[][] arr=new int[3][4];**//3行4列

**for(int i=0;i<arr.length;i++){**

**for(int j=0;j<arr[i].length;j++){**

**Arr[i][j]=100;**

**}**

**}**

#### 继承

1）作用：代码复用

2）通过extends实现继承

3）超类：所有派生类所共有的属性和行为

派生类：派生类所特有的属性和行为

1. 派生类继承超类后派生类具有：超类的+派生类的
2. 一个超类可以有多个派生类

一个派生类只能有一个超类------单一继承

1. 继承具有传递性
2. Java规则：构造派生类之前必须先构造超类

在派生的构造方法中若没有调用超类的构造方法

-----则默认super()调用超类的无参构造方法

在派生类得构造方法中调用了超类的构造方法

------则不在默认提供

super()调用超类构造方法，必须位于派生类构造的第1句

#### super

指代当前对象的超类对象

Super用法：

1. super.成员变量名------------访问超类的成员变量
2. Super.方法名（）------------调用超类的方法
3. Super();------------调用超类的构造方法

#### 向上造型

1. 超类型的引用指向派生类对象，叫向上造型
2. 能点出来什么，看引用类型,只能点超类里面的

扩展：**Student[] stu=null;**(**Student[] stu;**)

**for(int i=0;i<stu.length;i++){}**//空指针异常

**Student[] stu=new Student[0];**(**Student[] stu={};**)

**for(int i=0;i<stu.length;i++){}**//元素个数为0，但不为空，所以不报空指针异常

#### 方法的重写（Override）

重新写，覆盖

1. 发生在父子类中，方法名相同，参数列表相同，方法体不同
2. 重写方法被调用时，看对象类型（new谁调谁）

**Student zs=new Student("张三",18,"广东",20);**

**Person zss=new Student("李四",15,"广西",21);**

**zs.sayHi();**

**zss.sayHi();**//重写方法被调用时，看对象类型

//父类中没有此方法则点不出来，也没有该方法的重写

1. 重写遵循**“**两同两小一大**”**原则：

3.1）两同：

a、方法名相同

b、参数列表相同

3.2）两小：

1. 派生类方法的返回值类型小于或等于超类方法的
2. void是必须相等
3. 基本类型时必须相等
4. 引用类型时小于或等于
5. 派生类方法抛出的异常小于或等于超类方法的

3.3）一大：

1、派生类方法的访问权限大于等于超类的

#### 重写与重载的区别

1. 重写：

不是一定要发生。

* 1. 发生在父子类中，方法名相同，参数列表相同，方法体不同
  2. 遵循“运行期绑定”，看对象的类型来调用

1. 重载：
   1. 发生在一个类中，方法名相同，参数列表不同，方法体不同
   2. 遵循“编译期绑定”，看引用/参数的类型来调用

编译期:.ja

va源文件，经过编译，生成.class字节码文件(还没有对象）

运行期：JVM加载并运行**.class**

## 三、访问控制、static和final

### 1、package和import

**package:**

1. 作用：避免类名冲突
2. 包名可以有层次结构
3. 类的全名称为：**包名.类名**

域名反写 **.** 项目名称 **.** 模块名称 . 类名

cn.tedu **.** aproject **.** stumanager **.**

cn.tedu **.** aproject **.** teachermanager **.**

1. 建议：包名所有字母都小写

**import:**

同包中的类可以直接访问

不同包中的类不能直接访问，想访问只有如下两种方式:

1)先import声明类再使用类------建议

2)类的全称--------------太繁琐，不建议

### 2、访问控制修饰符

数据（变量）私有化（private），行为（方法）公开化(public)

访问权限：

1. public:公开的，任何类
2. private:私有的，本类
3. protected:受保护的，本类、子类、同包类(一般在超类中使用
4. )
5. 默认的：什么也不写，本类、同包类

说明：

1)类的访问修饰只能是public和默认的

2)类中成员的访问修饰如上四种都可以

### 3、final

最终的，不能改变的------应用率不高

1. 修饰变量：变量不能被改变
2. 修饰方法：方法不能被重写
3. 修饰类：类不能被继承

final修饰成员变量，只有两种初始化方式:

1）声明的同时初始化

2）在构造方法中初始化

final修饰局部变量，在使用之前赋值即可

### 4、static(难点）静态的

1)成员变量：

1.1)实例变量：没有static修饰，属于对象的，存储在堆中，

类中有几个对象中就有几个，通过对象名点来访问

1.2)静态变量：有static修饰，属于类的，存储在方法区中

只有一份，通过类名点来访问

堆:new出来的对象(包括实例变量)

栈:局部变量

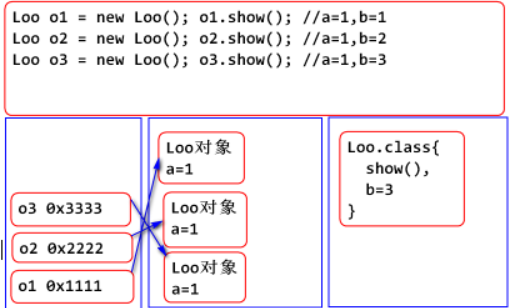
方法区:.class字节码文件(包括方法、静态变量)

2）静态变量：

1.1)由static修饰

1.2)属于类，存储在方法区中，只有一份（类只加载一次）

a++;b++;



1.3)常常通过类名点来访问

1.4)何时用:所有对象所共享的数据（图片、，音频、视频等）

3) 静态方法:

3.1）由static修饰

3.2）属于类，存储在方法区中，只有一份

3.3）常常通过类名点来访问

3.4）静态方法中没有隐式的this传递

静态方法中不能直接访问实例成员

//没有this就意味着没有对象，而实例成员必须通过对象点来访问

3.5）何时用：方法的操作仅与参数相关而与对象无关

4) 静态块:

4.1）属于类，在类被加载期间自动执行，

因类只加载一次，所以静态块也只执行一次，（可节省内存）

4.2）何时用：加载/初始化静态资源（图片、，音频、视频等）

5、static final常量

1）必须声明同时初始化

2）通过类名点来访问，不能被改变

3）建议：常量名字母都大写，多个单词用\_分隔

4）编译器在编译时将常量直接替换为具体的值,效率高

5）何时用：若有一个数据永远不变，并且经常使用

## 四、抽象类、接口和内部类

### 1、抽象方法

1）由abstract修饰

2）只有方法的定义，没有具体的实现（连大括号{}也没有）

何时用：所有对象所共有的行为，但每个对象的具体实现不同时设计为抽象方法

### 2、抽象类

1)由abstract修饰

2)包含抽象方法的类必须是抽象类

不包含抽象方法的类也可以声明为抽象类

3）抽象类不能被实例化（可以声明引用，不能new对象）

（abstract FlyingObject）

FlyingObject f=new FlyingObject();//编译错误，抽象类不能创建对象

可以new FlyingObject[5];//新的数据类型

可以声明引用：FlyingObject f;

1. 抽象类都是需要被继承的，派生类：

4.1）重写抽象类中的所有抽象方法----变不完整为完整

4.2）也声明为抽象类------没意义

（类的意义就是创建对象，单独的抽象类没有意义。）

1. 抽象类的意义：

5.1）封装派生类所共有的属性和行为-------代码复用

5.2）为所有的派生类提供统一的类型-------向上造型

5.3）可以包含抽象方法，为所有派生类提供统一的入口

派生类的具体实现不同，但入口是一致的（方法的不同实现，定义相同）

1):static final常量：应用率高。

1：必须声明同时初始化；

### 3、内部类

3.1成员内部类：应用率低

1. 类中套类，里面的称为Inner内部类，外面的称为Outer外部类
2. 内部类通常只服务于外部类，对外不具备可见性
3. 内部类通常是在外部类中创建的
4. 内部类中可以直接访问外部类的成员（包括私有的）eg: 外部类名.this.

因为：内部类中有一个隐式的引用指向了创建他的外部类对象

3.2）匿名内部类：

1）若想创建一个类的对象，并且对象只被创建一次，

此时该类不必命名，称之为匿名内部类

2）jdk1.7(含)版本之前，匿名内部类中想访问外面的变量，

要求该变量必须是final的

面试题：内部类有独立的**.class**吗？有

### 4、接口

1）是一种数据类型（引用类型）

2）由interface定义的，

3）只能包含常量和抽象方法

4）接口不能被实例化，

5）接口是需要被实现的，实现类中必须重写接口中所有的抽象方法

6）一个类可以实现多个接口，用逗号分隔

若又继承又实现时，先继承后实现

7）接口可以继承接口

同类型是继承，不同类型是实现

8)可以将部分派生类所共有的行为，抽到接口中，符合：既是也是原则时，使用接口。

9)接口是对继承单根性的扩展，实现多继承。

### 5、多态

1）意义：

1.1）同一类型的引用指向不同的对象时，有不同的实现

---------行为的多态（抽象方法都是多态的）

1.2）同一个对象被造型为不同类型时，有不同的功能

---------对象的多态（接口的实现

2）向上造型/自动类型转换

2.1）超类型的引用指向派生类的对象

2.2）能造型成功的类型有：超类+所实现的接口

2.3）能点出来什么，看引用类型

3）强制类型转换，成功的条件只有如下两种：

3.1）引用所指向的对象就是该类型

3.2）引用所指向的对象实现了该接口/继承了该类

4）强转时如果不符合以上两条件，则发生ClassCastException类型转换异常

建议：强转之前先通过instanceof来判断而后再强转

**If(o instanceof Coo){**

**Coo o4=(Coo)o;**

**}**

### 6、内存管理

1）堆：

1.1）存储new出来的对象（包括实例变量）

1.2）垃圾：没有任何引用所指向的对象，

垃圾回收器（GC）不定时到内存中清扫垃圾，

回收过程是透明的（看不到的），不一定发现垃圾就立刻回收

调用**System.gc()**可以建议虚拟机尽快调用GC来回收垃圾

1.3）实例变量的生命周期：

创建对象时存在堆中，对象被回收时一并被回收

1.4）内存泄漏：不再使用的对象还没有被及时回收（不用但有引用指向）

建议：不再使用的对象及时将引用设置为null

1. 栈

2.1）正在调用的方法中的所有的局部变量（包括方法的参数）

2.2）调用方法时，在栈中为该方法分配一块对应的栈帧，

栈帧中存储方法中的局部变量(包括方法的参数)

方法调用结束时，栈帧被清除，局部变量一并被清除

2.3）局部变量生命周期：

方法调用时存储在栈中，方法结束时与栈帧一并被清除

1. 方法区

3.1）存储.class字节码文件(包括方法、静态变量)

3.2)方法只有一份，通过this来区分具体的访问对象。

实例变量:

1)类中，方法外

2)创建对象时存在堆中，对象被回收时一并被回收

3)有默认值

局部变量:

1)方法中

2)方法调用时存储在栈中，方法结束时与栈帧一并被清除

3)没有默认值

### 7、面向对象三大特征（总结）

1. 封装
2. 类：封装的是对象的属性和行为
3. 方法：封装一段特定的业务逻辑功能
4. 访问控制修饰符：封装的是具体的访问权限
5. 继承
6. 作用：代码复用
7. 超类：所有派生类所
8. 共有的属性和行为

接口：部分派生类所共有的行为

派生类：派生类所特有的属性和行为

1. 特点：单一继承，多接口实现，传递性
2. 多态
3. 行为多态（所有对象方法都是多态的）

对象多态（所有对象都是多态的）

1. 向上造型、强制类型转换、instanceof
2. 多态的表现形式：

3.1）重写：根据对象的不同来多态

3.2）重载：根据参数的不同来多态

### 8、Debug调试工具

1.何时用:当程序的运行结果与你的预期不同时，需要调试

2.调试方式:

1)打桩: System.out.println(数据);

2)Debug调试工具:

2.0)添加断点

2.1)四个键:

2.1.1)F5:单步调试(会进入到方法中)

2.1.2)F6:逐过程调试(不会进入到方法中)

2.1.3)F7:结束方法的调试(从方法中返回)

2.1.4)F8:直接跳到下一个断点(若后面无断点则结束调试)

2.2)会看两个东西:

2.2.1)会看变量

2.2.2)添加监视

## 飞机大战分析与步骤

角色：英雄机，子弹，小敌机，大敌机，小蜜蜂，天空

关系：

1. 英雄机发子弹
2. 子弹射击小敌机，大敌机，小蜜蜂
3. 小敌机，大敌机，小蜜蜂撞击英雄机
4. 英雄机，子弹，小敌机，大敌机，小蜜蜂在天空非
5. 英雄机可单倍或双倍火力
6. 子弹打中小敌机------得1分

子弹打中大敌机------得3分

子弹打中小密封------英雄机得奖励（1条命或40火力值）

子弹发射一次双倍

### 第一天

设计对象类

1. 先找对象：天空，英雄机，子弹，小敌机，大敌机，小蜜蜂
2. 抽类:Hero,Sky,Airplane,BigAirplane,Bee,Bullet
3. 设计类中的成员变量和方法（业务，后面再详细做）
4. 创建对象并测试

Class Hero{//

Width,height,x,y,life,doubleFire

Void step(){}

}

### 第二天

：每个对象类设计构造方法

### 第三天

1.（World中）设计小敌机数组、大敌机数组、小蜜蜂数组、子弹数组，并测试

2.设计FlyingObject超类，6个对象类继承超类

3.给FlyingObject超类设计了两个构造方法，6个派生类中分别调用（super();）

### 第四天

1.将小敌机数组、大敌机数组、小蜜蜂数组合为FlyingObject数组

2.在6个派生类中重写step()

3.画窗口

Swing相关：

1. JFrame：框架（相框）
2. JPanel：面板

### 第五天

1、添加控制修饰符

方法：

1. 公开的

成员变量：

1. 超类中的-----protected
2. 普通类类中的------private

2、设置图片属性

超类FlyingObject导入图片操作类和图片操作方法loadImage(String fileName)

6个对象类添加静态图片变量和静态块

相应图片放到项目包中

### 第六天

1、数据永远不变，并且经常使用，设为static final常量（窗口宽和高，用到窗口设置、超类和天空Sky）

2、将超类中被重写的方法step()改为抽象方法，超类改为抽象类’

3、画对象：

1. 先画对象，得先去获取对象的图片，

每个对象都得获取图片，意味着获取图片行为为共有的行为

所以需要将获取图片的行为设计在FlyingObject超类中

每个对象获取图片的行为均都是不一样的，

所以获取行为的方法设计为抽象方法

--------在FlyingObject类中设计抽象方法getImage();

2) 在获取对象图片时，需要去考虑对象的状态（生命周期）

状态分三种：活着的，死了的（未删除），删除的

在实际应用中，状态都设计为常量

每个对象都有状态，所以将状态设计在FlyingObject超类中

-------------在超类中设计三个常量和变量（存储当前状态）

在获取图片时，需要去判断每个对象的状态，用于获取不同的图片

所以对象状态判断的行为为共有的行为，设计在超类中，

每个对象判断状态行为都是一样的，设计为普通方法

----------在FlyingObject中设计isLife()/ isDead()/ isRemove()

3）在派生类中重写getImage()用于在不同状态下获取不同图片

3.1）Sky天空，直接返回image

3.2)Bullet子弹，

若活着的，直接返回image

若死了，就删除

3.3)Airplane小敌机:

3.3.1)若活着呢，返回第1张图片

3.3.2)若死了呢，从第2张图到第5张图轮换，第5张图后可以删除了

3.4)Airplane小敌机:

3.4.1)若活着呢，返回第1张图片

3.4.2)若死了呢，从第2张图到第5张图轮换，第5张图后可以删除了

3.5)Airplane小敌机:

3.5.1)若活着呢，返回第1张图片

3.5.2)若死了呢，从第2张图到第5张图轮换，第5张图后可以删除了

3.6)Hero英雄机:

3.6.1)若活着呢，返回2张图片的切换

4）图片有了，就可以开始画了，6个对象都可以开画，属于共有的行为，所以将

画的行为设计在FlyingObject超类中

所有对象画的行为都一样，所以设计为普通方法

--------在超类中设计普通方法paintObject()画对象

1. 因为天空是两张图，所以需要重写超类中的paintObject()方法
2. 画的行为完成以后，只需要在窗口调用即可

--------在World类中重写paint()，而后paint()方法中调用paintObject()

在run()中调用repaint();//重画（重新调用paint)

### 第八天第七天

游戏步骤：

1. 敌人入场-------------定时发生
2. 因为敌人是窗口产生的所以在World中设计nextOne()生成敌人对象
3. 敌人入场为定时发生的，在run()中调用enterAction()实现敌人入场

在enterAction（）中：

每400毫秒调用nextOne()获取敌人，enemies扩容，装在最后一个元素上

1. 子弹入场-------------定时发生
2. 子弹是由英雄机发射出来的，所以在Hero中设计shoot()生成子弹对象
3. 子弹入场为定时发生的，在run()中调用shootAction()实现子弹入场

在shootAction()中：

每300毫秒调用hero的shoot()获取子弹（有单有双），bullets扩容，数组追加

1. 飞行物移动-------------定时发生

1)移动是所有对象所共有的行为，并且每个对象移动的行为都是不一样的，

所以在超类FlyingObject中设计了抽象方法step()，派生类重写step()

2)飞行物移动为定时发生的，在run()中调用stepAction()实现飞行物移动

在stepAction()中:

天空动，遍历敌人而后敌人动，遍历子弹而后子弹动

1、英雄机随着鼠标动----------事件触发

1. 先写行为：英雄机移动是英雄机行为，所以在Hero中设计moveTo(x,y)实现

英雄机随着鼠标移动

1. 英雄机随着鼠标动是事件触发的，在侦听器中重写mouseMoved()鼠标移动事件

在mouseMoved()中: 获取鼠标的x和y坐标，而后英雄机随着动

2、删除越界的飞行物（优化）

1）先写行为：

删除越界飞行物时，需先判断飞行物是否越界,每个对象都需判断，

所以将删除判断isRemove()在超类中

每个对象越界行为不一样，所以设计为抽象方法outOfBounds（），派生类中重写

1. 删除越界飞行物是定时发生，，在run()中调用outOfBoundsAction()

在outOfBoundsAction()中:

声明不越界敌人/子弹数组，遍历enemies/bullets，获取每个敌人或子弹

判断若不越界，则将对象添加到不越界敌人/子弹数组中，

将不越界敌人/子弹数组复制到enemies/bullets中

3、设计Enemy得分接口，Airplane和BigAirplane中实现接口

设计Award奖励接口，Bee实现接口

### 第九天

1. 子弹与敌人碰撞

1)在超类FlyingObject中设计hit()实现敌人与子弹/英雄机的碰撞

在超类FlyingObject中设计goDead()实现飞行物去死

在Hero类中设计addLife()增命、addDoubleFire()增火力

2)子弹与敌人的碰撞为定时发生的，所以在run()中调用bulletBangAction()实现子弹与敌人的碰撞

在bulletBangAction()中:

遍历所有子弹，获取每个子弹，遍历所有敌人，获取每个敌人，

判断是否撞上了，若撞上了则:

2.1) 子弹去死、敌人去死

2.2) 若撞了小敌机和大敌机，则玩家得分

若撞了小蜜蜂，则英雄机得奖励

1. 画分和画命

1)在Hero类中设计getLife()获取命

2)在paint()方法中，画分和画命

### 第十天

1. 英雄机与的人碰撞
2. 借用FlyingObject中的hit()碰撞行为

借用FlyingObject中的goDead()去死行为

Hero中设计subtractLife()减命、clearDoubleFire()清空火力

1. 因为英雄机与敌人碰撞为定时发生的，所以在run()中调用heroBandAction()

在heroBandAction()中：

遍历所有敌人，获取每个敌人，判断是否与英雄机撞上了，

若装上了：敌人去死，英雄机减命、英雄机清空火力’

2、检测游戏结束

1）借用Hero的getLife()获取命

2）因为检测游戏结束为定时发生的，所以在run()中调用checkGameOverAction()

在checkGameOverAction()中

判断英雄机的命数否<=0,若为true则。。。。。。。。

3、画状态:

1)设计了四个状态常量，当前状态state变量

设计了三个静态图片，在static块中初始化

2)在paint()中设计在不同状态下画不同的图片

3)将一堆action()设计为仅在运行状态下执行

将英雄机随着鼠标动设计为仅在运行状态下执行

4)重写mouseClicked()鼠标点击事件

重写mouseExited()鼠标点击事件

重写mouseEntered()鼠标点击事件

**swing相关的**：

1. 事件：发生了一件事
2. 事件处理：发生事件之后所做的操作
3. 侦听器：

3.1）有一个侦听器

3.2）安装侦听器

MouseListener

This.addMouseListener(1);//鼠标操作事件

This.addMouseMotionListener(1);//鼠标滑动事件

## 游戏扩展知识点

### 1、生成随机数

Random rand=new Random();

Int x=rand.nextInt(100);//

### 2、异常处理

需要对文件进行操作，对文件操作特别容易出现异常

------强制做文件操作时，必须要加入异常处理

### 3、图片切换

**return images[index++%2];**//若0,1切换取余2。若0,1,2切换取余3\*\*\*\*\*\*

### 4、定时器

**timer.schedule(new TimerTask(){**

**run(){**

**}**

**},10,10);**

第1个10（long）:从程序启动到第一次触发的时间间隔

第2个10（long）:从第一次触发到第二次触发的时间间隔

从第二次触发到第三次触发的时间间隔

从第三次触发到第四次触发的时间间隔

...

### 5、功能实现步骤

4.1.先写行为(方法):

1)若是某个对象特有的行为，则将行为设计在派生类中

若是所有对象共有的行为，则将行为设计在超类中

1.1)所有派生类行为都一样，设计为普通方法

1.2)所有派生类行为都不一样，设计为抽象方法，而后派生类重写

4.2.窗口调用:

1)定时触发的，在run()中调用

2)事件触发的，在侦听器中调用

3)画出来的，在paint()中调用

### 6、paint()调用

paint()调用的方式只有两种:

1)frame.setVisible(true);

2)repaint();

### 7、设计规则

1. 将所有派生类所共有的属性和行为，抽到超类中---------抽共性
2. 所有派生类的行为都一样，设计为普通方法

所有派生类的行为都不一样，设计为抽象方法

1. 将部分派生类所共有的行为，抽到接口中

符合既是也是原则时（若又继承又实现时），使用接口

接口是对继承的单根性的扩展-------------实现多继承

# **第三章、JAVASE01（标准Java版本）**

## 一、**API文档 、 字符串基本操作**

API：（Application Programming Interface,应用程序编程接口）是已写好，可供直接调用的功能（以类的形式封装）

### 1、String字符串基本操作/String及其常用API

1）String类使用了final修饰，不能被继承

String是不可改变对象，但字符串引用可以重新赋值

2）String常量池：（静态的重用，非静态的不重用）

静态字符串在常量池中创建并保存，并使用同一个引用,重用静态字符串

2.1）当再次使用相同字符串字面量时候，会使用上次的字符串引用

新创建String对象时引用不再相同，对象不同，但仍是同一个字符数组

栈 （堆）String char[] （a b c）

s1 String s1=“abc” value**1** a b c

s2 String s2=“abc” value**1** a b c

s3 String s3=new String(“abc”); value**2** a b c

2.2）字面量或常量的连接结果，也会重用字符串常量池中的对象

String s1="abc";

String s6="ab"+"c";

System.out.println(s1==s6);//true

2.3）字符串变量的连接结果不会常量池优化

String s7="ab";

String s8=s7+"c";//创建了新对象

System.out.println(s1==s8);//false，比较的是地址

String str=new String("ABC");

执行期间创建了2个字符串对象（“ABC”和new String）,3个对象(char数组）

String s1=new String("AB"+"C");

执行期间创建了2个字符串对象，3个对象（同上）

3）使用**length()**方法检查字符串中的字符个数

String的长度length()是方法

数组的长度是length属性

4）String 提供了**charAt（）**方法，获取指定位置上的字符

· 5）**indexOf(）、lastIndexOf();**

5.1）检查字符串中字符/字符串出现的位置(下标)

字符串提供了indexOf(被检索的字符/字符串);

如果找到第一次字符出现的位置，就返回字符位置

查找字符串时返回的是匹配字符串的起始位置

如果没有找到指定字符/字符串，则返回-1

5.2）indexOf(str,n)从指定位置开始检索字符/字符串

5.3）lastIndexOf()查字符最后一次出现的起始下标

lastIndexOf(str,n)从某处位置开始查字符最后一次出现的起始下标

6）**substring(起始位置，结束位置)：**截取子字符串,包含起始位置，不包含结束位置

**substring(起始位置)：**从起始位置到最后全部截取为子字符串

7）**trim()：**去除当前字符串的两端空白

trim()如果字符串两端有空白则返回这个字符串的一个两端没有空白副本

空白：编号小于等于32的字符(不仅是空格)

String name=" Tom ";

**name.trim();**

System.out.println(name);// **Tom** (前后有空格），去除空白后的副本没接收

**name=name.trim();**

System.out.println(name);//**Tom**(前后无空格）

8）**startWith();**检测字符串是否已制定字符串开始

**endWith();**检测字符串是否已制定字符串结束

### 2、StringBuil**d**er及其API

1）StringBuilder封装可变字符数组，对象创建以后可以通过调用方法改变其封装的字符序列

栈 （堆）String char[] （a b c）

s1 String s1=”abc” value1 a b c

s2 String s2=”abc” value1 a b c

s3 String s3=new String(“abc”); value2 a b c

2）StringBuilder常用方法

2.1）**str.append(String s);**追加字符串，用于字符连接

2.2）**str.insert(int index,String s);**在位置index前插入字符串

2.3）**str.delete(int start,int end);**删除字符串(包含开始，不包含结束）

2.4）**str.replace(int start,int end,String s);**替换字符串(包含开始，不包含结束）

2.5）**str.reverse();**字符串反转

**str.append(s).append(s).insert(index,s).**........

3）StringBuilder是可变字符串。

字符串的内容计算，建议采用StringBuilder实现

1. StringBuffer和StringBuilder

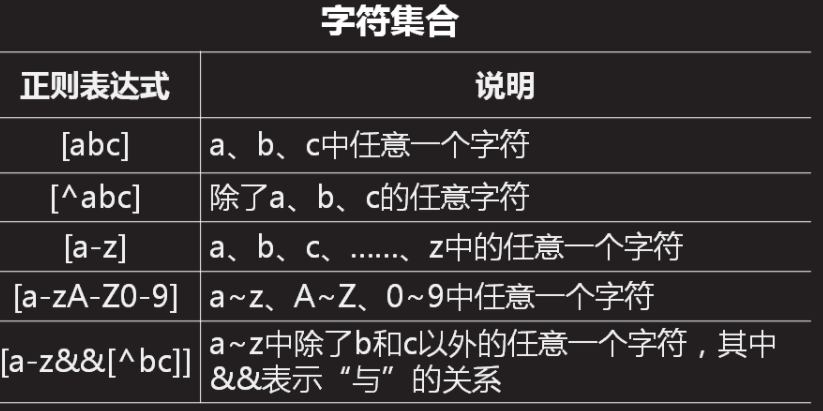
----StringBuffer是线程安全的，同步处理的，性能稍慢

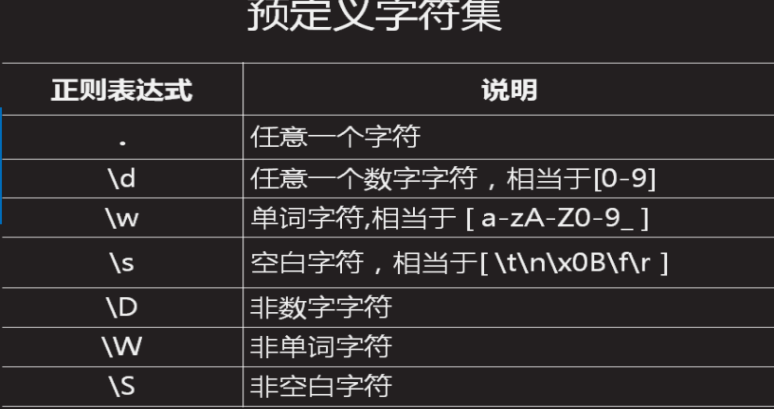
----StringBuilder是非线程安全的，并发处理的，性能稍快

## 二、正则表达式regex、Object、包装类

### 1、正则表达式

1）正则表达式简介







2）分组“（）”

将（）内一系列正则表达式看做一个整体，可用“|”表示“或”关系。

3）边界匹配“**^**”和“**$**”

--**^**代表字符串开始

--**$**代表字符串结束**^**\w{8,10}**$**;有从头到尾整体的限制。

### 2、String正则API

1）String提供了支持正则表达式的方法boolean matches(String regex) :

**boolean b=str.matches(regex);**使用给定的正则表达式验证当前字符串是否满足格式要求

需要注意，给定的正则表达式无论是否添加边界符(^$),都是做全匹配检查

2）java支持正则表达式方法String[] split(String regex):**String[] data=str.split(regex);**

将当前字符串按照满足正则表达式的部分拆分，将拆分出来的内容存入字符串数组并返回

注意： \* 若连续匹配了拆分部分，那么中间会拆分出一个空字符串

\* 若是在字符串末尾连续匹配，那么所有拆分出来的空字符串全部忽略。

3）字符串支持正则表达式方法：Stirng replaceAll(String regex,String str)

将当前字符串中满足正则表达式部分换成字符串str.

### 3、Object

在java类继承结构中，java.lang.Object类位于顶端，如果定义一个Java类时

没有使用extends关键字声明其父类，则其父类默认为java.lang.Object类

Object类型的引用变量可以指向任何类的型对象。

1）**toString**方法是java API中最常见被调用的方法，

通常我们定义的类都会重写这个方法

返回的字符串没有具体的格式要求，可结合捡来实际开发需求而定

原则上返回的字符串中应当包含当前对象属性信息。

2）**String toString()**

该方法的意义是将当前对象转换为一个字符串

Object已经实现了该方法，返回的字符串时当前对象的句柄（引用），

格式为：类名@地址

该字符串在实际开发中的意义不大，所以通常需要使用这个方法是要重写

1. **重写equals**

目的：是定义当前对象与给定的参数对象比较内容是否相同，

这里也没有严格要求必须两个对象所有属性都必须完全相同，可以根据开发情况而定。

Object提供的equals方法内部就是用“==”比较的，所以没有实际意义，

若需要比较两个对象内容是否相同应当重写该方法。Java提供的类都重写了equals。

public boolean equals(Object o){

if(o==null){

return false;

}

if(o==this){

return true;

}

if(o instanceof Point){

Point p=(Point)o;

return this.x==p.x &&this.y==p.y;

}

return false;

}

1. equals和==的区别

• ==用于比较变量的值，可以用于任何类型，

如果用于引用类型，比较的是两个引用变量中存储的值（地址信息），

判断两个变量是否指向相同的对象；

• equals是Object的方法，Object默认的equals方法比较规则与==一样。

重写以后，可以用于比较两个对象的内容是否相等

### 4、包装类

包装类

\* 由于8个基本类型没有面向对象特性，为了能让他们参与到

\* 面向对象的开发中，Java为这8个基本类型提供了对应的8个包装类

|  |  |  |
| --- | --- | --- |
| **基本类型** | **包装类** | **父类** |
| **byte** | **Byte** | **java.lang.Number** |
| **short** | **Short** | **java.lang.Number** |
| **int** | **Integer** | **java.lang.Number** |
| **float** | **Float** | **java.lang.Number** |
| **long** | **Long** | **java.lang.Number** |
| **double** | **Double** | **java.lang.Number** |
| **char** | **Character** | **java.lang.Object** |
| **boolean** | **Boolean** | **java.lang.Object** |

6个和数字相关的包装类继承自java.lang.Number

Number是一个抽象类，定义了不同类型的数字之间互相转换的方法，

所以6个数字类型的包装类都可以将其表示的基本类型在这6个类型之间转换

1. **valueOf：**包装类的静态方法 **，**将基本的类型转换为包装类

在byte范围内转换：

Integer i1=Integer.valueOf(127);

Integer i2=Integer.valueOf(127);

System.out.println(i1);

System.out.println(i1==i2);//true

System.out.println(i1.equals(i2));//t**rue**

1. **基本类型Value();**将包装类型数据转换为基本类

int d=i1.intValue();

double dou=i1.doubleValue();

byte bt=i1.byteValue();

3）**MAX\_VALUE**（最大值）和**MIN\_VALUE**（最小值）：数字类型的包装类都支持这两个常量，

分别表示对应的基本类型的取值范围。

int max=Integer.MAX\_VALUE;

int min=Integer.MIN\_VALUE;

4）**parseXX(String str)：**包装类够提供了一个静态方法，可以将字符串解析为对应的基本类型数据

但是前提是该字符串描述的数据能被基本类型正确表示

String str="123";

int d=Integer.parseInt(str);

double dou=Double.parseDouble(str);

String s="123.123";

int d1=Integer.parseInt(s);//整型装不了小数

double dou1=Double.parseDouble(s);

**5）自动拆装箱：**

JDK1.5之后推出了一个新的特性：自动拆装箱。该特性是编译器认可，而不是JVM。

编译器在编译代码时若遇到基本类型与包装类互相用时，会自动补全它们之间

转换的代码。这样做可以让我们在源代码中编写时不再考虑互转

**int i=new Integer(1);;//**触发了自动拆装箱,编译器会将代码改变为：

// int i=new Integer(1).intValue();

**Integer in=i;** 触发了自动拆装箱,编译器会将代码改变为：Integer in=Integer.valueOf(i);

## 三、文件操作

### 1、File对象

1）File的每一个实例是用于表示硬盘上的文件或目录（文件夹）

2）使用File可以：

2.1）访问其表示的文件或目录的属性信息（名字，大小，修改时间等）

2.2）操作文件或目录（创建，删除）

2.3）访问一个目录中的所有子项

注：File不能访问文件内的数据

3）访问当前项目目录中的demo.txt文件

3.1）在表示文件或目录时指定的路径尽量不要使用绝对路径，

因为不同的操作系统路径格式不一致。

3.2）相对路径是最常使用的，但是相对路径相对哪里要看程序在那个环境（项目）运行，

而运行环境决定者相对路径。

3.3）在eclipse中运行程序时，“当前目录”指定为该类所处的项目目录。

### 2、File表示文件信息

2.1、**getName()：**获取文件名字

File file=new File("./demo.txt");

String name=**file.getName();**

2.2、**length()：**文件长度：文件占有的字节量

long length=**file.length()**;

2.3、**canRead()：**文件是否可读可写

boolean cr=**file.canRead()**;

2.4**、canWrite()：**文件是否可写

boolean cw=**file.canWrite()**;

2.5、**isHidden()**是否为隐藏文件

boolean ih=**file.isHidden()**;

2.6、**createNewFile()：**在当前目录中创建一个文件test.txt

2.7、**boolean exists()：**该方法用于判断当前File表示的文件或目录是否存在

存在则返回true

File file=new File("./test.txt");

if(!**file.exists()**){

file.**createNewFile()**;

}else{

System.out.println("文件已存在");

}

2.8、**boolean delete()**：将当前目录或目录中的文件删除file.delete();

### 3、File表示目录信息

3.1、**boolean mkdir()**：在当前目录中新建一个目录dir.mkdir();

创建成功返回true，否则返回-1

3.2、**boolean mkdirs()**：在当前目录中创建多个目录(a/b/c):dir.mkdirs();

创建成功返回true，否则返回-1

3.3、**boolean isFile()**：判断当前Flile表示的是否为文件

**boolean isDirectory()**：判断当前File表示的是否为目录

3.4、**listFiles()：**获取当前目录中的所有子项

3.5、**boolean delete()**:删除当前目录中的目录。

删除目录有一个前提条件：该目录必须是一个空目录才可以删除

3.6、**递归调用**：在方法中调用自己

建议：递归调用耗费资源大，能不用就不用

将给定的File表示的文件或目录删除

public static void delete(File file){

if(file.isDirectory()){

File[] subs=file.listFiles();

for(int i=0;i<subs.length;i++){

File sub=subs[i];

delete(sub);//递归调用

}

}

file.delete();

}

3.7、**startsWith(String str)**:以str开头,file.getName().startsWith(".");

**endsWith(String str)**：以str结尾的,file.getName().endsWith(".txt");

3.8、**文件过滤器**

1）实现了文件过滤器接口后，要求必须**重写accept()**方法，

该方法的作用是定义过滤器规则，当参数file满足要求

该方法返回true即可

class MyFilter implements FileFilter{

public boolean accept(File file) {

return file.getName().endsWith(".txt");

}

}

2）使用**匿名内部类**形式完成文件过滤器创建

获取一个目录中满足条件的子项

File dir=new File(".");

File[] subs=dir.**listFiles(new FileFilter(){**

**public boolean accept(File file) {**

**return file.getName().startsWith(".");**

**}**

**});**

### 4、RandomAccessFile文件访问操作类

java.io.RandomAxxessFile:该类是专门用来读写文件数据的，其基于指针操作。

创建RandomAccessFile常用构造方法：

1）RandomAccessFile（String path,String mode)

2）RandomAccessFile（File file ,String mode)

\* 第一个参数是指定要进行读写操作的文件

\* 第二个参数为权限，常用的**r**:只读模式；**rw**:读写模式

\* 若文件（file）存在时可进行读写操作，若文件（path）不存在

**RandomAccessFile**在**rw**模式下则会自动创建一个文件（path）

eg:RandomAccessFileraf=new RandomAccessFile("./raf.dat","rw");

若不存在文件"raf.dat"，会在当前目录下创建一个文件"raf.dat"。

4.1、字节数据读写操作

1）**int read()**；

\* 从文件中读取1个字节，并以int形式返回。

\* 若返回值为-1则表示读取到了文件末尾

2）**void write(int d)**：RandomAccessFile提供的一个可以向文件中写出字节的方法

\* 向文件中写入1个字节数据，写的是给定的int值对应的2进制的“低八位”，raf.write(1);

3）**void close()**:RandomAccessFile在对文件的操作全部结束后要调用close()方法来

释放与其关联的所有系统资源。raf.close();

4）**int read(byte[] data)**：RAF提供的一次读写一组字节的方法：

4.1）一次性读取给定的字节数组总长度的字节量，

4.2）“**rw**”权限时，将这些字节存入到该数组中，返回值为本次实际读取到的字节量，

“**r**”权限时，不会将读取的字节存入该数组中。

4.3）若返回值为-1，则表示本次读取的是文件末尾

5）**void write(byte[] data)**：一次性将给定的字节数组中所有字节写入文件

**void write(byte[] data,int offset,int len)**：

\* 将给定数组(byte)中从下标offset处的连续len个字节一次性写出。

6）**byte[] getBytes()**：String提供的将字符串转换为字节的方法

\* 按照系统默认字符集转换（不推荐，存在平台差异）

**byte[] getByte(String csn)**： 按照给定的字符集转换，字符集的名字不区分大小写

\* byte[] data=str.getBytes("UTF-8");

\* 常见字符集：

\* GBK：国标编码

\* UTF-8：unicode的子集，也称为万国码

7）**String(byte[] data)**：String提供的构造方法，按照系统默认字符集将字节数组内容转换为字符串

**String(byte[] data,String csn)**：按照给定字符集将字节数组内容转换为字符串

**String(byte[] data,int offset,int len,String csn)**：

\* 将给定字节数组中从下标offset处的连续len个字节

\* 按照给定字符集转换为对应字符串

String str=new String(data);

String str=new String(data,0,len,"UTF-8");

8）**longgetFilePointer()**:获取指针位置.*long pos=raf.getFilePointer();*

\* RAF是基于指针进行读写数据的，即：RAF总是在指针当前位置读或写字节的

\* 并且读写完毕后指针会自动移动到下一个字节的位置。

\* 创建RAF后指针默认在文件的第一个字节位置

//将一个int最大值写入文件

9）**>>>位移运算**：将2进制整体向右移动，左面补0（位移产生新的数，原数不变）

01111111 11111111 11111111 11111111

max>>>24

00000000 00000000 00000000 01111111 11111111 11111111 11111111

max>>>16

00000000 00000000 01111111 11111111 11111111 11111111

int max=Integer.MAX\_VALUE;

raf.write(max>>>24);

raf.write(max>>>16);

raf.write(max>>>8);

raf.write(max);

10）**writeInt(int d)**：RAF提供的方便写出基本类型数据的方法，一次性将

给定的int 值对应的4字节全部写出。raf.writeInt(max);

**writeLong(max)**：long型连续写出8字节，raf.writeLong(max);

write基本数字类型()；

11）**void seek(long pos)**：将指针移动到指定位置raf.seek(0);

注意：\* 若读取4字节int值得过程中发现读取到了末尾，

\* 则抛出EOFException end of file异常 。

## 四、基本IO操作

### 1、IS与OS（输入流与输出流）

1.1、节点流与处理流

1）java.io

java提供了标准的IO操作，即：输入与输出

IO的作用是可以让我们的程序与外界进行数据交换。

比如从网络读写数据，读取键盘数据，读写文件数据等等

2）标准的IO将功能按照方向划分

输入：从外界到程序中的过程，只是“读”的过程

输出：从程序发送到外界的过程，这是“写”的过程

3）**学习IO的重点**：理解流连接操作，通过应用流连接，按照需求组合高级流与低级流，完成读写操作。

4）流分为：节点流（低级流），处理流（高级流）

4.1）节点流：实际连接程序与另一端的管道，负责在两端之间传送数据，

注意：读写一定是建立在节点流的基础上进行的。

4.2） 处理流：不能独立存在，可以连接在其他流上，处理流自带某种对数据的加工操作，

所以数据流经该流，会对这些数据进行处理，这样可以简化我们对数据的处理操作。

1.2、IS、OS抽象类，不能被实例化。

1）java.io.**InputStream**

所有字节输入流的父类，是一个抽象类，规定了所有字节输入流

都必须具备的读取字节的方法。

* 1. **int read()**：读取一个字节，以int形式返回，该int值的“低八位”有效，

若返回值为-1则表示EOF（end of file）

* 1. **int read(byte[] d)**：尝试最多读取给定数组的length个字节并存入该数组，

返回值为实际读取到的字节量。

2）java.io.**OutputStream**

所有字节输出流的父类，是一个抽象类，规定了所有字节输出流

都必须具备的写出字节的方法。

2.1）**void write(int d)**：写出一个字节，写的是给定的int的“低八位”。

2.2）**void write(byte[] d)**：将给定的字节数组中所有的字节全部写出。

### 2、文件流

2.1、文件流是一套低级流，作用是读写文件数据。

1）FileOutputStream:文件字节输出流FileOutputStream fos=new FileOutputStream("fos.txt");

若文件不存在会自动创建一个文件

1.1）**void write(int d)**：将指定字节写入此文件输出流。，这里只写int值的“低八位”。

2.2）**void write(byte[] d)**：将b.length个字节从指定byte数组写入此文件输出流中。

2.3）void write(byte[] d,int offset,int len)：将指定byte数组中从offset开始的len个字节

写入此文件输出流中。

2）FileInputStream:文件字节输入流

2.1）**int read()**：从此输入流中读取一个字节，若返回值为-1则表示EOF（end of file）

2.2）**int read(byte[] d)**：从此输入流中将最多b.length个字节的数据读入到字节数组b中。

返回值为实际读取到的字节量。

3）文件输出流支持的两种写模式：

3.1）覆盖写操作，即：若写出的文件有数据，则将原有数据全部删除，

将本次通过流写出的内容作为文件数据。

FileOutputStream(String path)

FileOutputStream(File file)

3.2）追加写操作，若文件有数据则全部保留，从该文件末尾追加内容。

FileOutputStream(String path,boolean append)

FileOutputStream(File file,boolean append)

**eg:**FileOutputStream fos=new FileOutputStream("fos.txt",true);

2.2、文件流与RandomAccessFile的对比：

1）论功能是一致的，都是用来读写文件数据

2）RAF对于文件即可读也可写，但是文件流不行，文件输入流

只用来读文件数据，文件输出流用来向文件中写入数据

3）流的特点是顺序读写操作，即：读写是不能回退的。

RAF是随机读写操作，因为依靠指针位置进行读写，

所以可以通过操作指针对文件任意位置随意读写

### 3、缓冲流

3.1、缓冲流是一对高级流，作用是提高读写效率。

\* java.io.BufferedInputStream

\* java.io.BufferedOutputStream

3.2、缓冲流实际上本质就是进行块读写操作，无论我们使用缓冲流进行何种

读写方式（随机读写，块读写），最终都会被缓冲流转换为块读写形式

通过文件流进行实际操作。缓冲流内部维护了一个8k的字节数组（缓冲区），

用于块读写操作。输入时先一次性读取若干字节存入缓冲区，再逐一返回字节；

输出时会先将数据存入缓冲区，当缓冲区存满时将数据一次性全部写出。

减少读写次数达到提高读写效率目的。

3.3、强制将当前缓冲区数据写出：bos.flush();

缓冲流close()方法时会自动调用flush.

3.4、扩展：有高级流时只需要关闭最外层的高级流即可

3.5、使用文件流进行文件复制操作

使用文件输入流读取原文件，使用文件输出流向目标文件中写出，

然后顺序从原文件读取字节写入目标文件完成复制操作

使用块方式复制

byte[] data=new byte[1024\*10];

FileInputStream fis=new FileInputStream("fos.txt");

int len=-1;

FileOutputStream fos=new FileOutputStream("fos\_cp.txt");

while((len=fis.read(data))!=-1){

fos.write(data, 0, len);

}

### 4、对象流

将对象转换为一组字节：对象序列化

将字节写入文件：数据持久化

对象持久化：先对象序列化再数据持久化。

4.1、对象流是一对高级流，可以方便我们将java中任何对象进行读写操作。

1）java.io.**ObjectOutputStream**：对象输出流，可以将对象转换为一组字节写出

2）java.io.**ObjectInputStream**：对象输入流，可以读取一组字节将其还原为对象，

前提是读取的这组字节应当是对象输出流将一个对象转换成的字节

4.2、oos.**writeObject(Object obj);**将给定对象转换为的一组字节写出

方法在当前案例中经过了两个流：oos，fos。

4.3 **对象序列化**：oos对象输出流将给定的对象转换为了一组字节的过程。

oos将对象转换为字节后将这组字节交给fos写入文件。

**数据持久化**：fos文件输出流将数据写入文件的过程，等于写入到硬盘中作长久保存

**对象持久化**：先对象序列化再数据持久化。

**对象反序列化**：ois对象输入流读取一组字节并将其还原为对象的过程

4.4、若一个类的实例希望能被对象流读写，那么要求该类必须实现java.io.**Serializable**接口

签名接口：**Serializable**没有任何内容，主要用于序列化识别

4.5、**transient**：当一个属性被transient修饰后,这个对象在序列化时该属性的值会被忽略，忽略

不必要的属性可以达到对象序列化（瘦身）的功能：private **transient** String[] otherInfo;

4.6、**serialVersionUID常量**：序列化版本号。

当一个类实现了Serializable接口后，应当定义一个常量：serialVersionUID

序列化版本号影响反序列化时是否成功，当对象输入流

在进行反序列化时会对比该对象与当前类的版本号是否一致，

一致则反序列化成功，不一致则抛出异常。

若我们没有指定版本号，编译器会在编译时按照当前类的结构生成一个，

但是这样做会导致当前类发生改变，那么版本号一定会变，

这样就无法控制反序列化规则：private static final long serialVersionUID = 1L;

## 五、文本数据IO操作

### 1、字符流

1.1、java将流按照读写单位划分为字节流与字符流

**InputStream**和**OutputStream**是所有字节流父类

java.io.**Reader**和java,io.**Writer**是字符流的父类

字符流只是方便我们读写字符，底层本质还是读写字节，

只是字节与字符的转换工作交给了字符流来完成。

1.2、Reader常用方法：

1）**int read()**：读取一个字符，返回的int值“低16位”有效。

2）**int read(char[] chs)**：从该流中读取一个字符数组的length个字符

并存入该数组中，返回值为实际读取到的字符量。

1.3、Writer常用方法：

1）**void write(int c)**：写出一个字符，写出给定int值“低16位”表示的字符。

2）**void write(char[] chs)**：将给定字符数组中所有字符写出。

3）**void write(String str)**：将给定的字符串写出。

4）**void write(char[] chs,int offset,int len)**：将给定的字符数组中从offser处

开始连续的len个字符写出。

### 2、字符转换流

java提供的其他高级字符流都有一个特点就是只能连在其他字符流上，

但是通常低级流都是字节流，这就导致字符流不能直接搭配字节流使用，

但是转换流例外，他们本身是字符流，而他们又**可以连接字节流**，

所以在实际开发中我们使用高级的字符流时，通常与字节流连接时要使

用转换流，他起到“将字符转换字节”的功能承上启下。

2.1、java.io.**InputStreamReader**：字符输入流，使用该流可以设置字符集，并按照指定的字符集从流中

按照改编码将字节数据转换为字符并读取。

2.2、 java.io.**OutputStreamWriter**：字符输出流，使用该流可以设置字符集，并按照指定的字符集

将字符转换为对应字节后通过该流写出。

### 3、缓冲字符流

3.1、**BufferedWriter**：缓冲字符输出流，内部提供了缓冲区，可以提高写出效率。

3.2、java.io.**PrintWriterWriter**：带有自动行刷新的缓冲字符输出流（常用）。

1）直接提供了向文件中写入字符串的构造方法，不用连接低级流也能向文件中写入数据。

\* PrintWriter(String path)

\* PrintWriter(File file)

PrintWriter pw=new PrintWriter("pw.txt","utf-8");

pw.println(String str1);

pw.println(String str1);

pw.close();

2）在流连接中使用时，需要在转换流中指定字符集

PrintWriter pw=new PrintWriter( //缓冲字符流 （行刷新）

new BufferedWriter( //缓冲字符流

new OutputStreamWriter( //转换字符流

new FileOutputStream("pw2.txt") //文件流

,"utf-8"

)

)

);

3）PrintWriter的构造方法中若第一个参数为流,那么就支持一个重载构造方法，

允许再传入一个boolean值类型的参数，若该值为true，则具有自动行刷新的功能

即：每当使用println方法写出一行字符串后就会自动flush（强制写出）

**PrintWriter pw=new PrintWriter(ows,true);**

PrintWriter pw=new PrintWriter(

new BufferedWriter(

new OutputStreamWriter(

new FileOutputStream(fileName),"utf-8")),true);

4）字符串当遇到字面量与变量进行内容比较时应当使用:**字面量.equals（变量）**

因为变量在实际开发中不能保证null，所以可能引发空指针异常，

字面量一定不是空所以不会出现该问题。

3.3、 java.io.**BufferedReader**：缓冲字符输入流，可以按行读取字符串

FileInputStream fis=new FileInputStream("./src/io/BufferedReaderDemo.java");

InputStreamReader isr=new InputStreamReader(fis);

BufferedReader br=new BufferedReader(isr);

1）**String readLine()**；连续读取若干字节，直到读取到换行符为止，然后将换行符之前

读取的字符以一个字符串形式返回。注意，返回的字符串中不含有最后读取到的换行符。

若返回值为null，则表示末尾

2）按行获取**键盘输入**的内容

BufferedReader br=new BufferedReader(

new InputStreamReader(**System.in**)

);

String line=null;

while(true){

line=br.readLine();

if("exit".equals(line)){//结束开关

break;

}

System.out.println(line);

}

br.close();

## 六、异常处理

当JVM运行程序时发现出现了某个异常时会自动实例化该类型的异常实例，并将代码执行过程

设置到该异常类实例中将其抛出。若抛出异常的代码没有被异常处理机制包含那么JVM会将该异常类

抛到当前方法之外，若抛出到main方法之外，则当前程序中断。（单线程情况下）

### 1、java异常捕获和处理机制

1.1、try-catch

1）语法：

**try{**

**代码片段**

**}catch(XXException e){**

**捕获try代码片段中出现的XXException并处理**

**}**

1. e.**printStackTrace()**：输出错误堆栈信息

e.**getMessage()**:获取错误信息

try{

}catch(Exception e){

e.printStackTrace();

String message=e.getMessage();

System.out.println(message);

}

3）try中出错代码以后的内容都不会再执行

4）、应当养成一个好习惯，在最后一个catch捕获Exception,

防止因为未捕获的异常导致程序中断：**catch(Exception e){}**

5）finally块：只能定义在异常处理机制的最后，可以直接跟在try后面或者最后一个catch之后。

finally可以保证只要程序运行到try当中，那么无论try当中的代码片段是否出现异常，

finally块里面的代码都必然执行。

通常把释放资源等操作放在finally中，比如流的关闭。

6）JDK1.7之后推出了一个新的特性：流自动关闭。AutoCloseable接口的子类可以定义在try**()**内

自动关闭语法是编译器认可，编译后的class文件中流的关闭还是在finally中进行的。

try(FileInputStreamfis=new FileInputStream("osw.txt");){

int d=fis.read();

System.out.println(d);

}catch(IOException e){

}

1.2、**throw**与**throws**

1）一个方法内部使用throw抛出一个异常，就要在方法上（方法签名后）使用throws

声明该异常的抛出以告知调用者处理这个异常。

**注意**：只有RuntimeException及其子类型异常在方法中抛出是不要求必须在方法上

声明该异常的抛出，其他类型异常则是必须的，否则编译不通过异常的抛出

2）**throw**关键字，在方法内，用于将一个异常抛出，

通常两种情况会主动抛出一个异常：

1.1）程序遇到一个满足语法要求，但是不满足业务逻辑要求的时候

可以主动抛出一个异常给调用者。

1.2）程序出现了异常，但是不应当在当前代码片段中解决该异常时可以抛出给调用者。

3） **throws：**满足语法，但是不满足业务逻辑要求的异常，又不希望在方法里处理这些异常，

这时可以使用throws声明这个方法将会抛出异常。

**注意**：当我们调用一个含有throws声明异常抛出的方法时，编译器要求我们必须

处理这个异常，处理方式有两种：

3.1）try-catch自行捕获并处理。

3.2）在当前方法上继续使用throws声明该异常抛出

1.3、自定义异常，异常类继承Exception,类中重写所有构造方法，

在会有该异常的方法内抛出该异常类，通常用于描述业务逻辑错误。

public class IllegalAgeException extends Exception{}

## 七、TCP通信

/\*\*

\* 聊天室服务端

\*

\*/

**publicclass** Server {

/\*

\* 运行在服务端的ServerSocket

\* 有两个主要作用：

\* 1、向系统申请对外的服务端口，客户端就是通过这个端口

\* 与服务端建立连接的。

\* 2、监听服务端口，等待客户端连接，一旦一个客户端

\* 通过Socket与服务端建立连接，那么ServerSocket

\* 会创建一个Socket与该客户端通讯

\*/

**private** ServerSocket server;

//该数组用于存放所有客户端的输出流

// private PrintWriter[] allOut=new PrintWriter[0];

**private** List<PrintWriter>allOut=**new** ArrayList<PrintWriter>();

// private List<PrintWriter> allOut=Collections.synchronizedList(new ArrayList<PrintWriter>());

**public** Server(){

**try** {

/\*

\* 创建ServerSocket的同时向系统申请对外的服务端口，

\* 注意，该端口不能和其他程序冲突，否则会抛出地址被占用的异常。

\*/

server=**new** ServerSocket(8088);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**publicvoid** start(){

**try** {

/\*

\* ServerSocket提供方法：

\* Socket accept()

\* 该方法是一个阻塞方法，调用后即等待客户端的连接，一旦一个客户端通过

\* 端口连接，那么accept方法会返回一个Socket实例，通过这个Socket实例就可以

\* 与刚连接的客户端交互了。

\*/

**while**(**true**){

System.***out***.println("等待客户端连接...");

Socket socket=server.accept();

System.***out***.println("一个客户端连接了");

//启动一个线程，处理该客户端交互工作

ClientHandler handler=**new** ClientHandler(socket);

Thread t=**new** Thread(handler);

t.start();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

**publicstaticvoid** main(String[] args) {

Server server=**new** Server();

server.start();

}

/\*

\* 该线程任务负责与指定Socket所对应的客户端进行交互。

\*/

**privateclass** ClientHandler **implements** Runnable{

**private** Socket socket;

/\*

\* 实例化时将对应客户端的Socket传入

\*/

**public** ClientHandler(Socket socket){

**this**.socket=socket;

}

**publicvoid** run(){

PrintWriter pw=**null**;

**try** {

InputStream in=socket.getInputStream();

BufferedReader br=**new** BufferedReader(

**new** InputStreamReader(in, "utf-8")

);

//获取输出流，用于给当前客户端回复消息

pw=**new** PrintWriter(

**new** BufferedWriter(

**new** OutputStreamWriter(

socket.getOutputStream(),"utf-8"

)

),**true**

);

//将当前客户端对应的输出流存入共享数组

**synchronized** (allOut) {

//

// //1、扩容数组

// allOut=Arrays.copyOf(allOut, allOut.length+1);

// //2、将输出流存入数组

// allOut[allOut.length-1]=pw;

allOut.add(pw);

}

/\*

\* 客户端在断开连接时不同系统有不同反应。

\* linux的客户端断开后，br.readLine方法会返回null。

\* windows的客户端断开连接后，br.readLine方法会直接抛出异常。

\*/

String message=**null**;

**while**((message=br.readLine())!=**null**){

System.***out***.println("客户端说："+message);

// pw.println("客户端说："+message);

//发送给所有客户端

**synchronized** (allOut) {

// for(int i=0;i<allOut.length;i++){

// allOut[i].println("客户端说："+message);

// }

//数组换成了集合

**for**(PrintWriter o:allOut){

o.println("客户端说："+message);

}

}

}

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

//处理客户端断开连接测操作

**synchronized** (allOut) {

//1、将该客户端的输出流从共享数组中删除

// for(int i=0;i<allOut.length;i++){

// if(allOut[i]==pw){

// //将最后一个元素放入当前位置

// allOut[i]=allOut[allOut.length-1];

// //缩容

// allOut=Arrays.copyOf(allOut, allOut.length-1);

// break;

// }

// }

allOut.remove(pw);

}

/\*

\* 2、客户端断开连接后，服务器关闭

\* 该客户端socket,释放资源

\*

\*/

**try** {

socket.close();

} **catch** (Exception e2) {

e2.printStackTrace();

}

}

}

}

}

## 八、多线程基础

### 1、进程与线程

1.1、**进程**是操作系统中运行的一个任务（一个应用程序运行在一个进程中）。

当操作系统创建一个进程后，该进程会自动申请一个主线程（首要线程）。

**线程**是进程的一个顺序执行流。

1.2、进程与线程区别

一个进程至少有一个线程。

进程在执行过程中拥有独立的内存单元，而多个线程共享内存。

线程不能独立执行，必须依存在应用程序中。

1.3、使用场合：多线程允许我们“同时执行多段代码。

程序中需要同时完成多个任务，可以将每个任务定义为一个线程，

使他们得以一同工作。多线程执行更快。

### 2、 并发运行

线程是并发运行的，线程调度会统一规划CPU时间，将CPU时间划分为若干片段，

然后尽可能的均匀分配给所有要并发运行的线程，每个线程在获得CPU时间片断后，CPU

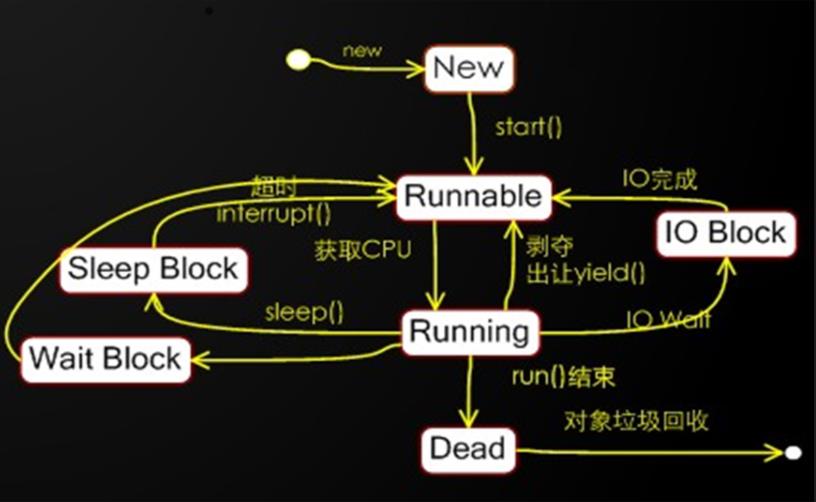
就来运行它的任务，当时间片断用完后，CPU会离开并执行下一个获取到CPU时间片段

的线程。所以所有线程并非真正的“同时”运行着代码，而是走走停停停，这种微观上走走

停停，宏观上感觉是同时运行的现象称为并发运行。

### 3、线程生命周期

线程创建，Runnable等待，Running并发运行，run()结束线程死亡回收Thread



### 4、线程的创建

线程提供了两种创建方式：

4.1、第一种创建线程的方式：

继承Thread并重写run方法来定义线程任务。

class MyThread1 extends Thread{

public void run(){

for(int i=0;i<1000;i++){

System.out.println("你是谁啊？");

}

}

}

注意：启动线程不要调用run方法，而是要调用start方法。

线程开始并发运行时会自动调用run方法。

Thread t1=new MyThread1();

t1.start();

第一种创建线程的方式有两个不足：

1）由于java是单继承，这就导致若继承了Thread就不能

再继承其他类，在实际开发中非常不方便，因为无法重用其他的某些方法。

2）由于继承Thread后重写run方法定义了线程要执行的任务，

这就导致线程与线程要执行的任务有一个必然的耦合关系，不利于线程重用。

4.2、第二种创建线程的方式：

实现Runnable接口单独定义线程任务。

class MyRunnable1 implements Runnable{

public void run(){

for(int i=0;i<1000;i++){

System.out.println("你是谁？");

}

}

}

注意：与第一种方式一样，启动线程不要调用run方法，而是要调用start方法。

线程开始并发运行时会自动调用run方法。

Runnable r1=new MyRunnable1();

Thread t1=new Thread(r1);

t1.start();

### 5、匿名内部类形式创建线程

5.1、匿名内部类直接创建Thread

Thread t1=new Thread(){

public void run(){

for(int i=0;i<100;i++){

System.out.println("你谁？");

}

}

};

t1.start();

5.2、匿名内部类创建Runnable，再由线程运行

Runnable r2=new Runnable(){

public void run(){

for(int i=0;i<100;i++){

System.out.println("我来查水表的！");

}

}

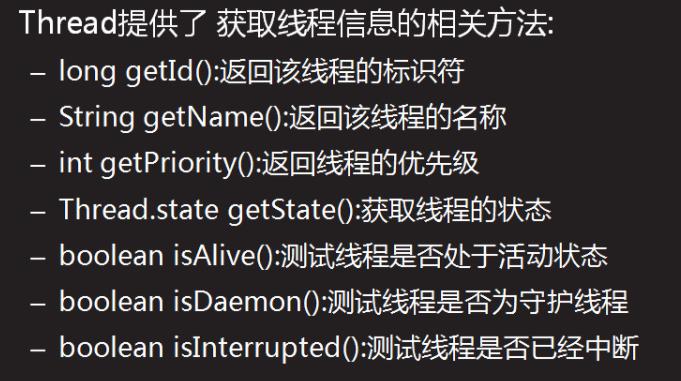
};

Thread t2=new Thread(r2);

t2.start();

### 6、线程操作API

6.1、获取线程信息



6.2、线程优先级

1）线程调用start方法后纳入线程调度统一管理，线程不能主动获取CPU时间片，只能被动分配。

调整线程优先级可以最大程度的改善某个线程获取CPU时间片的几率。

理论上线程优先级越高的线程获取CPU时间片的次数越多。

2）线程优先级有10个等级，用1-10表示。1最低，5默认，10最高。

3）**void setPriority(int p);**通过该方法可以设置线程的优先级。

Thread max=new Thread(){

public void run(){

for(int i=0;i<10000;i++){

System.out.println("max");

}

}

};

max.setPriority(Thread.MAX\_PRIORITY);

4）线程提供了3个常量分别表示了最低，默认和最高优先级。

**Thread.MAX\_PRIORITY**：最高优先级

**Thread.MIN\_PRIORITY**：最低优先级

**Thread.NORM\_PRIORITY**：默认优先级

6.3、**守护线程**又称为后台线程，默认创建的线程都不是守护线程。

1）**void setDaemon(boolean on)**：调用线程提供的方法设置，当参数为true时，

线程变为守护线程。在start调用前进行设置

jack.setDaemon(true);

jack.start();

2）守护线程使用与普通线程没有区别，但是结束时机上有一个区别之处，

即：当进程结束时，所有正在运行的守护线程会强制结束。

3）进程的结束：当一个进程中所有普通线程都结束时，进程结束。

6.4睡眠阻塞

1）static void **sleep(long ms)**

当一个线程执行sleep方法后就会进入阻塞状态指定的毫秒，超时后线程会

自动回到Runnable状态等待再次并发运行。该方法会要求必须处理InterruptedException，

即：当一个线程处于睡眠阻塞时被其他线程调用interrupt方法中断时会抛出该中断异常

并打断睡眠阻塞。

2）**interrupt**（）:是线程的一个方法，用于中断线程。但是若线程处于阻塞状态时是

中断阻塞，若线程没有处于阻塞状态

则线程直接被中断。

3）**扩展**：在JDK1.8之前，由于JVM内存分配问题，要求一个方法中的局部内部类若引用

这个方法的其他局部变量时，要求该变量必须是final的。

6.5、static void yield()：Thread的静态方法，用于使当前线程主动让出CPU时间片回到

Runnable状态，等待分配时间片。

6.6、static Thread **currentThread**()：线程提供了一个静态方法。

该方法可以获取运行这个方法的线程

1）java中所有的代码都是线程运行的，main方法也不例外。JVM启动后会启动一个线程

来执行main方法，通常我们称执行main方法的线程为“主线程”。

2）在**Spring**的**AOP**中有一个功能，对数据库事务的切面处理，这里本质上用到了

ThreadLocal这个类，而里面就要使用Thread.currentThread来得知运行代码的线程。

6.6、void **join()：**线程提供的一个方法用于等待当前线程结束，

该方法允许一个线程在另一个线程上等待，直到其执行完毕后再继续运行。

这样做可以协调线程间的“同步”运行。方法声明抛出InterruptException异常。

**同步操作（运行）**：代码执行有先后顺序（单线程运行时同步，多线程也可以进行同步运行操作）

**异步操作（运行）**：多线程并发操作，代码各执行各的（多线程下运行代码是异步的）

download.join()：其它线程在调用该方法后就进入了阻塞状态，

直到线程download运行完毕才会解除阻塞。

### 7、线程同步

7.1、多线程并发安全问题

当多个线程并发运行操作同一数据时，由于线程切换的时机不可控，可能会导致操作该数据时

的过程未按照程序设计的执行顺序运行，导致操作出现混乱，严重时可能会导致系统瘫痪。

7.2、当一个方法是用**synchronized**修饰后，那么该方法变为“同步方法”，多个线程不能同时

进入（同一个对象的）方法内容运行，而必须有顺序的一个一个运行，这样就可以避免并发安全问题。

1）在方法上使用synchronized，那么同步监视器对象就是该方法所属对象，即：方法中看到的this。

2）有效的缩小同步范围可以在保证并发安全的前提下尽可能提高并发效率。

7.3、**同步块:**

**synchronized(同步监视器-锁对象引用){**

**需要同步运行的代码;**

**}**

同步快可以更灵活准确的锁定需要同步运行的代码片段，这样可以有效缩小同步范围高并发效率。

但是需要注意：必须保证多个线程看到的同步监视器对象是“同一个”才可以。

7.4、静态方法若使用synchronized修饰后，那么该方法一定具有同步效果。

了解：静态方法的所属对象是当前类的对象 Class类（特殊的类），

Class类的每一个实例用于表示JVM加载的一个类，

当JVM加载一个类的时候就会实例化一个Class的实例用于表示他，

每个类在JVM内部都有且只有一个Class的实例，所以静态方法锁的

就是当前类对应的Class的实例

7.5、**互斥锁**

使用snychroneized锁定多段代码，而锁对象相同时，

这些代码片段之间就是互斥的，多个线程不能同时执行

这些方法

public synchronized void methodA(){}

public synchronized void methodB(){}

7.6、**死锁现象**

线程都保持着自己的锁，但是都等待对方先释放锁时

就出现了互相“僵持”的情况，导致程序不会再继续向下运行。

# **第四章、JAVASE02（标准Java版本）**

## 一、集合框架

### 1、Collection

1.1、java.util.**Collection**接口

Collection是所有集合的顶级接口，规定了所有集合都应当具备的方法

1.2、Collection下面有两个常见的子接口

java.util.**List**:可重复集，且有序

java.util.**Set**：不可重复集，大部分实现类无序。

注：元素是否重复依靠元素自身equals比较的结果而定。

1.3、创建**List**集合：Collection c=new ArrayList();

创建**Set**集合：Collention set=new HashSet();

1.4、集合操作方法：

1）boolean **add(E e)**;向当前集合中添加元素c.add("one"),成功添加则返回true.

2）int **size()**;返回当前集合的元素个数。

3）boolean **isEmpty()**;判断当前集合是否为空集，当集合中不含有任何元素是返回true

注意：与null区别，null表示没有集合。

4） void **clear()**;清空当前集合元素。c.clear();

5）boolean **contains(Object e)**;判断当前集合是否包含给定元素。集合会将给定元素与

集合元素顺序进行equals(重写）比较，若有比较为true的则认为集合包含该元素。

6）boolean **remove(Object o)**：删除给定元素，集合删除元素是根据给定元素与集合

元素顺序equals比较，删除为true的，对于List集合而言，若存在多个，则只删除一个。

1.5、集合持有对象的引用：集合中存储的都是引用类型元素，并且集合只保存每个

元素对象的引用，而并非将对象本身存入集合。

1.6、集合间的操作

1）boolean **addAll(Collection c)**;将给定集合元素添加到当前集合中。c1.addAll(c2);

2）boolean **removeAll(Collection c)**;删除当前集合中与给定集合的相同元素（交集）。

c1.removeAll(c3);

3）boolean **containsAll(Collection c)**:判断当前集合是否包含给定集合。c1.containsAll(c2);

### 2、迭代器Iterator（接口）

遍历集合元素 Collection提供了统一的遍历集合元素的方式：迭代器模式

2.1、Iterator **iterator()**;该方法可以获取一个用于遍历当前集合元素的迭代器实现类。

**Iterator it=c.iterator();**

1）不同的集合实现类都提供了一个可以用于遍历自身的迭代器。我们无需记住每个

迭代器实现类的名字，只需要以Iterator接口接收并调用迭代方法遍历即可。

2）迭代器遍历集合遵循：问（是否存在），取（取出），删的遍历过程，其中删元素不是必须操作。

3）boolean **hasNext()**;判断集合是否还有元素可以遍历

4）E **next()**;取出集合中下一个元素(**迭代器.next()**)

5）迭代器要求在遍历的过程中不允许通过集合的方法增删元素，否则会抛出异常。

删除元素可通过迭代器提供的remove方法删除，删除的是本次通过next方法迭代的元素

6）迭代器.remove():迭代器提供的方法,删除next()取出的集合元素

**Iterator it=c.iterator();**

while(**it.hasNext()**){

String str=(String)**it.next()**;

if("#".equals(str)){

// c.remove(str);

it.remove();

}

}

### ：增强型for循环

2.2、JDK1.5之后推出了一个新的特性

1）**增强型for循环**，也称为：新循环，for each

新循环是编译器认可，编译器会在编译源代码时 将新循环转换为迭代器

模式遍历集合（本质是迭代器），或改为使用传统for循环遍历数组。

新循环并不取代传统for循环的工作，他只用来遍历集合或数组时使用。

2）新循环语法：e用来接收遍历的集合或数组

**for(元素类型 e:集合或数组){**

**循环体**

**}**

3）新循环遍历集合就是迭代器遍历集合，所以注意，遍历过程中不能通过

集合方法增删元素。

### 3、泛型机制

3.1、**泛型**是JDK1.5之后推出的一个特性。

泛型也称为参数化类型，他可以用于指定一个类中的属性类型、方法参数的类型

以及返回值类型等。这样做的好处是将类型的定义权交给了调用者，而不再是提供者决定。提高了灵活性。

3.2、**泛型是编译器认可，而非虚拟机。泛型不指定，原型就是Object**.

3.3、泛型只是用来让编译器辅助做了一些事情

1）设置泛型类型值时，编译器辅助检查数据类型是否匹配

2）获取泛型类型值时，编译器会添加向下造型语句。

3.4、泛型在集合中广泛使用，用于指定集合中的元素类型。若不指定泛型，则默认为原型Object.

Collection<E>**c**=new ArrayList<E>();**E为泛型参数**。

1）Collection<String>**c**=new ArrayList<String>();

编译后的样子：Collection c=new ArrayList();

2）当调用方法传入泛型约束的参数时，编译器会检查实际参数是否符合该参数指定的

泛型类型，不符合编译不通过。

Iterator<String> it=**c**.iterator();

编译后的class文件的样子：Iterator it=c.iterator();

3）当获取一个泛型类型的值时，编译器会自动根据泛型指定的实际类型做向下造型操作。

String str=it.next();

编译后的class文件的样子（向下造型）：String str=(String)it.next()

4）不指定泛型则默认为原型Object

Collection c2=c1;//指向同一个对象，c1为Integer,c2为Object

c2.add("four");

遍历时会出现**类造型异常**

for(Integer i:c1){//c1指向的对象已经改变。

System.out.println(i);

}

## 集合操作——线性表

### List

1.1、java.util.**List**线性表，是Collection的子接口

1）List的特点：可重复集，并且有序。可以通过下标操作元素。

2）常用实现类：

**ArrayList**:内部由动态数组实现List接口，查询快，增删慢。适合随机访问。

**LinkedList**:内部由链表实现List接口，查询慢，增删快（首位增删效率最优）

对于性能不是极端苛刻情况下，通常使用ArrayList

3）定义的方法：

3.1）**E get(int index)**;获取指定位置对应的元素

传统for循环可以遍历List集合

for(int i=0;i<list.size();i++){

String str=**list.get(i)**;

System.out.println(str);

}

3.2）E set(int index,E e);将指定元素设置到指定位置，返回值为原位置对应的元素。

所以是替换元素操作。

String old=list.set(1, "2");

System.out.println(old);//输出two为替换前元素，不是2.

4）List提供了一对**重载**的add，remove方法

4.1）**void add(int index,E e)**;将给定元素插入到指定位置。

list.add(1,"2");

4.2）**E remove(index)**;删除并返回指定位置对应的元素

（需要就定义变量接收，不需要就直接写方法）

String old=list.remove(2);

5）子集操作

**List subList(int s,int e)**;截取指定的位置s(包含)到e(不包含)之间的子集。

对子集元素操作就是对原集合对应元素的操作。

（个人理解：截取是获取，子集引用指向原集的子集部分，并不是新建对象。）

List<Integer> subList=list.**subList**(3, 8);

将子集元素扩大到10倍：

for(int i=0;i<subList.size();i++){

subList.set(i, subList.get(i)\*10);

}

6）**void clear()**;删除List集合。（remove是删除元素）

list.subList(2, 9).clear();

7）集合数组相互转换

1）集合转换成数组

**Object[] toArray()**;

**<T>T[] toArray(T[] a)**;（常用）Collection提供了一个方法,将当前集合转换为数组

数组元素类型与集合类型一致。

2）数组转换为集合（不重复，只能转为List）

**staticasList(arr)**：集合的工具类Arrays的静态方法，将一个数组转换为List集合。

返回的List集合元素类型由传入的数组元素类型决定。

List<String> list=**Arrays.asList**(array);

2.1）对集合元素操作就是对数组元素操作

（个人理解：数组转换成的集合引用指向数组，而不是新建了集合对象。）

2,2）由于添加元素会导致数组扩容，那就不能表示原数组，

所以这样的操作在数组转换的集合中是不支持的。

调用add,remobe会抛出异常。

2.3）若想对集合增删元素，需复制一个集合（新建）。

List<String> list1=new ArrayList<String>(list);

list1.add("five");

### List排序

2.1、java.util.**Collections**：集合的工具类，其定义了很多静态方法，用于操作集合。

定义：**void sort(List<E> list)**;Collections工具类中的静态方法，对List集合进行自然排序（升序）。

：Collections.sort(list);

2.2、排序自定义类型元素的集合

1）sort方法要求（自定义）集合元素必须实现Comparable接口。

实现Comparable<E>接口后，要求必须重写int compareTo(自定义类型 o)方法：

该方法的意义是定义当前对象与参数对象的大小关系

返回值不关注具体取值，只关注取值范围。

当返回值>0:当前对象this大于参数对象o.

当返回值<0:当前对象this小于参数对象o.

当返回值=0:当前对象this等于参数对象o.

2）Collections的sort方法具有侵入性：

2.1)所谓侵入性，是指当我们使用某个功能时，该功能要求我们为其

修改代码，修改的越多侵入性越强，一旦不需要该功能时，这些改动

都没有意义了，这在实际开发中是很不推荐的。

2,2)除了有侵入性之外，该sort方法还有一个问题,就是若元素已经实现了

Comparable接口并定义了比较规则，但是该规则不满足我们实际排序

需求时也没办法，比如String，该类也不允许我们重写他的方法。

2.3)推荐的排序方式:

当遇到排序自定义类型元素，或者集合元素,自身的比较规则不满足排序时都建议使用下面的方式排序。

该sort方法要求额外传入一个**比较器**(匿名内部类)，并按照比较器规则,比较集合元素并进行排序。

因此该排序方法不要求集合元素实现Comparable接口。

Collections.sort(list,**new Comparator<String>()**{

public int **compare**(String o1,String o2){

return o1.length()-o2.length();

}

});

2.3、线程安全的集合

1）常用的集合实现类：ArrayList,LinkedList,HashSet

它们都不是线程安全的，多线程情况下使用会出现并发安全问题。

2）Collections提供了将集合转换为线程安全的方法。

list=**Collections.synchronizedList**(list);

set=**Collections.synchronizedSet**(set);

3）API手册中有说明，就算是一个线程安全的集合也不和迭代器遍历做互斥，

所以若并发操作时存在有遍历操作时要自行维护他们之间的互斥关系。

### 3、队列和栈

3.1、java.util.**Queue**：**队列**,规定了队列相关存取元素等操作。可以保存一组元素，

但是存取元素必须遵循：先进先出。

Queue接口继承自java.util.Collection接口,所以集合的相关方法他也具备。比如size,clear等

常用实现类：java.util.**LinkedList。**

创建队列：Queue<String> queue=new LinkedList<String>();

**对元素的操作方法**：(队列左首右尾）

1）**boolean offer(E e)**;入队操作。

将一个对象添加至队尾，添加成功则返回true。queue.offer("one");

2）**E poll()**;出队操作。从队首删除并返回一个元素。

String str=queue.poll();

3）**E peek()**;引用队首元素，不做出队操作。返回队首的元素但不删除。

String str=queue.peek();

4）遍历

4.1）一次性遍历，遍历后队列就不存在了（全取出了）

while(queue.size()>0){

str=queue.poll();

}

4.2）迭代器遍历，队列还存在

for(String s:queue){

System.out.println(s);

}

3.2、java.util.**Deque：双端队列**。该接口继承自java.util.Queue

双端队列的特点是两端都可以进出队

创建双端队列：Deque<String> deque=new LinkedList<String>();

**对元素的操作方法：**(**offer()、poll()、peek()**方法同queue)

1）**offerFirst(E e)**：将对象添加至队首

deque.offerFirst("three");

2）**offerLast(E e)：**将对象添加至队尾

deque.offerLast("four");

3）**pollFirst()：**从队首方向获取元素（同**poll()**）

str=deque.pollFirst();

4）**pollLast()**：从队尾方向获取元素，获取队尾元素并返回

str=deque.pollLast();

1. **peekFirst()**;引用队首元素(同**peek()**)，不做出队操作。返回队首的元素但不删除。

str=deque.peekFirst();

6）**peekLast()**:引用队尾元素，不做出队操作。返回队尾的元素但不删除。

str=deque.peekLast();

3.3、栈可以存取一组元素，但是存取元素必须遵循：先进后出原则。

栈可以通过双端队列Deque实现。通常栈用于完成“后退”这样的操作。

Deque<String> stack=new LinkedList<String>();

**对元素的操作方法：**

1）**push(E e)**;从队首方向入栈操作（压进添加）,将对象添加至队首

stack.push("one");

2）**pop()**;从队首方向出栈操作（弹出取出，栈中会删除）

String str=stack.pop();

## 三、查找表MAP

### 1、**java.util.Ma**p接口

1.1、Map称为查找表，该数据结构体现的样子是一个“多行两列”的表格。左列称为key,

右列称为value。Map总是根据key查找对应的value。存储元素也要求以key-value成对存入。

1.2、常用的实现类：java.util.**HashMap**散列表。HashMap是基于散列算法实现的Map，

当今世界上最快的查询结构。

1.3、Map定义的方法：（K是key的泛型类型，V是value值的泛型类型）

1）**V put(K k,V v)**;将给定的key-value对存入到Map中。

注意：

1.1）Map有一个要求，key不允许重复（key的equals比较决定）。若使用Map已有的key存入value，

则是替换value操作（位置不变）。put方法返回的就是被替换的value，若没有替换则返回null

1.2）对于value是包装类的情况，切记，不要直接用基本类型接收返回值，因为会触发自动拆箱特性：

Integer n=map.put("语文", 96);若写成int n=map.put("语文",96);

代码编译时会改为：int n=map.put("语文",99).intValue();

若key(String类型)在Map中不存在，则返回值为null，若拆箱会引发空指针异常

2）**int size()**;返回Map中实际存入的元素个数。int size=map.size();//几行

3）**V get(K k)**;根据给定的key获取对应的value，若给定的key不存在则返回值为null。

4）V remove(K k);删除当前Map中给定的key所对应的这组键值对。

返回值为该key所对应的value。若给定的key在Map中不存在，则返回值为null。

5）**boolean containsKey(K k)**;判断当前Map是否包含给定的key

**boolean containsValue(V v)**;判断当前Map是否包含给定的value

boolean ck=map.containsKey("化学");

### 2、HashMap

2.1、HashMap内部由数组保存键值对，存储元素是根据key的hashcode 值计算数组小标，

并将键值对存入，获取时也根据该值计算下标直接找到该元素。所以HashMap根据

这个方式避免了了查找元素时对数组的遍历操作,所以其不受元素的多少而影响查询性能.

2.2、由于Key的hashcode决定着键值对在HashMap中数组下标位置，而equals方法决定着

Key是否重复。所以这两个方法要妥善重写，当遇到两个Key的hashcode值一样，但是equals

比较不为true的情况时，会出现链表。链表会降低查询性能，应当尽量避免。

2.3、hashcode与equals方法是定义在Object中的，所有类都具有，Java提供的类都妥善的

重写了这两个方法。（可自动生成）

2.4、当我们定义的类需要重写equals或hashcode时必须遵循以下原则：

1）成对重写，当我们需要重写一个类的equals方法时就应当连同

重写hashcode方法。

2）一致性，当两个对象equals比较为true时，hashcode值应当相等，反之虽然不是必须的，

但是尽量保证当两个对象hashcode方法返回值相等时，equals方法比较为true。

否则会在HashM中出现链表。

3）稳定性，当参与equals比较的属性没有发生变化的情况下，多次调用hashcode方法返回的

数字不应当有变化。

### Map的遍历Entry接口（三种方式）

3.1、便利所有的key

**Set keySet()**;将当前Map中所有的key存入一个集合中后返回。

遍历这个Set集合等于遍历了所有的key

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

for(String key:keySet){

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

}

3.2、遍历所有的key-value对

在Map接口中定义了一个内部接口：java.util.Map.**Entry，**

Entry的每一个实例用于表示当前Map中的一组键值对，

其提供了两个常用方法：

1）**K getKey()**:获取key值

2）**V getValue()**:获取value值

不同的Map实现类都实现了Entry，并用实现类的每个实例表示一个具体的键值对。

**Set<Map.Entry<K,V>> entrySet()**;该方法会将Map中所有键值对存入一个集合后返回。

Map<String,Integer> map=new HashMap<String,Integer>();

......

Set<Entry<String,Integer>> entrySet=map.entrySet();

for(Entry<String,Integer> e:entrySet){

String key=e.getKey();

Integer value=e.getValue();

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

}

3.3、遍历所有的value（相对不常用）

**Collection values()**;将当前Map中所有的value以一个集合的形式返回。

Collection<Integer> values=map.values();

for(Integer value:values){

System.out.println(value);

}

### 4、有序的Map

LinkedHashMap维护者一个双向循环链表。此链表定义了迭代顺序，

该顺序通常就是存放元素的顺序。

Map<String,Integer> map=new LinkedHashMap<String,Integer>();

## 反射

反射是java提供的一套API

反射允许程序运行过程中取得任何一个已知名称的类的内部信息，

提供了检查一个对象内部结构的手段

利用反射API可以实现“动态”执行：

1、动态加载类，

2、动态创建对象

3、动态访问属性

4、动态调用方法

动态执行：只是在JVM运行期间才确定执行次序

静态执行：是指在编译以后就确定了程序的运行次序，JVM运行期间按照既定的次序执行。

反射检查对象内部结构

调用方法

test(“ABC”);

test(“ABC”);

test(“ABC”);

反射API可以动态检查对象的类型，类型的内部结构

检查对象类型的方法：Class cls=obj.getClass();返回值cls代表对象的类型

2、动态加载类

API方法 Class cls=Class.forName(类名);

3、动态创建对象

API方法

Instance 实例（对象）

创建cls代表的类型的实例

cls类型上必须包含无参构造器（可以是默认构造器）

newInstance就是调用这个无参构造器创建对象。

如果没有无参构造器则抛出异常。

Object obj=cls.newInstance();

可以利用Constructor调用有参构造器创建对象，用未使用少，这里省略。

4、动态访问属性

核心功能是动态执行：运行期间才决定加载那个类创建的哪个对象。

API方法

在Field类型上定义了get方法，可以用于获取对象的属性值。

如果需要动态获取属性的值：（若属性是私有的，不能获取异包类的属性值）

1. 获取Field类型对象

利用Class类型对象的方法，才能获取Field类型对象

——obj.getClass();

——Class.forName()

2）获得对象（操作的，要获得属性的对象）

--new一个对象

--newInstance创建对象

3）调用get方法

Object val=fld.get();

反射的用途：

解耦

利用反射可以实现一段程序与未来一个类之间偶合在一起，

这段程序就与未来的类之间是松耦合关系--也就是解耦了。

如：Eclipse可以开发任何未来的程序，解析任何未来程序的结构。

经典面试题目：Eclipse的快捷菜单用到了哪些技术：

答：Eclipse利用反射技术实现快捷菜单，

可以使Eclipse与被开发的类解耦。

1. 利用反射访问对象的方法

访问方法的核心API

invoke 调用

obj是包含当前method方法的对象

method.invoke(obj,参数1，参数2，...)

使用步骤：

--获得Class对象

Class.forName(类名);

--获得Class对象，才能利用其方法找到Method对象

Method getDeclaredMethod(方法名);根据方法名找到Method对象

--获得方法对象method

--得到包含方法的对象obj

Cls.newInstance();

--准备参数（有参时选项）

Object val=method.invoke(obj);

####getDeclaredMethod与getMethod

cls.getDeclaredMethod:在当前cls类型上查找当前类中声明的全部方法

--只查询当前类

cls.getMethod在当前cls类型以及全部继承的方法中查找声明的公有方法

--查询当前类和父类中继承的方法。

#反射

1. java的动态执行 API

--动态加载类

--动态创建对象

--动态访问属性

--动态执行方法

1. 可以用于实现软件组件的“解耦”

--可以实现现在组件和未来的组件耦合关系，调用未来的程序组件

例子：

Eclipde的快捷菜单使用反射技术

WebServer与未来的Servlet组件的耦合也是使用反射技术

重点掌握方法

1. Class.forName(类名)
2. Method.invoke(obj)
3. Class.newInstance()
4. setAccessible(true)

## XML语法、XML解析

### 1、XML语法

1.1、XML用途

XML是可扩展标记语言（EXtensible Markup Language）,是独立于软件和硬件的信息传输工具。

1. 简化数据存储
2. 简化数据传输

1.2、基本语法

1）元素（Element）和属性（Attribute）

1.1）XML元素指从开始标签直到结束标签的部分。元素可以包含其他元素（嵌套）、文本或两者混合。

1.2）元素可以有属性（开始标签中可包含Attribute，在尖括号内）。属性值必须被引号包围。

单引号双引号均可使用，若属性值本身有双引号，在标签中要用单引号包围他，或使用实体引用。

1.3）XML对大小写敏感，起始和结束标签相同。<letter>...</letter>

1.4)每个元素必须由起始标签和关闭标签组成。关闭标签多一个“/”,<letter></letter>

自关闭标签等同于无内容空元素,可以有属性<letter .../>

1.5)XML要求必须有根元素。根元素是不被其他元素包围。根元素只能有一个。

1.6）实体引用



### 2、XML解析

2.1、**XML解析方式**

1. **SAX**：(simple API for XML)是一种XML解析的替代方法。逐行扫描文档，一边扫描一边解析。

可以在解析文档的任意时刻停止解析。（对节点逐个解析）

优点：解析可以立即开始，速度快，没有内存压力，更有效。

缺点：不能对节点做修改。

2）**DOM**：(Document Object Model,即文档对象模型)

DOM解析器在解析XML文档时，会把文档中所有元素，按照其出现的层次关系，

解析成一个个Node对象（节点）。（对文档整体扫描整体解析）

优点：把xml文件在内存中构造树形结构，可以遍历和修改节点。

缺点：如果文件比较大，内存有压力，解析时间较长。

2.2、**Element（元素）**

Element的每一个实例用于表示xml文档中的一个元素（一对标签）。

其提供了很多获取其表示的元素相关信息的方法：

1）**String getName()**;获取当前元素的名字（标签的名字）

2）**String getText()**;获取当前元素中间的文本（开始和结束标签中间的文本信息）

3）**String elementText(String name)**;获取当前元素中指定名字的子元素中间的文本信息

例如：Strint str=e.**elementText**("name");等同于String str=e.element("name").getText();

4）**Element element(String name)**;获取当前元素中指定名字的子元素。

5）**List elements()**;获取当前元素中所有子元素。

6）**List elements(String name)**;获取当前元素中所有的指定名字的同名子元素。

2.3、**Attribute（属性）**

1. **String getName()**;获取当前属性的名字

2）**String getValue()**;获取当前属性的值

3）**Attribute attribute(String name)**;获取当前元素中指定名字的属性

4）**String attributeValue(String name);** 获取当前元素中指定名字的属性**值**

Attribute att=empEle.attribute("id");

int id=Integer.parseInt(att.getValue());

可以合为一句：

int id=Integer.parseInt(empEle.**attributeValue**("id"));

2.4、**读取XML（用SAXReader）**

SAXReader读取XML文档，需要导入dom4j-full.jar包。

注：若想使用dom4j首先先必须下载dom4j.jar包，并导入到项目环境中(maven要配置好)

使用DOM4J解析XML的大致步骤：

1）创建SAXReader

**SAXReader reader=new SAXReader();**

2）使用SAXReader读取xml文档并得到对应的Document对象

Document doc=reader.read(new File("emplist.xml"));

Document doc=reader.read(new FileInputStream("emplist.xml"));

3）通过Document获取根元素。

Element root=doc.**getRootElement();**

4）从根元素中按照文档结构逐级获取子元素，以达到遍历XML文档数据的目的

例如：

//获取根标签下所有<emp>标签

List<Element> empList=root.**elements**("emp");

//将每个<emp>标签表示的员工信息解析出来

以Emp实例形式存入list集合。

for(Element empEle:empList){

//获取员工名字

//1、获取<name>标签

Element nameEle=empEle.**element**("name");

//获取中将的文本数据

String name=nameEle.**getText**();

int age=Integer.parseInt(empEle.**element**("age").**getText**());

String gender=empEle.**elementText**("gender");

int salary=Integer.parseInt(empEle.**elementText**("salary"));

/获取属性id

int id=Integer.parseInt(empEle.**attributeValue**("id"));

//元素放入集合Emp中

Emp emp=new Emp(id,name,age,gender,salary);

list.add(emp);

}

for(Emp e:list){

System.out.println(e);

}

2.5、写XML(生成XML文件)

1）写出XML的大致步骤：写出后成一行，按Ctrl+Shift+F

1.1）创建一个Document对象表示一个空白文档

**DocumentHelper.createDocument();**自动创建Document对象

1.2）向Document中添加根元素

**Element addElement(String name);**Document提供的添加元素的方法。

doc.addElement(String name)向当前文档中添加给定名字的根元素，并以Element实例的形式返回，

以便于继续向根元素追加信息。需要注意，该方法只能调用一次，因为一个文档只能有一个根元素。

1.3）从根元素开始按照要生成的文档结构逐级添加子元素信息

元素.addElement(String name);向元素添加子元素。

1.4）创建XmlWriter

1.5）将Document通过XmlWriter写出以生成XML文档。

List<Emp> list=new ArrayList<Emp>();

list.add(new Emp(1,"张三",22,"男",4000));

try {

Document doc=DocumentHelper.createDocument();

Element root=doc.addElement("list");

/\*

\* 将每个员工信息以<emp>标签形式存入当根元素中

\*

\* Element提供了追加相关信息的方法：

\* Element addElement(String name)

\* 向当前元素中追加子元素并将其返回，以便继续对其追加。

\*

\* Element addText(String text);

\* 向当前元素的开始与结束标签之间添加文本，返回的仍然是当前元素，

\* 以便于对当前元素继续做其他追加操作。

\*

\* Element addAttribute(String name,String value);

\* 向当前元素中追加指定名字与对应值的属性，返回值仍然是当前元素。

\*/

for(Emp e:list){

//向根元素中追加子元素<emp>

Element empEle=root.addElement("emp");

//向<emp>中追加子元素<name>

Element nameEle=empEle.addElement("name");

//向<name>标签中添加文本

nameEle.addText(e.getName());

Element ageEle=empEle.addElement("age");

//int型转为String型+"";

//String.valueOf();

ageEle.addText(e.getAge()+"");

empEle.addElement("gender").addText(e.getGender());

empEle.addElement("salary").addText(Integer.valueOf(e.getSalary()).toString());

//向<emp>中追加id属性

empEle.addAttribute("id", e.getId()+"");

}

//org.dom4j.io.XMLWriter

XMLWriter writer=new XMLWriter(

new FileOutputStream("myemp.xml"),

//写出后成一行，按Ctrl+Shift+F

OutputFormat.createPrettyPrint()

);

writer.write(doc);

System.out.println("写出完毕");

//应在finally中关闭

writer.close();

} catch (Exception e) {

e.printStackTrace();

}

### 日期操作

### 3、线程池

# 第五章、WEBBASIC

李然，g-liran@tedu.cn

浏览器内核

1. webkit（safari内核（苹果），Chrome内核原型），开源的
2. Mozilla（FireFox）
3. Opera欧鹏

4、IE window10开始IE不再跟新，微软浏览器edge

5、web三要素：浏览器，服务器，HTTP协议。

## 一、HTML

### 1、HTML概述

1.1、HTTP协议是超文本传输协议

1.2、html是超文本标记语言，一种纯文本类型语言。

用来设计网页——勾勒出网页的结构和内容--骨架

1.3、html网页文档保存在服务器中，由浏览器（解释执行，即运行）下载html文档后，解析并且显示

1.4、html就是标签相对固定的xml

1）标签名固定

2）标签上可以使用的属性固定

3）标签与其它标签的嵌套固定

1.5、HTML工作原理：浏览器根据HTTP协议发出请求给服务器，服务器做出响应给浏览器返回一个HTML，

浏览器解释执行HTML，从而显示出内容。

总结：我们学习html，就是学习标签，学习属性，学习标签的嵌套关系。

### 2、HTML最基本结构

2.1、基本结构

<!DOCTYPEhtml>

<html><!--根元素-->

<head>

<metacharset=*"UTF-8"*>

<title></title>

</head>

<body>

<!--在页面中显示的元素-->

</body>

</html>

2.2、DOCTYPE

1）文档类型的声明，告诉浏览器当前文档是用的html标准是什么版本，按照哪种版本的规则去解析显示

2）<!DOCTYPE html>是按照**html5**的规则去解析。

3）如果不写DOCTYPE，不同浏览器解析标准不同

4）写了就没有差异了吗？不是的，差异会很小。

课堂操作：webServer/webapps/webbasic

2.3、head,描述当前页面的基本信息。

2.4、title，定义当前页面的标题。建议设置SEO：搜索引擎优化meta 设置浏览器端解码规则。

### 3、HTML常见标签

3.1、**文本标签**，跟论文格式很相似

1）标题 h1 h2 h3 h4 h5 h6 :用于声明文章中的标题

2）段落 p(paragraph) 前后有回车  
 3）属性 align=”center”：居中显示。

注意：所有的hn标签都可以使用align=“center”属性，但是现在的开发手段，

一般是使用css进行格式化（居中，字体颜色，字号大小等）

4）列表，显得有条理

4.1)无序列表 u--unordered无序，l——list——>ul；l——list，i——item——>li

<ul>

<li></li>

<li></li>

<ul>

4.2）有序列表o——ordered有序 l——list

<ol>

<li></li>

</ol>

5）分区：<div></div>

5.1)div默认是看不到的，它的存在是为了逻辑上作分区。

5.2）header footer nav的作用和div一模一样，只是可读性提高了，

是H5新标签，可以和div标签嵌套使用。

5.3）它们都属于**块级分区**，块级分区要独占一行，默认不允许其他元素在同一行。

5.4）行内分区,不会换行，会和其他文字混合显示在同一行上，如果只是一行文字内部

的区域进行属性的设置，请使用行内分区。<span></span>

6)元素的显示方式

6.1)块级元素，自己独占一行的元素

Hn p ul ol li header footer nav div

6.2）行内元素，可以和其他行内元素处于同一行

2.1）

span 行内分区

i/em 定义斜体

B/strong 定义粗体

U 下划线

del 删除线

2.2）行内元素的空格折叠现象：

写代码的时候，不管连写多少个空格、回车，都会显示一个空格。

2.3）实体引用

&nbsp;代表一个空格

<br> 代表一个回车

1.7、图片标签 <img>

1)网页中图片路径的写法

1.1）绝对路径：在tmooc，找到一个img标签，按F12，选中这个标签，

在控制台右键点击标签路径选中Open in tab在一个新选项卡中的地址栏

然后，放入img的src中，就可以显示其他网站的图片了。

这种行为称作图片的盗链，得到的是绝对路径。

总结：一般情况下，在一个页面中显示其他网站的图片，

经常会使用绝对路径，链接显示图片

行业中，称专门提供杨业图片的网站为**图床**

1.2）相对路径<img src="../img/06.png">

在一个网站内，自己的图片资源一般使用相对路径

2）总结：

2.1）使用其他网站的图片时，一般使用绝对路径

2.2）使用自己网站的图片资源，一般使用相对路径。

3）img标签的常用属性

3.1）

<img

src=”” 图片的路径

alt=”” 图片显示失败时，显示的文本

title=””图片的描述文本

width=””图片宽度

height=“”图片高度

>

3.2）为了防止图片失真，一般宽和高只写一个，让另一个属性取自适应。

3.3）width和height，一般使用css去编写，不适用img的属性了。

3.4）img标签支持的图片类型：

jpg 不支持透明色

png 支持透明色

gif 动态图 只支持256色，占用空间最小。

二）超链接 用于跳转页面

<a></a>

总结：1、a标签是行内元素

2、添加href属性，a标签才有效果

3、href中的url也分相对路径和绝对路径

访问本网站的其他页面使用相对路径

访问其他网站，使用绝对路径。

a标签的其他属性：

Target="\_blank":新页面，开启一个新窗口显示

Target="\_self":新页面，在原窗口显示（不设置则默认为self）

###超链接锚点的用法

页面之间的跳转，是超链接

一个页面内部的位置跳转是锚点。

H5的锚点设置方式：

<a href="#charAt">charAt 方法</a>

<h3 id="charAt">charAt 方法</h3>

H4的锚点设置方式：

<li><a href="#length">length 方法</a></li>

<a name="length"></a>

##网页中的表格标签

<table>

<!--行 table row-->

<tr>

<td></td><!--列 table data-->

<td></td>

</tr>

</table>

###表格的相关属性

border="2px"

align="center"

width="200px"

height="100px"

这些属性一般写在样式中，目前只在学习和测试过程中使用

###重要属性

1、单元格边框与内容的距离

cellpadding=""

2、单元格之间的距离

cellspacing=""

###列合并和行合并

colspan=”占据的列数”

rowspan=“占据的行数”

###表分组，跟分区类似

<thead></thead>

<tbody></tbody>

<tfoot></tfoot>

###总结：逻辑分组，没有显示变化

一般用于数据重构

th和td一样，但是th默认字体加粗居中。

## 表单标签

### 表单标签是用于在页面中显示一个可以收集用户填写的数据的控件.

### 学习表单,就是学习表单控件的结构和多种表单控件

![](4.png)

<form action="url" method="get">

<h3>登录学子商城</h3>

<div>

<label>请输入用户名:</label>

<input type="text" name="username">

</div>

<div>

<label>请输入密码:</label>

<input type="password" name="pwd">

</div>

<div>

<input id="auto" type="checkbox" name="auto">

<label for="auto">自动登录</label>

</div>

<div>

<input type="submit" value="登录">

</div>

</form>

## form表单主要属性

### action 表单提交url

### method 表单提交数据的方式get/post

### 所有的表单控件,都要在`<form>`标签的内部

### form表单,可以使用label

### 文本输入框/密码框 的主要属性

### value 用户自用输入的任何文本

### maxlength 限制输入的字符数

### readonly 设置只读,不能修改,但是数据可以提交

### placeholder 文本提示

### 单选框和多选框

### 单选框 radio

<input type="radio" name="gender" id="m">

<label for="m">男</label>

<input type="radio" name="gender" id="f">

<label for="f">女</label>

<input type="radio" name="gender"

id="unknow" checked="checked">

<label for="unknow">保密</label>

### 总结:name 属性,分组(同时保证可以提交)

### 属性checked,默认选中

### 多选框 checkbox

<h3>演示多选框</h3>

<form action="#">

<input id="study" type="checkbox" name="hobby" value="study">

<label for="study">学习</label>

<input id="sleep" type="checkbox" name="hobby" value="sleep">

<label for="sleep">睡觉</label>

<input id="teacherC" type="checkbox" name="hobby" value="teacherC">

<label for="teacherC">刘苍松老师</label>

<input id="swimming" type="checkbox" name="hobby" value="swimming">

<label for="swimming">游泳</label>

<input type="submit" value="提交">

</form>

### 按钮

submit 表单提交

button 普通按钮,调用javaScript代码

reset 重置按钮,清空表单内容

### 隐藏域

### 不希望用户看到的信息,同时又需要提交的时候上传

<input type="hidden" name="cartid" value="1234">

### 文件选择框 file

### 上传文件用的,比如上传头像

<input type="file" name="img" multiple="multiple">

### multiple="multiple" 多重的,可以同时选择多个文件

### 非input标签

1. 文本域,需要用户填写大量文本时使用

<form action="#">

<label>请输入送货地址</label>

<textarea></textarea>

</form>

属性:rows和cols,原本的意思是每一行占多少个字符,能显示多少行字符.

但是由于系统编码不同,浏览器解析不同,这个显示字符数差异很大.

readonly,只读

### 下拉选

<select name="tName">

<option>--请选择--</option>

<option>王克晶</option>

<option>范传奇</option>

<option>李然</option>

<option>刘苍松</option>

</select>

### 总结,如果下拉选想要提交数据,需要name属性

1. 如果option没有设置value属性,那么select的value就是选中的那个option的内容

2. 如果option设置了value属性,那么select的value就是选中的那个option的value

## 关于form表单控件的总结

1. 只有含有name属性的表单控件才可以提交数据,提交的是value值

2. 单选按钮和多选按钮,需要设置value值,不然就算选中,提交的数据也是"on"

3. 关于表单控件的默认值设置

3.1 文本框和密码框,设置value属性就可以显示默认文本.(提示文本placeholder)

3.2 单选框使用checked属性设置默认选中

3.3 下拉选设置selected属性,默认选中

## 二、CSS

--样式表--美化网页--皮肤--化妆--美颜--整容

css选择器

选择器的作用，就是为了选出当前页面中符合要求的一个或者多个标签。

1、标签选择器，一般用于定义默认样式，等待被其他样式覆盖。

h2{color:#f00;}

2、id选择器-由于一个html中，id都是唯一的，所以id选择器的作用，

是选取当前页面中符合要求的**一个**标签

<h3 id=”h”>选择器</h3>

#h{color:#f00;}

3、类选择器（class选择器）

<h2 class=”w300>类选择器</h2>

**.**w300{width:300px;}

4、派生选择器-可以通过父标签找到符合要求的子孙标签。

5、子类选择器-根据父标签，找到符合要求的子标签。ul>a{color:#990066;}

6、伪类选择器：hover-在一系列动作中添加某种样式。

:hover--鼠标悬停和鼠标离开。

总结：选择器（派生，子类）写的越具体，优先级越高。

css常用样式属性

1. 布局相关属性

width:单位：px 100%

height:

margin:设置外边距

margin：10px；同时控制四个边距

margin：10px 20px；上下为10px，左右为20px。

margin：0px auto；块级标记的水平居中写法

margin：10px 20px 30px 40px；上右下左。

margin-top:上外边距

margin-right:右外边距

margin-bottom:下外边距

margin-left:左外边距

padding:设置内边距

padding：10px；4个方向都是10px

Padding：10px 20px;上下10px，左右20px

Padding：10px 20px 30px 40px 上右下左

padding-top:上内边距

padding-right:右内边距

padding-bottom:下内边距

padding-left:左内边距

总结1：一个div如果不设置宽度，默认占父容器宽度的100%。

总结2：一个div如果不设置高度，默认高度为0px。如果有内容，那么内容多高，

div就撑起多高。

总结3、盒子模型，元素之间的留白，元素背景的填充范围，元素的大小，这些东西的

哦控制规定放在一起，就是盒子模型。

盒子模型一般用于计算整体布局的宽度。

一个元素占用页面的宽度计算公式为：左外边距+左边框+左内边距+内容区域宽度+右内边距+右边框+右外边距

2、背景相关属性(background)

background-color:背景颜色

background-image：背景图片

background-repeat：背景图片的平铺方式

background-position：背景图片的位置

background--size：背景图片的大小

2.1、background-color:背景颜色的颜色值可以如下设置

1）纯英文单词，类似于red，green等

2）#ffffff，6位16进制字符串，#006699

3）#f00，相当于# ff 00 00

4) rgb(255,255,255);十进制

5）rgba（255,255,255,0-1）;1不透明，0全透明

2.2、background-image:url(“...”);背景图片

div设置了背景图片，那么div内部的元素将会在背景图片之上显示（背景图片类似于桌布）

2.3、background-repeat：背景图片的平铺方式

1）background-repeat：no-repeat;不平铺

2）background-repeat：repeat;平铺

3）background-repeat：repeat-x;横向平铺

4）background-repeat：repeat-y;纵向平铺

2.4、background-position：背景图片的位置

1）center|top|bottom|left|right|具体数值

background-position：x轴 y轴;

background-position：right bottom;

background-position：270px 20px;

background-position：center center;

2.5、background--size：宽，高，设置背景图片的大小

background--size：330px 350px

3、字体相关属性

font-size:字体大小，单位：px cm em

em:是一个相对度量单位，若基础字号16px，font-size:1em;表示16px。

用于屏幕自适应。

font-family:字体 “微软雅黑” “黑体”

font-weight:字体权重，即字体的粗细（磅值pt）默认100pt

lighter 细一点

normal 正常粗细

bold 粗一点

bolder 比粗更粗

font-style:字体样式:normal,italic 斜体

4、文本属性

color:文本颜色

text-align：文本的对齐方式：left|right|center

区别margin:0px auto;块级水平居中

text-decoration:文本装饰(划线)

text-decoration:none;给a标签去除下划线

text-decoration:underline;

text-decoration:line-through;

text-decoration:overline;

line-height:文本行高。由于div的文本在默认情况下根据行高垂直居中，所以如果希望文本在div

垂直居中显示，只需要把行高属性设置为div的高度。

但是此处存在bug。如果字数过多需要折行，那么行高的效果会每一行都遵循。

1. 边框属性border

5.1、border:1px solid black;

border-width:边框的宽度

border-color；边框的颜色

border-style：边框的样式 solid(实线)

5.2、单独设置某一边；

border-top：2px solidblack;上边框

border-top-width：上边框的宽度

border-top-color：

border-top-style：

border-left：左边框

border-right：有边框

border-bottom:下边框

border：none;去除边框（特殊）

border-top:20px solid red;

border-left:3px dotted blue;/\* 点 \*/

border-right:30px double green;/\* 双实线 \*/

border-bottom:10px dashed yellow;/\* 虚线 \*/

5.3、设置边框的圆角

border-radius:3px;圆角，半径

div是正方形的，网页中显示圆形，只需要圆角半径设置成div宽或高的一半就可可以了。

总结：1、确定宽高（div不写宽默认是父级的宽）

2、考虑背景问题（颜色和图片）

3、内部字体问题（字号，颜色，字体，对齐）

4、位置的微调（margin，padding，浮动，定位）

注意：margin会移动元素的位置，padding会改变元素占据页面的大小

6、常见复杂属性

溢出属性overflow

visible 溢出部分可见

hidden 溢出部分隐藏

scroll 溢出部分显示滚动条

6.1、display控制元素的显示方式

display：block;按照块级元素进行显示

display：inline;按照行内元素进行显示，一般不应

display：none;隐藏元素

总结：行内元素：一行中可以有多个，从左往右依次摆放

行内元素，宽高，上下边距失效

块级元素：独占一行，只有块级元素可以设置宽高和上下外边距。

display：inline

display：inline

display：inline-block;行内块：与其他行内元素（行内块）共占一行，可以改变宽高、上下外边距。

行内块非常容易出bug，使用要小心。

7、浮动float

float：left;元素浮动起来往左放

float：right;元素浮动起来往右放

一旦元素设置了浮动属性，当前元素将会脱离默认文档流，在默认文档流之上进行渲染绘制。

总结：元素浮动后，先前的位置会被其他元素占用。

1、浮动时候，如果本标签，上面的标签还处于文档流中，本标签的浮动，只在当前行浮动。

2、如果本标签，上面的标签也浮动了，本标签会在上面标签那一行浮动。

3、clear：both;清除浮动，本标签不被上面其他浮动的标签覆盖。

4、浮动最常见的用法，是把块级元素横向排列。

（扩展）高度坍塌：外区div未设置高度，内区div设置浮动，因溢出外区导致外区高度消失，称为高度坍塌。

解决方法：在外区最下方添加一个空区块div，设置清除浮动。

原理：清除浮动后，浮动的部分不能覆盖，那么外区高会撑大包围所有内区。

8、定位position

position:absolute;绝对定位。

position:relative;相对定位。

总结：position会让标签脱离文档流，称为一个单独的层。

1、一个标签如果设置了position属性，就会解封另外4个属性：top,left,right,bottom

离上左右下边的距离。

2、absolute,定位之后，原位置会被其他元素占用。

3、relative，定位之后，原位置不会被其他元素占用。

4、如果父级标签没有写position，子标签absolute，那么子标签相对浏览器左上角原点定位。

子标签是relative，那么子标签相对父级标签左上角原点定位。

5、子标签是relative，不管父级是否有position属性，都是相对自己原来的位置。

6、父级标签写position，子标签absolute，相对于父级标签左上角定位，同时支持margin

7、子标签absolute，多重父级标签中以最近的那层写了position属性的

父级标签左上角为原点定位（从内往外找）。

总结写代码的套路：

1、如果标签原始位置不能被占用，使用relative，相对于自己原来位置去定位。

2、如果标签原始位置需要被占用，使用absolute，但是需要在父级标签中添加position：relative;

告诉子元素以该父级定位。

关于层数：后写定位的层数高，z-index可设置层数，数字越大层数越高。层数高的先显示。

position：fixed;相对窗口定位。

## 三、javaScript

--让网页呈现出动态的数据和效果

网页所有的交互都要使用javaScript

**javaScript的发展**：网景找到ECMA（欧洲计算机联盟协会）统一了javaScript标准

### 1、javaScript的特点：

1、可以使用任何文本编译工具编写

2、由浏览器内置的javaScript引擎执行代码

3、解释执行：事先不编译，逐行执行。

4、基于对象：内置了大量写好的对象。

**学习javaScript：**

1、学习如何找到标签对象

2、学习对标前对象的属性内容，进行增删改查。

### 2、javaScript的使用：

1、事件定义式，在事件定义时直接写js代码。

<!-- 事件：用户在做出某种操作时，会调用js代码 -->

<!-- 事件就是js调用时机，比如单击按钮等 -->

2、嵌入式，在`<script>`标签内，写js代码

3、文件调用式，

总结：

文件调用式，在单独的.js文件中写js代码，需要引用到网页才能使用。

引用时，script标签必须是双标签，哪怕没有内容

这个script作用就是导入外部的js文件，不允许同时引用文件又写js代码。

1. js注释：<script>标签中，是js底盘，要使用js的注释。/\*...\*/ //

js不区分单引号双引号。

### 3、js数据类型

<scripttype=*"text/javascript"*>

//声明变量：变量类型 变量名；

**var** x;

//给变量赋值：变量名=...

/\* x=9;

x="abc"; \*/

console.log(x);

</script>

总结：在<script>标签中写js代码一般有两种形式

1）封装成函数，该函数在页面加载完成之后，用户激活事件时被调用。

2）直接在标签中写js代码，这段js代码，这段js代码是在页面家在过程中

被直接调用，其调用时机甚至比body渲染还早。

#### 1.1、number类型

1）不区分整型和浮点型

2）所有的数字，都采用64位浮点格式存储。

3）js中没有double概念。

#### 1.2、String类型

1）由Unicode字符、数字、符号组成的字符序列

2）由首位一对单引号或者双引号括起。

3）单引号和双引号嵌套的时候，无需转义，其他特殊字符需要转义，\n

3）js中没有char的概念，字符就是一个长度的String。

#### 1.3、boolean类型

true和false，做数学运算的时候，自动转型为数值参与计算。

**var** s="hello";

**var** n=9;

**var** b=**true**;

console.log(s+n);//hello9

console.log(s+b);//hellotrue

console.log(n+b);//10

console.log(b+b);//2

在参与纯数学运算时，true---1 ，falst---0;

Js引擎对boolean类型的解释规则

1）非空字符串解释为true，“”字符串解释为false

2）非0数字解释为true，0解释为false。

4、强制类型转换

### 4、运算符

#### 4.1数学运算符

总结：**+ - \* / % ++ --**

1、纯数字的字符串”100”，除了+会变成字符串连接，其他的运算都会自动转成数字

2、js中的除法，如果除不尽会得到浮点数。100/3；

#### 4.2关系运算符

**><>= <= != == ===(全等：结构相同，类型相同)**

<scripttype=*"text/javascript"*>

**function** doClick(){

**var** s1="123";

**var** s2=123;

**var** s3=123;

console.log(s1==s2);//t

console.log(s1==s3);//t

//全等：

//数据类型相同

//数据值相同

console.log(s1===s2);//f

console.log(s2===s3);//t

}

doClick();

</script>

#### 4.3逻辑运算符

**！ && ||**；要注意短路问题。

#### 4.4、条件表达式

js可以使用任何数据做条件

当使用非boolean值做条件时，true--->非空字符串，非0数据；false--->nul,undefied,"",0,NaN

### 5、控制流程

**循环**

### 6、javaScript常用对象API

#### 6.1、内置对象

6.1.1、string对象

1）创建string对象的2中方式

**var str1=”hello world”;**

**Var str2=new String(“hello world”);**

2)大小写转换方法

**str.toLowerCase();**

**str.toUpperCase();**

3）获取指定的字符（串）

**str.charAt(index);**

4）查询指定的字符串（的下标）

**str.indexOf(“”);**

**str.lastIndexOf(“”);**

5）获取子字符串

**str.substring(star,[end]);**注意全小写,若有end,子字符串含头不含尾。

6）替换子字符串

**str.replace(str1,str2);**

str1----要找到的字符串

ste2----新的字符串

返回值是替换后的字符串

7）拆分子字符串

**str.split(str1,[length]);**

Str1----分割用的子字符串；length---指定返回数组的最大长度，可省略。

返回值为，分割之后的字符串数组。

6.1.2、number对象

**toFixed(length);**把number转换为字符串,保留小数点后一定的位数。

如果必要，该数字会被四舍五入，也可以用0补足位数。

6.1.3、数组

js中的数组特点：

1）Array在js中都是obj的数组

2）js中数组可以同时存放不同数据类型的值。

3）js中的数组长度可变

4）两者创建方式，不管哪一种创建出来都是object类型。

var arr=["新闻","炮兵",12.3,true];

var arr1=new Array();

arr1[0]="123";

arr1.push(23);像数组中添加元素。

5）数组常用API

5.1）arr.reverse();数组反转。

5.2）arr.sort();排序。是按照数组中每一个元素的“unicode”编码进行排序的，

比完第一位比第二位。string就是这样排序，但是这种排序不适用于number。

6.1.4、Math对象

Math.round(4.56);四舍五入

Math.floor(4.5);向下舍入。

6.1.5、Date日起对象

1)var d1=new Date();创建客户机当前时间

2)var d2=new Date("2016/12/12 12:12:12");创建指定时间的date对象（指定时间一般源于服务器）

3）getTime();setTime();读写事件的毫秒数

4）date的API，从服务器那到时间对象，在页面根据不同的需求显示。

d1.getFullYear();获取年

d1.getMonth();获取月，需要+1

d1.getDate();获取月的天

d1.getDay();获取周的天。

d1.getHours();获取小时

d1.getMinutes();获取分钟

d1.getDeconds();

5）date转化成本地时间格式

d1.toString();

d1.toLocaleTimeString();转换成本地时间格式

D1.toLocaleDateString();转换成本地年月日格式

6.1.6、正则

. 任意字符

\w 任意字母，数字，下划线

\s 任意空白字符

\d 任意数字

^ 字符串开头

$ 字符串结束

1）如何创建正则对象

---直接创建：var reg=/正则表达式/[模式];

var reg=/no/;匹配字符串里有no,正则式不需转义var reg=/\s\d/;(匹配空白字符)

---创建对象：var reg=new RegExp(正则表达式,[模式]);正则式需转义（是字符串）

var reg=new RegExp(‘\\s\\d’);(匹配空白字符)

全局模式，设定当前匹配为全局，g

设置当前匹配模式忽略大小写，i

2)正则对象的方法reg.xxx(str);

//reg.exec(str);从str中找到匹配正则的子串

//普通模式，返回第一个符合要求的子串

//全局模式下，第n次执行，返回符合要求的第n个子串

/\* console.log(reg.exec(str));

console.log(reg.exec(str));

console.log(reg.exec(str));

console.log(reg.exec(str)); \*/

//reg.test(str);判断str中是否包含reg匹配的子串。

console.log(reg.test(str));

3）字符串匹配正则的方法：str.xxx(reg);

3.1）str.replace(reg,"");将str中的与reg匹配的所有子串都替换成目标

3.2）str.match(reg);从str中找出和reg匹配的子串。返回值是数组

3.3）str.search(reg);从str中找出和reg匹配的第一个子串的索引。

4）表单验证

6.1.7、function对象

1）js中函数就是function对象

2）函数名就是指向function对象的引用

var fn1=function(){alert(111);}等同于function fn1(){alert(111);}

3）使用函数名可以访问函数对象。fn1;——>f(){alert(111);}

4）函数名后面跟上()，是调用函数。fn1();——>执行alert(111);

5）函数的返回值。

5.1）不定义返回值的类型

5.2）默认返回值是undefined

5.3）可以使用return返回具体的值

6）函数的参数

js函数没有重载。函数被调用时，只要函数名一样，无论传入多少个参数，

调用的都是同一个函数对象，所以js没有重载。

但是可以实现和重载一样的调用方式，使用arguments

function add (){

var sun=0;

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

sum+=arguments[i];

}

return sum;

}

js在调用函数的过程中，只检测函数名，不检测参数列表。

如果参数名称匹配，则直接调用。

可以使用arguments访问传递过来的参数列表。

js中没有重载，如果出现相同函数名的两个函数，后一个有效。

匿名函数：就是不给函数起名字。

如果一个函数再别的地方不再被调用了，就可以使用匿名函数。

全局函数：可用于所有的javaScript对象，不需要对象调用。

常见的全局函数：typeof();isNaN();parseInt();parseFloat();eval();...

eval();用于 计算表达式 字符串。var str=”2+3”;eval(str);

用于执行字符串中的js代码.var str+”alert(111);”;eval(str);运行alert（111）。

eval最重要的作用，是动态执行服务器传过来的js代码。

#### 6.2、外部对象

6.2.1、BOM与DOM

BOM:Browser Object Model,浏览器对象模型

DOM:Document Object Model,文档对象模型

总结：用过BOM，可以移动窗口，更改状态栏文本。

执行其他不与页面内容发生直接联系的操作（不操作标签）

BOM是没有标准，却被浏览器厂商广泛支持。

总结：DOM定义了访问和操作HTML的标准方法

通过对DOM树的操作，来实现对HTML文档数据的操作。

6.2.2、js相关的BOM操作

1. Window表示整个浏览器窗口

所有js的全局对象，全局函数以及全局变量，都自动成为window对象

的成员（window可以点出来）

2）window的常用操作。

document 窗口中显示的HTML文档对象

history 浏览器的历史记录对象

location 窗口文件地址对象

screen 屏幕对象

navigator 浏览器相关信息对象

**6.2.3、window对象**

1）普通弹出框

window.**alert**("你好");

2）弹出确认框

**var** flag=window.**confirm**("how are you!");

3）弹出输入框

**var** str=window.**prompt**("中午吃的什么？");

6.2.4、**定时器**：主要用于网页动态时钟，倒计时，轮播图，无缝滚动，跑马灯效果。

1）周期性定时器：以一定的时间间隔执行代码，循环往复。

setInterval(exp,time);

exp:要执行的js语句，一般为匿名函数。

time:时间周期，单位为毫秒

返回值：返回已经启动的定时器ID

clearInterval(ID);关闭定时器

总结：后启动定时器就相当于启动了一个子线程，当前方法fn4相当于主线程。

2个线程之间并发执行，相互不等待。

2）一次性定时器（延迟执行）：在一个设定好的时间间隔之后执行代码

setTimeout(exp,time);

exp:要执行的js语句，一般为匿名函数。

time:延迟，单位为毫秒

返回值：返回已经启动的定时器ID

clearTimeout(ID);关闭定时器

6.2.5、window常见属性：

1）**location对象**，包含当前页面的URL的信息。常用语获取和改变当前浏览器的网址。

1.1）属性href，location.href=url(绝对地址);

1.2）**reload()**;重新载入当前网页（刷新）

**function** fn1(){

/\* location.href="http://www.tmooc.cn/web/login\_new.html" \*/

//刷新

/\* location.reload(); \*/

**var** flag=confirm("你确定要离开吗");

**if**(flag){

location.href="http://www.tmooc.cn/web/login\_new.html";

}

2）**screen对象**，包含有关客户端显示屏幕的信息，常用于获取屏幕的分辨率和色彩。

1、width/height

2、availWidth/availHeight:可用宽/可用高

总结：可用高，除了window的任务栏之外的高度

3）**history对象**，包含用户在浏览器窗口中访问过的URL。

length属性，浏览器历史列表中的url数量

方法：

3.1）back()

3.2）forward();

3.3）go(num);//0是当前页，-1是上一页，1是下一页

4）**navigator对象**，包含浏览器的相关信息

userAgent属性，当前浏览器的版本。

5）**DOM对象**的操作

1、查找节点

**var** p1=document.getElementById("p1");

**var** p2=document.getElementsByTagName("p")[1];

console.log(p2);

//console.log(p1);

//console.log(p1.nodeName);

//console.log(p1.nodeType);

总结：nodeType返回值，对应了节点的类型：

1——元素节点;2——属性节点;3——文本节点

8——注释节点;9——文档节点

nodeName和nodeType，在写js框架时用的非常多，正常开发很少使用。

2、读取节点信息

双标签叫内容，单标签叫值(value)

所有的双标签都有内容，一般表单中的控件，数据称之为值。

1、innerHTML，认识标签，会解析标签。

2、innerText，不认识标签，会把标签当成字符串。

p1.innerHTML="1.<u>读写</u>节点";

p1.innerText="1.<u>读写</u>节点";

3、读写值 btn.value

4、读写节点的属性

通过方法读写属性

通过标准属性名读写属性

通过不标准属性名读写属性，只有高版本浏览器才支持。

总结：不标准的属性建议使用方法处理。

标准的属性中的style，室内联样式，除了学习和测试不建议使用。

鼠标悬停和鼠标离开事件

onmouseover

onmouseout

3、修改节点信息

1、查询节点

如果想要操作HTML元素，必须先要找到该元素

查询节点方式方法

1. 通过id查询
2. 通过层次（节点关系）查询
3. 通过标签名称查询
4. 通过name属性查询

4、创建新节点

document.createElement(TagName);

TagName:要创建的元素的标签名称。

返回值就是这个标签的对象。

把这个新标签对象挂到dom树上。

插入，追加新节点

**function** fn1(){

**var** oLi=document.createElement("li");

oLi.innerHTML="重庆";

**var** oUl=document.getElementsByTagName("ul")[0];

oUl.appendChild(oLi);

}

**function** fn2(){

**var** oLi=document.createElement("li");

oLi.innerHTML="苏州";

//插入需要父级对象和弟弟对象

**var** oUl=document.getElementsByTagName("ul")[0];

**var** gz=document.getElementById("gz");

//把新节点插入到父节点下级，弟弟之前。

oUl.insertBefore(oLi,gz);

}

5、删除节点

总结：需要父级去删除子元素

### 7、javaScript事件处理

事件：指页面元素状态的改变，用户在操作鼠标或键盘是触发的动作

7.1、鼠标事件

--onclick

--ondblclick

--onmouseover//鼠标悬停

--onmouseout//鼠标离开

--onmousedown//鼠标按下事件

--onmouseup//鼠标抬起事件

7.2、键盘事件

--onkeydown//键盘按下

--onkeyup//键盘弹起

7.3、状态改变事件

--onload

--onchange//值改变

--onfocus//焦点获取

--onblur//失去焦点(光标)

--onsubmit//

事件绑定

1、在标签属性中直接处理事件

<input type="button" value="按钮1" onclick="fn1();">

2、js代码中动态定义，可以吧html代码和js代码分离，好维护

3、取消事件 onXXX=”return false”

事件处理机制

--事件冒泡

由内向外依次触发事件称为事件冒泡。

发生事件冒泡的要求：1、嵌套关系。2、内层和外层标签必须是同一个事件

--取消时间的冒泡。由于浏览器厂商使用的内核不同有两种方法取消。

1、event.stopPropagation();

总结：以前只支持firefox chrome.现在高版本的ie已经支持了，ie8.0及其以下的版本不支持

2、event.cancelBubble=true;

总结：以前只支持ie，现在高版本的FF和chrome中已经支持了

7.4、event对象，事件对象

事件触发后或自动产生一个event

1、获取event对象

任何事件触发后都会产生一个event对象。

event对象记录事件发生的鼠标位置，键盘按键状态和出发对象（事件源）等信息。

1. 在html属性中直接使用event对象

支持ie ff chrome但是这种写法没有意义。

我们希望html代码和js代码分离，松耦合。

2）在js代码块中直接使用event对象。

IE和chrome浏览器，chrome对ie做了兼容。

3）在函数中使用event对象，必须以传惨的方式

这种方式，ff,ie,chrome都支持，所以开发要求这种写法。

2、通过event对象获取事件源

1）IE和chrome，chrome对IE做了兼容。

2）ff和chrome，高版本IE也支持，8.0版以下不支持

## 四、jQuery

--是js的框架，提高了javaScript的开发效率。

jquery对象本质上就是Dom数组

### 1、jQuery常用选择器

jQuery选择器可以准确选取到想找的元素。

选择器允许HTML元素组或单个元素进行操作。

//1、基本选择器

$("li");

$("#gz");

$(".sz");

$("ul li");

$("ul>#gz");

$("ul>.sz");

//2、层次选择器

$("#gz+");//选择他的弟弟；$("#gz+li");

$("#gz~");//选择他的（所有）弟弟们

//3、过滤选择器

1）基本过滤器 $("li:first");//第一个li; $("li:lt(2)");//下标小于2的

$("li:odd");

$("li:last");

$("li:eq(0)");

$("li:not(#gz)");

2）内容过滤器,不是:开头，是中括号[]!

$("li:contains('州')");//匹配包含给定内容的元素

$("li:empty");//内容为空的li元素

3）可见性过滤器

1. 属性过滤器

$("[style]");//有属性style的元素；$("li[style]");//li标签带有style属性

$("[href='#']");//带有href属性并且值为'#'；

$("[href!='#']");/带有href属性并且值不为'#'；

$("[href$='.jpg']");/带有href属性并且值以'.jpg'结尾；

5）状态过虑器

$("input:disabled");<!-- 不可用状态 -->

$("input:*readonly*");<!-- 只读状态 -->

$("input:checked");<!-- 默认选中状态 -->

6）表单过滤器

4、表单选择器

### 2、jQuery操作DOM

2.1、jQuery对内容和值的操作。

var str=obj.html(); obj.html("<span>123</span>");

var str=obj.text(); obj.text("123");

obj.val(); obj.val("abc");

obj.attr("属性名");obj.attr("属性名","属性值");

2.2、jQuery增删节点

2.2.1、创建新节点

var oSpan=$("<span>你好</span>");

2.2.2、将新节点挂到DOM树上:

parent.append(obj);作为最后一个子节点添加进来。

parent.prepend(obj);作为第一个子节点添加进来。

brother.before(obj);作为上一个兄弟节点

brother.after(obj);作为下一个兄弟节点

2.2.3、删除DOM节点

obj.remove();删除这个节点

obj.remove(selector);只删除满足selector条件的节点

parent.html("<span>你好</span>");创建新节点就删除了旧节点。

2.3、样式操作add remove has css toggle

obj.addClass("样式类名"); 添加样式。$("p").addClass("red").addClass("big");

obj.removeClass("样式类名");移除指定的样式

obj.removeClass();移除所有样式

obj.hasClass("样式类名");判断是否有某个样式

obj.toggleClass("样式类名");切换样式，有就删，没有就添上

obj.css("样式属性"，"样式的值");设置具体的样式

obj.css("样式属性");获得具体样式的值

### jQuery方法的返回值

3.1、任何通过$得到的对象都是jQuery对象

1）选择器过滤器 $("p")

2）转型：Dom——>jQuery对象 $(Dom)

var p=docemnnt.getElementByTagName("p")[0];

$(p).addClass(...);

3）创建新节点 $("<p></p>");

3.2、修改方法一般都返回jQuery对象

obj.html("abc"); obj.attr("src","../img/06.png");

obj.addCladd("red").removeClass("red");

3.3、查询到的元素一般都是jQuery对象

3.4、读取到的文字，返回值是string。

obj.val();

obj.html();

obj.text();

3.5、万能确认对象方式：

控制台输出看对象结构

### 4、通过节点关系，查找节点

obj.children();获取所有子元素/obj.children(selector);获取满足条件（选择器）的子元素

obj.next();obj.next(selector);下一个节点，弟弟

obj.nextAll();所有的弟弟

obj.prev();/obj.prev(selector);上一个节点，哥哥。

obj.prevAll();所有的哥哥

obj.siblings();/obj.siblings(selector)//查询所有的兄弟，结果不含自己。

obj.find(selector);查找满足条件的所有后代。

obj.parent();

### 5、jQuery事件

Dom事件查找繁琐，并且有严重的兼容性问题。

jQuery对dom做了封装，简化了操作，消除了兼容性问题。

5.1、动态绑定事件

$(function(){

Obj.click(function(){});//简写形式

//完整写法

Obj.bind(“click”,function(){});

});

关于js的window.onload和$(function(){});

window.onload在同一个HTML文件上有多个，后者会覆盖前者。

$(function(){});有多个，每一个都有效。

5.2、事件对象event

$(**function**(){

$(":button:first").click(**function**(event){

console.log(event);

});

});

5.3、事件机制--事件冒泡

事件冒泡机制，事件从内向外传播

作用：少些事件，通过获取事件源绑定一个事件

js:event.stopPropagation();

event.cancelBubble=true;

jquery中取消冒泡的做法：只需要在函数中添加return false即可。

5.4、获取事件源

js:var obj=event.srcElement || event.target;

jQ:var obj=event.target;jq直接兼容。

总结：jQuery为我们提供了一套统一简洁的API

jQuery对事件操作（重要）

5.5、合成事件（jq对事件的特殊贡献，是噱头，不是重点,不实用）

(**function**(){

$("img:first").hover(**function**(){

$("img:first").addClass("big");

$("img:first").attr("src","../img/14.png");

},**function**(){

$("img:first").removeClass();

$("img:first").attr("src","../img/06.png");

});

setInterval(**function**(){

$("img:eq(1)").toggle(100);

},200);

});

hover();相当于onmouseover和onmouseout的结合

toggle();相当于hide()和show()方法的结合

5.6、模拟事件

电脑模拟人的行为，做一些操作，激发了某些事件。

比如点奥帮你点击一个网站显示广告，可以认为点X关掉，不点的话，

2秒后电脑模拟点击X自动关掉，同时调用了绑定事件函数。

obj.trigger(事件类型); $(":button").trigger("click");

### 6、jQuery动画

6.1、显示、隐藏

show();/hide();

作用：通过同时改变元素的宽高来实现显示或隐藏（透明度）

用法：obj.show(执行时间,回调函数);

回调函数：传给某一个方法的函数，在方法结束时运行。可不写

执行时间：slow,normal,fast,写毫秒数

**function** show(){

$("img:eq(0)").show(500,**function**(){

console.log("显示完成");

});

}

**function** hide(){

$("img:eq(0)").hide(500,**function**(){

console.log("隐藏完成");

});

}

动画的本质，是使用定时器连续不断的修改样式。

启动了定时器，相当于启动了子线程，事件本身是主线程，相互不等待。

6.2、上下滑动式

slideDown();向下全部显示/slideUp();向上渐渐收起

作用：通过改变高度来实现显示或者隐藏效果。

用法同上。

**function** down(){

$("div:eq(1)>div").slideDown(2000);

}

**function** up(){

$("div:eq(1)>div").slideUp(2000);

}

6.3、淡入淡出

fadeIn();/fadeOut();

作用：通过改变透明度opacity来实现显示或者隐藏

用法同上。

6.4、自定义动画

自定义动画基于相对/绝对/固定定位。

animate(偏移位置，执行时间，回调函数);

偏移位置：{'left':'500px'};描述动画执行之后元素距离方向位置

# 、DATABASE数据库

刘国斌

## 一、数据库简介

1、IO流文件存储的弊端：

1.1、效率低（开发效率低，执行效率也低）

1.2、数据的增删改查非常麻烦

1.3、只能保存小量数据。

1.4、只能存储文本数据。

2、名词解释：

2.1、DB:Database:数据库，实际上就是一个文件集合，本质就是一个文件系统，数据按照特定的

格式存储到文件中，使用sql语言对数据进行增删改查操作。

2.2、DBMS:DataBaseManagementSystem,数据库管理系统，管理数据库文件的软件。

指一种操作和管理数据库的大型软件，用于建立、使用和维护数据库，对数据进行

统一的管理和控制。用户通过DBMS访问数据库中的数据。

3、常见数据库：Mysql,oracle,db2,sqlserver,sqlite ...

3.1、mysql:开源，公开源代码，免费，市场排名第一

弊端：通过卖服务赚钱，赚钱慢

闭源：不公开源代码，收费，卖产品，赚钱快。

3.2、oracle:oracle公司，性能最强大的数据库，收费最贵市场排名第二

3.3、DB2：IBM公司，闭源收费，通常一些银行项目会使用。

3.4、sqlserver:微软公司，，闭源收费，提供整套解决方案（web服务器，操作系统，

数据服务器等）市场排名第三

3.5、sqlite:轻量级数据库，应用在嵌入式或移动设备中，大小只有几十K，

功能和性能较大型数据库要少很多。

4、数据库的分类：

4.1、关系型数据库：经过数学理论验证，可以将现实生活中存在的各种关系保存到数据库中。

在此数据库中，以表的形式保存数据之间的关系。

4.2、菲关系型数据库：主要为了解决特定的应用场景，如缓存，高并发访问等。

存储数据的方式有多种。如redis是常见的非关系型数据库，以键值对的形式保存数据。

5、sql安装

参见sql安装文档，注意两点：1、端口号选择默认的3306,2、字符集选择utf-8

6、什么是数据库服务器

6.1、服务器：一台高配置电脑

6.2、ftp服务器：安装了ftp服务软件，提供了ftp服务功能的高配置电脑

6.3、邮件服务器：安装了邮件服务软件，提供了邮件服务功能的高配置电脑

6.4、数据库服务器：安装了数据库服务软件，提供了数据库服务功能的高配置电脑

6.5、web服务器：安装了web服务软件的高配置电脑。

## 二、SQL基础

### 1、连接数据库

--打开终端或命令行，在终端中输入以下命令：mysql -uroot -p回车，然后再敲回车

--退出指令：exit.

### 2、什么是sql：

Stuctured Query Language:结构化查询语言，使用sql语言和数据库服务器

进行交互，通过sql告诉数据库服务器对数据进行什么操作。

### 3、sql规范：

--以；（分号）结尾

--关键字之间有空格，通常只有一个，但多各也可以

--可以存在换行

--数据库名称和表名称区分大小写

### 4、Sql语句

#### 4.1、和数据库相关的SQL

--每一个工程对应一个数据库，存储数据需要先创建一个数据库，然后再数据库中创建表。

1）查看数据库:show databases;

2）创建数据库:create database 数据库名称;create database db1;

创建时指定字符集：create database 数据库名称 character set 字符集;

create database db2 character set gbk;

3）查看指定数据库详情：show create database 数据库名称;

4）删除数据库：drop database 数据库名称; drop database db2;

5）使用数据库：use 数据库名称; use db1;

#### 4.2、和表相关的sql

--什么是表：表是关系型数据库存储数据的单位，数据库中存储数据需要先

创建表，网标准存储数据。

1）创建表：create table 表名(字段1名 字段1类型,字段1名 字段1类型,...);

create table stu(id int,name varchar(10),age int,chinese int,math int,english int);

在创建表sql语句的执行过程中：在终端中写完sql语句后敲回车终端会把sql通过

网络传输到DBMS（mysql）,DBMS对sql语句进行解析，然后对数据库中的数据进行操作。

2）查看当前数据库中的所有表：show tables;

3）查看指定表详情：

3.1）show create table 表名;

3.2）desc 表名;查看表字段信息。

4）创建表指定引擎和字符集：create table 表名(...) engine=引擎类型 charset=字符集类型;

表的引擎

--innodb:支持数据库的高级操作，包括：事务，外键等。

--myisam:仅支持数据的增删改查操作。

5）删除表：drop table 表名;

6）表的修改

6.1）修改表名：rename table 原名 to 新名;

6.2）修改表的引擎和字符集:alter table 表名 engine=引擎类型 charset=字符集;

6.3）添加表的字段(添加一个)：

默认在最后面，alter table 表名 add 字段 字段类型;

在最前面，alter table 表名 add 字段 字段类型 first;

在某个字段后面，alter table 表名 add 字段 字段类型 after 字段名;

添加多个字段：alter table 表名 add (字段1 类型，字段2 类型，...);

alter table tb add (age int,aa int);

6.4）删除字段(删除一个)：alter table 表名 drop 字段名;

删除多个字段：alter table 表名 drop 字段1,drop 字段2;....;

6.5）修改字段名称和类型

alter table 表名 change 原字段名 新字段名 字段类型;

6.6）修改字段的类型和位置

alter table 表名 modify 字段名 字段类型 first/after 字段名;

#### 4.3、数据相关的sql

1）插入数据

--全表插入,要求插入的数据的数量和顺序要和表字段的数量顺序一致。

insert into 表名 values(值1,值2,...);

--指定字段插入:

insert into 表名(字段1，字段2，，....) values(值1，值2，...);

--批量插入：values后面写多套值。

insert into 表名 values (值1,值2,...),(值1,值2,...), ...;

insert into 表名(字段1，字段2，，....) values(值1，值2，...),(值1，值2，...)...;

2）查询数据

select \* from 表名;\*是所有字段。

select 字段1,字段2, ... from 表名;

3）删除数据

delete from 表名;删除表中所有数据。

delete from 表名 where 字段名=值;

delete from 表名 where 字段名 is null;

4）修改数据

update 表名 set 字段名1=值1,字段名2=值2，... where ...;

注意：windows电脑出现命令行中无法插入中文数据的解决方案。

在命令行中先登录mysql，然后执行set names gbk;通知mysql数据库

服务器，客户端（命令行）的编码格式为gbk.

### 5、主键约束

什么是约束：就是创建表的时候给表字段添加的限制条件。

主键约束的特点：唯一并且非空,一个表中只有一个主键

如何使用：创建表时在字段后添加primary key

create table t1(id int **primary key**,name varchar(10));

### 6、主键+自增：

create table t2(in int primary key auto\_increment,age int);

自增机制：

--当自增字段的值为null时会自动赋值并自增。

--以表中曾出现的最大值+1.

--删除数据自增值不减

--Delete 删除所有数据，自增值在原来基础上继续+1.

### 7、注释：comment

在创建表的时候，可以通过comment对字段进行描述。

create table t3(id int primary key auto\_increment comment ‘这是主键’，

comm int comment ‘这是奖金’);

### 注意：使用eclipse执行sql乱码问题

在建立连接时，修改url，在数据库名称后面添加：

数据库名?useUnicode=true&characterEncoding=UTF-8

### 8、`和'区别

**`**是在创建表时，修饰表名和字段的名的，可以省略。

Create table `t4`(in int,`age` int);

**'**是用来表示字符串的。

### 9、数据冗余-拆分表

如果数据库设计不合理，保存大量数据后会出现大量的重复数据，这种现象称为数据的冗余。

通过拆分表格的形式，把可能大量重复的数据，用单独的一张表保存，在原表中只需要通过id建立关系即可。

### 10、事务

事务是数据库中执行sql语句的最小工作单元，在同一个事务中的sql语句要么同时成功，要么同时失败。

--mysql数据默认sql语句是自动提交的。

--关闭数据库的自动提交。

--查看自动提交的状态：show variables like‘%autocommit%’;

--关闭自动提交:set autocommit=0;

--打开自动提交:set autocommit=1;

--验证转账流程：uadate person set money=800 where id=1;

--此时打开一个新的终端，在新终端中查看是否修改（没改）

--回到原窗口执行下面sql，commit之后再去新窗口验证（改了）。

update person set money=700 where id=2;

commit;

回滚rollback

执行rollback会回滚到上次提交的点挥着关闭自动提交的点。

--保存回滚点 savepoint s1(标识);

update person set money=100 where id=1;

savepoint s1;

update person set money=200 where id=1;

rollback to s1;

## 三、SQL分类及数据类型

### 1、SQL分类

#### 1.1、DDL

Data Definition Language,数据定义语言

--包括create ,drop,alter，truncate(删除表的指令）

--不支持事务

truncate table 表名；

作用：删除表并创建一张空表，自增(auto\_increment)数值清零

#### 1.2、DML:

Data Manipulation Language,数据操作语言

--包括：insert,update,delete,select

--支持事务

#### 1.3、DQL:

Data Query Language,数据查询语言

--只有select，也属于DML，没有修改数据，因此没有事务概念

#### 1.4、TCL：

Transaction Control Language,事务控制语言。

--包括：commit,rollback,savepoint,rollback to

#### 1.5、DCL:

Data Control Language,数据控制语言。

--分配用户权限的相关sql

### 2、数据库数据类型

#### 2.1、整型

--常用：int(m),bigint(m);m代表显示长度，如果字段数值长度不到m时，会在数值的前面补零，

但是一定要和zerofill结合使用。

create table t\_int(num int(10) zerofill);

Insert into t\_int values(15);

#### 2.2、浮点数

--常用：double(m,d);m代表总长度，d代表小数长度。

如：76.344——m=5,d=3。

--知道：decimal(m,d);超高精度小数，需要涉及高精度运算时使用。

#### 2.3、字符串

--char(m):长度不可变，m=20,abc 20,执行效率高，最大值255；

--varchar(m):长度可变，m=20,abc 3,节省资源，最大值65535，但是超过255简易使用text.

--text：可变长度，最大值65535.

#### 2.4、日期类型

--date:只能保存年月日

--time:只能保存时分秒

--datatime：保存年月日时分秒 9999-12-31，默认值为NULL.

--timestamp:保存年月日时分秒 2038-01-19，默认值为当前时间

create table t\_date(d1 date,d2 time,d3 datetime,d4 timestamp);

## 四、sql查询

### 1、没有条件的查询

select \* from 表名；

### 2、带条件的查询

#### 2.1、列值为null和不为null

1）查询没有上级领导的员工编号，姓名，工资

**select** empno,ename,sal **from** emp **where** mgr **isnull**;

2）查询有上级领导的员工编号，姓名，领导编号

**select** empno,ename,mgr **from** emp **where** mgr **is notnull**;

#### 2.2、别名

在要查询的字段后添加别名。

**select** empno **as**'员工编号',ename **as**'姓名'**from** emp;

**select** empno '员工编号',ename '姓名'**from** emp;

**select** empno 员工编号,ename 姓名 **from** emp;

#### 2.3、去重复

在要查询的字段前添加distinct

**selectdistinct** job **from** emp;

#### 2.4、比较运算符

**> ，< ，= ，>= ，<= ，!=，<>**

案例：查询不是10号部门的所有员工姓名，工资，部门编号（两种写法）

**select** ename 姓名,sal 工资,deptno 部门编号 **from** emp **where** deptno !=10;

**select** ename 姓名,sal 工资,deptno 部门编号 **from** emp **where** deptno <>10;

#### 2.5、and和or

1）查询不是10号部门，工资小于3000的员工编号，姓名，工资，部门编号

**select** empno,ename,sal,deptno **from** emp **where** deptno!=10 **and** sal<3000;

2）查询部门是30或上级领导为7698的所有员工的姓名，部门编号，上级领导编号

**select** ename,deptno,mgr **from** emp **where** deptno=30 **or** mgr=7698;

#### 2.6、in关键字

如果查询字段的职位多个的时候可以使用in关键字

1）查询年龄是25,26,30,22的所有信息

**select** \* **from** person **where** age **in** (25,28,30,22);

#### 2.7、(not)between...and..

在某两个数值之间，包含and两边的数值。

1）查询员工工资在500-1000的所有员工姓名和工资。

#### 2.8、(not)like: \_,%

\_：代表单个未知字符

%：代表多个未知字符

如：以a开头 a%；以a结尾 %a；第二个字符是a \_a%；包含a %a%

倒数第三个字符是a %a\_\_；第二个和最后一个是a \_a%a

1）查询名字以k开头的所有员工名字

**select** ename **from** emp **where** ename **like**'k%';

2）查询商品标题包含记事本的商品

**select** title **from** t\_item **where** title **like**'%记事本%';

### 3、查询结果排序 order by

3.1、单个字段排序：**order by 字段名 排序类型**；

放在查询语句条件最后面。默认为升序，升序：asc;降序：desc

查询员工姓名和工资，按照工资降序排序

**select** ename,sal **from** emp **orderby** sal **desc**;

3.2、多字段排序，当第一个字段有相同值时，第二个字段排序开始

**order by 字段1 排序类型，字段2 排序类型....**

### 4、limit分页查询

4.1、limit d1(跳过条数),d2(查询条数);放在排序后面。

查询拿工资最高的员工信息

**select** \* **from** emp **orderby** sal **desc** limit 0,1;

### 5、数值计算

+，-，\*，/，%（mod()）

查询所有商品的单价，库存，总价:select price,num,price\*num from t\_item;

--%和mod都是取余的作用，7%2 等效 mod(7,2)

### 6、日期相关函数

1）获取当前日期+时间now()：select now();与表无关，可以不写表名。

2）获取当前日期：select curdate();

3）获取当前时间：select curtime();

4）从日期和时间中提取日期:select date(获取的日期时间字段名);

5）从日期和时间中提取时间：select time(获取的日期时间字段名);

案例：查询商品创建的年月日：select date(created\_time) from t\_item;

6）提取年，月，日，时，分，秒

**select** extract(year **from**获取的日期时间字段名);

**select** extract(month **from** now());

**select** extract(day **from** now());

**select** extract(hour **from** now());

**select** extract(minute **from** now());

**select** extract(second **from** now());

7）日期格式化：date\_format(时间，格式)

格式：(**%Y**)4位年,(**%y**)2位年,**(%m**)2位月,(**%c**)1位月,(**%d**)日,(**%H)**24小时,(**%h**)12小时,(**%i**)分,(**%s**) 秒。

**select** date\_format(now(),'%Y年%m月%d日%H时%i分%s秒');

8）把不规则日期格式转成标准格式：

**str\_to\_date(日期字符串,格式)**

### 7、ifnull(x,y)

判断x是否为null，如果是取值y,如果不是取值x，age=ifnull(x,y);

案例：把员工表没有奖金的，奖金改为0；**update** emp **set** comm=ifnull(comm,0);

### 8、聚合函数

对多行数据进行合并统计

1）求和：sum(字段名)

2）平均值：avg(字段名)

3)最大值：max(字段名)

4）最小值：min(字段名)

5）统计数量：count(\*),满足条件的数据条数。

select count(\*) from person where age<50;

### 9、字符串相关函数

1、concat(a,b);字符串连接函数

select ename,concat(sal,'元') from emp;

2、char\_length(str);获取字符串的长度

**select** ename,char\_length(ename) **from** emp;

3、instr(str,subStr);获取subStr在str中的位置

**select** instr('NBA','B');

4、locate（subStr，str）；获取subStr在str中的位置。

**select** locate('B','NBA');

5、insert(str,start,length,newstr)插入字符串

在字符串str中 第start个位置插入newstr

6、lower(str)：转小写；

7、upper(str):转大写

8、trim(str);去两端空白 select trim（‘ abc ‘）;

9、left(str,length):从str左边截取length个字符

10、right(str,length):从str左边截取length个字符

11、substring(str,index,length) 从str中截取字符串，index开始位置，

12、replace(str,old,new):把str中的old部分替换new字符串

13、repeat(str,num);重复

14、reverse(str);反转

### 10、数学相关函数

1、向下取整 floor(num)

2、四舍五入 round(num);round(num,n)保留n为小数，n可为负

select round(2.7);

3、truncate(num,n)非四舍五入保留n为小数

4、随机数 rand(),范围为[0,1)

select floor(rand()\*6);[0,6)

## 五、分组、子查询、关联

### 1、分组查询 group by

分组查询通常和聚合函数结合使用

查询条件中每个xxx就以xxx作为分组的条件

格式：select deptno,avg(sal) from emp group by deptno;

### 2、 Having 有条件的分组统计

--where后面只能对普通字段进行筛选

--having写在group by 后面，通常是和group by结合使用

--普通字段的条件写在where后面，聚合函数条件下载having后面，having写在group by 后面。

--sql中各个关键字的顺序：

select...from 表名 where...group by...having...order by...limit

### 3、子查询

嵌套在sql语句里面的查询sql语句，称为子查询。

子查询可以有多层嵌套。

子查询可以写在的位置有：

1. 写在where/having后面做查询条件的值。
2. 写在from后面当成一张新的表，**必须起别名**。

3、写在创建表的时候，与表名之间用as连接。将查询的内容放进新表。

**createtable** newemp **as** (**select** ename,sal **from** emp);

### 4、关联查询

同时查询多张表的数据称为关联查询。

笛卡尔积：

关联查询如果不写关联关系，则查询结果为两张表的乘积，乘积称为笛卡尔积。

笛卡尔积是一种错误的查询结果，工作中不能出现。

##等值连接和内连接

等值连接和内连接都是关联查询的查询方式，效果相同。

等值连接格式：select \* from A,B where A.x=B.x ...

内连接格式：select \* from A join B on A.x=B.x where ...

内连接可读性更高，使用多

##外连接

关联查询有时候只查询两张表有关系的数据不能满足需求，如果需要查询某一张表所有的数据

（白喊没有关系的），则使用外连接查询方式。

--左外连接：以join 左边表为主查询所有数据，右边表只查询有关系的数据。

--右外连接：以join 右边表为查询所有数据，左边表只查询有关系数据。

## 六、表设计——关联关系

### 1、一对一

--两张表A和B，A表中有一条数据对应B表中的一条数据，称为一对一关系。

--应用场景：用户表和用户信息扩展表

==如何建立关系：在从表中添加一个字段记录主表的id。用户表和用户信息扩展表中，

用户表为主表，信息卡扩展表为从表，用户表中的id称为主键，从表中记录主表id

的字段称为外键。主键用来表示数据的唯一性，外键用来和其他表建立关系。

### 2、一对多

--AB两张表中A表中一条数据对应B表中多条数据，并且B表中一条数据对用A表中

一条数据，两张表的关系为一对多。

--应用场景：部门和员工，商品和分类

--如何建立关系：一对多的两张表，在多的表中添加一个字段记录另一张表的id。

1）查询每个部门对应的每个员工

2）查询每个员工对应的每个部门只显示工资大于2000的信息。

**select** \* **from** t\_emp e **join** t\_dept d **on** e.deptid=d.id ;

**select** \* **from** t\_emp e **join** t\_dept d **on** e.deptid=d.id **where** e.sal>2000;

### 3、多对多

--AB两张表，A表中的一条数据对应B表中的多条数据，同时B表的一条数据对应A表的

多条数据，这种关系称为多对多。

--应用场景：老师表和学生表，用户表和角色表

--如何建立关系：需要创建一个关系表，在关系表中记录两个表的id

### 4、自关联

--一张表当成两张表

--在一张表中有一个字段指向当前表的id，称为自关联。

--使用：当保存的数据有层级关系并且不确定有多少层的时候使用自关联。

1）查询员工姓名和对应的上级姓名

**select** e.ename,m.ename **from** emp e **join** emp m **on** e.mgr=m.empno;

### 5、连接方式和关联关系

连接方式：包括内连接和外连接（左外和右外），是指通过sql查询两张表数据时

使用的查询方式

关联关系：一对一，一对多，多对多，是指设计表时两张表之间存在的逻辑关系。

## 案例

### 1.权限管理的表设计

1.创建5张表user(id,name),role(id,name),module(id,name),

用户和角色关系表u\_r(uid,rid),角色和权限表r\_m(rid,mid)

**createtableuser**(id **intprimarykey** auto\_increment,name **varchar**(10));

**createtable** role(id **intprimarykey** auto\_increment,name **varchar**(10));

**createtable** module(id **intprimarykey** auto\_increment,name **varchar**(10));

**createtable** u\_r(uid **int**,rid **int**)

**createtable** r\_m(rid **int**,mid **int**)

2、插入数据user:刘德华，张学友，凤姐；role:男游客，男会员，女游客，女管理员

module：男浏览，男发帖，女浏览，女发帖，女删除

**insertintouservalues** (**null**,'刘德华'),(**null**,'张学友'),(**null**,'凤姐');

**insertinto** role **values**(**null**,'男游客'),(**null**,'男会员'),(**null**,'女游客'),(**null**,'女管理员')；

**insertinto** module **values**(**null**,'男浏览'),(**null**,'男发帖'),(**null**,'女浏览'),(**null**,'女发帖')，(**null**,'女删除');

3、建立关系：刘德华是男游客，张学友是男会员，凤姐是女游客和男会员，

男游客对应男浏览，男会员对应男浏览和男发帖，女游客对应女浏览，女管理员对应女浏览女发帖和女删除

**insertinto** u\_r **values**(1,1),(2,2),(3,2)(3,3);

**insertinto** r\_m **values**(1,1),(2,1),(2,2),(3,3)(4,3),(4,4),(4,5);

1）查询每个用户对应的所有权限

**select** u.name,m.name **fromuser** u **join** u\_r ur **on** u.id=ur.uid

**join** r\_m rm **on** ur.rid=rm.rid

**join** module m **on** rm.mid=m.id;

2）刘德华对应的所有权限

**select** u.name,m.name **fromuser** u **join** u\_r ur **on** u.id=ur.uid

**join** r\_m rm **on** ur.rid=rm.rid

**join** module m **on** rm.mid=m.id

**where** u.name='刘德华';

3）有男发帖的用户都有哪些

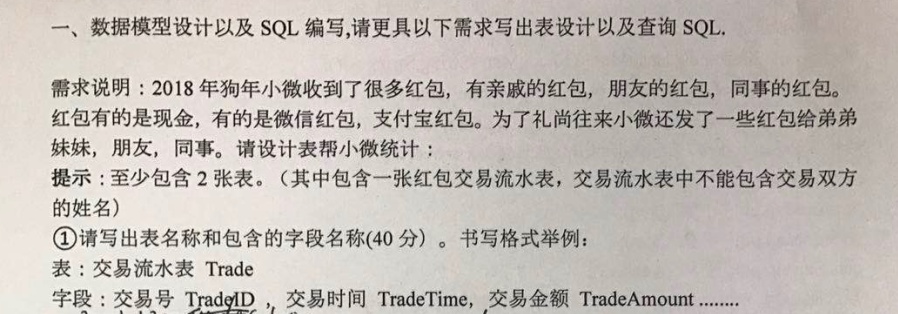
**select** m.name,u.name **fromuser** u **join** u\_r ur **on** u.id=ur.uid

**join** r\_m rm **on** ur.rid=rm.rid

**join** module m **on** rm.mid=m.id

**where** m.name='男发帖';

### 2、面试题



1)--创建表

**createtable** trade(id **intprimarykey** auto\_increment,**timedate**,money **double**,**typevarchar**(5),pid **int**);

**createtable** person(id **intprimarykey** auto\_increment,name **varchar**(10),gender **varchar**(5),rel **varchar**(5));

--插入数据

**insertinto** person **values**(**null**,'刘德华','男','亲戚'),(**null**,'刘若英','女','亲戚'),

(**null**,'马云','男','同事'),(**null**,'特朗普','男','朋友'),(**null**,'貂蝉','女','同事');

**insertinto** trade **values**(**null**,'2018-02-20',-20,'微信',1),

(**null**,'2018-02-21',500,'现金',2),

(**null**,'2018-02-21',-50,'现金',2),

(**null**,'2018-02-23',50000,'支付宝',3),

(**null**,'2018-02-23',-100,'支付宝',3),

(**null**,'2018-02-20',1000,'微信',4),

(**null**,'2018-02-20',-20000,'现金',5);

3)统计2月15号到现在的所有红包收益

**select**sum(money) **from** trade **wheretime**>str\_to\_date('2018年2月15号','%Y年%m月%d号');

4)查询2月15号到现在金额大于100所有女性亲戚的名字和红包金额。

**select** p.name,t.money **from** person p **join** trade t **on** p.id=t.pid

**where** t.money **notbetween** -100 **and** 100**and** p.gender='女'**and** p.rel='亲戚'

**and** t.**time**>str\_to\_date('2018年2月15号','%Y年%m月%d号');

5)查询三个平台分别收入的红包金额

**selecttype**,sum(money) **from** trade **where** money>0 **groupbytype**;

## 七、视图、约束、索引

### 1、视图

数据库中存在多种对象，表和视图都是数据库中的对象，创建视图时名称不能和表名重名。

视图实际上是代表了一段sql查询语句，可以理解成视图是一张虚拟的表，

表中的数据会随着原表的改变而改变。

--因为有些数据的查询需要书写大量的sql语句，每次书写比较麻烦，使用视图可以起到sql重用的作用。课 可以隐藏敏感信息

1.1、创建视图

create view 视图名 as 子查询；

**createview** v\_emp\_10 **as** (**select** \* **from** emp **where** deptno=10);

--隐藏员工表的工资字段

**createview** t\_emp\_nosal **as** (**select** empno,ename,job,mgr,deptno **from** emp);

1.2、视图分类

--简单视图：创建视图的子查询中不包含：去重，函数，分组，关联查询的视图称为简单视图。

可以进行增删改查操作

--复杂视图：和简单视图相反。只能查。

在简单视图中进行增删改操作，与对表的操作一样

1）视图中插入数据

insert into v\_emp\_10 (empno,ename,,deptno,sal) values (10001,’张三’,10,300);

2）数据污染：往视图中插入一条视图中不显示，但是原表会显示的数据称为数据污染。

如在10号部门视图表插入20号部门数据，视图中不会显示，原表中会显示，就是数据污染。

如果需要避免数据污染的出现，创建视图时需要使用with check option 的关键字，

插入不匹配数据时会失败。

3）修改和删除只能操作视图中存在的数据。

1.3、修改视图

create or replace view 视图名 as 子查询

1.4、删除视图

drop view 视图名

drop view if exists 视图名,如果存在就删除，不存在也不报错。

1.5、视图别名

如果创建视图时使用了别名，则对视图操作的时候只能使用别名。

视图总结：

1. 视图是数据库中的对象，代表一段sql语句
2. 作用：重用sql，隐藏敏感信息。
3. 分类
4. 工作中一般使用视图时只进行查询操作，如果需要增删改则直接对原表进行操作。

### 2、约束

约束是给表字段添加的限制条件。

#非空：not null。添加非空约束的字段，值不能为空。

#唯一：unique。添加唯一约束的字段，值不能重复。

#主键约束：primary key。添加了主键约束的字段，值不能为null也不能重复。

--创建表时添加主键约束

--创建表后添加主键约束。

create table t (id int);

alter table t add primary key(id);

--删除主键约束：alter table 表名 drop primary key;

#自增

1. 当字段赋值为null，字段会自动增长
2. 如果删除数据，自增数值不会减
3. 如果指定插入比较大的值，下次插入数据时会从最大值的基础上+1
4. 如果使用delete删除全表数据，自增值不变。
5. 使用truncate关键字删除，自增数值清零。

#默认约束 default

--给字段设置默认值，当字段不赋值的时候，默认值生效。

create table t(id int,age int default 10);

#检查约束 check

Mysql 不支持，但语法通过，不会报错。

#外键约束

--作用：用来保证两张表之间数据一致性和完整性的约束

--添加约束后，外键的值可以为null，可以重复，但不能是另外一张表不存在的数据。

--添加约束后，外键指向的表不能先删除，如果需要删除，先删除外键约束或存在外键的表。

--添加约束后，外键指向的数据不能先删除

--外键的值通常指向另外一张表的主键。

--使用外键必须两张表使用相同的引擎（innodb）,myisam不支持外键。

--工作中除非特殊情况，一般不使用外键约束，使用java代码逻辑对插入和删除的数据

进行限制，因为加了外键约束后不方便测试。

添加外键约束的格式：

create table emp(id int,age int,deptid int,

constraint 约束名 foreign key(字段名) references 关联表名(关联的字段名));

### 3、索引

实验表

--在终端中先登录mysql，在db6下面执行 source 命令

source 文件的绝对路径。

--学生机路径：source/home/soft01/桌面/item\_backup.sql

--windows电脑路径：把文件放在d盘下

source d:/item\_backup.sql

--导入完成之后 执行：select count(\*) from item2;测试

3.1、索引是用来提高查询速度的技术，类似一个目录。

如果不使用索引数据会零散的保存在磁盘块中，磁盘块大小（4-8k），查询数据时

需要挨个遍历每一个磁盘快，直到找到数据为止。

使用索引之后会在磁盘中将数据以树状结构进行保存，查询数据时从树状结构中进行查询，

可以大大降低磁盘快的访问数量，从而提高查询速度

3.2、索引会占用磁盘空间，所以创建时需谨慎。根据查询需求来决定创建什么索引。

3.3、索引需要建立在大量数据的表中，如果数据量不够大，有可能会降低想效率。

3.4、索引的分类

1）聚集索引（聚簇索引）：数据保存在树状结构中，一张表只有一个聚集索引，

数据库会自动为添加了主键的表创建聚集索引。

2）非聚集索引：树状结构中没有数据，保存的是磁盘块的地址。可以有多个。

3.5、创建索引（非聚集索引）

create index 索引名 on 表名（字段名[（长度）]）;

--创建索引之前先查询看时间

select \* from item2 where title='100'; 1.07s

--创建title索引

create index index\_title on item2(title);

--再次查询

select \* from item2 where title='100';

3.6、查看索引:show index from 表名;

3.7、删除索引:drop index 索引名 on 表名;

3.8、复合索引：创建索引的时候添加多个字段

--当频繁使用多个字段作为查询条件的时候使用复合索引。

创建：create index 索引名 on 表名(字段1，字段2...);

3.9、创建表时添加索引：create table t(id int,age,int,index index\_age(age));

索引总结：

1. 索引会占磁盘控件，不是越多越好。
2. 数据量小的表不要创建索引
3. 主键会自动创建聚集索引
4. 对经常出现在where/order by/distinct后面的字段创建索引可以提高效率，效果更好。
5. 不要再修改太频繁的表中创建索引。

### 4、事务

4.1、事务：数据库执行sql语句的最小工作单元，不可拆分，同时成功或同时失败。

4.2、事物的ACID特性（常见面试）

1）Atomicity:原子性，最小，不可在拆分

2）Consistency:一致性，同时成功，同时失败

3）Isolation:隔离性，多个事务之间互不影响

4）Durability：持久性，事务完成后数据持久保存到数据库

4.3、sql中事务的指令

1）查看自动提交的状态:show variables like ‘%autocommit%’;

2）设置自动提交的状态：set autocommit=0/1;

3）提交：commit

4）回滚：rollabck;

5）保存回滚点：savepoint s1;

6）回滚到某个点：rollback to s1;

# 第七章、JDBC

--JDBC:Java Database Connectivity:Java数据库连接，实际上JDBC是java中一套和

数据库进行交互的api（application program interface）应用程序编程接口。

--因为java程序员可能有需求连接各种数据库，为了避免java程序员每一种数据库都学习

一套新的api，sun公司提出了一个叫JDBC的接口，各个数据库厂商根据这个接口去写实现类（驱动），

这样的话java程序员只需要掌握JDBC接口的调用就可以操作任意数据库。

## 一、JDBC连接数据库基本操作

### 1、创建JDBC

1.1、创建**Maven**工程

**Maven**：是项目管理工具，可以构建工程，通过使用坐标的形式导入jar包，

使用maven会自动导入依赖jar包

1.2、下载mysql相关的jar包，http://maven.tedu.cn，搜索mysql，找mySql：mysql-connnector-java

左侧目录下5.1.6展开，单击mysql-connector-java-5.1.6.jar，右侧代码拷贝。

### 2、JDBC访问数据库

2.1、步骤：

maven文件下5.1.6jar包下com.mysql.jdbc包展开，找到Driver.class打开，

在Attach Source按钮下方 public class 后面为**驱动类**全名com.mysql.jdbc.Driver

**1）注册驱动**

格式：Class.*forName*("驱动类名");

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

**2）获取连接对象**

格式：Connection conn=DriverManager.*getConnection*("数据库url", "用户名", "密码");

Connection conn=DriverManager.*getConnection*("jdbc:mysql://localhost:3306/db6", "root", "1234");

**3）创建sql执行对象**

格式：Statement stat=连接对象.createStatement();

Statement stat=conn.createStatement();

**4）创建sql语句** if not exists不存在则创建，存在不报错

String sql="create table if not exists t\_jdbc(id int,name varchar(10))";

**5）执行sql语句**

stat.execute(sql);

2.2、方法解释

1）**execute** 此方法可以执行任意sql，返回值true代表有结果集（查询语句时会有结果集），

False为没有结果集。通常情况下execute方法用于执行DDL（数据定义语言）。

2）**executeUpdate** 此方法执行增删改操作，执行DML，返回值为int型，代表生效的行数执行DML

3）**executeQuery** 执行查询操作，执行DQL，返回值类型为ResultSet，里面保存查询到的所有结果

4）**ResultSet**对象：此对象里面装着查询到的结果数据，见到ResultSet就用while遍历。

5）**next()** 方法会先判断是否右下一条数据，有返回true，没有返回false，如果返回true

内部游标（指针）会往下移动。

6）从ResultSet对象中获取数据

--通过字段的名称获取数据

**int**id=rs.getInt("id");

String name=rs.getString("name");

--通过字段的位置获取数据，从1开始

**int**id=rs.getInt(1);

String name=rs.getString(2);

7）数据库字段类型和java类型对比

数据库字段类型 ————java获取数据

Int——————————getInt

Varchar————————getString

double ————————getDouble

datetime/timestamp——getDate

8）关闭资源

--关闭连接对象Connection，使用完之后的连接要及时关闭，避免浪费资源。

--关闭Statement，会占用空间，而且Statement对象有上限

--关闭ResultSet,里面保存的是查询结果，用完就释放，节省空间。

--关闭顺序：先建后关，ResultSet—>Statement—>Connection

3、封装工具类：复用代码

3.1、DBUtils 第一次封装

3.2、DBUtils 第二次封装

把连接数据库的信息存放到了配置文件中，这样换数据库的话只需要该配置文件即可，

把读取数据放到了静态块中，保证读取数据的代码只执行一次。

二、

模板代码

自定义快捷代码

1. 复制每次想重复的代码
2. 单击window选中preferences并打开，选择Java，Editor,Templates,点击New
3. 在新出的窗口中Name处添加快捷输入名，Description为描述，
4. 将复制的代码粘贴在Pattern处
5. 希望快捷输入代码块后光标自动出现在需要编辑的地方时，可将光标停在此处，

点击Insert Variable按钮，找到双击Cursor，点OK即可。

#Properties 属性配置对象

1、程序员可以把工程中的某些数据以配置文件的形式进行保存，此对象就是处理

\*.properties文件的对象，数据是以键值对的形式进行保存的。

2、如何使用：

1）创建 my.properties

数据库连接池DBCP

DataBase Connection Pool

如果没有数据库连接池，每次和数据库交互都需要一次建立连接和断开连接操作，若果有一万次交互就有一万次建立连接和断开连接，频繁的开关连接非常消耗服务器资源，使用连接池，可以设置一个最大连接数量，如果需要连接，连接池要有空闲连接就给并执行，没有则等待归还之后在执行，这样大大降低了开关连接的次数，把连接的利用率最大化。

使用数据库连接池

导包：内网maven.tedu.cn,外网maven.aliyun.com

##PreparedStatement

是Statement的子接口

好处：

--代码可读性高，避免了拼接sql语句的麻烦。

--带有预编译效果，可以避免sql注入。因为预编译的时候已经把sql的逻辑固定，

不允许再次修改，所以可以避免注入。

创建用户表：create table user(id int,username varchar(10),password varchar(10));

Insert into user values(1,

##数据库中有什么表就需要在Java代码中创建什么对象，表中有什么字段，对象中就有什么属性。

数据库中有什么表就需要在Java代码中创建什么DAO类(Data Access Object 数据访问对象）

练习：

##批量操作

因为每次执行sql都需要和数据库服务器进行数据交互，多次执行sql添加到一个批量处理里面，

可以把多次交互变成一次交互，从而提高执行效率。

分页查询

练习：控制台输入查询的页数和每页的条数，然后返回响应的数据PageTest.java

##事务

创建person表 id,name,money

插入1，超人，500；2，蝙蝠侠，5000；

实现：超人跟蝙蝠侠借钱2000.

实现效果：如果蝙蝠侠的钱足够就借，输出借钱成功，不够就输出蝙蝠侠加以没钱了。

1、事务相关方法

设置自动提交方法：conn.setAutoCommit(true/false);

提交：

练习：创建部门表dept(id,name),创建员工表emp(id,name,deptid)

通过代码插入

获取元数据

1、数据库元数据：包括数据库版本，用户名，密码，连接地址。

# 第八章、ServletJSP

[程祖红chengzh@tedu.cn](mailto:程祖红chengzh@tedu.cn)

Servlet/jsp mvc框架

## 一、Servlet

### 1、servlet介绍

sun公司制定的一种用来扩展web服务器功能的组件规范。

（1）用来扩展web服务器功能

Web服务器只能处理静态资源请求（即需要事先将html文件准备好，并存放到指定的文件夹下面）

不能够处理动态资源的请求（即需要计算，生成动态页面），所以需要扩展其功能。

Servlet可以扩展web服务器功能，，当web服务器收到请求之后，如果需要计算，可以调用servlet来处理。

（2）组件规范

1. 什么是组件：符合规范，实现部分功能，并且需要部署到相应的容器里面才能运行的软件模块。

Servlet就是一个组件，需要部署到servlet容器里面才能运行。

2）什么是容器：符合规范，提供组件运行环境的程序。

Servlet容器需要符合servlet规范，为servlet提供网络相关的服务。

Tomcat容器，www.apache.org

### 2、安装Tomcat

(1)检查是否有Tomcat：JavaEES环境下点Window，Show View，点击打开Servers界面，

即可显示是否有Tomcat.若没有再安装。

1. 安装Tomcat：Window下点开Preferences，展开Server下Runtime Environments运行环境

点ADD按钮选与tts9文件下tomcat文件名相同版本的Apache Tomcat,然后下一步，选择

Browse下对应的tts9/apache-tomcat文件，JRE下选择jdk,（若没有jdk,点JRE左边Installed jRE...

按钮安装jdk，点ADD选则Standard VM安装。）

选好jdk完成。Servers界面下双击Tomcat，Server Locations下选第二项，

右上角Publishing下选择第一项即可，右下角可看端口。

### 3、如何写一个servlet

Step、写一个java类，实现Servlet接口或者继承HttpServlet抽象类，

注：一般选择继承

Step2、编译。

Step3、打包（变成一个组件）

appname（应用名）

WEB-INF

classes(放.class文件)

lib(可选，放.jar文件)

web.xml(部署描述文件)

Step4、部署。

将step3创建的整个文件夹拷贝到容器里。

注：可以使用jar命令将step3创建的整个文件夹压缩成”.war”结尾的文件，

然后再拷贝这个文件就可以了。

Step5、启动servlet容器，访问servlet。

访问：<http://ip:port/appname/url-pattern>

注：url-pattern在wen.xml文件当中设置。

<http://localhost:8080/day01/hello>

Hello Kitty

搭建maven工程：

（1）鼠标右键new一个maven项目，选Maven Project，下一步NEW Maven project

界面勾选第一个创建简单项目，下一步：填写组名工程名，Packaging选war包完成。

（2）新建的工程会有小红叉报错，打开项目，右键单击第一项Deployment....，选择有小奶瓶标记的 Generate Deployment...(打包，加载部署文件；目录src/main/webapp/WEB-INF/web.xml).

（3）若项目下没有Deployment....,则点击JavaEE再操作第二部即可。

（4）部署到容器里。

右键单击项目，选最后一项Properties,找到Targeted Runtimes(运行目标），

勾选目标容器Apache Tomcat.

（5）启动servlet容器

### 4、servlet是如何运行的

比如在浏览器地址栏输入

<http://ia:port/day01/hello>

Step1、浏览器依据ip,port(端口)与tomcat建立连接

Step2、浏览器将请求数据打包。

注：即按照http协议的要求，创建请求数据包，并把该请求包发送给tomcat。

Step3、tomcat收到请求数据包之后，拆包，并将解析到的数据存放到request对象里面，

同时创建一个response对象。

注：拆包，即按照http协议要求，解析请求数据包的内容。

**Step4**、tomcat将servlet实例化，然后调用实例的service方法。

注：tomcat会将request对象和response对象作为参数传递给service方法。

开发人员可以通过调用request对象提供的方法获取请求数据包里面的数据，也可以

将处理结果写入response对象。

Step5、tomcat会从response对象中获取处理结果，然后打包发送给浏览器。

Step6、浏览器解析响应数据包，生成页面。

### 5、常见问题

（1）404

1）含义：404是一个状态码，服务器依据请求路径找不到对应的资源。

2）原因

a、请求路径写错。

b、忘记部署或者部署失败。Tts9/apache-tomcat/wtpwebapps可以查看是否有项目。

（2）500

1）含义：服务器运行时发生异常。

2）原因

a、代码不规范（没有按照servlet规范）

比如：没有继承HttpServlet,配置文件中

B、代码不严谨，如对请求参数值没有做检查就做类型转换。

（3）405

1）含义：服务器找不到处理方法

2）原因：没有正确override HttpServlet的service方法。

java工具：端口监视（服务端代理）

（1）Window选项，点击Show View 展开Other，Debug文件夹下选中TCP/IP Monitor

（2）TCP/IP Monitor界面下左上角空白格中，鼠标右键点击properties,然后点即Add按钮，

弹出New Monitor小窗口，Local monitor port处本地端口填4个数字（8888），

下方Host name填ip,（如localhost：8080），Port为当前Servlet使用的端口（如8080,8088）。

Local monitor port：本地监事端口

Host name:要访问的服务器ip

Port：要访问的服务器端口

（3）选中新添加的TCP/IP Monitor，Start运行，

（4）通过本地ip+本地端口访问到要访问的服务器端口

## 二、请求方式及乱码问题

### 1、http协议（了解）

（1）http协议：是一种网络应用层协议，规定了浏览器与web服务器之间如何通信以及相应的数据包的格式。

注：TCP/IP 传输层和网络层协议，可以确保数据可以可靠的传递。

Http协议将数据打包之后，需要依赖TCP/IP协议来传递数据包。

1）通信过程

建立连接

发送请求

发送响应

关闭连接

2）优点：web服务器可以利用有限的连接数为尽可能多的请求服务。

（2）数据包的结构

1）请求数据包

a、请求行（请求方式 请求资源路径 协议和版本）

b、若干消息头

注：消息头是一个键值对（以“：”隔开），浏览器与web服务器之间可以通过发送消息头来

传递一些特定的信息，比如浏览器可以发送“user-agent”消息头告诉服务器，浏览器的类型和版本。

c、实体内容：get请求没有数据，只有发送post请求时，才会有数据（请求参数）

2）响应数据包

a、状态行（协议和版本 状态码 状态描述）

注：200：正确

404：依据请求路径找不到对应的资源

500：系统出错

b、若干消息头

注：服务器也可以将一些消息头发送给浏览器，比如发送“content-type”消息头告诉浏览器，服务器发送过来的数据类型以及编码。

c、实体内容：程序处理的结果，浏览器需要解析出来，生成相应的页面。

（3）两种请求方式

1）get请求

a、使用场景：在地址栏输入某个地址；点击链接；表单默认的提交方式

b、特点：

--会将请求参数显示在浏览器地址栏，不安全。

注：有一些网络设备（比如路由器）会记录所有的请求地址，

--会将请求参数添加到请求行（也就是请求资源路径的后面），只能提交少量的数据（2k）。

如果包含了敏感的请求参数，则不安全

2）post请求

a、发送场景：设置表单的method属性为“post”。

b、特点：

--浏览器地址栏不会显示请求参数，相对安全。

注：不管是哪种请求类型，都不会对数据加密。对于敏感数据，一定要加密处理。

--请求参数会添加到实体内容里面，可以提交大量的数据。

注：实体内容里面添加的数据量理论上是没有限制的。

### 2、读取请求参数值

（1）方式一

String request.getParameter(String paramName);

注：paramName：请求参数名。

如果请求参数名写错，返回null。

（2）方式二

String[] request.getParameterValues(String paramName);

注：有多个请求参数名相同时，用该方法。

### 3、Servlet输出中文时产生的乱码问题

（1）乱码原因

out输出时，默认会使用iso-8859-1来编码

（2）解决方法

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

### 4、表单包含有中文参数值产生的乱码问题

（1）乱码原因：表单提交时会对中文参数值进行编码。，比如：使用utf-8来编码，而服务器默认会

使用iso-8859-1来编码，所以产生了乱码。

注：会按照打开该表单所在的页面是的字符集来编码

（2）如何解决

1）情形1：post请求

读取请求参数前：request.setCharacterEncoding("utf-8");

2)情形1：get请求

修改server.xml,65行添加

<ConnectorURIEncoding=*"utf-8"*/>

http协议

What

数据包结构

两种请求方式

读取请求参数值

输出中文

response.setContentType

表单包含有中文参数值

post

Request.setchar

get

## 三、重定向

1、重定向：

（1）服务器通知浏览器重新访问一个新的地址。

注：服务器可以发送一个302状态码及一个Location消息头（该消息头的值是一个地址，

一般称之为重定向地址）给浏览器；浏览器收到后，会立即向重定向地址发请求。

（2）如何重定向：response.sendRedirect(String url);

注：url为重定向地址。

（3）特点

1）重定向地址是任意的

2）重定向之后，浏览器地址栏的地址会发生变化。

（4）一个细节（了解）

重定向之前，容器会先清空response对象上存放的所有数据。

步骤：

1. 创建工程
2. 导包：jdbc驱动，连接池
3. 添加jdbc.properties文件
4. DBUtil类，并测试
5. 实体类
6. DAO类

## 四、JSP

### 1、jsp概念

Java server page

sun公司制定的一种服务器端动态页面技术规范。

注：a、虽然使用servlet也可以生成动态页面，但是过于繁琐（需要时有out.println语句输出），

并且维护麻烦（要修改页面，就必须修改java代码）。所以sun才制定了jsp规范。

b、jsp是一个以.jsp为后缀的文件，主要内容是html和少量的java代码。

容器（如tomcat）会将该文件换成一个对应的servlet然后执行。

c、jsp本质就是一个servlet。

### 2、如何写jsp

**Step1**、添加一个以.jsp为后缀的文件。

**Step2**、在该文件里面，可以添加如下内容：

（1）html(css,javascript):直接写即可

（2）Java代码：

第一种形式：java代码片段 <% java代码 %>

第二种形式：jsp表达式 <%=表达式 %>；相当于out.println(表达式)，如：<%=u.getId()%>

第三种形式：jsp声明 <%! 声明一个方法或者变量 %>//servletDay10

（3）隐含对象

（4）指令

### 3、隐含对象

什么是隐含对象：在jsp文件里面，直接可以用的对象，

为什么可以直接使用这些隐含对象：

jsp会被容器转换成一个对应的servlet，在转换过程中，容器会自动添加获得这些对象的代码。

（1）输出流对象out，

（2）请求对象request，

（3）响应对象response，

（4）session，

（5）application

（6）exception：只有在isErrorPage=“true”是才能使用。

（7）pageContext，（a6.jsp,a7.jsp）

1）pageContext是什么

容器会为每一个jsp实例都会创建一个唯一的一个符合pageContext接口要求的对象。

2）特点：

唯一性：一个jsp实例对应一个pageContext。

持久性：只要jsp实例还在，则pageContext会一直存在。

3）作用：

作用1：绑定数据

注：绑定到pageContext上的数据，只有对应的jsp实例能访问到。

作用2：提供了获得其他所有隐含对象的方法。

（8）config：ServletConfig

（9）Page：jsp实例本省。jsp对应的那个servlet实例

### 4、jsp指令

1）什么是指令

通知容器，在将jsp转换成servlet时，做一些额外的处理，比如导包。

2）语法：**<%@ 指令名 属性=值 %>**

3）page指令：

**a、import**属性：指定要导的包，如：<%@ page import=”java.util.\*” %>

注：如果要导多个包，报预包之间用“，”隔开。

**b、contentType**属性：设置response.setContentType方法的参数值。

**c、pageEnconding**属性：告诉容器，再读取jsp文件的内容时，使用指定的字符集去解码。

pageEnconding=*"utf-8"*注：有部分容器，在读取jsp文件的内容时，

默认使用iso-8859-1来解码。

//servletDay10

**d、session**属性：缺省值时true，如果设置为false，则不能使用session隐含对象。

e、errorPage属性：用来指定一个异常处理页面，当jsp运行出错,容器会调用异常处理页面

f、isErrorPage属性：缺省值时false，如果值是true，则可以使用exception隐含对象。

4）include指令

**file**属性：指定要包含的文件名。如<%@ cinclude file=”header.jsp” %>

注:容器再讲jsp转换成servlet时，会将file属性指定的文件的内容插入到该指令所在的位置。

若file指定的jsp文件内容只是jsp片段，则该文件不会被转换成.java文件。

5）taglib指令：用于导入jsp标签。

### 5、jsp注释

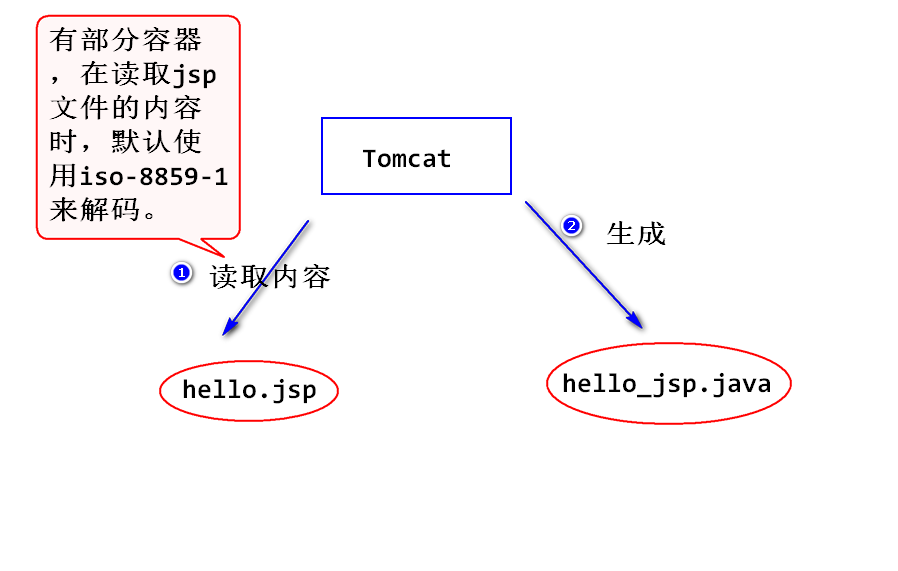
（1）<!-- 注释内容-->

注：如果注释的内容是java代码，则java代码会执行，（在注释里执行，所以不显示）

（2）<%-- 注释内容--%>

注：如果注释的内容是java代码，java代码不会执行

### 6、jsp如何运行的



阶段一、容器将.jsp文件转换成一个.java文件

1）html——>在service方法里面，使用out.write输出。

2）<% %>——>在service方法里面，照搬java代码。

3）<%= %>——>在service方法里面，使用out.print输出

4）<%! %>——>为servlet增加新的属性和方法。

阶段二、容器再执行该servlet。包括编译、实例化。调用service方法。

### 7、jsp标签和EL表达式

**（1）什么是jsp标签**

Sun公司指定的一种用来替换jsp中java代码的技术，语法类似于HTML标签，容器遇到jsp标签之后，会找到标签对应的标签类然后执行。

注：

a、因为，直接在jsp文件中写java代码，不利于jsp文件的维护，

比如，将包含有java代码的jsp交给美工去修改就很不方便。

所以，sun公司才制定了jsp标签技术规范。

b、使用jsp标签，jsp文件会变得跟简洁，方便维护，同时，方便代码的复用。

**（2）EL表达式**

是一套简单的运算规则，用于给jsp标签的属性赋值，也可以脱离jsp标签直接使用。

**（3）EL表达式的使用**

1）访问bean的属性。

注：javabean

如果一个java类满足如下几个条件，就可以称之为Javabean：

public class

public 构造器

Implement Serializable(序列化接口)

属性

get/set方法

方式一、${user.username}（servletDay10/e1.jsp）

注：a、执行过程：

容器依次从pageContext->request->session->application中查找**绑定名称**为”user”

的对象（即调用getAttribute方法），找到就不再向下查找。找到该对象之后，调用

getUsername方法，然后输出计算结果。

b、会将null转换成“”输出。

c、如果依据绑定名找不到对应的值，不会报空指针异常，任然输出“”。

d、可以使用pageScope、requestScope、sessionScope、applicationScope来指定查找范围，

比如，从session对象中查找：${sessionScope.user.username}

方式二、${user[‘username’]}

注：a、执行过程：同方式一。

b、允许[]里面使用绑定名。

c、允许[]里面使用从0开始的下标，用于访问数组中的某个元素。

${user.interest[1] }

**（4）EL做一些简单的运算**

注：运算结果可以用来给jsp标签的属性赋值，也可以直接输出。

1）算术运算：+、-、\*、/、%

${100+100 }${'1'+'2' }

注：**+**号只能求和，不能连接字符串

2）关系运算：>、>=、<、<=、==、!=

${1<2 }//true

<%request.setAttribute("s1", "abc");%>

${s1=="abc" }<//true

${sessionScope.s1=="abc" }//false

3）逻辑运算: **&&**、**||**、**！**

${1<2 && 3>5 }

${user.gender=='m' || user.age>30 }

4）empty运算：

判断集合内容是否为空，或者是否为一个空字符串。

<%

request.setAttribute("s2", "");

List list1=**new** ArrayList();

request.setAttribute("list1", list1);

%>

**空字符串：${empty s2 }**

**空的集合：${empty list1 }**

**值为null：${empty null }**

**找不到对应的值：${empty abc }**

**（5）读取请求参数值**

${param.username}等价于 request.getParameter(“username”);

${paramValues.interest}等价于request.getParameterValues(“interest”)，返回数组;

### 8、jstl标签

（1）jstl是什么

(java standard tag lib 即java标准标签库)

apache开发的一套通用的jsp标签，后来捐献给了sun，sun将其命名为jstl。

（2）变成步骤

Step1、导包。pom.xml文件中导入jstl1.2包

Step2、使用taglib指令导入要使用的标签。

uri属性：指定要导入的标签的命名空间。

prefix属性：指定命名空间的别名。

注：命名空间：是为了区分同名的元素而在元素前添加的一个限定。通常是一个域名。

<%@tagliburi=*"http://java.sun.com/jsp/jstl/core"*prefix=*"c"*%>

（3）核心标签

1）if标签

a、语法：

<c:if test=””>标签体</c:if>

b、注：当test属性值为true时，容器执行标签体的内容。

c、test属性值可以使用el表达式来赋值。

d、var属性指定绑定名，scope属性指定绑定的范围。

2）choose标签（servletDay11/e4.jsp）

a、语法

<c:choose>

<c:when test=””></c:when test=””>

<c:otherwise></c:otherwise>

</c:choose>

b、when可以出现1次或者多次，表示一个分支；otherwise可以出现0次或者1次，表示例外。

c、当test属性值为true是，执行标签体的内容，可以使用el表达式来给该属性赋值

3）forEach

遍历集合或者数组。

a、语法

<c:forEach items=“” var=”” varStatus=””></c:forEach>

b、items属性：用来指定要遍历的集合或者数组，可以使用el表达式来赋值。

c、var属性：用来指定绑定名。

注：绑定范围固定是pageContext。该标签每次从集合或者数组中取一个元素，然后将

钙元素绑定到pageContext上。

d、varStatus属性：用来指定绑定名。

注：绑定范围固定是pageContext。绑定值是一个特殊的对象，该对象提供了几个方法用来获得

当前遍历的状态。

getIndes():获得当前正在被遍历的元素的下标（从0开始）

getCount():获得当前是第几次遍历（从1开始）

（4）自定义标签（了解）（servletDay11/hello.jsp）

1）编程步骤

Step1、写一个java类，继承SimpleTagSuppor类，属于简单标签技术。

Step2、override doTag方法，在该方法里面，编写相应的处理逻辑。

Step3、在.tld文件里面描述标签。

2）标签的运行过程

Step1、容器依据标签的命名空间，找到标签的描述文件。

比如，以hello.jsp,容器依据命名空间(“http://tedu.cn/tag”)找到标签的

描述文件（mytag.tld）

Stpp2、容器依据标签的名字找到对应的标签类

比如，标签名字为”hello”,对应的标签类是”tag.HelloTag”.

Step3、容器调用标签类的构造器，创建其实例

Step4、容器将标签的属性值赋给标签对象对应的属性。

比如，将”hello Kitty”赋值给HelloTag对象对应的info属性。

（通过调用对应的set方法）

Step5、容器调用标签对象的doTag方法。

## 五、转发

1、什么是转发

一个web组件将未完成的处理交给另外一个web组件继续做。

注：web组件指的是jsp或servlet。

常见的场景是一个servlet获取处理结果之后，转交给一个jsp类展现。

2、如何转发

Step1、绑定数据到request对象上：request.setAttribute（String name，Object obj）；

注：name：绑定名，obj一般称之为绑定值。

Object request.getAttribute(String name);

Step2、获得转发器

RequestDispatcher rd=request.getRequestDispatcher(String uri);

注：（1）RequestDispatcher 是一个接口，该方法会返回一个符合该接口要求的对象，

我们一般称之为转发器。

（2）uri是转发的目的地，比如一个jsp的地址。

（3）转发的本质是一个web组件通过转发器，通知容器去调用另外一个web组件。

Step3、转发

rd.forward(request,response);

3、特点

（1）转发之后，浏览器地址栏的地址不变。

（2）转发的目的地地址是有限制的，要求是一个应用中的某个web组建的地址。、

4、转发与重定向区别

（1）能否共享request对象：

转发可以，而重定向不行。

原因：

1. 当请求到达容器时，容器会立即创建request，response对象，当响应发送完毕，容器会立即销毁这两个对象。也就是说，request和response的生存时间试一次请求与响应期间存在。

2）转发是一次请求，重定向是两次请求。

（2）浏览器地址栏的地址有无变化

转发无变化，而重定向有变化

（3）目的地有无限制

转发有限制（同一个应用），而重定向没有任何限制

登录：

流程

用户填写有户名和密码并提交，服务端检查是否有匹配条件的记录，如果有，则登陆成功，跳转到用户列表页面；否则登录失败，跳转到登录页面并提示用户。

步骤：

Step1、UserDao类

Public boolean find(String username,String pwd);

如果用户名已存在，返回一个完整的user对象（该对象包含了用户的所有信息），

如果用户名不存在，返回null。

Public User find(String username);

Step2、测试UserDao

Step3、login.jsp

Step4、LoginServlet,读取用户名和密码，调用UserDao的find方法

登录成功：重定向list

登录失败：转发到login.jsp

## 六、路径问题

1、什么是路径问题

链接地址、表单提交、重定向、转发都需要填写一个路径（或者也可以称之为地址）。

<a href=”addUser.jsp”>

<form action=”login”>

Response.sendRedirect(“”)

Request.getRequestDispatcher(“”)

2、相对路径：不以”/“开头的路径

3、绝对路径：以”/“开头的路径

4、如何写绝对路径

链接、表单提交、重定向从应用名开始写，转发从应用名之后开始写。

注：不要将应用名直接写在路径里面！

而应该使用以下方法获得实际部署时的应用名：

String request.getContextPath();

## 七、状态管理

### 1、什么是状态管理

将浏览器与web服务器之间多次交互当作一个整体来处理，并且将多次交互所涉及的

数据（即状态）保存下来。

### 2、如何进行状态管理

（1）将状态保存在浏览器端。

Cookie技术

（2）将状态保存在服务器端。

Session技术

### 3、Cookie

（1）什么是Cookie：服务器临时存放在浏览器端的少量数据。这些数据用来跟踪用户的状态。

（2）工作原理

当浏览器访问服务器时，服务器会将少量数据以set-cookie消息头的形式发送给浏览器。

浏览器会将这些数据保存下来。当浏览器再次访问服务器时，会将这些数据以cookie消息头

的形式发送给服务器。

（3）添加cookie

创建cookie对象：

Cookie c=new Cookie(String name,String value);

response.addCookie(c);

注：name为cookie名，value为cookie值，要求都是String。

（4）读取cookie

Cookie[] request.getCookies();

注：a、一个Cookie对象封装了一个Cookie中的所有数据。

b、该方法有可能返回null。

String cookie.getName();

String cookie.getValue();

（5）编码问题

a、什么是编码问题

cookie只能保存合法的ascii字符。对于中文需要编码处理，即将中文转换成合法的

ascii字符串的形式。

b、如何处理

String URLEncoder.encode(String str,String charset);

String URLDecoder.decode(String str,String charset);

c、建议

在添加cookie时，不管是否是中文，建议统一进行编码处理。

（6）生存时间问题

a、默认情况下，浏览器会把cookie保存在内存里面，浏览器只要不关闭，cookie就会一直存在。

浏览器关闭，则cookie会被删除。（浏览器关闭，进程结束，gc线程回收）

b、可以调用setMaxAge方法来设置cookie的生存时间。

cookie.setMaxAge(int seconds);

单位是秒。比如保存一年：setMaxAge(365\*24\*60\*60);

seconds值大于0：浏览器会将cookie保存在硬盘上（一般是以文件的形式存放），如果超过

指定的时间，cookie会被删除。

seconds值小于0：缺省值（即将cookie保存在内存里面）。

seconds值等于0：删除cookie

（7）cookie的路径问题

a、什么是cookie的路径问题？

浏览器访问服务器上的某个地址时，会比较该地址是否与cookie的路径匹配，只有匹配的

cookie才会被发送。

b、cookie默认的路径：等于添加该cookie的web组件的路径。

比如servletDay06/biz01/addCookie.jsp添加了一个cookie，则该cookie默认的路径就是

“servletDay06/biz01”。

c、匹配规则

如果请求路径是cookie路径或者其子路径，则该cookie会被发送。

比如：cookie路径是“servletDay06/biz01”，如果请求路径是

“servletDay06/findCookie.jsp” :no

“servletDay06/biz01/findCookie.jsp” :yes

“servletDay06/biz01/abc/findCookie.jsp”:yes

d、可以调用setPath方法类修改cookie的路径

比如：cookie.setPath(String path);

（8）实现一个简单的cookie工具（cookieUtil）扩展

（9）cookie的限制

a、可以被用户禁止。

b、不安全

注：对于敏感数据，不建议保存在cookie里，如果一定要保存，则一定要加密。

c、只能保存少量数据。大约4k。

d、cookie的数量有限制。

注：浏览器大约只能保存几百个cookie。

e、只能保存字符串

### 4、Session（会话）

（1）什么是Session

服务器端为了保存用户状态而创建的一个特殊的对象。

（2）工作原理

当浏览器访问服务器时，服务器创建session对象（该对象有一个唯一的id，一般称之为sessionId），

服务器会将sessionId以cookie的形式发送给浏览器。

当浏览器再次访问服务器时，会将sessionId发送过来，服务器利用sessionId找到对应的session对象。

（3）如何获得session对象

1）方式一

HttpSession request.getSession(boolean flag);

注：a、HttpSession是一个接口。

b、当flag为true时，服务器先查看请求当中是否有sessionId，如果没有，

则创建一个session对象；

如果有sessionId，则服务器依据该sessionId去查找对应的session对象，如果找到了，

则返回该对象，如果找不到，则创建一个新的session对象。

c、当flag为false时：

先查看请求当中是否有sessionId，如果没有，返回null；

如果有sessionId，则依据该sessionId去查找对应的session对象，如果找到了，则返回该对象，

如果找不到，返回null。

2）方式二

HttpSession s=request.getSession();

等价于request.getSession(true);

（4）绑定数据相关的方法

setAttribute(String name,Object obj);

Object getAttribute(String name);

注：如果找不到对应的值，返回null

removeAttribute(String name);解除绑定

（5）session超时

1）什么是session的超时

服务器会将空闲时间过长的session对象删除掉。

注：a、这样做的目的是为了节省内存空间

b、缺省的超时时间长度为30分钟。

2）修改超时时间

方式一：servers/web.xml文件Ctrl+F搜索session-config

<session-config>

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

</session-config>

方式二

setMaxInactiveInterval(int seconds)

两次请求之间的间隔不能超过指定的时间长度。如果超时，session对象会被删除。

（6）删除session

invalidate(); 把对象中的所有数据都删掉，使失效。session成为一个新的对象.

（7）session验证（案例）

Step1、登录成功后，在session对象上绑定一些数据，比如：

session.setAttribute(“user”,user);

Step2、当用户请求某个地址时，进行session验证

Object ont=session.getAttribute(“user”);

If(obj==null){

//没有登录，重定向到登录页面

Response.sendRedirect(“login.jsp”);

}

## 八、servlet处理请求路径问题

### 1、容器如何处理请求资源路径

比如，在浏览器地址输入http://ip:port/dar07-2/abc.html

Step1、容器会默认认为访问的是一个servlet，去查看web.xml，看是否有匹配的servlet。

注：有三种匹配方式

1. 精确匹配 <url-pattern>/abc.html</url-pattern>
2. 通配符匹配

使用“\*”来匹配零个或者多个任意的字符，比如

<url-pattern>/\*</url-pattern>

<url-pattern>/demo/\*</url-pattern>

c、后缀匹配

使用“\*.”开头，后接一个任意的后缀，比如

<url-pattern>\*.do</url-pattern>

以上配置会匹配所有以.do结尾的请求。

Step2、如果没有匹配的servlet，再去查找对应位置的文件。

### 2、一个Servlet处理多种请求

Step1、该servlet采用后缀匹配。

<servlet-mapping>

<servlet-name>actionServlet</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

Step2、分析请求路径，依据分析结果，进行不同处理。（分支结构）

## 九、servlet的生命周期

1、什么时servlet的生命周期

servlet容器如何创建servlet对象，如何对其进行初始化处理，如何调用其方法处理请求，

以及如何销毁该对象的整个过程。

2、生命周期分成哪几个阶段

（1）实例化

1）什么时实例化

容器调用servlet的构造器，创建servlet对象。

2）什么时候实例化

情形一：容器收到请求之后。

情形二：容器启动之后，立即创建。

注：需要额外的配置。

配制方法：

在部署文件web.xml中配置servlet时，

<servlet>

<servlet-name>someServlet</servlet-name>

<servlet-class>web.SomeServlet</servlet-class>

<!--

配置启动加载：即容器启动之后，会立即创建相应的实例。

注：值是一个大于等于零的整数，值越小，优先级越高（优先被创建）

-->

<load-on-startup>1</load-on-startup>

</servlet>

3）容器只会创建一个实例

（2）初始化

1）什么时初始化

容器调用servlet的init方法。该方法只会执行一次。

2）GenericServlet已经提供了init方法的实现。

将容器传递过来的ServletConfig对象保存下来了，并且提供了一个方法（getServletConfig）

来获得该对象。

3）如何实现自己的初始化处理逻辑

只需要override GenericServlet的init()方法(无参)。

4）初始化参数

Step1、配置初始化参数

<!-- 配置初始化参数 -->

<init-param>

<param-name>company</param-name>

<param-value>IBM</param-value>

</init-param>

Step2、使用ServletConfig提供的getInitParameter方法来读取初始化参数。

（3）就绪（调用）

1）什么时就绪

容器收到请求之后，调用servlet的service方法来处理请求。

2）HttpServlet已经提供了service方法的实现。

该方法依据请求类型调用对应的doXXX方法。

比如，get请求调用doGet方法，post 请求调用doPost方法（这两个方法只是抛出了一个异常）。

3）我们写servlet，可以有两个选择：

一是override HttpServlet的service方法，

二是override HttpServlet的doGet方法和doPost方法。

（4）销毁

1）什么是销毁

容器在删除servlet对象之前，会调用destroy方法。

注：该方法只会执行一次。

2）GenericServlet已经提供了destroy方法的实现。

注：该方法是一个空方法，什么都没做

3）如何实现自己的销毁处理逻辑

override GenericServlet的destroy方法。

3、相关的接口与类（了解）

1）Servlet接口

init(ServletConfig config)

service()

destroy()

2)GenericServlet抽象类

实现了Servlet接口中的部分方法（init和destroy）

3）HttpServlet抽象类

继承了GenericServlet，实现了service方法。

## 十、servlet上下文

1、什么是servlet上下文

容器启动之后，会为每一个web应用创建唯一的一个符合ServletContext接口要求的对象，

该对象一般我们称之为servlet上下文。

注：唯一性：一个web应用对应一个上下文。

持久性：只要求不关闭，应用没有被卸载，那么上下文就会一直存在。

2、如何获得Servlet上下文

GenericServlet提供了getServletContext()方法来获得上下文。

3、作用

（1）绑定数据

setAttribute

getAttribute

removeAttribute

注：request，session，servlet上下文都提供了绑定数据相关的方法，如何选取？

这三个对象生存时间：request < session < servlet上下文

在满足使用条件的情况下，尽量优先使用生命周期短的。

绑定到上下文的数据能被所有用户访问

绑定到session的数据，只有该session对应的用户访问。

（2）读取全局的初始化参数

a、什么是全局的初始化参数

可以被同一个应用中的所有组件共享的初始化参数。

b、如何使用

Step1、

<!-- 配置全局的初始化参数 -->

<context-param>

<param-name>company</param-name>

<param-value>IBM</param-value>

</context-param>

Step2、读取

ServletContext.getInitParameter(String paramName);

## 十一、过滤器、监听器

### 1、过滤器

（1）什么是过滤器

Servlet规范当中定义的一种特殊的组件，用来拦截servlet容器的调用过程。

注：容器收到请求之后，会先调用过滤器，再调用后续的其他组件（servlet）。

（2）如何写一个过滤器

Step1、写一个java类，实现Filter接口

Step2、在接口方法当中，实现拦截处理逻辑。

Step3、配置过滤器（web.xml）

**<filter>**

**<filter-name>commentFilter</filter-name>**

**<filter-class>web.CommentFilter</filter-class>**

**</filter>**

**<filter-mapping>**

**<filter-name>commentFilter</filter-name>**

**<url-pattern>/comment</url-pattern>**

**</filter-mapping>**

（3）过滤器的优先级

当有多个过滤器都符合拦截的要求，则依据配置的先后顺序来执行。

（4）初始化参数

Step1、配置初始化参数

Step2、调用FilterConfig的getInitParameter方法来读取初始化参数。

（5）过滤器的优点

a、在不修改原有代码的基础上，为系统增加一些新的简单的功能。

b、将多个模块相同的功能集中写在一个类里面，方便代码的维护。

### 2、监听器

（1）什么是监听器

servlet规范当中定义的一种特殊的组件。用于监听servlet容器产生的事件并进行相应的处理。

注：容器会产生两大类事件。

1）生命周期相关的事件。

容器创建或者销毁了request，session，servlet上下文是产生的事件。

2）绑定数据相关的事件。

调用了request，session，servlet上下文的setAttribute和removeAttribute方法是产生的事件。

（2）如何写一个监听器

Step1、写一个java类，依据监听的事件类型，选择实现对应的监听器接口。

注：比如，要监听session创建可销毁对应的事件，需要实现HttPSessionListener接口。

Step2、在接口方法当中，实现监听处理逻辑。

Step3、配置监听器（web.xml）。

## 十二、servlet线程安全问题

1、为什么说servlet会有线程安全问题？

（1）容器只会创建一个servlet实例。

（2）容器收到一个请求，就会启动一个线程，由这个线程来调用servlet实例来处理请求。

这样一来的话，就有可能有多个线程同时去调用某个servlet实例，就有可能发生线程安全问题

比如说这些线程要修改某个属性值。

2如何解决

使用synchronized对有可能产生线程安全问题的代码块枷锁。

注：加锁之后，会影响性能。

## 十三、MVC（Model View Controller 模型 视图 控制器）

1、什么是MVC

是一种软件架构思想，其核心思想是，要将数据处理与数据展现分开，按照这种思想，

我们一般将软件分成三种不同的模块，分别是：

**a、模型：**用来处理业务逻辑，即数据处理。

**b、视图**：数据展现，包括用户交互的界面。

**c、控制器**：用来协调模型和视图的。

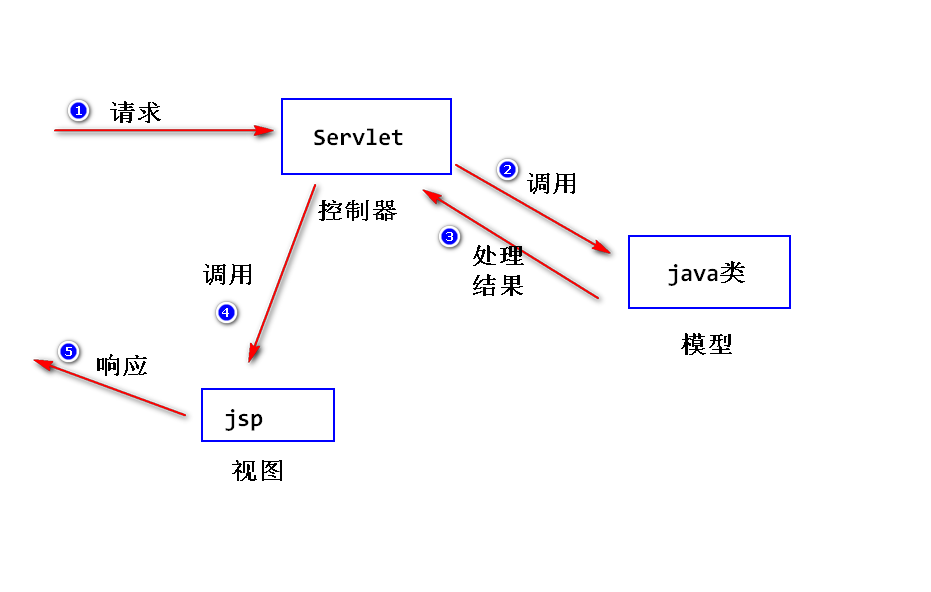
注：对协调的理解

用户通过视图将请求发送给控制器，有控制器选择合适的模型来处理；

模型返回的处理结果也要先交给控制器，由控制器选择合适的视图来展现处理结果。

2、如何使用MVC

在web开发当中，我们可以使用java类充当模型，使用Servlet充当控制器，使用jsp充当视图。



3、MVC的优点

（1）方便代码的维护：

模型或者视图发生了改变，不会彼此影响。

比如，计算bmi指数，可以使用不同的视图来展现处理结果，添加新的视图时，并不影响模型；

反过来，如果有新的模型来计算bmi指数，也并不影响视图。

（2）方便测试：

将业务逻辑写在Java类里面，可以直接测试，如果将业务逻辑写在servlet里面，

需要部署整个应用才能测试。

（3）方便分工协作

4、MVC缺点

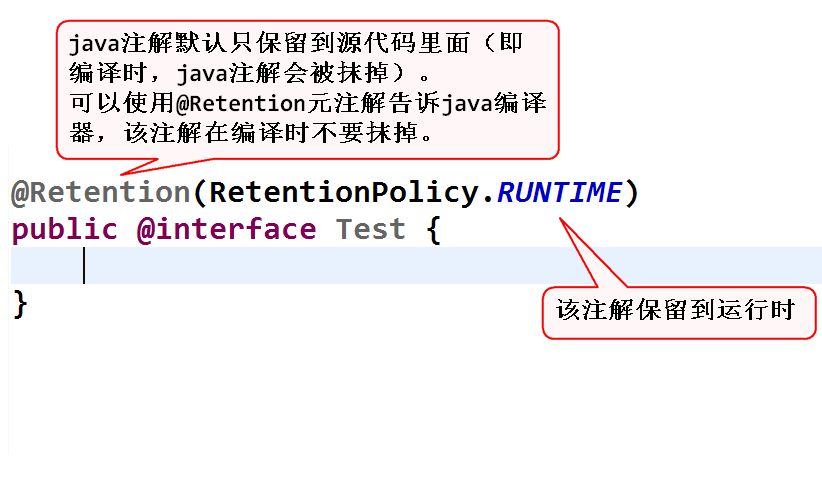
使用MVC，会增加代码量，也会增加软件设计的难度，相应会增加软件开发的成本。

所以对于相对比较简单的应用，不需要使用MVC。

二、实现一个简单的MVC框架

1、java反射基础

2、java注解

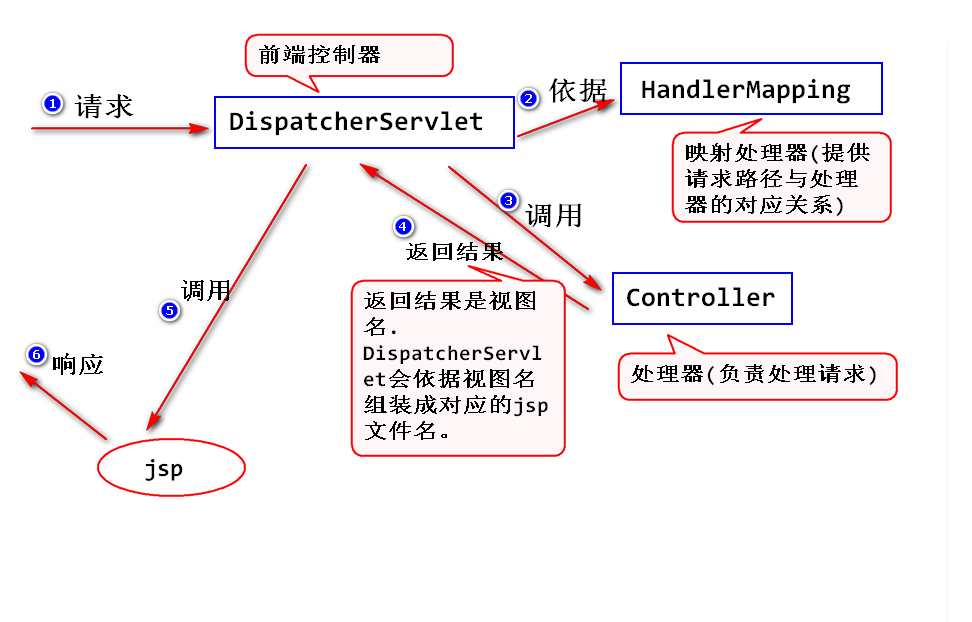


3、实现一个简单的MVC框架

（1）设计思路：

因为MVC中的C，即Controller（控制器）主要负责两件事，一是依据请求路径调用合适的模块来处理请求，另外，依据模型返回的处理结果调用合适的视图来展现处理结果。所以我们可以实现一个通用的控制器，该控制器可以通过读取配置文件或者java注解中的配置信息，来调用对应的模型或视图。

（2）框架



（3）使用MVC

Step1、导包dom4j

Step2、将smartmvc的核心包拷贝过来:base包

Step3、配置DispatcherServlet

Step4、添加smartmvc.xml配置文件。

Step5、添加处理器（如LoginController）

在该处理器里面，方法前添加@Requestmapping注解。

方法的返回值是视图名

step6.添加jsp。

注意jsp文件名应该符合 "/WEB-INF/" + 视图名 + ".jsp"。

step7.在smartmvc.xml配置文件中，添加处理器的配置。

# 第九章、Spring框架

一、Spring介绍

1、什么是Spring

Spring是一个开源的轻量级的应用开发框架，目的是用于简化企业级应用程序开发 ，降低侵入性。

Spring提供的IOC和AOP功能，可以将组件的耦合度降至最低（解耦），便于系统日后的维护和升级。

Spring为系统提供了一个整体的解决方案，开发者可以利用它本身提供的功能外，也可以与第三方框架

和技术整合应用，

## 一、Spring容器

### 1、Java Bean

符合一定规范的java对象

类的定义符合如下规则的java对象称为Java Bean

（1）必须有包

（2）必须有无参构造器

（3）必须实现序列化接口

（4）有getXXX,setXXX方法声明的Bean属性。

--Bean属性 不是对象属性。

--boolean类型的 get属性方法，可以声明为isXXX方法

### 2、Spring IOC

用于管理（创建，修改属性等）java bean对象

（1）Spring IOC 可以创建Java Bean对象

（2）Spring IOC 可以修改Bean属性

（3）Spring IOC 可以作为Java Bean 的工厂使用

### 3、Spring可以管理Bean的属性

Bean属性注入：修改Bean属性。实际上是调用了setXXX方法，修改了Bean属性

### 4、Spring创建对象的3种方式

Spring帮助创建JavaBean对象，Spring支持3种对象创建方式

（1）利用构造器创建JavaBean

（2）利用静态“工厂方法“”创建对象

（3）利用动态工厂方法创建对象

工厂方法：简单理解就是利用方法创建对象。将对象复杂的创建过程封装到方法中，

使用时调用方法就重用了复杂的对象创建过程，简洁的得到对象。

如静工厂方法

Connection conn= DBUtils.getConnection();

Calendar rightNow=Calendar.getInstance();

动态工厂方法

Statement st=conn.createStatement();

Date time=cal.getTime();

### 5、Bean 的作用域

Spring在默认情况下，其中Bean都是单例的。

单例：在软件中一个永远唯一的对象，称为单例

### 6、Bean的生命周期管理

Spring中的Bean对象，由Spring控制对象的创建和销毁，这个过程称为Spring

中Bean对象的生命周期管理。

（1）单例对象 singleton（单例）

创建：在Spring容器初始化时候，创建单例对象，如果设置了init-method属性，则在创建对象以后调用初 始化方法。

使用：每次调用getBean方法时，返回的都是同一个对象。

销毁：在Spring容器关闭时，Spring会自动销毁单例对象，如果设定了destroy-method属性，则会在销毁之 前执行销毁方法

（2）多例对象 prototype（原型）

创建：在调用getBean方法时，创建对象，如果设置了init-method属性，则在创建对象以后调用初 始化方法。

使用：每次调用getBean方法时，都会返回一个新对象。

销毁：Spring不管。也不会调用destroy-method。

### 7、Bean生命周期与延迟实例化

问题：单例对象默认情况下在容器启动时立即初始化。

如果这些对象内存耗用高，则启动会很慢。

解决办法：

对于内存占用高，使用少的对象，可以设置延迟（懒惰，按需）实例化。

在bean标签上使用懒惰属性设置即可。

设置了懒惰属性为true：

1）容器启动时，bean不实例化

2）在第一次getBean时实例化

3）是单例bean对象

### 8、IOC

Inversion of Control,控制反转。

IOC指程序中对象的获取方式发生反转，由最初的new方式创建，转变为有第三方框架创建、

### 9、DI

注入（依赖注入）

降低了对象之间的耦合度

## 二、Spring MVC

### 1、搭建Spring MVC

（1）导入Spring-mvc包

（2）在web.xml中配置前端控制器

（3）为前端控制器提供spring配置文件

（4）在Spring配置文件中配置Spring MVC的组件。

2、ModelAndView

Spring MVC认为web控制器处理结果后，需要将数据（Model）发送到视图（View）显示，

ModelAndView就是一个封装显示数据和视图的Java Bean对象。

2、Spring MVC：将userDao注入控制器

Web应用程序框架，封装web应用中常用功能，只需要按照Spring MVC约定编写代码，

就可以快速实现web应用。

（1）搭建Spring MVC项目

（2）导入jdbc,dbcp,jstl包

（3）复制dao层

-dao/UserDao.java

-entity/User.java

-util/Dbutil.java

-jdbc.properties

（4）UserDao.java添加注解@Repository

-目的：UserDaoo对象有Spring容器管理

（5）添加applicationContext.xml用于扫描dao包中注解

（6）导入JUnit包

（7）创建测试案例，测试UserDao对象

（8）复制视图jsp到webapp

-css/\*

-img/\*

-WEB-INF/\*.jsp

（9）删除header.jsp中的自定义标签

（10）开发控制器UserController

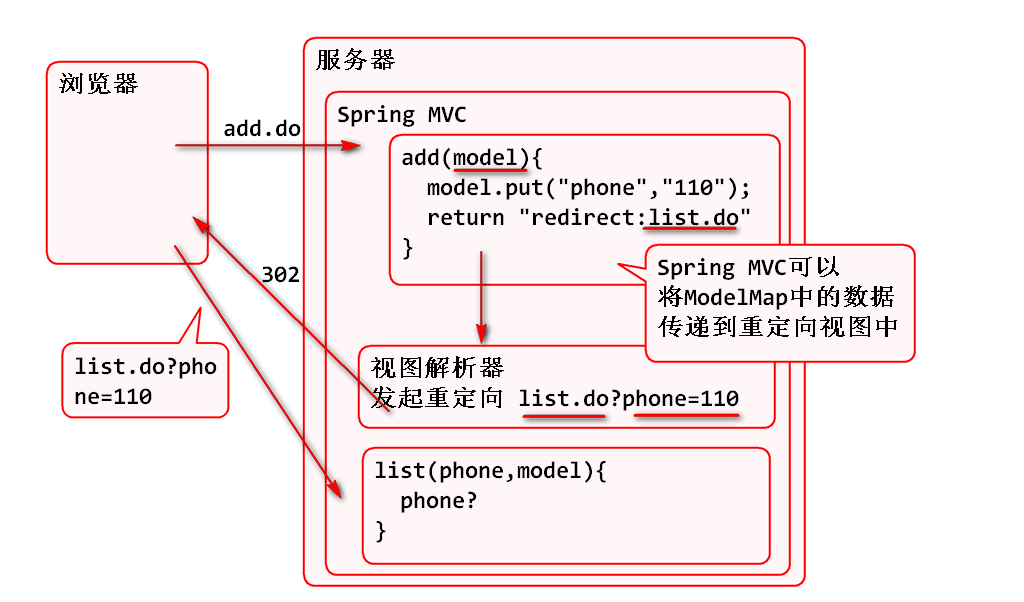
（11）添加Spring容器的初始化监听器web.xml:

（12）

## 三、Spring

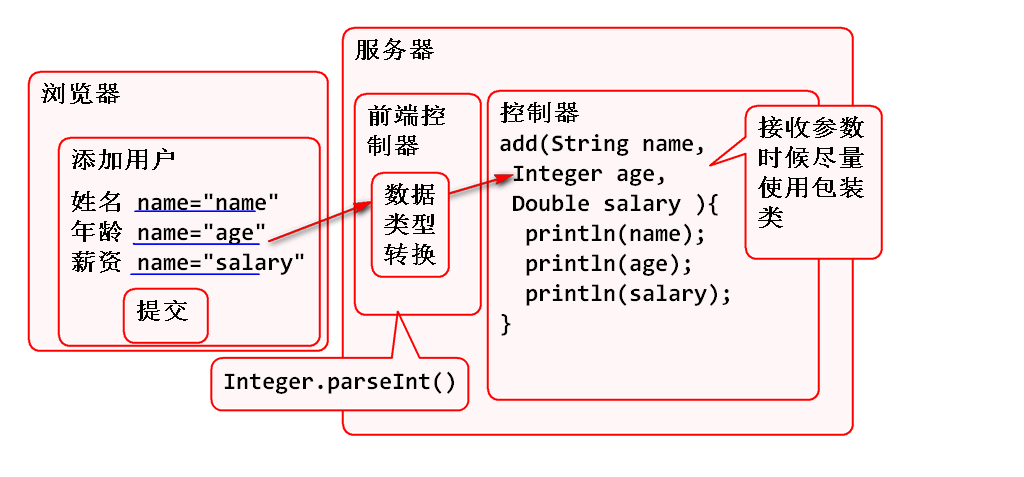
## Spring MVC 重定向参数传递

Spring MVC 控制器向视图界面传递参数时候, 不仅能够向转发视图传递参数, 而且还能向重定向视图传递数据:



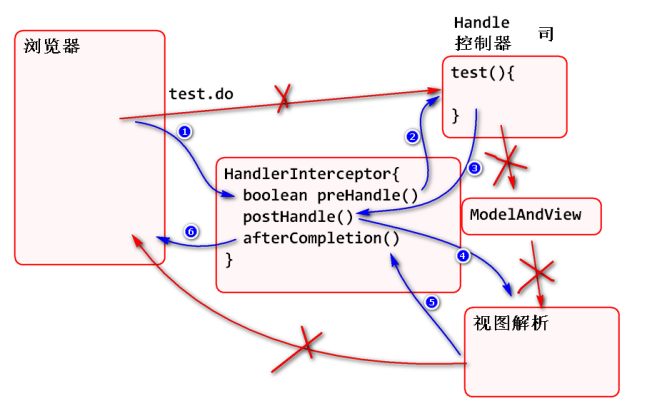
## Spring MVC 数据类型转换

Spring MVC 在接收浏览器参数时候, 可以进行自动数据类型转换, 为了更加方便的接收参数, 建议使用包装类接收参数:



## Spring MVC 拦截器(Interceptor)

Spring MVC 提供了与 Servlet Filter 类似的拦截器功能, 其执行原理为:



演示案例:

1、编写拦截器类:

@Component

public class DemoInterceptor implements HandlerInterceptor {

// 在控制器之前执行

public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)

throws Exception {

System.out.println("preHandle");

return true;

}

//在控制器之后执行

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {

System.out.println("postHandle");

}

//视图处理以后执行

public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)

throws Exception {

System.out.println("afterCompletion");

}

}

2、在spring-mvc.xml 配置拦截器

<!-- interceptor 拦截器在 spring-mvc中配置拦截器的拦截位置 -->

<mvc:interceptors>

<mvc:interceptor>

<!-- mapping 用于设定拦截器的拦截位置 -->

<mvc:mapping path="/test.do"/>

<!-- 引用@Component注解声明的Bean demoInterceptor -->

<ref bean="demoInterceptor"/>

</mvc:interceptor>

</mvc:interceptors>

3、添加控制器方法:

@RequestMapping("/test.do")

public String test(){

System.out.println("test()");

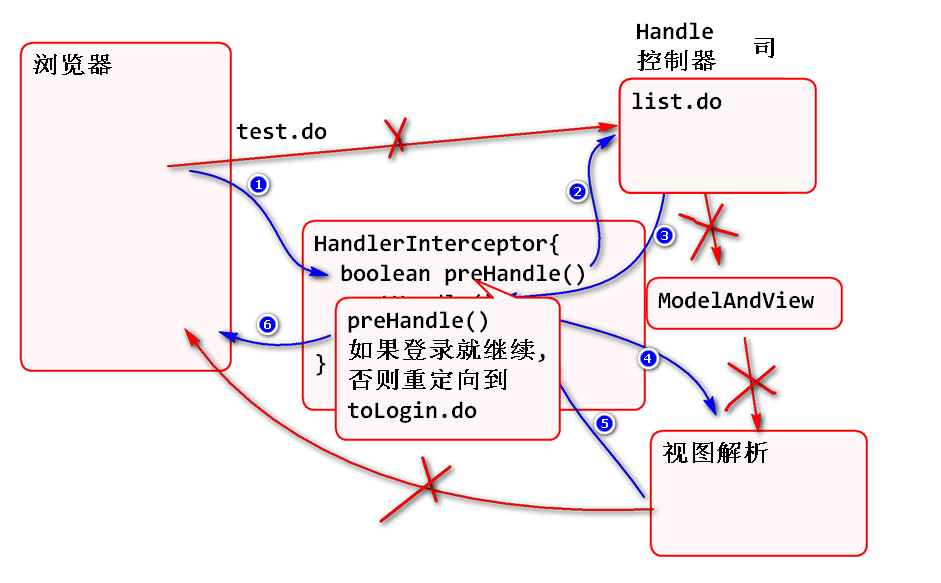
return "test";

}

4、测试

http://localhost:8088/spring-mvc-d2/test.do

利用拦截器可以解决权限管理问题:



步骤:

1、编写拦截器:

//继承HandlerInterceptorAdapter的好处是 可以减少实现HandlerInterceptor接口时候的空方法.

//称为: 适配器(Adapter)模式!

@Component

public class AccessInterceptor

extends HandlerInterceptorAdapter{

@Override

public boolean preHandle(

HttpServletRequest request,

HttpServletResponse response,

Object handler)

throws Exception {

//检查用户是否登录

User user=(User)request.getSession().getAttribute("loginUser");

if(user==null){

System.out.println("没有登录");

String path="toLogin.do";

response.sendRedirect(path);

return false;

}

return true;

}

}

2、配置拦截器 spring-mvc.xml

<!-- 配置权限拦截器 -->

<mvc:interceptor>

<mvc:mapping path="/\*.do"/>

<!-- exclude-mapping配置放过的请求 -->

<mvc:exclude-mapping path="/toLogin.do"/>

<mvc:exclude-mapping path="/login.do"/>

<ref bean="accessInterceptor"/>

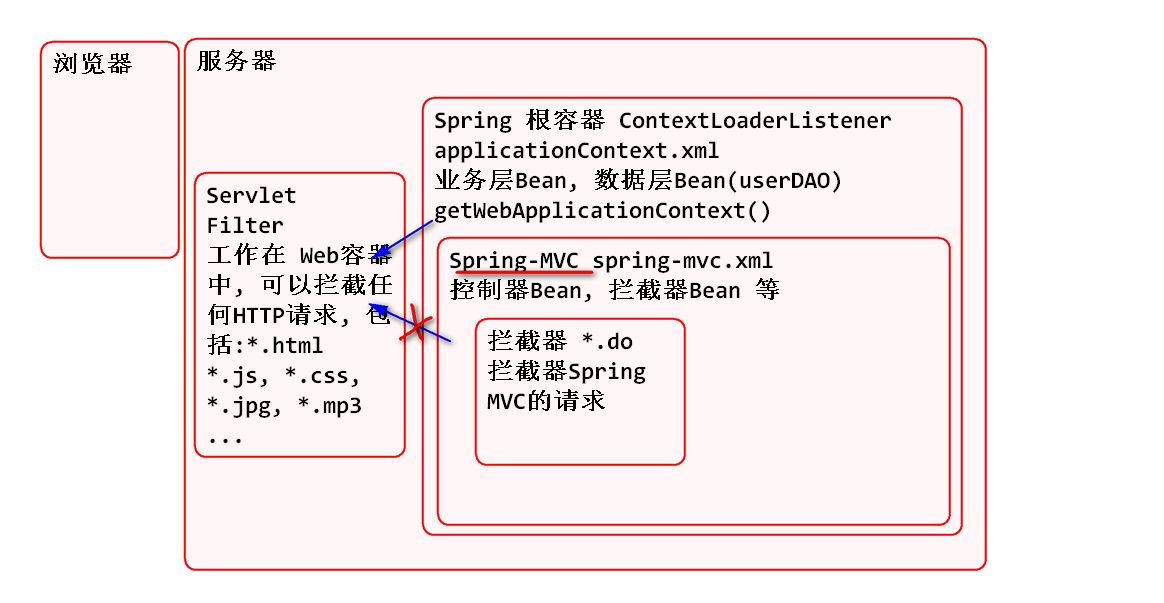
</mvc:interceptor>

3、部署测试, 没有登录就会转到登录页

## Servlet Filter 与 拦截器

Servlet Filter 是工作在Web容器中, 其拦截范围非常宽, 任何HTTP请求都可以被拦截处理

Spring MVC 拦截器, 工作在Spring MVC容器中, 只能拦截Spring MVC的请求.



拦截\*.css \*.js的案例:

1、编写过滤器

public class DemoFilter implements Filter{

public void init(FilterConfig config)

throws ServletException {

}

public void doFilter(ServletRequest request,

ServletResponse response,

FilterChain chain)

throws IOException, ServletException {

HttpServletRequest req=

(HttpServletRequest)request;

StringBuffer url=req.getRequestURL();

System.out.println(url);

chain.doFilter(request, response);

}

public void destroy() {

}

}

2、配置:

<filter>

<display-name>DemoFilter</display-name>

<filter-name>DemoFilter</filter-name>

<filter-class>controller.DemoFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>DemoFilter</filter-name>

<url-pattern>\*.css</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>DemoFilter</filter-name>

<url-pattern>\*.js</url-pattern>

</filter-mapping>

3、测试, 需要清理浏览器缓存.

## Spring MVC 异常处理

软件在运行过程中很有可能出现异常, 而在控制器中进行大量的try catch编码显然是非常繁琐的, Spring MVC提供了异常方案, 可以大大简化异常的处理.

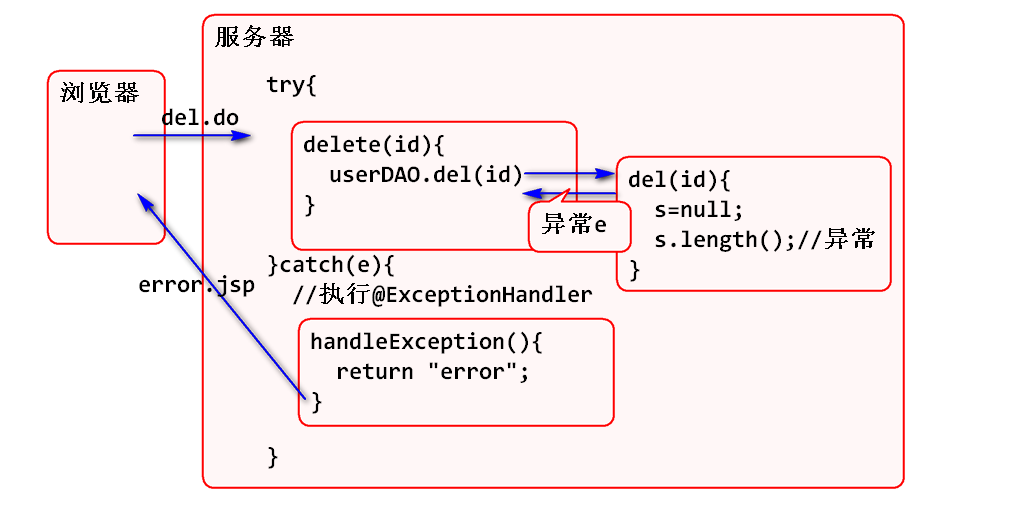
Spring MVC 中可以使用多种异常处理

1、在控制器中直接使用try ... catch 语句进行处理.

2、在控制器中使用 @ExceptionHandler 注解进行异常处理.

3、在Spring MVC 配置文件中进行异常处理.

目前最常用的方式是使用 @ExceptionHandler 注解进行异常处理



案例步骤:

1、在控制器中添加异常处理方法

//当控制器出现 Exception 类型异常时候会执行 handleException 方法利用 @ExceptionHandler 注解处理异常的

//好处是可以避免每个控制器方法中都写 try...catch 来处理异常.在 异常处理方法中不能使用 ModelMap 参数!

//@param e 控制器方法中出现的异常,@return 视图名

@ExceptionHandler(Exception.class)

public String handleException(

Exception e,

HttpServletRequest request){

e.printStackTrace();

request.setAttribute(

"msg", "异常:"+e.getMessage());

return "error";

}

注意: 在异常处理方法中不能注入 ModelMap 对象!

2、为软件添加测试异常:

// 删除用户

public void delete(int id){

Connection conn = null;

PreparedStatement ps = null;

try {

String s = null;

s.length(); //引起异常

conn = DBUtils.getConn();

String sql = "DELETE FROM t\_user WHERE id=?";

ps = conn.prepareStatement(sql);

ps.setInt(1, id);

ps.executeUpdate();

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException(e);

}finally{

DBUtils.close(conn, ps, null);

}

}

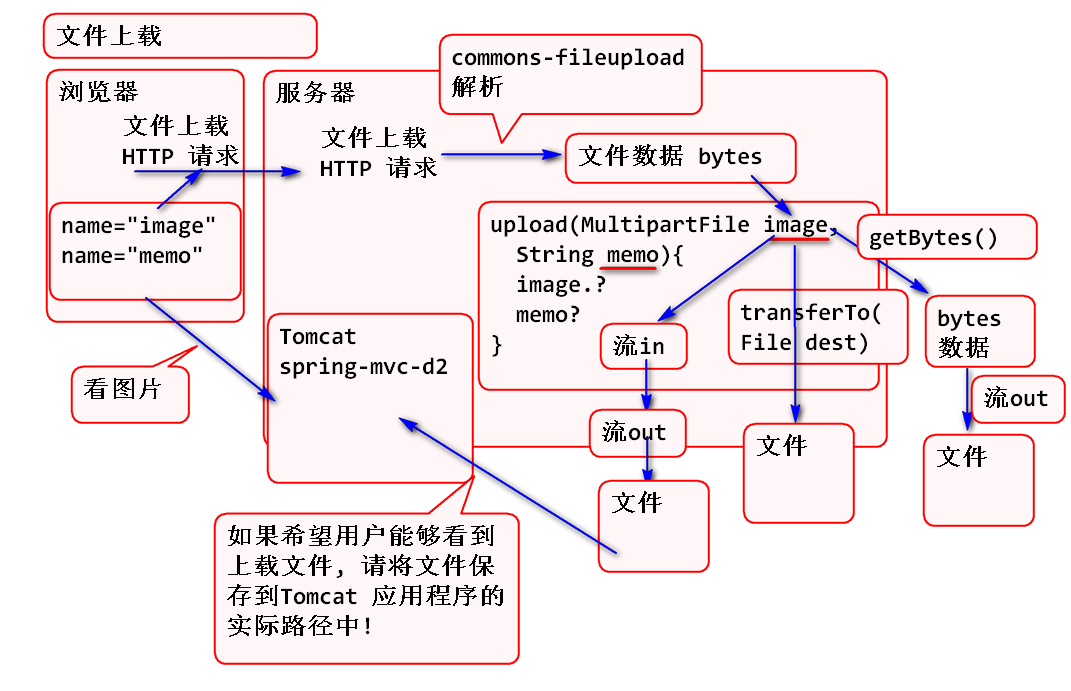
String s = null; s.length(); 会引发空指针异常

3、测试, 在执行删除功能时候, 会跳转到异常处理页面.

## 文件上载(上传 upload)

HTTP协议支持文件上载功能, 具体内容请参看 [http://doc.tedu.cn/rfc/rfc1867.txt](http://doc.tedu.cn/rfc/rfc1867.txt" \o "RFC1867)

原理:



浏览器和Spring MVC按照这个 RFC1867 标准实现文件上载, 具体步骤

### 1、单个文件上传

1、客户端

使用form 标签并且使用 method="post" 和 enctype="multipart/form-data"

使用 input type=file 选择文件

2、服务器端

导入 commons-fileupload

在Spring中配置 multipartResolver 组件

在控制器利用 MultipartFile 接收文件

案例步骤:

1、编写客户端 upload.html

<h1>文件上载</h1>

<!-- 文件上载表单必须有 method="post" 和enctype="multipart/form-data"-->

<formaction=*"upload.do"*method=*"post"*enctype=*"multipart/form-data"*>

<div>

<label>选择文件</label>

<!-- input type="file" 用于选择上载文件 -->

<inputtype=*"file"*name=*"image"*>

</div>

<div>

<label>说明</label>

<inputtype=*"text"*name=*"memo"*>

</div>

<inputtype=*"submit"*value=*"上传"*>

</form>

提示: 利用浏览器的F12可以检查文件上载的http消息

2、在项目中导入 commons-fileupload 组件

<!-- 上传文件 -->

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.3</version>

</dependency>

3、在spring-mvc.xml 中配置上载解析器

<!-- 配置文件上载解析器 -->

<beanid=*"multipartResolver"*

class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<!--配置上载编码，否则中文文件名会乱码 -->

<propertyname=*"defaultEncoding"*value=*"utf-8"*></property>

<!-- one of the properties available; the maximum file size in bytes -->

<!-- 最大上载字节总数 -->

<propertyname=*"maxUploadSize"*value=*"10000000"*/>

</bean>

4、编写控制器方法:

/\*文件上载

\* method=RequestMethod.POST 限定用户提交的请求必须是post请求，

\* 如果不是post请求，则会出现500错误

\*/

@RequestMapping(value="/upload.do",method=RequestMethod.***POST***)

**public** String upload(

MultipartFile image,

String memo,

HttpServletRequest request) **throws** Exception{

//文件相关信息从image对象中获取获取文件名

String filename=image.getOriginalFilename();

//获取文件长度

**long**length=image.getSize();

//获取文件中全部的数据

**byte**[] data=image.getBytes();

//获取<input name="?">name属性值

String name=image.getName();

System.***out***.println(filename);

System.***out***.println(name);

System.***out***.println(length);

System.***out***.println(memo);

// //保存到文件/home/soft01/files

// File dir=new File("D:/files");

//获取Tomcat中实际的路径：servletContext.getRealPath();

String path="/files";

path=request.getServletContext().getRealPath(path);

System.***out***.println(path);

//将文件是保存到Tomcat的实际路径中

File dir=**new** File(path);

**if**(!dir.exists()) dir.mkdir();

File file=**new** File(dir,filename);

// //保存到文件

// image.transferTo(file);

//利用流保存文件

InputStream in=image.getInputStream();

FileOutputStream out=**new** FileOutputStream(file);

**int**b;

**while**((b=in.read())!=-1){

out.write(b);

}

in.close();

out.close();

**return**"ok";

}

5、测试....

### 2、多文件上传

（1）实现多文件上传要在选择文件节点添加属性multiple=*"multiple"*

<div>

<label>选择文件</label>

<!-- input type="file" 用于选择上载文件 -->

<inputtype=*"file"*multiple=*"multiple"*name=*"image"*>

</div>

（2）控制器需传参MultipartHttpServletRequest（接口：提供了List<MultipartFile>接收多个文件）

@RequestMapping(value="/uploads.do",method=RequestMethod.***POST***)

**public** String uploads(MultipartHttpServletRequest req) **throws** Exception{

List<MultipartFile>files=req.getFiles("image");

System.***out***.println(files);

//获取<input name="?">name属性值

**for** (MultipartFile image : files) {

String filename=image.getOriginalFilename();

**long**length=image.getSize();

**byte**[] data=image.getBytes();

//获取<input name="?">name属性值

String name=image.getName();

//保存到文件/home/soft01/files

File dir=**new** File("D:/files");

//如果不存在则创建

**if**(!dir.exists()) dir.mkdir();

File file=**new** File(dir,filename);

// //保存到文件

// image.transferTo(file);

//利用流保存文件

InputStream in=image.getInputStream();

FileOutputStream out=**new** FileOutputStream(file);

**int**b;

**while**((b=in.read())!=-1){

out.write(b);

}

in.close();

out.close();

}

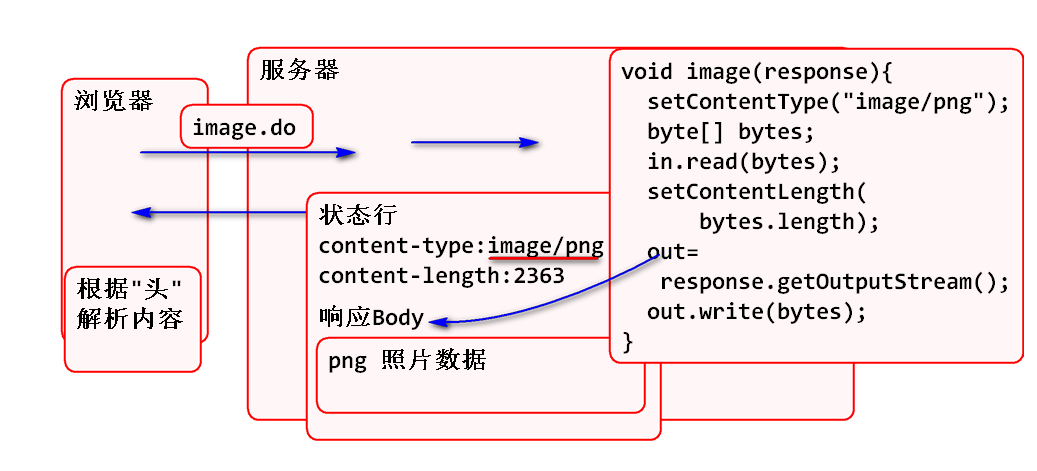
**return**"ok";

}

## Spring MVC 文件下载

### 1、直接下载图片

Spring MVC 对文件下载做了支持, 原理为:



使用步骤:

1. 编写控制器:

/\*

\* 利用Spring 下载文件，在页面显示

\*/

@RequestMapping("image.do")

**publicvoid** image1(HttpServletResponse response) **throws** Exception{

//读取照片数据

String path="D:/files/1.png";

**byte**[] bytes = readFile(path);

//设置响应头

response.setContentType("image/png");

response.setContentLength(bytes.length);

//设置响应正文（Message Body）

OutputStream out=response.getOutputStream();

out.write(bytes);

out.close();

}

**privatebyte**[] readFile(String path) **throws** FileNotFoundException, IOException {

File file=**new** File(path);

**byte**[] bytes=**newbyte**[(**int**)file.length()];

FileInputStream in=**new** FileInputStream(file);

in.read(bytes);

in.close();

**return**bytes;

}

注：配置拦截器, 放过image.do请求

2. 利用浏览器测试:

http://localhost:8088/spring-mvc-d2/image.do

### 2、@ResponseBody注解

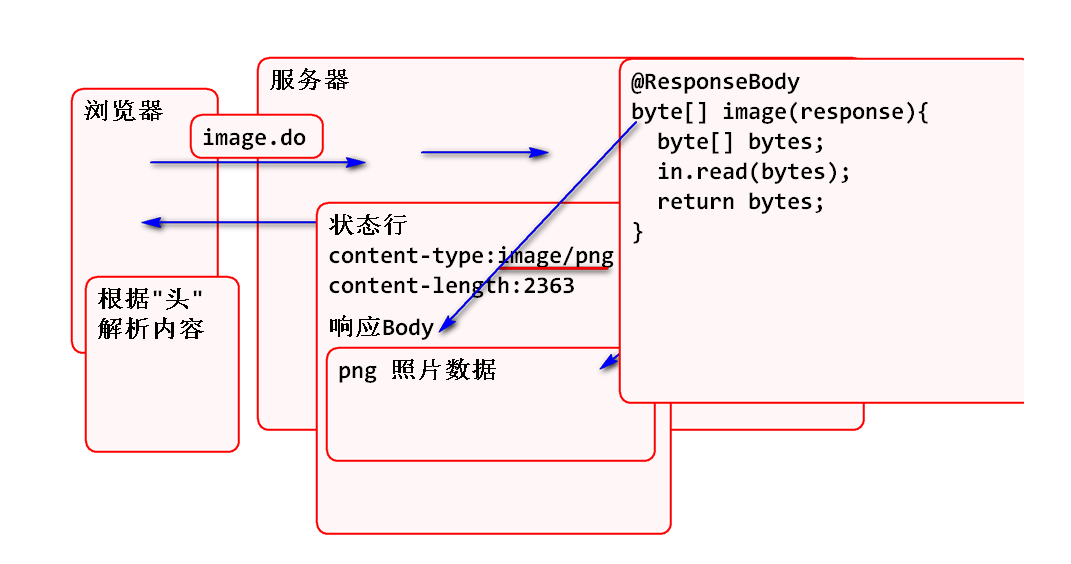
Spring MVC 提供了@ResponseBody注解, 这个是一个自动处理响应Body的注解:

1. 当返回值是 byte[] 时候会自动设置ContentLength, 自动填充到 响应body中.

2. 当返回值是Java Bean 时候自动转换为 JSON 字符串(后面讲解)

3. 还有其他功能 ...

原理:



案例:

1. 编写控制器:

/\*利用Spring 下载显示一张照片\*/

@RequestMapping(value="image2.do",produces="image/png")

@ResponseBody

**publicbyte**[] image2() **throws** Exception{

//读取照片数据到bytes数组中

String path="D:/files/1.png";

**byte**[] bytes = readFile(path);

//设置响应头

//设置响应正文（Message Body）

**return**bytes;

}

**privatebyte**[] readFile(String path) **throws** FileNotFoundException, IOException {

File file=**new** File(path);

**byte**[] bytes=**newbyte**[(**int**)file.length()];

FileInputStream in=**new** FileInputStream(file);

in.read(bytes);

in.close();

**return**bytes;

}

注：配置拦截器, 放过image2.do请求

2. 编写网页 download.html:

<styletype=*"text/css"*>

*.png*{width:*80px*}

</style>

<body>

<h1>下载演示</h1>

<h2>显示照片</h2>

<p>

<imgclass=*"png"*alt=*""*src=*"image.do"*>

<imgclass=*"png"*alt=*""*src=*"image2.do"*>

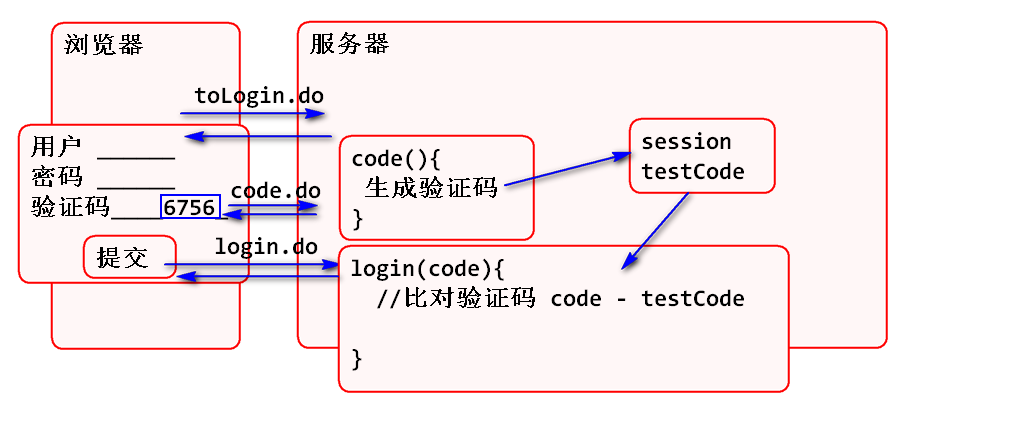
</p>

</body>

3. 测试图片的显示

## ## 验证码功能

网站经常利用验证码功能, 避免网站被机器自动访问. 其原理为:



实现步骤:

1. 利用控制器生成验证码图片, 并且将验证码保存到Session中.

2. 更新login.jsp 显示验证码

3. 更新登录控制器, 验证处理验证码.

案例:

1. 添加生成验证码的控制器

@RequestMapping(value="/code.do",produces="image/png")

@ResponseBody

**publicbyte**[] code(HttpSession session) **throws** IOException{

String code=code(4);

//将验证码保存到Session中

session.setAttribute("testCode", code);

**byte**[] bytes=createPng(code);

**return**bytes;

}

/\*

\* 生成验证码字符串

\* @param n 验证码位数

\* @return 随机的验证码

\*/

**static** String *c*="abcdefhjkpqrdsyx345678ABCDEFH";

**public** String code(**int**n){

**char**[] code=**newchar**[n];

Random r=**new** Random();

**for**(**int**i=0;i<n;i++){

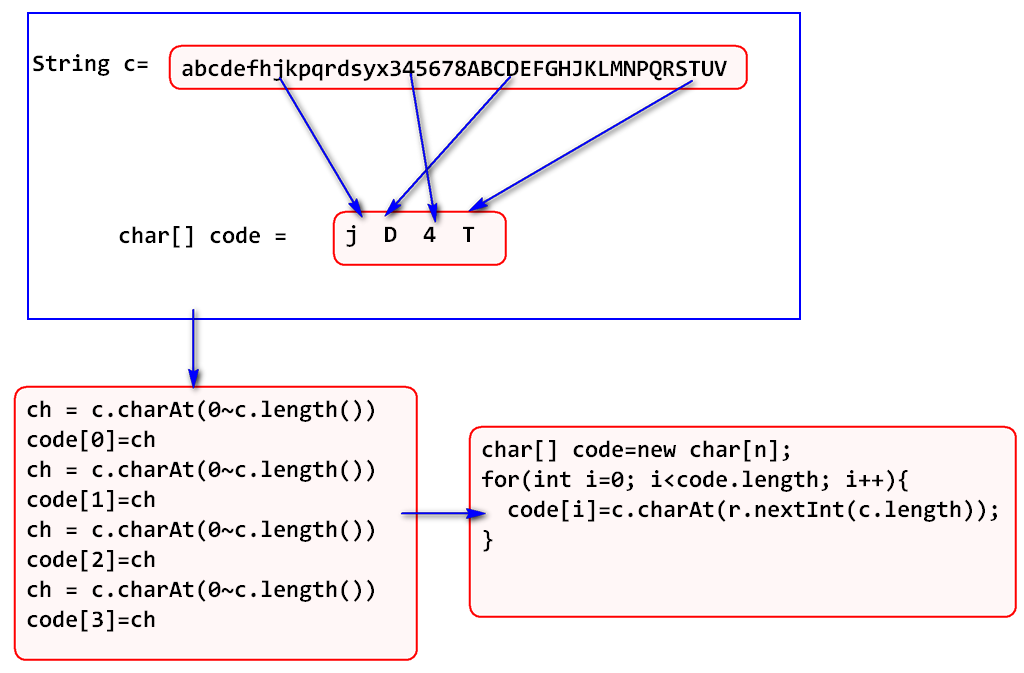
code[i]=*c*.charAt(r.nextInt(*c*.length()));

}

**returnnew** String(code);

}

验证码算法原理:



/\*

\* 生成验证码图片

\*/

**privatebyte**[] createPng(String code) **throws** IOException {

BufferedImage img=**new** BufferedImage(100,40,BufferedImage.***TYPE\_3BYTE\_BGR***);

img.setRGB(0, 0, 0xf0000);

img.setRGB(50, 20, 0xffff00);

Random r=**new** Random();

**for**(**int**i=0;i<100;i++){

**int**x=r.nextInt(100);

**int**y=r.nextInt(40);

**int**color=r.nextInt(0xffffff);

img.setRGB(x, y, color);

}

Graphics2D g=img.createGraphics();

//设置字体

Font font=**new** Font(Font.***SANS\_SERIF***,Font.***PLAIN***,30);

g.setFont(font);

//设置颜色

g.setColor(**new** Color(r.nextInt(0xffffff)));

g.drawString(code, 10, 30);

ByteArrayOutputStream out=**new** ByteArrayOutputStream();

//将图片数据倒入“酱油瓶”

ImageIO.*write*(img, "png", out);

//“盖上盖子”

out.close();

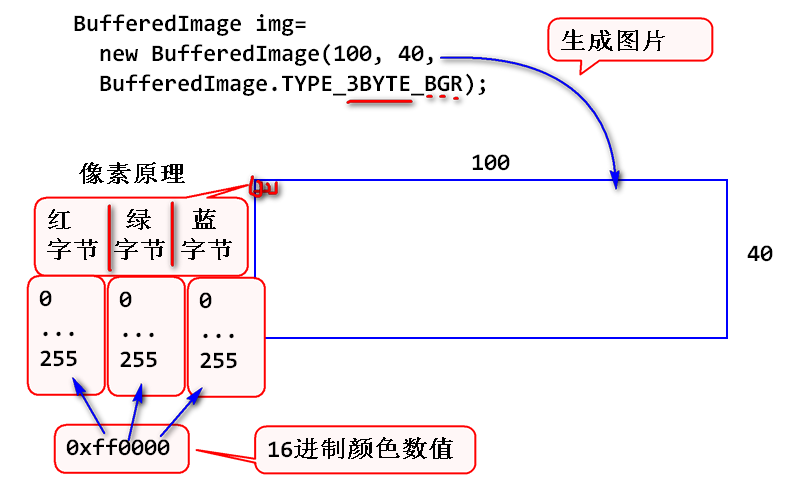
//将“酱油瓶”中的数据（酱油）倒出来。

**byte**[] bytes=out.toByteArray();

**return**bytes;

}

注：验证码图片绘图原理:



2. 更新login.jsp页面显示验证码:

<tr>

<tdvalign=*"middle"*align=*"right"*>验证码:</td>

<tdvalign=*"middle"*align=*"left"*>

<inputtype=*"text"*class=*"inputgri"*name=*"code"*/>

<imgonclick="this.src='code.do?'+new Date().getTime();"src=*"code.do"*>

</td>

</tr>

3. 更新登录控制器, 处理验证码:

/\*处理登录请求的控制器方法 \*/

@RequestMapping("/login.do")

**public** String login(

@ModelAttribute(name="username") String username,

String pwd,

String code,

ModelMap model,

HttpSession session,

HttpServletRequest request,

HttpServletResponse response){

String testCode=(String)session.getAttribute("testCode");

//添加验证 equalsIgnoreCase忽略大小写并相等

**if**(code==**null** || !code.equalsIgnoreCase(testCode)){

model.put("login\_fialed", "验证码错误");

**return**"login";

}

**......**

}

4. 在拦截器中放过验证码的 url

<mvc:exclude-mapping path="/code.do"/>

5. 测试...

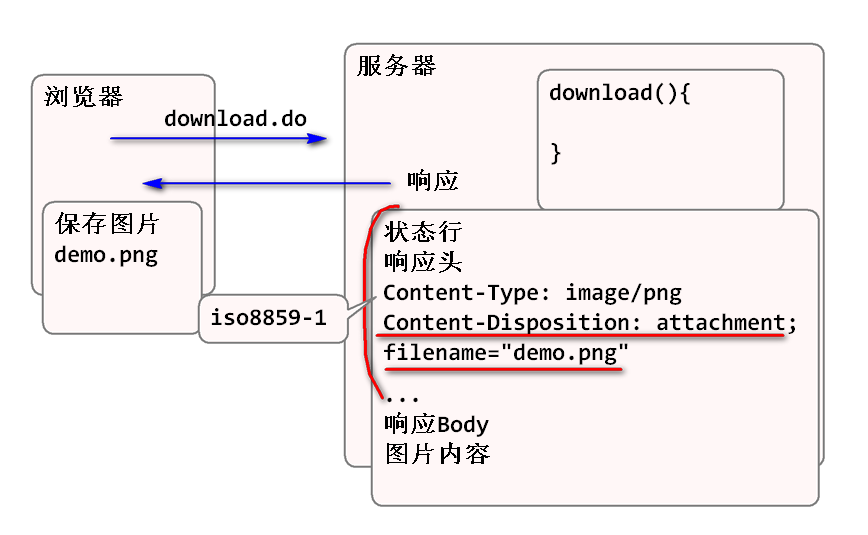
## ## 下载保存图片

### 1、下载图片

图片默认情况下, 浏览器进行内联(inline)显示, 如果需要实现下载功能, 则需要设置一个响应头 Content-Disposition

具体请参看: http://doc.tedu.cn/rfc/rfc2616.txt 的 19.5.1 章节

下载图片的原理是:



案例:

1. 编写控制器:

@RequestMapping(value="/download.do",produces="image/png")

@ResponseBody

**publicbyte**[] download(HttpServletResponse response) **throws** IOException{

String file=URLEncoder.*encode*("实例1.png","UTF-8");

response.setHeader("Content-Disposition", "attachment;filename=\"fname.ext\""+file+"\"");

System.***out***.println("\"");

**byte**[] bytes=readFile("D:/files/1.png");

**return**bytes;

}

2. 编写客户端

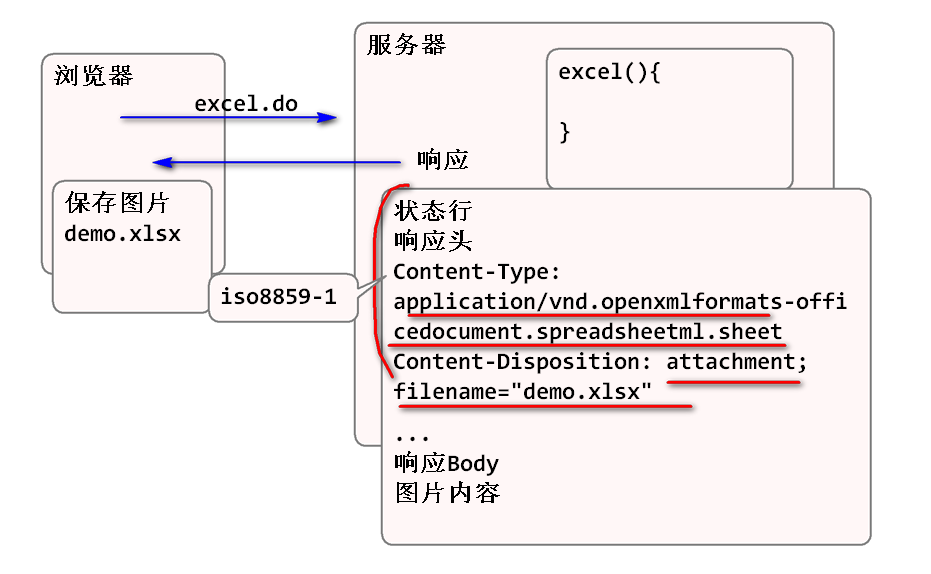
<p>

<ahref=*"download.do"*>下载图片</a>

</p>

3. 测试

**2、下载Excel**:



步骤:

1. 导入Excel API

<!-- POI是Apache组织提供的开源 Office API -->

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-ooxml</artifactId>

<version>3.17</version>

</dependency>

2. 编写控制器

/\*下载保存excel表\*/

@RequestMapping(value="excel.do",

produces="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet")

@ResponseBody

**publicbyte**[] excel(HttpServletResponse response) **throws** IOException{

String file=URLEncoder.*encode*("实例1.xlsx","UTF-8");

response.setHeader("Content-Disposition", "attachment;filename=\""+file+"\"");

**byte**[] bytes=createExcel();

**return**bytes;

}

**privatebyte**[] createExcel() **throws** IOException {

//创建工作簿（就是Excel文件）

XSSFWorkbook workbook=**new** XSSFWorkbook();

//在工作簿中创建工作表

XSSFSheet sheet=workbook.createSheet("演示");

//在工作表中创建行，参数是行号0，1,2,3,4，........

XSSFRow row=sheet.createRow(0);

//在行中创建格子，参数是列号0，1,2,3,4，........

XSSFCell cell=row.createCell(0);

//在格子中添加数据

cell.setCellValue("Hello World");

ByteArrayOutputStream out=**new** ByteArrayOutputStream();

workbook.write(out);

workbook.close();

out.close();

**byte**[] bytes=out.toByteArray();

**return**bytes;

}

3. 编写客户端:

<p>

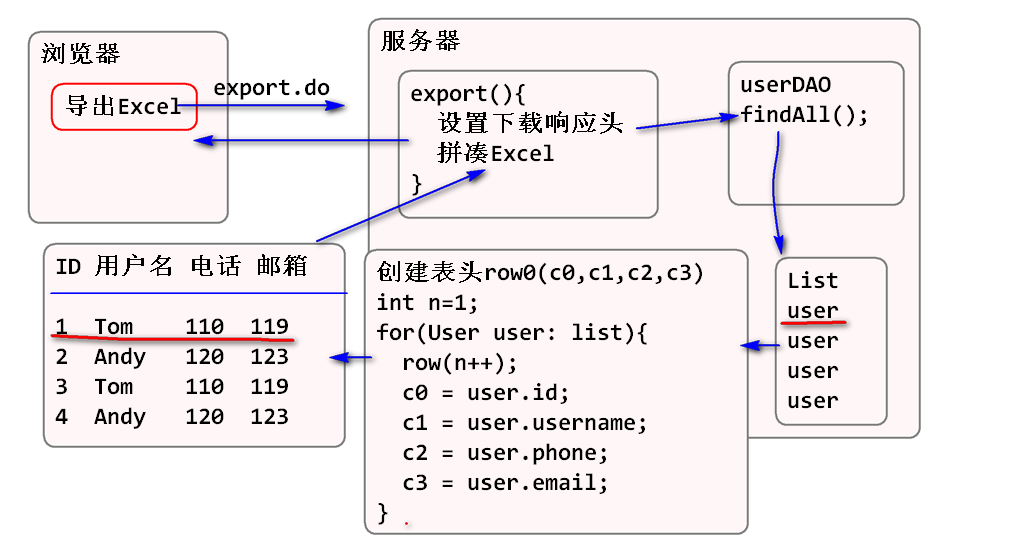
<ahref=*"excel.do"*>下载excel</a>

</p>

4. 测试

## ## 添加用户列表导出功能

导出原理:



导出为Excel表步骤:

1. 编写控制器:

/\*将用户列表导出为Excel表\*/

@RequestMapping(value="export.do",

produces="application/vnd.openxmlformats-officedocument.spreadsheetml.sheet")

@ResponseBody

**publicbyte**[] export(HttpServletResponse response) **throws** IOException{

String file=URLEncoder.*encode*("用户列表.xlsx","UTF-8");

response.setHeader("Content-Disposition", "attachment;filename=\""+file+"\"");

**byte**[] bytes=usersExcel();

**return**bytes;

}

**privatebyte**[] usersExcel() **throws** IOException {

XSSFWorkbook workbook=**new** XSSFWorkbook();

XSSFSheet sheet=workbook.createSheet("用户列表");

//创建表头

XSSFRow header=sheet.createRow(0);

header.createCell(0).setCellValue("编号");

header.createCell(1).setCellValue("姓名");

header.createCell(2).setCellValue("电话");

header.createCell(3).setCellValue("邮箱");

List<User>list=userDao.findAll();

**int**n=1;

**for** (User user : list) {

XSSFRow row=sheet.createRow(n++);

row.createCell(0).setCellValue(user.getId());

row.createCell(1).setCellValue(user.getUsername());

row.createCell(2).setCellValue(user.getPhone());

row.createCell(3).setCellValue(user.getEmail());

}

ByteArrayOutputStream out=**new** ByteArrayOutputStream();

workbook.write(out);

workbook.close();

out.close();

**byte**[] bytes=out.toByteArray();

**return**bytes;

}

2. 更新客户端listUser.jsp添加导出按钮

<input type="button" class="button"

value="导出到Excel"

onclick="location='export.do'"/>

3. 测试

# 第十章、SpringMybatis

## 一、Mybatis

1、

1、持久层的框架，支持普通的sql语句，存储过程（项目）

2、封装可底层的jdbc

3、提高了开发效率。

4、sql语句可以配置到xml文件中。也可用注解方式，但不建议，不方便。

### 2、框架的使用

建库

create database mydb;

（1）建表

create table t\_user(id int auto\_increment primary key,name varchar(50),pwd varchar(32),

Email varchar(50);phone varchar(32));

（2）新建工程

1）新建maven工程

2）添加web.xml

3）tomcat运行环境

4）添加依赖jar包

Mybatis.jar,mysql.jar,junit.jar

（3）创建实体类

（4）配置mybatis框架

1）配置文件（连接数据库）

2）映射文件（写sql语句）

（5）编写持久层

1）定义接口

2）定义接口的实现类

3）sql语句设置到映射文件中

SqlSession 提供的方法:insert,update,delete,selectOne,selectList.

### 3、基于Mapper映射器的mybatis框架的使用（重点）

（1）建表

create table t\_address(id int auto\_increment primary key,province varchar(50),

city varchar(50),area varchar(50),user\_address varchar(50))

（2）创建工程

1）新建maven工程

2）添加web.xml

3）tomcat运行环境

4）添加依赖jar包

Mybatis.jar,mysql.jar,junit.jar

（3）添加mybatis配置文件

1）把SqlConfig.xml放到resources文件夹下；此文件配置的信息用来实现数据库连接的配置信息。

2）在resources文件夹中，创建新的文件夹mappers，在mappers文件夹中，

放置映射文件AddressMapper.xml

（4）对表完成crud的操作

（5）持久层

1）

2）映射文件

## 二、ssm整合

### 1、新建工程

1、新建maven工程

2、添加web.xml

3、tomcat运行环境

4、添加依赖jar包

Spring-webmvc,

mybatis.jar, mybatis-spring,

mysql.jar, commons-dbcp, spring-jdbc,

jstl,junit.jar

5、规范包名

cn.tedu.ssm.bean:实体类

cn.tedu.ssm.mapper:持久层接口

cn.tedu.ssm.service:业务层,包含接口和类

cn.tedu.ssm..controller:控制器层

6、配置文件

spring-mvc.xml（spring-mvc的配置文件）

spring-mybatis.xml(applicationContext.xml)

db.properties

DeptMapper.xml(映射文件）

7、web.xml（配置web容器组件）

（1）ContextLoaderListener:spring容器中（初始化）

（2）DispatcherServlet：spring-mvc容器中（初始化）

（3）设置编码格式的Filter

### 2、设计表t\_dept

create table t\_dept(

id int auto\_increment primary key,

Dept\_name varchar(50),

Dept\_loc varchar(50));

### 3、部门管理

1、添加部门

1.1、添加部门——持久层

新建Dept实体类

//1、set/get方法

//2、如果定义构造方法，必须定义无参的构造方法

//3、重写equals(),hashCode(),toString()

//4、实现序列化接口

（1）接口中定义方法

publicinterface DeptMapper{

void insertDept(Dept dept);

}

（2）映射文件编写sql语句

在DeptMapper.xml文件中，定义insert节点，编写insert语句

<insert id=”insertDept” parameterType=”cn.tedu.ssm.Dept”>

</insert>

（3）测试：

1.2、添加部门——业务层

（1）定义接口

Public interface DeptService{

Void addDept( Dept dept);

}

（2）编写实现类：实现接口中方法

@Service

Public class DpteServiceImpl implements DeptService{

@Resource

Public void addDept(Dept dept){

//调用持久层方法

deptMapper.insertDept(dept);

}

}

1.3、添加部门——控制器层

（1）显示页面

/showAdd.do

请求参数：无

请求方式：get/post（带有密码或数据量大时用post）

响应方式：转发

@Controller

@RequestMapping(“/dept”)

public class DeptController{

@RequestMapping(“/showAdd.do”)

public String showAdd(){

Return “add”;

}

}

定义add.jsp,在web文件夹中定义add.jsp

（2）定义url处理添加按钮功能

/addDept.do

请求参数：name,loc

请求方式：post

响应方式：转发---ok.jsp

@RequestMapping(“/addDept.do”)

Public string addDept(String name,String loc){

//1、调用业务方法

//2、return “ok”

}

在web文件中定义ok.jsp

2、显示部门信息

2.1、显示部门信息——持久层

（1）接口中定义方法

List<Dept> selectAll();

（2）映射文件中编写sql语句

<select id=”selectAll” resultType=”xx.Dept”>

Select id,dept\_name deptName,dept\_loc deptLoc from t\_dept

</select>

（3）测试

2.2显示部门信息——业务层

（1）接口中定义方法

List<Dept> getDeptAll();

（2）在实现类中实现接口中的方法

方法的功能：调用持久层的方法，返回list

（3）测试

2.3显示部门信息——控制器层

请求url

/getAll.do

参数列表：ModelMap（放到request）——把list传递到页面

请求方式：get

享用方式：转发

@RequestMapping（“/getAll.do”）

public String getAll(ModelMap map){

//1、调用业务层方法，返回list

//2、把list添加到map

//3、return “showDept”;

}

2.4、显示部门信息——页面

3、部门管理——删除

3.1、删除部门——持久层

（1）在接口中定义方法

void deleteById(Integer id);

（2）在映射文件中东一delete语句

<delete id=”deleteById”>

Delete from t\_dept where id=#{id}

</delete>

3.2、删除部门——业务层

（1）在接口中定义方法

void removeById(Integet id);

（2）在实现类中实现方法：

方法功能：调用持久层方法

3.3、删除部门——控制层

url

/removeDept.do

参数列表：id

请求方式：get

响应方式：重定向——到getAll.do

public String removeDept(Integer id){

//1、调用业务层方法

return “redirect:/dept/getAll.do”;

}

3.4、删除部门——页面

<a href=”${}/dept/removeDept.do?Id=${}dept.id}”>删除</a>

小结：

1.spring-mybatis整合

2、ssm环境的搭建

3、分析问题：页面——数据库

解决问题：数据库——页面

4、开发步骤：

1）db table

2)持久层

3）业务层

4）控制器

5）页面

# 第十一章、ajax

## 一、ajax

### 1定义

（1）ajax不是新的技术，是javascript,xml,XMLHttpRequest的结合体，完成异步提交功能。

（2）同步提交：用户提交数据后，当前页面不可以操作；页面响应成功才可以操作页面。

（3）异步提交；用户提交数据后，还可以在当前页面上操作；和服务器响应是否成功没有太大影响。

（4）特点：从服务器端获取一小部分数据、

### 2、原理

用户提交 交给ajax的xhr处理，然后xhr把请求提交给服务器，服务器响应数据给xhr，

由xhr把数据接收回来，通过javascript设置数据到页面。

### 3、xhr的创建和函数属性

（1）创建xhr对象：if(window.XMLHttp)

var xhr=new XMLHttpRequest();

//IE5 IE6

var xhr=new ActiveXObject(“Microsoft.XMLHttp”);

（2）open()；打开连接

open(“get/post”,”url”,true);//异步请求为true

（3）send();发送请求

（4）onreadystatechange:从创建xhr对象到响应数据成功，监听5种状态变化。

（5）readyState：获取状态

0：xhr未初始化（没open，send）

1：初始化成功，发送请求

2：接收数据（响应）

3：解析数据

4：响应成功

（6）status:获取响应状态码：200,404,405,500

（7）responseText：接受服务器，响应的数据

### 4、使用ajax异步验证用户名

get 请求

（1）创建工程

创建maven

添加web.xml

添加tomcat运行环境

依赖jar包：spring-webmvc,junit

配置文件：spring-mvc.xml

在web.xml中，配置前端控制器和编码Filter

（2）

（3）控制器

### 5、二级联动菜单

post请求

## 二、Json

### 1、

1、客户端和服务器端传递数据的一种解决方案。

2、语法：key-value

**var** jsonObject={"state":0,"message":"失败"};

alert(jsonObject.state);

3、好处：易于编写，易于阅读，已与解析

### 2、使用json

（1）一个k-v的json对象

**var** jsonObject={"state":1};

alert(jsonObject.state);

3、json字符串-服务器端

4、jQuery实现异步提交

$.ajax({

url:”../user/xx.do”,

data:”name=”+name+”&pwd”+pwd，

type:”get/post”,

dataType:”json”,//服务器响应数据格式

success:function(obj){}

});

# 项目案例：学子商城

14天

学子商城新扩展

1、Mybatis动态语句

If foreach

2、存储过程

3、AOP面向切面编程

目的：提高分析问题和解决问题的能力

## 一、功能模块

用户管理模块

1、注册，

2、登录，

3、修改个人信息（基本信息，密码）

二、地址管理模块

1、添加地址

2、显示地址

3、管理地址（修改，删除）

三、商品管理模块

1、商品分类，

2、商品展示，

3、商品详情

四、购物车模块（会讲存储过程）

1、添加购物车

2、管理购物车商品

## 二、开发流程

1、项目需求——需求说明书（项目原型）

2、概要设计：模块功能总体设计

3、详细设计：详细子模块

4、编码

5、测试

6、上线（产品交付）

7、维护

三、项目

1、创建项目

（1）

（4）添加依赖jar包

Spring-webmvc

Mysql,Commons-dbcp,Spring-djdbc

Mybatis,Mybatis-spring

Junit

Jstl

Jackson\*3

Fileupload,Commins-io

（5）规范包名

Cn.tedu.store.bean

Cn.tedu.store.mapper

Cn.tedu.store.service

Cn.tedu.store.controller

（6）配置文件

Spring-mvc.xml前端控制器相关

Application-service.xml数据库相关

Db.properties

Mapper/xxMapper.xml

（7）web.xml

2、数据表

（1）建库:tedu\_store

（2）建表:t\_user

（3）分析

id int auto\_increment primary key

username varchar(50) not null unique

password varchar(50) not null

email varchar(50) not null

phone varchar(50) not null

image varchar(100)

gender int(1)

created\_user varchar(50)

created\_time data

modified\_user varchar(50)

modified\_time data

create database tedu\_store;

create table t\_user(

id int auto\_increment primary key,

username varchar(50) not null unique,

password varchar(50) not null,

email varchar(50) not null,

phone varchar(50) not null,

image varchar(100),

Gender int(1),

created\_user varchar(50),

create\_time date,

modified\_user varchar(50),

modified\_time date

)default charset=utf8

3

（1）实体类

（2）在mapper包定义接口UserMapper，定义方法

（3）在mappers文件夹，定义UserMapper.xml（把xxMapper.xml改成UserMapper.xml）,在此文件中定义insert节点，编写insert sqly语句，在mapper节点中namespace

2.1.2、注册——业务层

1、在service包中定义接口，定义方法

Public interface InUserService{

Void register(User user);

}

2、在service包中定义接口的实现类，实现接口中方法

在service.ex包中定义异常类UsernameAlreadyExistException；该类继承RuntimeException

@Service

Public class UserService implements IUserService{

@Resource

Private UserMapper userMapper;

Public void register(User user){

//1、调用持久层selectUserByUsername

(user.getUsername());返回User对象user1；

//2、if（user==null){

//3、调用insertUser（user);

}else{

//4、抛出异常UsernameAlreadyExistException

}

}

}

2.1.3、注册-用户名，邮箱，电话号码的异步验证

邮箱持久层：

（1）在UserMapper接口中定义方法，查询邮箱

Integet selectByEmail(String email);

（2）在UserMapper.xml中定义select节点，编写select语句根据邮箱查询

<select id=”selectByEmail” resultType=”java.lang.Integer”>

Select count(\*) from t\_user where email=#{email}

</select>

4、显示首页

新建MainController类，定义方法，显示首页

调整页面

（1）在心图片前面

# 二进制

1、什么是二进制

逢二进一的计数规则

2、16进制

逢16进一的技术规则

其本质目的是用于3简写2进制。2进制从后向前（低位到高位）

每4位课可以缩写为一个16进制数。

计算机专业人员需要使用2进制时候，都是采用16进制。

3、补码

补码是计算机中处理“有符号数”的编码规则。

补码的编码核心思想：将固定位数的2进制数分一半作为负数使用。

固定位数2进制在计算时候超过的位数自动溢出。

4、2进制计算

运算符：& |~>>>>><<

（1）& 与运算

运算规则（逻辑乘法）

0 & 0 ->0

0 & 1 ->0

1 & 0 ->0

1 & 1 ->1

两个数据对齐位数进行按位 & 计算

（2）| 或运算（逻辑加法）

0 | 0 ->0

0 | 1 ->1

1 | 0 ->1

1 | 1 ->1

两个数字对齐位置，每一位进行 | 计算

（3）<< 左移位计算

运算规则：讲一个数字的整体向左移动，高位自动溢出（舍弃），低位补0

（4）>>> 右移位计算

数字整体向右移动，低位自动溢出，高位补0。

（5）>> 数学右移位

低位溢出，高位为0则补0，高位为1则补1。

移位计算的数学意义：