Java学习笔记

目录

[一、配置： 7](#_Toc536455944)

[二、特点： 7](#_Toc536455945)

[三、基础知识： 7](#_Toc536455946)

[1.标识符： 7](#_Toc536455947)

[2.注释： 7](#_Toc536455948)

[3.关键字： 7](#_Toc536455949)

[Final： 8](#_Toc536455950)

[4.运算符： 8](#_Toc536455951)

[5.修饰符 8](#_Toc536455952)

[6.数组： 9](#_Toc536455953)

[一维数组： 9](#_Toc536455954)

[多维数组： 10](#_Toc536455955)

[对象数组： 10](#_Toc536455956)

[四、Java的数据类型： 11](#_Toc536455957)

[常量： 11](#_Toc536455958)

[变量： 11](#_Toc536455959)

[类型转换： 13](#_Toc536455960)

[编译优化 13](#_Toc536455961)

[五、语句结构 13](#_Toc536455962)

[循环： 13](#_Toc536455963)

[Break与continue 14](#_Toc536455964)

[If条件语句： 14](#_Toc536455965)

[switch多分支选择结构： 14](#_Toc536455966)

[六、面向对象 15](#_Toc536455967)

[基本概念 15](#_Toc536455968)

[封装： 15](#_Toc536455969)

[privte关键字： 16](#_Toc536455970)

[继承： 16](#_Toc536455971)

[1.含义： 16](#_Toc536455972)

[2.内容： 16](#_Toc536455973)

[3.特点： 16](#_Toc536455974)

[重名问题： 16](#_Toc536455975)

[super关键字： 17](#_Toc536455976)

[this关键字： 17](#_Toc536455977)

[类： 18](#_Toc536455978)

[方法： 19](#_Toc536455979)

[重载： 19](#_Toc536455980)

[构造方法：Constructor 19](#_Toc536455981)

[static关键字： 20](#_Toc536455982)

[静态代码块： 20](#_Toc536455983)

[静态实例初始化器：？？？ 21](#_Toc536455984)

[return语句： 21](#_Toc536455985)

[内部类： 21](#_Toc536455986)

[权限修饰符与类关系 23](#_Toc536455987)

[对象： 23](#_Toc536455988)

[匿名对象： 24](#_Toc536455989)

[对象数组： 24](#_Toc536455990)

[多态： 25](#_Toc536455991)

[abstract抽象类： 26](#_Toc536455992)

[接口： 26](#_Toc536455993)

[泛型 28](#_Toc536455994)

[集合 29](#_Toc536455995)

[迭代器： 35](#_Toc536455996)

[foreach 36](#_Toc536455997)

[容器： 36](#_Toc536455998)

[异 常： 37](#_Toc536455999)

[七、线程 39](#_Toc536456000)

[1.进程与线程 39](#_Toc536456001)

[2. Thread线程类和Runnable接口 40](#_Toc536456002)

[3.线程的创建： 40](#_Toc536456003)

[4.线程安全问题： 41](#_Toc536456004)

[5.进程通信： 42](#_Toc536456005)

[6.等待和唤醒机制： 43](#_Toc536456006)

[7.线程池： 43](#_Toc536456007)

[8.函数式编程： 44](#_Toc536456008)

[IO: 45](#_Toc536456009)

[File类： 45](#_Toc536456010)

[过滤器： 46](#_Toc536456011)

[流： 46](#_Toc536456012)

[函数式编程： 54](#_Toc536456013)

[Stream流： 55](#_Toc536456014)

[方法引用： 56](#_Toc536456015)

[软件测试： 58](#_Toc536456016)

[注解： 58](#_Toc536456017)

[反射： 59](#_Toc536456018)

[八、数据库： 62](#_Toc536456019)

[MySQL: 62](#_Toc536456020)

[九、JDBC 76](#_Toc536456021)

[概念： 76](#_Toc536456022)

[一般步骤： 77](#_Toc536456023)

[JDBC连接池: 80](#_Toc536456024)

[十、Html 82](#_Toc536456025)

[Css 82](#_Toc536456026)

[JavaScript 85](#_Toc536456027)

[Jqurey 85](#_Toc536456028)

[JQ基本使用： 85](#_Toc536456029)

[\*JS与JQ转换 86](#_Toc536456030)

[\*选择器 86](#_Toc536456031)

[\*内容操作： 89](#_Toc536456032)

[\*属性操作： 89](#_Toc536456033)

[\*样式控制： 90](#_Toc536456034)

[\*CRUD 91](#_Toc536456035)

[动画的显示和隐藏： 92](#_Toc536456036)

[\*遍历: 92](#_Toc536456037)

[\*事件： 94](#_Toc536456038)

[BootStrap 94](#_Toc536456039)

[XML 94](#_Toc536456040)

[Tomcat 95](#_Toc536456041)

[安装目录： 95](#_Toc536456042)

[常见问题： 95](#_Toc536456043)

[部署项目 95](#_Toc536456044)

[IDEA配置Tomcat: 96](#_Toc536456045)

[IDEA创建JavaEE项目 96](#_Toc536456046)

[原理： 97](#_Toc536456047)

[小技巧： 97](#_Toc536456048)

[Servlet 97](#_Toc536456049)

[概述： 97](#_Toc536456050)

[servlet3.0: 98](#_Toc536456051)

[servlet体系结构： 98](#_Toc536456052)

[servlet相关配置 99](#_Toc536456053)

[Http 99](#_Toc536456054)

[概述： 99](#_Toc536456055)

[双R原理： 100](#_Toc536456056)

[Request 100](#_Toc536456057)

[请求消息数据格式： 100](#_Toc536456058)

[体系结构： 101](#_Toc536456059)

[功能： 101](#_Toc536456060)

[Response: 103](#_Toc536456061)

[概述： 103](#_Toc536456062)

[组成： 103](#_Toc536456063)

[功能： 104](#_Toc536456064)

[会话技术： 106](#_Toc536456065)

[Cookie： 106](#_Toc536456066)

[Session: 106](#_Toc536456067)

[JSP 107](#_Toc536456068)

[El表达式: 107](#_Toc536456069)

[JSTL标签: 107](#_Toc536456070)

[Fliter过滤器: 107](#_Toc536456071)

[Listener 108](#_Toc536456072)

[综合案例 108](#_Toc536456073)

[Ajax: 108](#_Toc536456074)

[1.原始JS方式 108](#_Toc536456075)

[2.Ajax方式： 108](#_Toc536456076)

[3.$.get()/post(); 109](#_Toc536456077)

[Json: 109](#_Toc536456078)

[语法： 109](#_Toc536456079)

[Json解析器： 111](#_Toc536456080)

[Java对象转换Json字符串 111](#_Toc536456081)

[集合转Json字符串 111](#_Toc536456082)

[Json字符串转换 Java对象 111](#_Toc536456083)

[案例： 112](#_Toc536456084)

[Redis 113](#_Toc536456085)

[概念 113](#_Toc536456086)

[安装： 113](#_Toc536456087)

[应用场景： 113](#_Toc536456088)

[Redis数据结构： 114](#_Toc536456089)

[命令操作： 114](#_Toc536456090)

[redis持久化 115](#_Toc536456091)

[Java客户端 Jedis 115](#_Toc536456092)

[jedis连接池： JedisPool 115](#_Toc536456093)

[Spring data redis: 116](#_Toc536456094)

[Question: 116](#_Toc536456095)

[Maven: 118](#_Toc536456096)

[简介： 118](#_Toc536456097)

[安装： 119](#_Toc536456098)

[配置: 119](#_Toc536456099)

[maven安装目录： 122](#_Toc536456100)

[maven项目： 122](#_Toc536456101)

[框架 123](#_Toc536456102)

[Mybatis: 123](#_Toc536456103)

[简介 123](#_Toc536456104)

[相关概念： 123](#_Toc536456105)

[环境搭建： 124](#_Toc536456106)

[Xml配置： 124](#_Toc536456107)

[mybatis的具体使用： 127](#_Toc536456108)

[API分析: 129](#_Toc536456109)

[Mybatis连接池 129](#_Toc536456110)

[Mybatis事务： 130](#_Toc536456111)

[动态sql： 131](#_Toc536456112)

[Mybatis表关系： 135](#_Toc536456113)

[延迟加载和立即加载 136](#_Toc536456114)

[Mybatis缓存： 136](#_Toc536456115)

[自定义框架： 137](#_Toc536456116)

[Spring 138](#_Toc536456117)

[Bean: 139](#_Toc536456118)

[IOC： 141](#_Toc536456119)

[DI依赖注入： 141](#_Toc536456120)

[SpringAOP相关注解： 142](#_Toc536456121)

[基于XML的spring声明式事务控制 143](#_Toc536456122)

[基于注解的springAOP 事务控制 144](#_Toc536456123)

[优化业务层代码： 145](#_Toc536456124)

[Spring提供的事务控制API：； 145](#_Toc536456125)

[spring中基于XML的声明式事务控制配置步骤 147](#_Toc536456126)

[Spring基于注解的声明式事务控制 149](#_Toc536456127)

[全文检索Lucene： 149](#_Toc536456128)

[常用类： 152](#_Toc536456129)

[System类： 152](#_Toc536456130)

[Arrays类： 152](#_Toc536456131)

[Colloections类： 152](#_Toc536456132)

[Math类： 153](#_Toc536456133)

[Random类： 153](#_Toc536456134)

[Object类： 153](#_Toc536456135)

[Date类： 154](#_Toc536456136)

[DateFormat类： 155](#_Toc536456137)

[SimpleDateFormat类 155](#_Toc536456138)

[calender类： 155](#_Toc536456139)

[String:类 155](#_Toc536456140)

[字符串缓冲区： 156](#_Toc536456141)

[系统类库： 158](#_Toc536456142)

[问题与解疑： 158](#_Toc536456143)

[Java单词： 158](#_Toc536456144)

# 一、配置：

# 二、特点：

跨平台：依靠JVM实现，但JVM不是跨平台，每一个OS有对应版本的JVM

开源

面向对象

# 三、基础知识：

## 1.标识符：

类名，方法名，变量名...

**命名规则（硬性）：**

①由26个字母（区分大小写），数字0-9，下划线和$，中文字符（后2不推荐）组成

②首字母不能是数字

③不能是关键字

④中间不能有空格和特殊符号

**命名规范（软性）：**

①类名规范：首字母大写，后面每个单词首字母大写（大驼峰式）

②变量名规范：首字母小写，后面每个单词首字母大写（小驼峰式）

③方法名规范：同变量名

## 2.注释：

\*>单行注释：//

\*>多行注释:/\*...\*/

\*>文档注释:/\*\*...\*/

\*单行注释可以嵌套，多行不可以嵌套

## 3.关键字：

赋予特殊含义的单词不能随意使用

保留关键字：goto ;const

特点：小写字母；特殊颜色

### Final：

**用法：**

被final修饰的常量名称，一般都有**书写规范，所有字母都大写。**

1.可以修饰局部变量：**一次赋值，无法改变**;必须保证实际值没有变化：做参数时在方法中可以变，但实际值没变

基本：数据值

引用：地址值

2.可以修饰成员变量：一次赋值，无法改变；用了final，**不会给默认值；必须手动赋值；** 二选一：

直接赋值；

构造赋值：**必须保证所有重载构造方法都对final成员变量赋值**

3.可以修饰方法：**不能被覆盖重写**

4.可以修饰类：被修饰的类**无法有子类**，但可以有父类；其中所以方法无法覆盖重写

Tips：

对于类与方法：**abstract与final不能同时使用**

**Abctract关键字，不可与private；static;final同用**

## 4.运算符：

三元运算符：

常用形式：

1.数据类型 变量 = 关系表达式？ 结果1：结果2；

2.System.out.println(关系表达式？ 结果1：结果2);

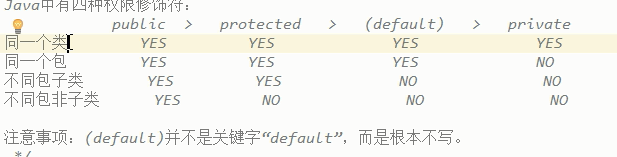
tips：

1.结果必须被使用

2.两种可能结果的**数据类型**必须与被赋值的变量一致

## 5.修饰符

权限修饰符：***public>protect>default(默认)>private***



静态修饰符:static:类，方法，变量；静态不可访问非静态

其他修饰符：abstract抽象类;default接口默认方法；final

## 6.数组：

**同类型；(length)大小不能改变；元素个数可以是变量；**可以读写（类c）

**引用数据类型**

### 一维数组：

**1.声明：**①类型 []数组名； ②类型 数组名[]；

***Tips:下标[]在数组名前或者后是有区别的***

int a[],b; //a是一维数组，b是变量

int []a,b; //a和b都是一维数组

**tips:不管【】的位置在哪，数组new时的行【】长度不能省略，列可以省**

**2.创建数组：**格式：new 数据类型[长度] 如：new int[9]

Tips:该数组没有名字，不能单独使用，需要和上面的声明配和使用：

int a[]=new int[9];

声明和分配空间可以分开，也可以放在一起；

**静态初始化：**指定内容

数据类型【】数组名=new 数据类型【】{...具体元素 }

省略格式：数据类型【】数组名= {...具体元素 }

**动态初始化：**指定长度

数据类型【】数组名=new 数据类型【数组长度】

静态初始化和动态初始化都可以拆分成两步：

即 定义；

创建；

**Tips：**

1.静态初始化省略模式，不可以拆分

2.**静态适用于知道数组的元素**，由系统自动计算长度

3.**动态适用于知道数组的长度，**而不知道数组的元素

**3.使用数组：**

①任何一个数组都有length属性：用法：数组名.length

**注意：数组的长度length是final variable型，值不可改变；**

**需要改变数组的表示长度时，可利用一个变量接收数组实际长度**

②访问数组元素：数组名 [下标]

***Tips：与c不同，java 数组可以赋值；如数组a和数组b***

***a=b; //把b的地址赋值给a***

### 多维数组：

**二维数组：**

**1.声明：**①类型 数组名[][] ②类型 [][]数组名 ③类型 []数组名[]

***Tips:下标位置不同，声明的结果也不同***

int a[][],b; //a是二维数组，b是简单变量

int []a[],b; //a是二维数组，b是一维数组

int [][]a.b //a和b都是二维数组

**2.分配空间：**

①每行元素相同：

new int[3][4]; //共三行，每行都有四个元素

②**每行元素不同：**

new int[3][];

a[0]=new int[3]; //第1行有3个元素

a[1]=new int[4]; //第2行有4个元素

a[2]=new int[5]; //第3行有5个元素

**3.使用数组：**

①**数组名.length 表示数组的行数； a[i].length表示第i行元素的个数**

②访问数组元素：数组名 [下标][下标]

③初始化： 类似c

### 对象数组：

对象数组是元素的管理者

元素是对像的数组

**数据类型 []对象数组名 = new 数据类型[数组大小]；**

*Tips:*对象数组可用forech循环，但与普通数组不同，**对象数组的forech循环会改变值**， 普通的不会

对象数组的赋值时，需要再次为每个对象分配空间：

例：employee[2]= new Employee(…)

1.**对象数组直接打印元素，结果是地址值；调用对应的属性，或者重写元素类型所在类的toString（方法），才能将值打印**

**String 数组是值**

如：Student[] array = new Student[5];

Student one = new Student(...);

...

array[0] = one;

...

System.out.println(array[0]);//输出的是地址

Student temp = array[0];

System.out.println(temp.name);//输出的是对象的属性值

# 四、Java的数据类型：

## 常量：

字面值常量

自定义常量

1、 整型常量：

10进制；16进制：0x或0X；8进制0；二进制数以0B或0b开头

整型常量默认int

Tips:Java中无无符号数；**long型后需要加l或L；**

2、浮点型常量:

十进制形式（必须含小数点，.154合法）；科学计数法

浮点型**默认double**；声明float型必须加f或F

字符串常量

字符常量

布尔常量

空常量：null

## 变量：

java和c一样，是一种强类型语言；变量使用前要定义，程序运行中也不可以改变变量的类型

格式：变量类型 变量名称

分为两大类：

Ⅰ、基本数据类型：

①数值型：整数型（byte,int,short,long）（1，2，4，8）

浮点型（float,double）（4，8）

②字符型：char（一字符占两字节）（2）

③布尔型：boolean（逻辑运算只能取值true和false，不可以0或非0代替）

Ⅱ、引用数据类型：

①类：class

②接口:interface

③数组

④集合

⑤String

**Java中都是值传递：**

**基本数据类型做参数传递的是值；**

**引用数据类型做参数和返回值时，传递的是地址；String除外，它做参数传递的是具体值；副本，实参值不变**

Tips:

浮点型可能只是一个近似值；

取值范围与所占字节无必要联系（float>long）

默认类型

**变量不能重复定义，**

**先定义再使用**

区别：内存空间：

基本数据类型变量在栈里：这里

引用数据类型变量引用到堆：哪里（类似指针）

赋值时 ：

基本类型复制的是值

引用类型复制的是引用

包裹类型：包裹类型可以直接使用相应的基本类型

**初始化：**

显示初始化：

定义常量时，直接赋值

构造初始化：

创建对象，利用构造方法

**成员变量与局部变量区别：**

1.默认值

成员：有默认值

局部：无默认值，必须先初始化，再使用

2.作用域

成员：整个类

局部：所在括号内

3.位置

成员：类中方法外

局部：方法内，声明处

4.内存位置

成员：堆：随对象的new而进入堆

局部：栈：随方法的调用而进栈

5.生命周期

成员：随对象的创建而存在

局部：随方法的调用而存在

## 类型转换：

**显示类型转换**：范围小的向范围大的转换，系统自动完成（小->大）

**强制类型转换：**范围大的向范围小的转换，手动完成 （大->小）

格式：1.目标类型 变量名 = （目标类型）需要转换的数据

**强制类型转换易产生精度损失和数据溢出（不建议强转）**

1.byte,char,short变量***参与运算时会先提升为 int***,再进行计算

2.boolean***不能进行计算，也不能进行类型转换***

3.byte,char ,short，如果右侧没有超过左侧范围，编译器会自动补上对应的强转（byte），（char），（short）；如果超过范围， 会报错

4.在给变量赋值时，右侧表达式全是常量时，编译器自动计算表达式结果、

## 编译优化

例：

byte b,b1 = 2 ,b2 = 3, b3;

b = b1+b3;//编译失败

b = 3+4;//编译成功

b = 111+111;//编译失败

**编译时期**，**jvm不知道变量的值，**只有在运行时才知道变量的数值

全**常量参与运算**时，jvm在**编译时期已经计算出结果**，然后判断结果是否在左侧数据类型范围内，如果在，就赋值给左侧，如果不在，就报错

# 五、语句结构

## 循环：

1.循环内容:

\* 初始化语句: 只执行一次

\* 条件判断语句：判断条件是否成立，如果成立就执行循环体，如果不成立就结束循环

\* 循环体: 重复要做的内容

\* 步进语句: 进行扫尾工作,每次循环结束后,都会执行一次

2.三种循环区别:

1.执行次数:do...while至少执行一次；for与while循环至少执行0次；

2.循环变量的作用域：for循环变量相当于局部变量

for与while唯一的区别是初始化语句作用域的问题

？？？性能

tips：

**循环次数已知的推荐用for循环，次数未知的用while循环**

## Break与continue

循环控制：break语句与continue语句：

1.break：跳出循环

1.常用switch 多分支结构与循环语句中

2.多种嵌套循环，根据位置来判断跳出那一层循环

2.continue：终止本次循环，开始下一次循环；并未跳出整个循环

3.

Break语句：结束循环体

形式：1.基本形式：…break；…

2.***带标号的***：标号： //标号是任何合法的标识符 循环语句

break 标号；//使标号所指向的循环提前结束

Continue语句：结束本次循环

形式：1.基本形式：…continue；…

2.***带标号的***：标号： //标号是任何合法的标识符 循环语句

continue 标号；//使标号所指向的循环当循环提前结束，后面循环的继续执行

## If条件语句：

if...else语句与三元运算符相互转换的问题

\* 能使用三元运算符就一定能使用if...else格式的

\* 能使用if...else格式的.不一定能使用三元运算符

## switch多分支选择结构：

基本形式：

Switch(表达式){

case 常量值1：语句；break;//语句可以省略

case 常量值1：语句；break;

case 常量值1：语句；break;

...

Default:语句；break;

}

1、switch语句中，表达式的数据类型：

**基本数据类型：byte，short，int，char，**

**引用数据类型：enum，String；**

2、case:常量值不能重复；

3、省略break语句时要注意穿透问题

发生穿透时，不会判断之后的case值，而是直接执行语句，直到遇到**break退出**

**4.switch语句形式灵活，default语句也可以放在第一行，但也先执行**

# 六、面向对象

## 基本概念

1.面向对象技术三大特征：封装（类）、继承、多态

①、类 ：类是对同一类对象的抽象描述，它包括对象的属性和行为，是面向对象的基础

②、继承：在现有类的基础上生成新的类，通过继承可以实现代码复用，提高编程效率

③、多态：同一对象有不同行为：提高效率，程序的可维护，可扩充性

2.对象：客观世界中的任何事物都是对象（不论生命或肉眼所见）

对象有两特征：属性和行为（功能）

3.类 ：类是对同一对象属性和行为的封装 （用类描述对象）

是一类对象的原型

是组成java的基本要素

抽象:抽象是忽略事物与当前目标无关的非本质特征，而着眼于与当前目标有关的本质特征，**从而找出同一类事物的共性，并把具有共性的事物归为一类，**得到一个抽象的结果

***Tips:***①抽象是对事物的高度概况（属性与行为的概括）

②根据问题的需要，抽象的结果也不同，如：

（学籍系统:注重课程与成绩；而学生健康：注重身高，体重，病历）

二者都是描述学生的一个类，但二者的属性不同

③ 类是规范，根据类的定义创建对象；

对象是实体，需要被创建

## 封装：

将一些细节信息隐藏起来，对于外界不可见

1.方法：将相同功能代码封装成方法，提高复用性，减少冗余

2.private关键字:狭义，提高安全性

3.工具类

### privte关键字：

对需要保护的成员变量进行保护；间接访问，通过一些代码排除不合理使用

可用set与get设置与获取被保护的成员变量

其中set必须有参数，无返回，相应，get无参数，有返回值，与成员对应

间接访问private成员变量，就是定义一对get和set方法，必须叫setXxx或getXxx命名

对于基本类型中的**boolean**，**get方法**一定要写成**isXxxx形式**，**setXx不变**

**扩展：**

狭义封装：成员变量私有化，只能在本类使用；外界可通过公用set与get方法访问

①不能直接访问 ② 处理非法访问

广义封装：将相同功能代码封装成方法，提高复用性，减少冗余

将相同功能的代码抽取到一个类中---工具类

## 继承：

### 1.含义：

继承要解决的问题是：共性抽取

### 2.内容：

父类：基类，超类

**父类只能使用父类的东西**

子类：派生类

**子类既可以直接使用父类非私有成员和方法的，也可以使用自己的**

### 3.特点：

Java支持单重继承（类），多级继承（接口）；不支持多重继承。

即一个子类只能有一个直接父类，但一个父类可以有多个子类

一个接口可继承多个接口

### 重名问题：

1.成员变量重名：

子类对象访问规则：

直接：直接通过子类对象访问成员变量：

创建对象时，等号左边是谁，就优先使用谁，没有往上找，没有就报错

间接：通过成员方法访问成员变量，谁的方法，优先使用谁，没有往上找

三种变量重名问题：

**父类成员变量：super.成员变量**

**本类成员变量：this.成员变量**

**局部变量 ： 直接使用**

2.重名方法：

谁创建（new ）用谁，没有往上找

**Tips：**

**无论成员方法，还是成员变量，如果没有都是向上找父类，绝不会找子类**

**方法的覆盖重写：**

New的是谁，优先使用谁的方法，没有上找

子类方法的返回值必须小于或等于父类方法的返回值的范围

子类方法的权限必须大于等于父类方法的权限修饰符

Public > protected> (default) >private: default 不是关键字

### super关键字：

访问父类内容

1.在子类中访问父类的成员变量

2.在子类中访问父类的成员方法

3.在子类的构造方法中访问父类的构造方法：super();

子类构造方法中默认有，不写也存在

Tips：

1.子类构造方法中默认隐含super()；调用父类无参构造方法；

如果父类构造方法有参，就没有默认无参构造方法

3.Super父类的构造调用，必须是子类构造方法中的第一个语句；不能调用多次super构造，只能调用一个；

4.只有子类构造方法中才能调用父类构造方法

Tips:子类必须调用父类构造方法，不写就默认super（）,写了就调用指定的父类构造

### this关键字：

用法：访问本类内容

在本类成员方法中，访问本类成员变量

在本类成员方法中，访问本类另一个成员方法

在本类构造方法，访问本类另一个构造方法

A：this（）构造调用也必须是每个构造方法的第一个语句，且只能有一个

但不能循环调用

B：**super和 this两种构造调用不能同时使用；**

如下例

例：

Public zi (){

This(123);//本类无参构造调用有参构造

This（...）//错误，只能有一个，同super

}

Public zi(int a){

}

①this是对象名，对象的引用

②this的作用域在方法中

③**如果类中域名与方法中的本地变量重名，本地变量会覆盖域，需要用this关键字，<this.域名>表示域**

**二者对比：**

super用于访问父类内容，this用于访问本类内容

二者都必须是构造方法中的第一行语句；所以二者不能同时使用。

## 类：

1.类的定义：

2.基本形式：class 类名 //首字母大写

｛

域 //域是对象的属性，也是对象的数据

方法 //方法是对象的行为，是对对象的域进行处理的工具

｝

定义类时，应先根据问题需要抽象出对象的域

定义方法时，

3.类体：

Ⅰ、域：

定义形式： 数据类型 域名 //变量

类型前可加修饰符

Tips：①域的默认值

②类型前可加修饰符

（***视类型而定：数值型：0字符型：\u0000；布尔型：false；引用型：null***）

域的初始化：

①定义域时可以给域赋初值，也可默认

②创建对象时，利用静态初始化器，实例初始化器或构造方法对域初始化

Ⅱ、方法

4.使用：

导包：java.lang包中类不用导；同包不导，不同包：包路径.类名

对象：

JVM垃圾回收机制：

### 方法：

基本形式：

修饰符 返回值类型 方法名 （参数列表）

｛方法体｝

Tips：①方法类型是方法的返回值的类型，没有返回值就是void

②方法名是***首单词首字母小写，其余单词首字母大写***

③形参可有可无

④方法体是方法的功能实现部分

⑤类型前可用修饰符

**⑥方法定义先后顺序无关系，与调用有关**

⑦方法定义后只有调用才会执行

**⑧方法定义不可嵌套**

1、方法的使用：调用与递归调用

**方法的调用：**

单独调用：本类中的静态方法：静态方法名（）；

在本类中调用本类中的方法

输出调用（不推荐） System.out.println(方法名（）);（必须有返回值）

赋值调用（推荐） 创建一个变量类接收调用结果

tips：

无返回值方法只有单独调用一种方式

### 重载：

含义：同一作用域内，可以有一组***相同方法名***，***不同参数表***的方法

功能：通常用来命名一组功能相似的方法，

可减少方法名的数量，提高可读性和方法的调用性

Tips：①不同参数表是指***参数个数或者参数类型,或者多个参数数据类型顺序不同***；重载是根据它来区分的。

②当调用重载方法时，系统会***根据实参***的不同***自动调用相应***的方法

③方法的重载与参数名称，修饰符与返回值类型无关

### 构造方法：Constructor

功能： 构造方法是类中的方法；

***用来创建对象以及初始化对象的域***

***有的可以进行类型转换***

定义：构造方法名（形参）

｛构造方法体｝

Tips:① 构造方法没有类型名，既没有返回值

②***构造方法名与所在类类名相同***

③方法体可以为***空***

④构造方法可以***重载***，多数情况定义为public

⑤***调用构造方法前先调用实例初始化器***

⑥***默认构造方法***：如果用户没有定义构造方法，系统会默认一个无参数，空体的构造方法，使用它也可以创建对象

### static关键字：

本类中的静态方法可直接调用；也可类名调用，也可以对象调用（不推荐）

普通方法：对象调用

1.作用：

修饰成员变量，方法，

**类变量：**属于所在类，多个对共享同一份数据；一个对象调用，作用于所以对象

**类方法：**推荐用类名.静态方法方式调用；也可用对象.静态方法名调用，但不推荐，javac会编译成前者

**tips：**本类中，**非静态方法**中**可**直接调用**其他本类方法**

**静态方法**中**不**能直接调用**非静态方法。**

**⑥静态方法不能直接访问非静态成员变量;静态方法中也不能直接访问非静态方法**

**（原因是内存先有静态，后有非静态）**

**⑦静态方法中不能用this关键字:this代表当前对象，谁调用谁就是当前对象**

**⑧静态方法也不能用super**

### 静态代码块：

类中；**一次性对静态成员变量进行赋值**

形式：

static{//初始化语句} //静态代码块：

tips：

静态代码块

构造代码块

同步代码块

局部代码块

应用：

加载驱动

加载配置文件

Tips:

①***只有方法体，没有方法头;***它也是***类中***的一个方法

②一个类中***最多一个***实例初始化器，可以定义，也可不定义

***③首次使用本类时，首先执行且只执行一次***

⑤***静态代码块在构造方法前被调用***

（静态内容总是优先于非静态，所以静态代码块先于构造方法执行）

### 静态实例初始化器：？？？

定义： ｛方法体｝

Tips:①它也是***类中***的一个方法

②***只有方法体，没有方法头***

③一个类中***最多一个***实例初始化器，可以定义，也可不定义

④***实例初始化器在构造方法前被调用***

### return语句：

1.结束方法；

2.返回与方法类型相同的值

Tips:

1. 无返回值方法void,最后一行reutrn可以省略；

若不省略，只能是return ;

2.一个方法可有多个return，但不能连写，且必须保证同时只有一个被执行；

### 内部类：

类A包含类B，A外部类，B内部类

**命名规范：避免$符，因为JVM在编译内部类时，外内两类之间用 $分割**

#### 成员内部类：

**类中方法体外**

**定义格式：**

class 外部类 {

class 内部类{

}

}

在描述事物时，若一个事物内部还包含其他事物，就可以使用内部类这种结构。比如，汽车类 Car 中包含发动机 类 Engine ，这时， Engine 就可以使用内部类来描述，定义在成员位置。

**访问特点**

**内部类可以直接访问外部类的成员，包括私有成员。**

**外部类要访问内部类的成员，必须要建立内部类的对象。**

**内部类对象格式**：

**外部类名.内部类名 对象名 = new 外部类型().new 内部类型()；**

使用方式：

1.间接方式：外部内方法内使用内部类；main调用外部方法，间接访问内部类

2.直接方式：直接创建内部类对象：

外部类名.内部类名 对象名 = new 外部类型().new 内部类型()；

重名问题：

变量重名：

外部成员变量:外部类名称.this.外部类成员变量

内部成员变量：：this.变量

方法局部变量：就近原则

#### 局部内部类：

**类中，方法内**；作用域：所在方法内：只有所属方法内才能使用

**形式：**

Class 外部类名称{ //外部类

外部类方法{ //外部类方法

Class 内部类名称{。。。} //方法内内部类

}

}

**使用：**

1.**创建外部类对象调用内部类所在方法使用，然后在方法内定义内部类对象使用内部类中的成员；或者内部类使用所在方法中的final局部变量**

2.局部内部类只能在所属方法中使用；局部内部类对象只能在所属方法内创建

**3.局部内部类如果想访问所属方法中的局部变量，必须保证该变量值不变；即被final修饰**

Tips：Java8+开始Final可省略

原因：局部变量随方法进栈与出栈；局部内部类new出来的对象在堆中，

方法运行结束后，局部变量消失，而对象还在堆中，直到被回收机制回收；

如果对象还想使用局部变量，必须保证局部变量值不变，将其作为一个常量使用，即使方法结束，也可通过常量访问

#### \*匿名类：

属于局部内部类

是内部类的简化写法。*它的本质是一个带****具体实现的父类或者父接口****的****匿名的子类对象****。* 开发中，最常用到的内部类就是匿名内部类了

如果接口的实现类或者父类的子类只需要使用唯一的一次，那么这种情况下就可以省略该类的定义，而改用匿名内部类

**格式：**

接口名称 对象名 = new 接口名称{//接口不能直接实例化，此处用接口=接口+{}+；代表匿名类

//覆盖重写所有抽象方法

}；//注意分号

用创建的对象调用成员

1.大括号{}内才是匿名内部类内容；注意大括号之后的分号”；”

2.匿名内部类创建对象只能使用唯一一次，如果想多次使用，需再进行单独定义实现类(就是重复定义多个匿名类对象，)或者不用匿名类，定义一个接口的实现类；

3.匿名对象，在调用方法的时候，只能使用一次；如果希望同一个对象，调用多次方法，必须给对象命名：对象名是匿名类的对象

4.匿内部类省略了实现类/子类名称，但是匿名对象省略了对象名称

匿名类不等于匿名对象

5. **New 后的接口，是匿名类需要实现的接口**

Tips:

**使用匿名类，但不是匿名对象：**

MyInterface obj = new Myinterface(){

//覆盖重写接口中的方法

};

Obj.method();

**使用匿名类，且使用匿名对象**

new Myinterface(){

//覆盖重写接口中的方法

}.method();

### 权限修饰符与类关系

外部类：public;default只能用这两个

成员内部类：四个

局部外部类：什么都不能用

## 对象：

1.含义：由类所定义的变量——对象型变量

Tips:对象型变量特殊之处在于他有属性和行为

2.声明：类名 对象名 Book myBook

创建：new 构造方法（实参） new Book()

表示：声明+创建 （可用一条语句）Book myBook =new Book()

3.对象的使用：

对象名.域名

对象名.方法名（）

### 匿名对象：

1.含义：没有名字的对象

没有名引用堆内存的对象

**2.格式：new 类的名称()；**

3.用法：

1.**可以通过匿名对象调用方法，给变量赋值**

例：new Student().study();

new Student().name =”没卵用”

2.**一个对象只使用一次的情况**

**作为方法的参数与返回值**

(1)匿名对象做方法**参数：**

public void main(){

method(new Scanner(System.in));

}

public void method(Scanner sc)){

int num = sc.nextInt;

System.out.println(num);

}

(2)匿名对象做**返回值：**

return new Scanner(System.in);

对象数组：对象数组是元素的管理者

元素是对像的数组

数据类型 []对象数组名 = new 数据类型[数组大小]；

*T****ips:*对象数组可用forech循环，但与普通数组不同，对象数组的forech循环会改变值， 普通的不会**

对象数组的赋值时，需要再次为每个对象分配空间：

例：employee[2]= new Employee(…)

1.**对象数组直接打印元素，结果是地址值；调用对应的属性，才能将值打印**

**String 数组是值**

如：Student[] array = new Student[5];

Student one = new Student(...);

...

array[0] = one;

...

System.out.println(array[0]);//输出的是地址

Student temp = array[0];

System.out.println(temp.name);//输出的是对象的属性值

## 多态：

继承（类类，接口与接口）或者实现（类实现接口）是多态的前提

对象的多态性：一个对象具有多个形态

体现：父类对象引用子类对象

格式：

1.父类名称 对象名 = new 子类名称（）；

2.接口名称 对象名 = new 实现类名称（）；

Fu obi = new Zi();

①访问成员方法：new的是谁，就用谁的方法，没有就向上找

②访问成员变量的两种方式：

直接：通过对象名.成员变量：

=号左边是谁，就优先用谁的，没有向上找（不会向下）

间接：通过成员方法访问成员变量：

看该方法属于谁，优先用谁，没有向上找

子类没有覆盖重写就是父，否则就是子

多态成员方法特点：

动态绑定

**方法：编译看左，运行看右。**

**变量：编译看左，运行看左。**

**对象的向上转型:**

父类名称 对象 = new 子类名称（）

其实是多态写法；在右侧创建一个子类对象把他当做父类使用

对象一旦向上转型为父类，那么无法调用子类特有内容

向上转型一定是安全的，从小范围到大范围

**向下转型：**

子类类型 对象名 = （子类类型）父类对象

其实是一个还原的动作；将父类对象还原成为原来的子类对象

注意：

1.必须保证对象本类创建时，就是猫，才能乡下转型

2、若果对象创建的时候本来不是猫现在非要向下转型成为毛，就会拨错

3.**一定要进行instanceof判断**

instanceof

格式：

对象 instanceof 类名称，返回一个布尔值；判断前面的对象是不是后面类型的实例；

### abstract抽象类：

如果父类中的方法不确定如何实现，这个方法应该是抽象方法

**抽象方法：只有方法头，没有方法体；**

如：public abstract void method();

**tips：**

**抽象类可以包括抽象方法和普通成员方法；但抽象方法只能在抽象类中**

**Abctract关键字，不可与private；static;final同用**

**抽象类：**

**1.抽象类不能直接实例化，可以让一个类继承抽象类，然后用子类重写实现抽象类抽象方法**

**2.抽象类中可以包含构造方法,初始化数据:创建子类对象时，先调用父类构造方法，再调用子类构造方法。**

**3.一般，子类必须重写抽象类的所有抽象方法；但如果子类没有重写所以抽象方法，则这个子类必须是抽象类。**

**Tips：**

**1.抽象子类继承抽象父类；可有选择实现；**

**普通类继承抽象子类时，需要重写实现父类没有重写实现的抽象方法**

**2.普通类继承抽象类，必须重写实现所有抽象方法；**

**重点：**

**抽象类（接口）做参数与返回值 用的是子类（或者实现类）**

### 接口：

Interface是一种特殊的Class

是若干类的公共规范；是若干常量和方法的声明的集合

一种引用数据类型

解决java单继承

解耦

**1.格式：** public interface 返回值类型 接口名{

//成员常量

//抽象方法

//私有方法

//默认方法

//静态方法

} **}**

**2.内容：**

**-成员常量：**

**形式：public static finale 数据类型 常量名 =值**

**默认修饰符：public static final：可以省略**

**必须手动赋值，一旦赋值，无法改变;**

**不赋值，就是默认值，因为final无法改变**

**命名规范：使用完全大写字母，\_下划线分割**

**-成员方法**

所有版本都支持抽象方法

1.Java7：常量，抽象方法

2.Java8：额外包含默认方法和静态方法

3.Java9：额外包含私有方法

**0-抽象方法：**

**修饰符固定是public abstract;可以选择性省略(全省或者部分省)，但不能是其他**

**抽象方法没有方法体**

**1-默认方法：**

格式：public **default** 返回值类型 方法名（参数表）{**//default不可省略**

//方法体

}

解决接口升级的问题

调用默认方法，如果实现类没有，就向上找接口

默认方法会被实现类继承：**默认方法可被实现类的对象直接调用，实现类也可以重写默认类**

**2-静态方法：**

**格式：**

**Public static 返回值类型 方法名 （参数）{**

**//方法体**

**}**

**不能通过接口的实现类的对象调用接口中的静态方法；而是，通过接口名，直接调用：**

**接口名.静态方法（参数）**

**3-私有方法：**

**普通私有方法：解决多个默认方法代码重复的问题，类不能使用 ，是私有的**

**Private 返回值类型 方法名 （参数）{**

**方法体**

**}**

**静态私有方法：解决多个静态方法间重复代码的问题**

**Private static 返回值类型 方法名 （参数）{**

**方法体**

**}**

**Tips：私有方法只能被接口调用，不能被实现类调用**

**3.使用步骤：**

1.接口不能直接实例化，需要一个实现类来实现来实现该接口：class implements 接口1，接口2...;

2.创建实现类的对象来使用

**Tips:**

1.一个类可以实现多个接口，但只能继承一个父类

2.接口的实现类必须 覆盖重写接口中所有抽象方法

如果实现类没有覆盖重写接口中所有的抽象方法，那这个实现类必须是抽象类

3.接口不能有静态代码块：static{}；接口不能有构造方法

4.实现类实现的多个接口存在重复的抽象方法，只需覆盖重写一次即可

5.只有类实现接口，接口不能实现接口

6.接口可以继承接口，但不能继承类

7.不管是不是抽象类,实现类实现的多个接口，存在重复的默认方法，那么实现类的一定要重写该方法

8.一个类如果直接父类中的方法，与接口中默认方法冲突，优先使用父类中的方法

***继承优先于接口实现***

class Zi extends Fu impelents InterfaceDemo{}

即类Fu中的方法与接口InterfaceDemo中的默认方法同名，优先使用Fu类中的方法

9.方法三要素可以随意 定义

Tips：：

接口的多继承：一个接口继承多个接口

多个父接口中的抽象方法重复没关系

多个父接口中的默认方法重复，有关系子接口必须进行默认方法的重写[而且带default关键字]

**Tips：**

面向接口的编程方式：

设计程序先设计接口，再实现类

适合多人合作

但是导致代码臃肿

### 泛型

针对不同的类有相同的处理方法；实现多态的又一种方法

优点：

1.更安全:避免类型转换的异常

2.将运行时期的异常提前到了编译时期

3.适用更广泛，针对不同的类有相同的处理方法，但类之间不一定有继承关系

Tips：

泛型只能是引用类型，因为保存的是地址值，但可以使用基本类型对应的包装类：一般首字母大写，特殊int：Integer;char:Character

**1.自定义泛型类：**<>放类名后面

定义：【修饰符】 class 类名<类型变量1，类型变量2...>｛｝

①泛型类可有多个类型变量

②类型变量类型变量指定方法的返回类型以及域和局部变量的类型

**2.自定义泛型方法：**<>放在方法名前

定义：【修饰符】 <类型变量> 返回值类型 方法名（参数）｛｝

泛型方法可以定义在普通类中，也可定义在泛型类中

当调用一个泛型方法时，在方法名前的尖括号内放入具体的类型

例：class ArrayAlg //这是一个普通类

{

Static <T> T method (T a)//两个T意义不同

{return (a.length/2);}

}.

**3.可变参数泛型方法**

. 类型变量：

①类型变量使用大写；较短

②java库中使用变量E表示集合的元素类型；K和V分别表示表的关键字和值的类型；T（或相邻U；S）可以表示任意类型

**4.泛型对象**

类名<实际类型> 对象名 = new 类名<实际类型>()

**5.定义和使用含有泛型的接口**

\* 格式:

修饰符 interface接口名<代表泛型的变量> { }

\* 如何使用:

public interface MyGenericInterface<E>{

public abstract void add(E e);

public abstract E getE();

}

**5. 泛型通配符**

\* ? 通配符: 表示任意类型，（object类）

\* 泛型的限定

\* 上限

格式: ? extends E:只允许本类(E)和子类(E子类)

\* 下限

格式: ? super E:只允许父类（E的父类）和本类（E）

**例：**

**泛型的上限：此时的泛型?，必须是Number类型或者Number类型的子类**

**public static void getElement1(Collection<? extends Number> coll){}**

泛型的下限：此时的泛型?，必须是Number类型或者Number类型的父类

public static void getElement2(Collection<? super Number> coll){}

### 集合

集合框架体系：  
 1.Collection接口:  
   List: ArrayList; LinkedList; Vector(少用)   
   Set : HashSet ; LinkedHashSet(顺序); TreeSet ;    
  
 2.Map接口：      
       HashMap ; LinkedHashMap；TreeMap；Hashtable(过时)

子接口：

①List:有序;有索引，允许重复；

②Set :HashSet与TreeSet无序,LinkedHashSet有序;不允许重复元素，无索引

Map接口： 映射

键——值对的集合

Tips：

1.\*创建集合不带泛型(默认的类型是Object)

ArrayList list = new ArrayList();

\*创建集合带泛型

ArrayList<引用数据类型> list = new ArrayList<>();

**2.数组和集合的区别**

数组:

\* 长度是不可变的

\* 数组可以存储基本数据类型,也可以存储引用数据类型

\* 整数索引

集合:

\* 长度是可以变的

\* 集合只能存储引用数据类型

\* 整数索引ArrayList集合

3.泛型只能是引用类型，因为保存的是地址值，但可以使用基本类型对应的包装类：一般首字母大写，特殊int：Integer;char:Character

4.引用类型（String除外）直接打印集合得到的是地址；

String类型集合直接打印为值；集合为空：打印[];集合不为空：打印值

关键看所存储的元素类是否重写toString方法，没有重写直接打印就是地址，重写了，按照重写，输出。

5.System.out.println();在打印时，默认调用toString()方法

.

6.list增加元素需要判断是否成功：ArrayList 不用，其他需要

ArrayList<Student> stL = **new** ArrayList<>();

System.***out***.println(**"直接打印集合名"**+stL);*//各元素的地址值*System.***out***.println(**"打印集合内某个元素"**+ stL.get(0));*//地址值*

**for** (**int** i = 0; i < stL.size(); i++) {  
 Student temp = stL.get(i);  
 System.***out***.println(temp.getName()+temp.getAge());*//元素的值*}

ArrayList<String> sL= **new** ArrayList<>();

System.***out***.println(**"直接打印集合名:"**+sL);*//值:[老三。老四]*System.***out***.println(**"打印集合内某个元素:"**+ sL.get(0));*//第一个元素值：老三*

5、集合做参数传递的也是地址值,String型集合传递的是值；

ArrayList<String> list = new ArrayList<String>;//new后的类型可省

**Collection接口常用功能**

\* public boolean add(E e): // 添加指定的对象到集合中,用于都返回true

\* public void clear() : // 清空集合

\* public boolean remove(E e): // 删除指定的对象

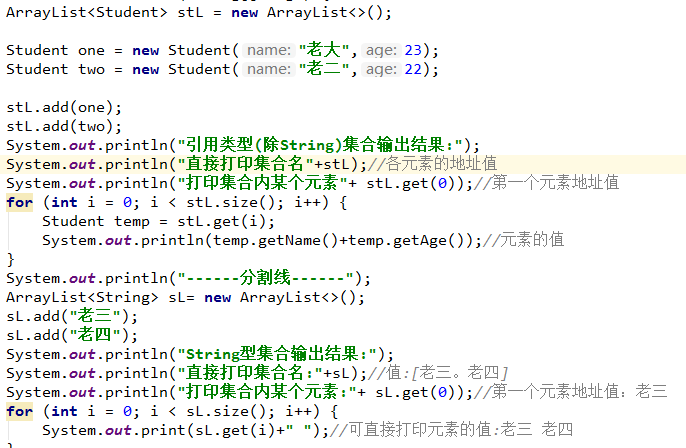
\* public boolean contains(E e): // 判断是否包含指定的对象

\* public boolean isEmpty(): // 判断集合是否为空 如果为空返回true 否则返回false

\* public int size(): // 获取集合的元素的个数

\* public Object[] toArray(): // 将集合转换为数组

\* public <T> T[] toArray(T[] a):// 将集合转换为指定类型的数组



红黑树

插入：换色旋转

#### 集合具体：

**Collection接口：**

包含一些公共方法，实现类可用

**子接口**：

**List**:增加一些根据索引进行操作的方法

**Set:**它与 Collection 接口中的方法基本一致，并没有对 Collection 接口进行功能上的扩充，只是比 Collection 接口更加严格了。与 List 接口不同的是，Set接口中无索引，无序（LinkedHashSet有序）并且都会以某种规则保证存入的元素不出现重复。

-----------------------------------------------------------------------------

##### List接口：

有序，有索引，允许重复

**tips:**

操作索引的时候,一定要防止索引越界异常

IndexOutOfBoundsException:索引越界异常,集合会报

ArrayIndexOutOfBoundsException:数组索引越界异常

StringIndexOutOfBoundsException:字符串索引越界异常

**1.ArrayList:**

查询快，增删慢。

底层数据结构：数组

底层原理是数组拷贝，增删慢。

例子：package com.my.day07.demo01.Demo02ArrayList

tips：

remove()方法是重载的；直接写数字，默认使用int index;想使用Object， 类型.valusesOf()

**2.LinkedList:**

双向链表

底层数据结构：链表

**3.Vector：**（目前很少用）

单线程同步的，效率慢，安全。

底层数据结构：数组

-----------------------------------------------------------------------------

##### Set接口：

共性：无索引，不允许重复;不能使用普通for循环；只能用迭代器和增强for循环

**1.HashSet:**

无序；

（无序:元素的存取顺序有可能不一样）

底层数据结构：哈希表： 查询和增删都比较快

1.8前： 数组+链表

1.8后：数组+链表+红黑树

--->哈希表：数组+链表(+红黑树)

**2.TreeSet:**

基于 TreeMap 的 NavigableSet 实现。使用元素的自然顺序对元素进行排序，或者根据创建 set 时提供的 Comparator 进行排序，具体取决于使用的构造方法。

**3.LinkedHashSet:**

有序（增加了一个链表，用于记录顺序）

底层数据结构：哈希表+双重链表

**\*\*\*重点：**

HashCode();String重写该方法，相等

HashSet：增加时，判断重复,根据哈希值与equlas()方法判断，

存储的元素必须重写这两方法：

TT重，其他异

利用Object类中的int hashCode()方法与boolean equals(parm)方法进行判重：

Set集合在add()一个元素时，首先会调用hashCode方法，判断要增加的元素的哈希值是否与集合内已有元素重复，如果不重复，就增加；

否则，再调用equlas()方法判断二者的值是否相同，如果值相同，就认为重复，不增加，否则，增加。

\*\*\*自定义类型元素判重：重写hashCode与equlas方法

-----------------------------------------------------------------------------

##### 可变参数：

JDK1.5新特性

1.用法：参数数据类型确定，但参数个数不确定

2.格式：参数数据类型...变量名

JDBC常用

3.底层是数组。数组用法

**tips：**

**\*一**个方法的参数列表，只能有一个可变参数

method(int ...a,int ...b)错误

\*如果方法的参数有多个，那么可变参数必须是参数列表的末尾

method(int ...a,String s )错误

\*特殊写法：

Object ...

-----------------------------------------------------------------------------

##### Collections集合工具类：

包：java.util.Collections;

功能：

1.public static <T> boolean addAll(Collection<T> c , T ...elements)往集合中添加一些元素

参数1：集合类； 参数2：可变参数 ； 可用于一次性往集合中添加多个元素

2.p s void shuffle(List<?> list);打乱List集合中元素的顺序；

3.p s <T> void sort(List<T> list)将List集合中元素按照默认规则排序（升序）

使用前提：被排序集合中存储的元素的类必须实现Comparable<T>接口，重写接口中的compareTo方法，来自定义排序的规则：

自己（this） - 参数：升序

参数 - 自己：降序

基本类型和String已重写，自定义类型一定要实现接口，重写方法

4.p s <T> void sort(List<T> list ， Comparator<? super T>):将List集合中元素按照指定规则排序

**注意：与3不同:**

Comparable:自己与参数比较，需要实现接口，重写compareTo方法

Comparator:相当于第三方判断：两个第三方参数比较，需要创建一个Compartor接口匿名内部类，重写compare方法

第一个参数 - 第二个-->升序

第二个参数 - 第一个-->降序

扩展：组合排序：

首先按数字排序，相等的再按首字母排序

----------------------------------------------------------------------------

##### Map：

1.HashMap:

底层：哈希表

特点：

无序的

键值对应，一键一值；不可同键，可同值，后来居上

可存储null

多线程的

例子：package com.my.notes.set.map

方法：

构造方法：

无参：默认容量与加载因子；

有参：容量/因子/容量，因子

功能方法：

K键类型；V值类型

1.**V put（K,V）**:将键与值对应起来；一键一值，后者居上；允许值重复

增加与修改

2.int size():获取键-值关系个数

3.boolean containsKey(K):判断是否有该键

boolean containsValues(V)：判断是否有该值

4.V remove(Object key) :删除指定键对应的值：若 没有该键返回null；有返回被删除的元素

5.V get(K):返回指定键对应的值

6.Set<K> keySet():返回键的set集合

7.Collection<V> values():返回值的Collection集合

\*8.**Set(Map.Entry<K,V>) entrySet();**

将K与V**封装成一个键值对对象（ Entry对象），**存放在一个容器

Entary是Map接口的一个内接口，它有两个方法，可获取键与值：

**K getKey(); 获取键**

**V getValue(); 获取值**

tips：这两个方法是Entary接口的方法，不是集合的，需要用Entary对象或者匿名对象来调用

**2.LinkedHashMap:**

**有序**；在HashMap下面有一个子类LinkedHashMap，它是链表和哈希表组合的一个数据存储结构

**底层：**

哈希表+双重链表（记录元素顺序）

**3.Hashtable:**

JDK1.1，1.2被取代，目前只有其子类properties活跃

**底层**：哈希表

**特点**：

**非空**：只能存储非null键与值

**同步**，单线程

Properties见IO流

**tips：**集合中只有Hashtable不能存储null，

集合中过时的：Vector ; Hashtable

集合中同步的：Vector ; Hashtable

集合中有序的：List接口：ArrayList;LinkedList;Vector;

Set接口 :LinkedHashSet

Map接口：LinkedHashMap

集合中无序的：Set接口：HashSet; TreeSet

Map接口：HashMap; Hashtable;

-----------------------------------------------------------------------------

**JDK9对于集合的优化**：

Set , List , Map三接口 of()方法，用于创建值已经确定，不准备更改的集合

只能用于这三个接口，不能用于其实现类

该种方法创建的集合，不能更改；

### 迭代器：

对集合提供了一种公共的遍历 的方式

每一个集合都有自己的迭代器

**Iterator接口的常用方法**

\* public boolean hasNext(): 判断是否有下一个元素,如果有返回true,否则返回false

\* public E next(): 取出下一个元素

**使用步骤:**

\* 使用集合对象中的iterator()获取迭代器对象

\* 判断是否有下一个元素 hasNext()

\* 如果有使用next(),将下一个元素给取出来

Collection<String> coll = **new** ArrayList<>();

Iterator<String> iterator= coll.iterator();  
**while** (iterator.hasNext()){  
 String s = iterator.next();  
 System.***out***.print(s+**"\t"**);  
}

**迭代器的实现原理**

\* 在使用迭代器进行对集合遍历的时候,首先要判断是否有下一个元素,如果有,就把下一个元素给取出来,然后在进行判断是否有下一个元素,如果有在进行取出来,以此类推,直到把集合中的元素都取出来为止.

### foreach

\* 增强for循环 方便遍历数组和集合

\* B:foreach 作用

\* 遍历打印

\* C:foreach 格式

for (数组/集合元素的类型 变量名 : 数组/集合对象 ) {

System.out.println(变量名);

}

Tips:迭代器和for增强循环只能进行查找遍历操作，不能进行其他操作，否则会报错

### 容器：

共性： ①容器类似一个空间无限的对象数组，所以可以进行对象的一系列操作

②容器可以直接输出：System.out.println()

③容器也可以使用forech循环

④形式：容器类型 元素类型 对象管理者 构造方法

其中容器类型是系统内部类

范型容器：ArrayList<元素类型>

**系统内部一个类** （类似一个空间无限的对象数组）

定义：容器类型 元素类型 对象管理者 构造方法

例 ：ArrayList<String> notes = new ArrayList<String>（）

Tips:①存放string类型的一个容器

②ArrayList<String>是类

③notes是对象管理者，即对象名，通过他调用其他域和方法

④

⑤**存入的元素按顺序排列，且会自动增加一个下标。类似数组的按下标排列**

**可通过下标（索引）访问元素**

⑥容器也可以使用forech循环，**与普通数组不同，可以改变值**

集合容器：

HashSet<元素类型> 对象名 =new HashSet<元素类型>（）

Tips：**集合容器中的元素不重复**

哈希表

是一种特殊容器

形式：HashMap<Integer,String> coinnames = new HashMap<Integer,String>（）

类 对象名 构造方法

①一个键对应一个值，键是唯一的

②**若同键多值，只有最后一个值有效**

③**<Integer,String>是键和值的类型，注意此处类型是引用类型，不能是基本元素类型**

**（如int,float…都会出错）**

④ .put（键，值）

⑤ .keySet().size得到哈希表键的个数 （键是唯一的）

## 异 常：

**概念：**指的是程序在执行过程中，出现的非正常的情况，最终会导致JVM的非正常停止。

**处理：**中断处理

Tips:语法错误不是异常

**体系：**

**根类:**java.lang.Throwable

**子类1**：java.lang.Error :错误，系统资源问题，无法处理，必须修改源代码

**子类2**：java.lang.Exception编译期异常,程序员可以进行更正，使程序正常运行

**子类1**:RuntimeException运行期异常,在运行过程中出现的，可处理也可不处理

...

**RuntimeException ：**

是那些可能在 Java 虚拟机正常运行期间抛出的异常的超类。

可能在执行方法期间抛出但未被捕获的 RuntimeException 的任何子类都无需throws子句中进行声明。

处理机制：异常要么被捕获处理，要么上抛

当发生异常时，由JVM根据异常产生的原因创建一个异常对象，存储相关信息（类型，原因，位置），如果发生异常的代码处没有异常处理逻辑，就会把异常上抛给调用者，如果调用者也没有异常处理逻辑，就会一直上抛，...直至抛给JVM虚拟机，由JVM处理：

1.打印异常信息

2.中断程序

**throw关键字：**

在指定的方法内，抛出指定的异常(制造异常:合法性判断）

格式：

**throw new** Exception或其子类(提示信息)；

推荐抛出运行期异常

1.如果是RuntimeException，可处理，也可不处理，默认交给JVM处理;如果是编译期异常，必须处理

2.如果是编译期异常，必须处理,要么throws，要么try ...catch

**处理方法：**

1. throws方法抛出异常 ：交给别人处理

2. try...catch...finaly语句 ；自己处理

Tips:如果不想处理，或者不能处理，就用throws上抛

1.throws方法：

格式：**方法头 throws Exception异常或其子类**{

...

}

Tips:

位置：throws抛出，必须位于方法声明的后面

可抛出多个异常，如果多个异常有**子父类关系，直接用父类。**

2.try...catch...finaly

格式：

try{

//可能出现异常的代码

}

catch(异常类型A e1){

//当try中出现B类型异常,就用该catch来捕获.

处理异常的代码 //记录日志/打印异常信息/继续抛出异常

}

finaly{

//扫尾工作

}

**流程：**

如果try内的语句发生异常， try块内异常后面的代码不会被执行，直接跳到catch块匹配，但final块中的语句一定会执行

**Tips：**

1.catch块：当try中出现相应的异常,就用该catch来捕获处理异常的代码

//记录日志/打印异常信息/继续抛出异常

2.finaly块：

①.**finally不能单独使用**

**②.finally块一定会执行**

**③finally内有return语句，永远返回**

当我们在try语句块中打开了一些物理资源(磁盘文件/网络连接/数据库连接等),我们都得在使用完之后,最终关闭打开 的资源----**在finaly块中关闭**

**3.多个catch连用：**

**一次捕获，多次处理**

这种异常处理方式，要求多个catch中的异常不能相同，并且若catch中的多个异常之间有**子父类异常**的关系,那么子类异常在上面，**父类异常在最后面**

**Throwable中的常用方法：**

1.public void printStackTrace() :

打印异常的详细信息。 包含了异常的类型,异常的原因,还包括异常出现的位置,在开发和调试阶段,都得使用printStackTrace。

2.public String getMessage() :

获取发生异常的原因。 提示给用户的时候,就提示错误原因。

3.public String toString() :

获取异常的类型和异常描述信息(不用)。

**自定义异常类：**

根据自己业务的异常情况来定义异常类。继承Exception或子类

继承：

Exception异常 ：编译期

RuntimeException ：运行期

所有的异常类都带有一个带异常信息的构造方法，方法内部调用父类带异常信息的虚方法，让父类来处理异常信息

**子父类异常：**

如果父类抛出了多个异常，子类重写父类方法时，抛出与父类相同的异常或者是父类异常的子类，或者不抛出异常

父类方法没有抛出异常，子类重写父类方法时也不可抛出异常。此时子类产生该异常，只能捕获处理，不能声明抛出

**父类异常什么样，子类异常什么样**

# 七、线程

### 1.进程与线程

①进程是内存中运行的程序，每个进程都有自己独立的内存空间

②线程是进程的一个执行流程。是进程的执行单元。一个进程可有多个线程，线程总是属于某个进程

③多线程共享进程的地址空间，而进程有独立的空间

④操作系统以进程为单位分配资源；同一进程内的线程共享进程的资源

⑤CPU调度以线程为基本单位

**Tips：**

**1.主线程**：执行主方法的线程

jvm执行main方法，main方法进入栈内存中,jvm开辟到cpu的执行路径,主线程

**2.新线程**：当创建新的线程对象，调用start方法时，会新开辟一个栈，再开辟一条路径，执行该线程的run方法，根据他们优先级不同，执行顺序也不同。多线程间互不影响。

3.Java使用的为**抢占式调度**，优先级高的先执行，同级别的随机

4.多次启动一个线程是非法的。

### 2. Thread线程类和Runnable接口

**(1).Thread类：**

**所属包：**java.lang.Thread

**常用构造方法：**

Thread(String name) //创建指定名称的线程

Thread(Runnable target) //创建指定目标的线程

Thread(Runnable target ,String name)//创建指定目标和名称的线程

**常用功能方法：**

**void start()**开始线程，**自动调**用重写的**run方法**

**void sleep(long millis)**使线程在指定时间内休眠

void join()等待该线程终止

boolean isAlive（）判断线程是否在活动状态

**static Thread currentThread()**返回当前线程的对象引用

**String getName()**获取当前线程的名字

**void setName()**设置线程的名字

**(2).Runnable接口：**

**抽象方法：**void run();实现该接口的类必须重写**无参run**方法

**Tips：run()方法内存放线程要执行的任务**

**Thread类**：

### 3.线程的创建：

（1）、创建线程类Thread的子类

（2）、创建实现Runnable接口的实现类

（3）、匿名类：是在前两者基础上的优化

（4）、Lambta表达式创建

**子类创建：**

1、创建子类继承Thread类，重写run方法

2、创建子类对象，调用start()方法，开启新线程,执行任务

**接口创建：**

1、创建Runnable接口实现类，重写run方法

2、创建实现类对象

3、创建Thread类对象，调用构造方法，**传入实现类对象**

4、Thread类对象调用start()方法，开启新线程，执行任务

**匿名类创建：**

**匿名内部类创建线程：**

**把子类继承父类，重写父类的方法，，创建子类对象合一完成**

**把实现类实现接口，重写方法，创建实现类对象合一完成**

**结果：子类/实现类对象，（没有名字）**

**（1）子类：** **（2）实现类：**

**new Thread(){ (Runnable r =)new Runnable(){**

**//重写run方法 //重写run方法**

**}.start(); }**

**new Thread().start();**

**(3).new Thread(new Runnable(){ //3是对2的优化**

**run();**

**}).start();**

**lambda表达式**

**new Thread(()->{…}).start();**

**Thread和Runnable区别：**

Runnable接口创建线程的好处：

1.避免了单继承的局限性（单重继承）

2.适合多个相同的程序代码的线程去共享同一个资源。

把设置线程任务和开启新线程分离

实现类 Thread类

3. 增加程序的健壮性，实现解耦操作，代码可以被多个线程共享，代码和线程独立。把设置线程任务和开启新线程分离

4. 线程池只能放入实现Runable或Callable类线程，不能直接放入继承Thread的类。

**tips：**

在java中，每次程序运行至少启动2个线程。一个是**main**线程，一个是**垃圾收集线**程。因为每当使用java命令执行一个类的时候，实际上都会启动一个JVM，每一个JVM其实在就是在操作系统中启动了一个进程。

### 4.线程安全问题：

多线程访问共享数据会产生线程安全问题。

解决：同步（提高了安全性，降低了效率）

**同步机制：**

**1. 同步代码块。**

**2. 同步方法。**

**3. 锁机制。**

**1.同步代码块：**

**synchronized** 关键字可以用于方法中的某个区块中，表示只对这个区块的资源实行互斥访问。

**格式:**

**synchronized(同步锁){**

**需要同步操作的代码**

**}**

**原理：**

线程1会检测同步代码块是否有锁，有就获取，直到执行完同步代码块，才释放锁；

线程2检测同步代码块，发现没有锁，被阻塞，等待线程1释放锁，再去获取。

**同步锁:**

对象的同步锁只是一个概念,可以想象为在对象上标记了一个锁.

1. 锁对象可以是任意类型。

**非静态同步**：

①可以新建一个Object变量做锁类型，

②也可直接用this关键字，因为代表当前对象的引用。

③方法或块的所在类

**静态同步**：本类字节码文件(类名.class对象)

2. 多个线程对象 要使用同一把锁。即保证同步锁的唯一性

**tips:**在任何时候,最多允许一个线程拥有同步锁,谁拿到锁就进入代码块,其他的线程只能在外等着

**2.同步方法:**

使用synchronized修饰的方法,就叫做同步方法,保证A线程执行该方法的时候,其他线程只能在方法外等着。

**格式1：非静态**

**public synchronized void method(){**

**可能会产生线程安全问题的代码**

**}**

**格式2：静态**

**public static synchronized void method(){**

**可能会产生线程安全问题的代码**

**}**

**3.Lock锁**

**java.util.concurrent.locks.Lock** 机制提供了比synchronized代码块和synchronized方法更广泛的锁定操作,同步代码块/同步方法具有的功能Lock都有,除此之外更强大,更体现面向对象。

需要使用方法来创建Lock类的实例：

Lock lock = new ReentrantLock();

Lock锁也称同步锁，加锁与释放锁方法化了，如下：

**public void lock() :加同步锁。**

**public void unlock() :释放同步锁。**

**使用步骤：**

1.创建Lock对象， Lock lock = new ReentrantLock();

2.由Lock对象调用lock()与unLock（）方法。一般一个在前，一个在后

### 5.进程通信：

**概念：**多个线程在处理同一个资源，但是处理的动作（线程的任务）却不相同。

**为什么要处理线程间通信：**

多个线程并发执行时, 在默认情况下CPU是随机切换线程的，当我们需要多个线程来共同完成一件任务，并且我们 希望他们有规律的执行, 那么多线程之间需要一些协调通信，以此来帮我们达到多线程共同操作一份数据。

**如何保证线程间通信有效利用资源：**

多个线程在处理同一个资源，并且任务不同时，需要线程通信来帮助解决线程之间对同一个变量的使用或操作。 就 是多个线程在操作同一份数据时， 避免对同一共享变量的争夺。也就是我们需要通过一定的手段使各个线程能有效 的利用资源。而这种手段即—— **等待唤醒机制。**

**Object对象（锁对象）**

锁对象可以是任何类型的：如byet [] byte = new byte[0];

void wait() 无限等待，

void wait(long millis); 等待指定时间，时间到时，自动醒来，进入可运行或者阻塞

void notify()唤醒单个等待线程

void notifyAll（）唤醒所有的等待线程

**唤醒之后，会继续执行wait之后的代码**

**tips：**

①wait与notifiy必须在同步代码块或同步函数中使用内，保证等待与唤醒只有一个在执行

②同步使用的锁对象必须保证唯一

③只有锁对象，才能调用wait和notify，锁对象可以是任意对象(Object)

④被唤醒线程不能立即回恢复执行，因为它在同步代码块中中断，而此刻它不持有锁，所以它需要再次尝试去获取锁，（竞争）,成功后才能在当初用wait()方法之后的地方恢复执行。

**总结：**如果能获取锁，线程waiting到runnable；否则，从等待队列出来，又进入阻塞队列,线程就从waiting到blocked

### 6.等待和唤醒机制：

就是在一个线程进行了规定操作后，就进入等待状态（wait()）， 等待其他线程执行完他们的指定代码过后 再将其唤醒（notify()）;在有多个线程进行等待时， 如果需要，可以使用 notifyAll()来唤醒所有的等待线程。

**例：生产者与消费问题**

### 7.线程池：

**一个容纳多个线程的容器，其中的线程可以反复使用，省去了频繁创建线程对象的操作，节省资源**。

Java里面线程池的顶级接口是 java.util.concurrent.Executor ，但是严格意义上讲 Executor 并不是一个线程 池，而只是一个执行线程的工具。真正的线程池接口是 java.util.concurrent.ExecutorService 。

**JDK1.5以后自动创建**

**容器-->集合**

多个线程存入集合，使用时，remove，使用完毕后，归还线程

**好处：**

1.降低资源消耗：减少创建和销毁线程的次数，可复用

2.提高响应速度，任务到达时，不必等待创建就可立即执行

3.提高线程的可管理性。可以根据系统的承受能力，调整线程池中工作线线程的数目，防止因为消耗过多内存，使服务器累趴（每个线程约1MB内存，线程开的越多，消耗的内存越大）

**创建：**

1.**java.util.concurrent.Executors.**线程池的工厂类，用来生成线程池

静态方法：

**static ExecutorService newFixedThreadPool(int nThread)**返回线程池接口

2.**java.util.concurrent.ExecutorService** 线程池接口

**1.submit(Runnable task)**提交一个Runnable任务执行

**2.shutdown()**关闭/销毁

**使用步骤：**

1.创建线程池对象： Executors工厂类创建线程池对象；返回ExecutorsService接口；

2.创建Runnable实现类 ：创建Runnable接口的实现类,重写run方法；

3.提交Runnable任务: ExecutorsService接口对象调用submit()提交一个Runnable任务执行。

tips:

1.submit()后，线程池会一直开启，

2.**线程使用后，会自动归还**

3.shuodown()摧毁线程池后，再调用submit就会报错

### 8.函数式编程：

面向过程：步骤

面向对象：对象

面向函数：目的

**函数接口：**

有且仅有一个抽象方法的接口

**Lambta表达式：**

**格式：(参数类型 参数名称) ->{代码语句}**

**组成：**

**参数：（）** --->接口中抽象方法的参数，无空，有写

**符号: ->** --->传递参数给方法体

**方法体:{}**  --->重写接口中抽象方法的方法体

**省略模式：**

**1. 小括号内参数的类型可以省略；**

**2. 如果小括号内有且仅有一个参，则小括号可以省略；**

**3. 如果大括号内有且仅有一个语句，则无论是否有返回值，都可以省略大括号、return关键字及语句分号。**

**使用前提：**

1. 使用Lambda**必须具有接口**，且要求接口中有且**仅有一个抽象方法。**

无论是JDK内置的 Runnable 、 Comparator 接口还是自定义的接口，只有当接口中的抽象方法存在且唯一时，才可以使用Lambda。

2. 使用Lambda**必须具有上下文推断。**也就是**方法的参数或局部变量类型必须为Lambda对应的接口类型，**才能使用Lambda作为该**接口的实例**。

## IO:

**路径分隔符：** pathSeparator

Window路径分隔符：分号

Linux路径分割的：冒号

**Tips：创建文件时最好使用分隔符，保证跨平台**

**文件/文件夹的路径:**  Separator

Window系统名称分隔符：\...\...\

Linux系统名称分割的: /.../.../

**绝对路径：从盘符到文件**

**相对路径：**使用当前项目的根目录

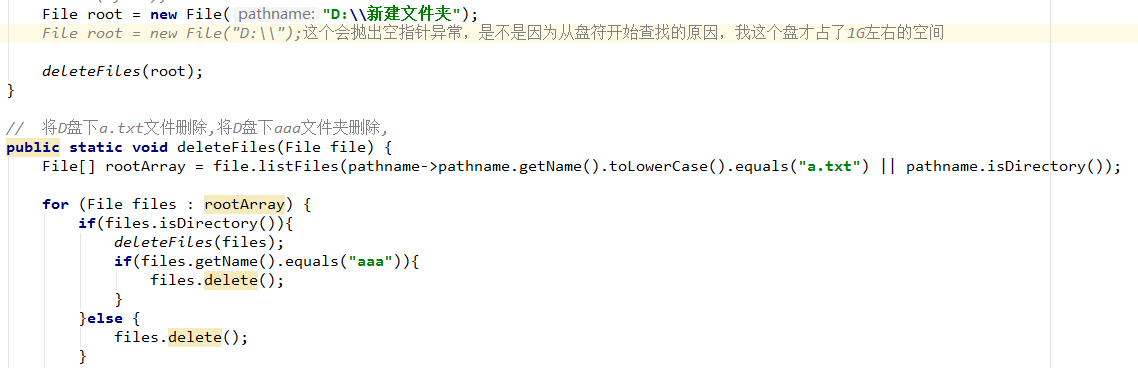
Tips：

路径不区分大小写

编写时用“\\”是因为\\是转义字符；-->’\’，

### File类：

File类的实例不可变.**package** io.code.homework.Root.



**构造函数：**

File(String name) 将字符串封装成File对象

File(String parent,String child)传入一个父路径和一个子路径

File(File parent,String child)传入一个File类父路径和一个子路径

**Tips；创建file对象时，并没有在硬盘中创建一个实际存在的文件，只有用creatNewFile与mkdir或mkdirs时，才在硬盘中创建一个实际存在的文件或者目录**

**功能方法：**

Boolean exists(),文件或目录存在true，否则false（）

Boolean isDirectory()是否是目录

Boolean isFiles()是否是文件

Boolean createNewFile(),指定文件不存在并创建成功返回true;否则,false,

如果路径不存在，会抛出异常

**Tips:需要处理异常（IO/Security +Exception）**

Boolean mkdir()创建单级目录

Boolean mkdirs()**既可以创建单级目录，也可以创建多级**

Boolean delete()删除文件或目录，仅当目录为空时才能删除目录

String getName()获取文件名（路径结尾部分）

String getParent()获取父路径

File getPatentFile() 获取父路径(不能获取相对路径的父路径，返回 null )

**Long** length()获取文件的字节，不存在返回OL（不能获取目录的，需要所有文件字节累加）

String getPath();返回构造方法封装的路径

String getAbsolutePath();获取绝对路径

**String[] list()返回字符串数组形式的抽象路径名**

**list(FilenameFileter filter)返回满足指定过滤条件的文件或目录**

**File[] listFiles()返回抽象路径名组**

**listFiles(FileFilter filter)返回满足指定过滤条件的文件或目录**

**listFiles(FilenameFileter filter) 返回满足指定过滤条件的文件或目录**

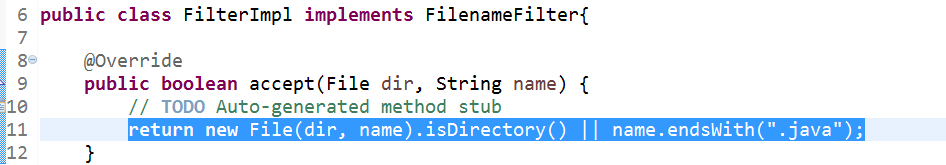
**tips:如果路径不存在或者参数不是目录，会抛出空指针异常**

**可返回隐藏文件**

### 过滤器：

**FilenameFilter接口**

boolean accept([File](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\JDK_API_1.6.CHM::/java/io/File.html" \o "java.io 中的类) dir, [String](mk:@MSITStore:C:\\Users\\Administrator\\Desktop\\JDK_API_1.6.CHM::/java/lang/String.html" \o "java.lang 中的类) name); 测试指定文件是否应该包含在某一文件列表中。 **路径** **文件名：二者合在一起才是一个完整的文件对象**



**注意蓝色部分：需要将dir与name拼接成一个File对象，再判断是否是目录**

**FileFilter接口**

boolean accept(File pathname)

FileFilter不是一个函数式接口，它一共有两个抽象方法;

### 流：

**IO将数据持久化保存在硬盘;**

#### IO流体系

流对象读取数据原理：

**程序🡪JVM🡪OS（调用方法，读取文件）🡪数据**

**按流向分：**

**输入流：**将硬盘中数据读取到内存

**输出流：**将内存中的数据写出到硬盘

**按类型分：**

**字节流与字符流**

Tips:字符流原理也是字节流

计算机一切数据都是字节

**IO流体系顶层根类**

**字节流：**

输入流：InputStream：

输出流：OutputStream

**字符流：**

输入流：reader

输出流：writer

##### 文件流

**FileInputStream:**

文件输入流，可以将文件中的内容读取出来

**构造函数：**

FileInputStream(File file)由指定的文件对象创建一个输入流

FileInputStream(String name)由指定的文件名创建一个输入流

**Tips：**

如果指定**文件不存在，**或者它**是一个目录**，而不是一个常规文件，抑或因为其他某些原因而无法打开进行读取，则**抛出 FileNotFoundException**

**功能方法：**

int read()从此输入流中读取一个数据字节

int read(byte[] b)读取最多b.length个字节

int read(byte[] b，int off,int len)在数组b中，从off开始最多读取len个字节

**tips：三者都是读取到末尾，返回-1**

long skip(long n)跳过并丢弃n个字节 数组

void close()

**FileOutputStream：**

**文件输出流，可以将内容从内存中写入硬盘中的文件。**

**如果文件不存在，就创建，存在，就清空原数据**

**构造函数：**

FileOutputStream(File file)由指定的文件对象创建一个输入流

FileOutputStream(String name)由指定的文件名创建一个输入流

FileOutputStream(File file,boolean append)增添了是否追加写入的功能

FileOutputStream(String name,boolean,append) 增添了是否追加写入的功能

**区别：**

**前二者，多次写入，是覆盖，后两者，是补充**

**构造函数作用：**

**创建对象**

**会使用传进的路径在硬盘内创建一个空文件（不存在，创建，存在，就清空原数据）**

**把流指向这个文件**

**功能方法：**

void write(byte[] b)

void write(int n)写入下一次字节;要写入的字节是参数 b 的八个低位。b 的 24 个高位将被忽略。

void close()释放资源（**流使用完毕后一定要关闭**）（**使用后，不能再写入数据，否则，报异常**）

(1).void **write（byte[] b）**

**参数：byte[] b，存储读取到的多个字节（缓冲数组）大小通常为1024或其整数倍**

**返回值：int 每次从数组中读取的字节个数**

**如果数组空间过大，会浪费空间，出现空格，可设置一个整型变量len，存储每次读取到的有效字节，然后再打印len个字节：**

**1.String类的 new String(byte[] b,int off,int len)**

**2.string(byte[] byte).trim();去空**

**如果第一个字节是正数（0-127）,查询ASCII表**

**如果第一个字节是负数，就会显示中文，查询系统默认码表（如，中文：GBK）**

**TIPS:**

**编码表：**

**GBK2个字节为一个数据； UTF-8 3个字节一个数据**

**(2).void write(int n)**

会把10进制写成二进制形式，实际上存储的是二进制数字

**虽然参数为int类型四个字节，但是只会保留一个字节的信息写出。**

**Tips:**

**任意文本编辑器打开文件时，都会查询编码表，把字节转换成字符表示**

**0-127 查询ASCII表**

**其他值：系统默认码表（如，中文：GBK）数字？？？**

**续写：在创建输出流时，多次重复使用同一个路径的文件的对象时，不是覆盖，而是续写**

**true续写，false,不续写；**

**换行：**

**Windows: \r\n**

**Mac:/r**

**Linux:/n**

##### 缓冲流：

缓冲流是在基本流对象的基础上的加强。增加一个内置默认的缓冲区（数组），减少IO次数，提高读取速度

**字节缓冲流：**

**输入流：**BufferedInputStream (extends InputStream)

**构造函数：**

(1) BufferedInputStream(InputStream in)创建一个缓冲流，传递InputStream对象

(2) BufferedInputStream(InputStream in , int size)创建一个缓冲流，具有指定大小的缓冲区

**使用：**

1、创建FileInputStream基本流对象，构造方法绑定数据源

2.创建缓冲流对象，传递FileInputStream对象，以提高基本流读取效率

3.缓冲流调用read方法，读取数据

4.缓冲流释放资源

Tips：读取数据不用flush刷新;写入需要

InputStream

**输出流：**BufferedOutputStream (extends OutputStream)

**构造函数：**

(1)BufferedOutputStream(OutputStream out)

(2) BufferedOutputStream(OutputStream out ,int size)

**参数:**OutputStream:字节输出流，

可以传递FileOutputStream，缓冲流会给它增加一个缓冲区，提高写入效率

int size:指定缓冲流内部缓冲区的大小，不指定默认

**使用：**

1.创建FileOutputStream,底层字节输出流，构造方法绑定数据的目的地

2.创建BufferedOutputStream字节缓冲输出流，构造方法传递FileOutputStream

3.使用缓冲流对象调用write方法，写入数据

4.使用缓冲流对象调用flush方法，把内部缓冲区中的数据，刷新到文件中

5.释放资源close方法

①只close缓冲流对象即可

②close方法会先调用flush刷新数据，所有第4步可省略

**字符缓冲流：**

**构造函数和使用方法类似字节缓冲流，只是添加一些特殊方法,**

**输入流**：BufferedReader

public **String readLine()** : 读取一行数据

行的终止符号：遇见下列字符之一即认为某行已终止：

换行，回车，或回车+换行

返回值：**包含该行内容的字符串，不包含任何终止符**，如果**读到末尾返回null**

**String line;**

**While((line = bor.readLine())!=null){**

**System.out.println(line)**

**}**

**输出流**：BufferedWriter

public **void newLine()** : 写一行行分隔符,由系统属性定义符号。

##### 字符流（文件）

**java.io.Writer 抽象类**是表示用于写出字符流的所有类的**超类**，将指定的字符信息写出到目的地。它定义了字节 输出流的基本共性功能方法。

void **write(int c)** 写入单个字符。

void **write(char[] cbuf)** 写入字符数组。

abstract **void write(char[] cbuf, int off, int len)** 写入字符数组的某一部分,off数组的开始索引,len 写的字符个数。

void **write(String str)** 写入字符串。

void **write(String str, int off, int len)** 写入字符串的某一部分,off字符串的开始索引,len写的字符个 数。

void flush() 刷新该流的缓冲。

void close() 关闭此流，但要先刷新它。

**输入流：FileWriter**

**构造方法：**

FileWriter(File file) ： 创建一个新的 FileWriter，给定要读取的File对象。 FileWriter(String fileName) ： 创建一个新的 FileWriter，给定要读取的文件的名称。 当你创建一个流对象时，必须传入一个文件路径，类似于FileOutputStream。

**输出流：FileReader**

**java.io.Reader 抽象类**是表示用于读取字符流的所有类的**超类**，可以读取字符信息到内存中。它定义了字符输入 流的基本共性功能方法。

public void **close()** ：关闭此流并释放与此流相关联的任何系统资源。

public int **read()** ： 从输入流读取一个字符。

public int **read(char[] cbuf)** ： 从输入流中读取一些字符，并将它们存储到字符数组 cbuf中 。

**构造方法：**

**FileReader(File file) ： 创建一个新的 FileReader ，给定要读取的File对象。** **FileReader(String fileName) ： 创建一个新的 FileReader ，给定要读取的文件的名称**。 当你创建一个流对象时，必须传入一个文件路径。类似于FileInputStream 。

##### 转换流

InputStreamReader:将字节转换成字符🡪解码 extends Reader

InputStreamWriter:将字符转换成字节🡪编码 extends Writer

构造方法：

**InputStreamReader(InputStream in)** : 创建一个使用默认字符集的字符流。 **InputStreamReader(InputStream in, String charsetName)** : 创建一个指定字符集的字符流。

构造方法：

**OutputStreamWriter(OutputStream** out**)** : 创建一个使用默认字符集的字符流。 **OutputStreamWriter(OutputStream** out**, String charsetName) :** 创建一个指定字符集的

参数：

OutputStream out :字节输出流，将转换后的字节写入到文件

CharsetName :编码，不区分大小写

字符流。

**编码问题：**

**编码**：字符转换成字节

**解码：**字节转换成字符

**字符集（编码表）**

ASCII

Unicode:

GBK:

Tips:

**GBK一个汉字两字节，UTF-8一个汉字三字节**

**数据源编码和目的地编码一直才不会乱码**

##### 序列流

1.ObjectOutputStream：

序列流：把对象以流的方式写入到文件中保存

**\* 构造方法:**

ObjectOutputStream(OutputStream out)

创建将对象写入指定OutputStream的ObjectOutputStream。

**\* 成员方法:**

void writeObject(Object obj) 将指定的对象写入 ObjectOutputStream。

**\* 使用步骤:**

1.创建ObjectOutputStream对象,构造方法中传递字节输出流

2.使用ObjectOutputStream对象中的方法writeObject,把对象写入到文件中

3.释放资源

**\* 注意事项:**

**用对象输出流写出对象,那么必须是用对象输入流读取对象**

**写入的对象所在类必须实现接口java.io.Serializable接口**

2. ObjectInputStream

对象的反序列化流:将文件中保存的对象读出

\* 构造方法:

ObjectInputStream(InputStream in)

创建从指定 InputStream 读取的 ObjectInputStream。

\* 成员方法:

Object readObject() 从 ObjectInputStream 读取对象。

\* 使用步骤:

1.创建ObjectInputStream对象,构造方法中传递字节输入流

2.使用ObjectInputStream对象中的方法readObject读取保存对象的文件

3.释放资源

4.使用读取出来的对象(打印)

transient关键字

瞬态关键字

InvalidClassException异常

\* InvalidClassException异常产生的原因 ?

\* 在进行对像的序列号和反序列的时候，我们修改了类中的字段。这个时候我们在进行反序列化的时候

首先检查文件中的序列号和对象中默认的序列号是否一致，如果不一致，说明操作的不是同一个对象，所以就会报出 InvalidClassException 异常

\* 如何解决InvalidClassException异常 ?

\*

###06 练习\_序列化集合

需求：

当我们想在文件中保存多个对象的时候

可以把多个对象存储到一个集合中

对集合进序列化和反序列化

思路：

1.定义一个存储Person对象的ArrayList集合

2.往ArrayList集合中存储Person对象

3.创建一个序列化流ObjectOutputStream对象

4.使用ObjectOutputStream对象中的方法writeObject,对集合进行序列化

5.创建一个反序列化ObjectInputStream对象

6.使用ObjectInputStream对象中的方法readObject读取文件中保存的集合

7.把Object类型的集合转换为ArrayList类型

8.遍历ArrayList集合

9.释放资源

##### 打印流

\* PrintStream特点:

1.只负责数据的输出,不负责数据的读取

2.与其他输出流不同，PrintStream 永远不会抛出 IOException

3.有特有的方法,print,println

void print(任意类型的值)

void println(任意类型的值并换行)

\* 构造方法:

PrintStream(File file):输出的目的地是一个文件

PrintStream(OutputStream out):输出的目的地是一个字节输出流

PrintStream(String fileName) :输出的目的地是一个文件路径

注意:

如果使用继承自父类的write方法写数据,那么查看数据的时候会查询编码表 97->a

如果使用自己特有的方法print/println方法写数据,写的数据原样输出 97->97

\*System.setOut(PrintStream out)

重新分配标准输出流：

使用System.setOut方法改变输出语句的目的地改为参数中传递的打印流的目的地

输出语句,默认在控制台输出

public class Demo02PrintStream {

public static void main(String[] args) throws FileNotFoundException

System.out.println("我是在控制台输出");

PrintStream ps = new PrintStream("10\_IO\\目的地是打印流.txt");

System.setOut(ps);//把输出语句的目的地改变为打印流的目的地

System.out.println("我在打印流的目的地中输出");

ps.close()；}}

==============================================================================

##### 扩展PrintWriter：

\* PrintWriter 打印字符流

\* 打印流的特点

\* 自动换行 println()

\* 不能输出字节 可以输出字节以外的内容

\* 必须是通过配置 自动刷新 (println,printf,format)

\* 包装流本身没有写出功能

\* 将字节输出流转换字符输出流

\*构造方法

\* PrintWriter(String fileName) : 指定文件的名称并创建PrintWriter对象

\* PrintWriter(Writer out, boolean autoFlush) : 根据字符输出流,并是否开启自动刷新 创建PrintWriter对象

\*特有方法

\* void write(String s): 写一个字符串

\* void print(String s): 输出字符串, 没有换行

\* void println(String s): 输出字符串并换行. 如果启动了自动刷新, 则会执行自动刷新写入数据

\* void printf(Locale l, String format, Object... args): 使用指定格式字符串和参数将格式化的字符串写入输出流. 如果启动了自动刷新, 则会执行自动刷新写入数据

\* void format(Locale l, String format, Object... args): 使用指定格式字符串和参数将格式化的字符串写入输出流. 如果启动了自动刷新, 则会执行自动刷新写入数据

##### 扩展：Properties:

**Hashtable的子类，Properties是以个属性集，他的K和V都默认是String类型**

\* **常用方法**

\* void load(InputStream in): 将文件内容加载到properties集合中

\* void load(Reader reader): 将文件内容加载到properties集合中

\* void store(Writer writer, String comments): 将properties集合中的内容写入到文件中,并添加文件的描述

\* void store(OutputStream out, String comments)：将properties集合中的内容写入到文件中,并添加文件的描述

\*读写操作步骤

\* 写出到文件

1.创建Properties集合对象,添加数据

2.创建字节输出流/字符输出流对象,构造方法中绑定要输出的目的地

3.使用store,把集合中的临时数据,持久化写入到硬盘中存储

4.释放资源

\* 从文件读取

1.创建Properties集合对象

2.使用Properties集合对象中的方法load读取保存键值对的文件

3.遍历Properties集合

注意:

1.存储键值对的文件中,键与值默认的连接符号可以使用=,空格(其他符号)

2.存储键值对的文件中,可以使用#进行注释,被注释的键值对不会再被读取

3.存储键值对的文件中,键与值默认都是字符串,不用再加引号

## 函数式编程：

**函数式接口：**

1.接口中的抽象方法只能有一个

2.可以有其他非抽象方法

检测是否是函数式接口

@FunctionalInterface

作用：

一般做方法的参数和返回值类型

参数：使用函数式接口，作为参数

返回值：返回值类型是函数式接口，可直接返回一个表达式；（返回一个接口实现类对象），作为返回值

接口的使用：

1、实现类

2.接口的匿名内部类

3.Lambta表达式

4。简化Lambta表达式

匿名内部类有，xxx$.class

**Lambta表达式没有class文件，减少内存的占用**

**Lambta表达式: ---延迟执行**

日志：可以帮助快速定位问题，记录程序运行中的情况，以便项目的监控和优化

Lambta优化性能：

lambta只是把参数传递过去，只有满足条件才会调用方法；因为延迟加载

非函数式接口：

**常见函数式接口1：**

**常用函数式接口2：**

**1.Supplier接口<T>：**

抽象方法：T get（）;用来获取一个泛型参数指定类型的对象数据。

**2.Consumer接口<T>：**

抽象方法：void accept(T t);消费一个指定泛型的数据。

默认方法：andThen

如果一个方法的参数和返回值全都是 Consumer 类型，那么就可以实现效果：消费数据的时候，首先做一个操作，然后再做一个操作，实现组合。而这个方法就是 Consumer 接口中的default方法 andThen 。

Consumer c1; Consumer c2;

c1.andThen(c2).accept(T t)

**3.Predicate接口<T>：**

对某种类型的数据进行判断，从而得到一个boolean值结果

抽象方法： boolean test(T t) 。用于条件判断的场景：

默认方法：

与 and

或 or

非 negate

**4.Function <T,R>**

接口用来根据一个类型的数据得到另一个类型的数据，前者称为前置条件，后者称为后置条件。

抽象方法：R apply(T t)

根据类型T的参数获取类型R的结果。

使用的场景例如：将 String 类型转换为 Integer 类型。

默认方法：andThen

andThen 方法，用来进行组合操作。JDK源代码如：

## Stream流：

**延迟方法：**方法返回一个新的Stream流可使用链式编程

**终结方法：**该方法使用后，流被关闭，不可使用链式编程 **count与forEach**

**\*\*\*一个流只能使用一次。**

**创建流：**

1.根据集合创建流：

Collection集合中的默认方法：

Stream<T> stream(); 返回一个T型的流

tips：Map，需要转换成Collection集合才能创建流：

1.\*keySet(); values();

2.\*entrySet()

2.根据数组创建流：

Stream接口中的静态方法of(T ...values)（可变参数）

tips：

\*int【】 arr={1,2};

Stream.of(arr);创建的是Stream<int[]>

\*Integer[] arr1 ={1,2,3};

Stream.of(arr1);创建的是Stream<Integer>

**使用流：**

1.**void forEach(Consumer<? super T> action)**;遍历一个流

2.**Stream<T> filter(Predicate<? super T> predicate)**;过滤一个流，得到一个子集

3.**<R> Stream<R> map(Function<? super T, ? extends R> mapper);** 映射获得一个新流，类型转换

5.**Stream<T> limit(long maxSize);** 截取一个流；如果流中元素长度大于maxsize则截取前maxSize个，否则不进行操作

6.**Stream<T> skip(long n)**;如果流的当前长度大于n，则跳过前n个；否则将会得到一个长度为0的空流

7.**static <T> Stream<T> concat(Stream<? extends T> a, Stream<? extends T> b)**合并两个流

tips：Stream 接口的静态方法 concat ：

合并的两个流的类型必须相同

## 方法引用：

**\*\*\*使用方法引用的前提：**

1.Lambta表达式(有函数式接口)--->用方法引用替换

2.""的左右边都存在相应的解决方案

**\*\*\*常见方法引用：**

\*\*\*以下用法仅视频演示，具体情况具体分析\*\*\*

\*\*\*方法引用就是用更简洁的代码

--->省略（参数)

--->代替Lambta表达式{}内的内容

\*\*\*一般，可根据Lambta表达式内的{}内的内容使用方法引用

\*\*\*Lambta表达式的作用：可看做一个函数式接口的实现类，（比匿名内部类更简洁，但使用条件也更苛刻： 必输是函数式接口）

--->（）内是接口中抽象方法的参数；

---> {} 内是对接口中抽象方法的具体实现

**\*\*\*代码分析：**

**接口：**

Interface MyInterface{

T/void method(有参/无参)；

}

**使用类：**

public class UseLambta{

public static void main(String[] args) {

show(【参数】，（【参数】->{

//接口中方法的具体实现 //方法的具体实现

}）;)

}

public void/T show(【有参/无参】，MyInterface mi){

【return】 mi.method(【有参/无参】); //只有调用作用，不关注方法的具体实现}}

**分析：**

主函数main中调用show函数，show函数再利用用函数式接口对象调用接口中的函数：

传递相应的参数：函数式接口参数位传入Lambta表达式（匿名类/实现类）,Lambta{}内的内容是对接口中抽象方法的实现，

**1.打印引用：**

**形式**：**System.out::println**

适用于Lambta表达式{}内：打印输出

**2.静态方法引用：**

**形式：静态类名::静态方法**

适用于Lambta表达式{}内：使用静态方法

如：Math::abs

**3.对象引用：**

**形式：对象::对象所在类的方法**

适用于Lambta表达式{}内：某类的对象调用本类中的方法

如： Person pc = new Person();

pc::method

**4.类的构造器引用**

**形式：类名::new**

适用于Lambta表达式{}内：创建一个类的实例

**5.数组构造器引用**

**形式：数组类型::new**

适用于Lambta表达式{}内：创建一个数组

**6.super引用**

**形式：supper::父类方法**

适用于Lambta表达式{}内：调用父类方法

**7.this引用**

**形式：this::本类方法**

适用于Lambta表达式{}内：调用本类方法

## 软件测试：

黑盒测试：功能测试，数据驱动测试，是一种从用户观点出发的测试

　 黑盒测试主要测到的错误类型有：不正确或遗漏的功能；接口、界面错误；性能错误；数据结构或外部数据访问错误；初始化或终止条件错误等等。

白盒测试：结果测试，逻辑驱动测试；是针对被测单元内部是如何进行工作的测试

它根据程序的控制结构设计测试用例，主要用于软件或程序验证。

扩展：

\* @Before:

\* 修饰的方法会在测试方法之前被自动执行

\* @After:

\* 修饰的方法会在测试方法执行之后自动被执行

\* 即使方法出现异常，Before和After修饰的方法也会执行

Junit使用：白盒测试

使用步骤：  
 **1.定义一个测试类(测试用例)**     \* 建议：测试类命名规范：被测试的类名Test  
        包名：xxx.xxx.xx.test

**2.定义测试方法：可以独立运行**

建议：命名规范： test+被测试的方法

返回值：void

参数：无参

**3.给测试方法加注解：**@Test

如果报错，是因为缺少junit依赖环境：需要导入junit包

**4.判断测试结果**：

红色：失败

绿色：成功

一般使用断言判断结果，而不是输出语句（因为可能有输出，但不是期望的，也是绿色）

**Assert断言类静态方法：**

Assert.assertEquals(**期望的结果,运算的结果**);

## 注解：

**概念：**说明程序的。给计算机看的；JDK1.5之后的新特性

**定义：**注解（Annotation），也叫元数据。一种代码级别的说明。它是JDK1.5及以后版本引入的一个特性，与类、接口、枚举是在同一个层次。它可以声明在包、类、字段、方法、局部变量、方法参数等的前面，用来对这些元素进行说明，注释。  
**使用注解：**@注解名称

**\* 作用分类**：  
    **①编写文档**：通过代码里标识的注解生成文档【生成文档doc文档】

Javadoc xxx.java生成JavaDoc文档  
 **②代码分析：**通过代码里标识的注解对代码进行分析【使用反射】  
 **③编译检查：**通过代码里标识的注解让编译器能够实现基本的编译检查【Override】**JDK中预定义的一些注解**  
   **\* @Override：**检测被该注解标注的方法是否是继承自父类(接口)的  
   **\* @Deprecated：**该注解标注的内容，表示已过时  
  **\* @SuppressWarnings**：压制警告  
       一般传递参数all  @SuppressWarnings("all") **自定义注解**  
 **\* 格式：**  
      元注解  
      public @interface 注解名称{  
         属性列表;//实际上是若干抽象方法  
      }

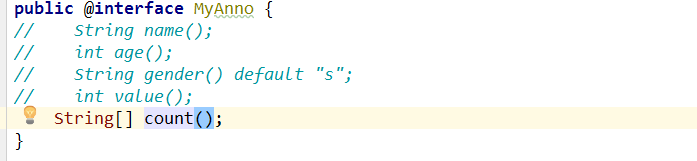
**\* 本质：**注解本质上就是一个接口，该接口默认继承Annotation接口  
    \* public interface MyAnno extends java.lang.annotation.Annotation {}  
   **\* 属性**：接口中的抽象方法

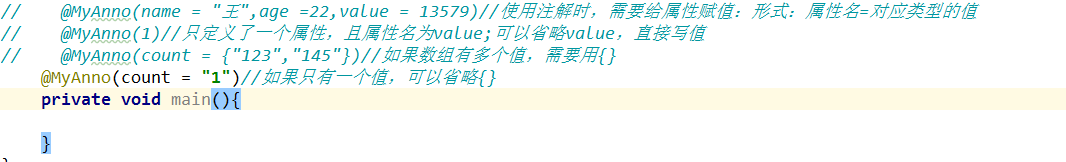
**使用注解时，需要给属性赋值*：***

***形式：属性名=对应类型的值*** *\* 注解中使用default的属性，在使用时可以不赋值，因为有默认值了  
\* 如果注解中只定义了一个属性，且属性名为value，那么在使用时，可以省略value，直接写值*

**注解中属性的类型：** *\* 基本类型；String;枚举；注解类型以及上述类型的数组类型  
\* 数组型的属性,如果数组有多个值，需要用{}；如果只有一个值，可以省略{}*

**示例：**





## 反射：

\* 框架：半成品软件。可以在框架的基础上进行软件开发，简化编码  
\* 反射：将类的各个组成部分封装为其他对象，这就是反射机制  
   **好处**：  
     1. 可以在程序运行过程中，操作这些对象。  
     2. 可以解耦，提高程序的可扩展性。  
      java代码三阶段：  
        源代码阶段：  XXX.java--->XXX.class

Tips：通过该类.getClassLoader方法将字节码加载到内存中

        class对象阶段：将类的各部分封装成对象  
        Runntime阶段：

获取Class对象的方式：  
   1. Class.forName("全类名")：**\*\*将字节码文件加载进内存，返回Class对象\*\***  
       \* 多用于配置文件，将类名定义在配置文件中。读取文件，加载类  
   2. 类名.class：通过类名的属性class获取  
       \* 多用于参数的传递  
   3. 对象.getClass()：getClass()方法在Object类中定义着。  
       \* 多用于对象的获取字节码的方式  
    \* 结论：同一个字节码文件(\*.class)在一次程序运行过程中，只会被加载一次，不论通过哪一种方式获取的Class对象都是同一个。

\* Class对象功能：

**1.获取成员变量们**

    \* Field[] getFields() ：获取所有public修饰的成员变量  
    \* Field getField(String name)获取指定名称的 public修饰的成员变量

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

\* Field getDeclaredField(String name) 获取指定名称的成员变量

**2.获取构造方法：**

Constructor<?>[] getConstructors()

Constructor<T> getConstructor(Class<?>... parameterTypes)//形参字节码

Constructor<T> getDeclaredConstructor(Class<?>... parameterTypes)

Constructor<?>[] getDeclaredConstructors()

**3.获取成员方法**

Method[] getMethods()

Method getMethod(String name,Class<?> ...parameterTypes)方法名，形参字节码

Method[] getDeclaredMethod()

Method getDeclareMethod(String name ,Class<?> ...parameterTypes)

(方法名， 方法形参类型.class)

**Tip：**

无返回方法：有输出的，自动输出，不需要设置变量接收，接收打印也是null

有返回方法：需要一个变量来接收方法执行的结果，再打印输出，否则，不会自动输出

**扩展：**

**暴力反射：**

成员变量，构造方法，成员方法都有暴力反射的方法，用于获取私有的成员

当使用忽略类型的获取方法时，会获取四种类型的成员，包括private，但private是私有的，直接获取会报错：

所以需要使用暴力反射来获取：

Filed/Constructor/Method/.setAccessible(**true**);

一般用在真正使用Filed/Constructor/Method之前，刚获取之后

**Filed：**

1. 设置值

\* void set(Object obj, Object value) （对象，值）

2. 获取值

\* Object get(Object obj) (对象)

3. 忽略访问权限修饰符的安全检查

\* setAccessible(true):暴力反射

**Constructor:构造方法**

\* 创建对象：

\* T newInstance(Object... initargs) （初始化值）

如果是空参，就创建的是空参构造器，操作可以简化：

Class对象的newInstance

**方法:**

\* Method：方法对象

\* 执行方法：

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

\* 获取方法名称：

\* String getName:获取方法名

**public class** IntegaterCase {  
  
 **public static void** main(String[] args) **throws** Exception {  
 *//1.创建属性集* Properties pp = **new** Properties();  
 *//2.加载配置文件  
 //2.1获取配置文件* InputStream is = IntegaterCase.**class**.getClassLoader().getResourceAsStream(**"pro.properties"**);  
 *//2.2属性集加载配置文件* pp.load(is);  
 *//释放资源* is.close();  
 *//2.3获取类路径和方法名* String className = pp.getProperty(**"className"**);  
 String methodName = pp.getProperty(**"methodName"**);  
  
 *//利用构造器创建对象（有/无参）无参可以直接使用类创建 ，简洁  
 //Constructor constructor = className.getClass().getConstructor();//(class<?> ...)  
 //Object o1 = constructor.newInstance();//(与上面的类型对应的构造函数实参)  
  
 //3.开始使用  
 //3.1加载该类进入内存* Class cls = Class.*forName*(className);  
 *//3.2创建对象* Object obj = cls.newInstance();  
 *//3.3创建方法对象* Method method = cls.getDeclaredMethod(methodName);*//(方法名，形参：类型.class)  
 //3.4安全起见，暴力反射* method.setAccessible(**true**);  
 *//3.5使用方法* method.invoke(obj);*//invoke(对象，实参)这里定义的是无参，无返回方法  
// T result = method.invoke(对象，实参) 如果方法有返回值，需要接收  
// System.out.println();* }  
}

# 八、数据库：

**数据库：**

按照某种数据结构存储，管理数据的文件系统

**特点：**

DOS常用命令：

常见问题：

？？？？**1.数据库无法插入中文：**

解决：修改数据库或表的字符集

### MySQL:

#### 安装与卸载：

#### 基础使用：

启动：

**\*1.手动：**

\*手动：计算机管理-服务

\*cmd:services.msc(开启)计算机服务-手动开启

**\*2.自动：**

**启动MySQL服务**：**net start MySQL(服务名)** （默认是mysql；如果安装时修改了，需要使用修改过的，如我的是：mysql57）

**关闭MySQL服务**：**net stop MySQL(服务名)**

2.登录与退出：

**\*本地登录：**

**mysql -u 用户名 -p 密码**

会提示输入密码：enterpassword: 输入密码

tips:

需要先启动mysql服务

**???非管理员模式下，不能直接使用，需要进入MySQL的bin目录,才能使用**

**\*远程登录：**

需要设置可以远程登录：

**前提是设置过**

Enable root access from remote machines

**mysql -hIP地址 -u root -p 密码（目标的IP与密码）**

**mysql --hostIP地址 --user=用户 --password=目标的密码**

**\*退出：**

**exit**

**quit**

**/q**

MySQL目录结构：

**\*安装目录**：

bin :二进制可执行文件

date:数据：日志

lib:运行需要的库文件

share：错误文件

my.ini：配置文件

**\*数据目录：**

数据库：文件夹

表：文件

数据：表中数据

#### SQL:

Structured query language: 结构化查询语言

\*定义了操作所有关系型数据库的规则；

\*方言：每一种数据库操作的方法存在不一样的地方

\*\*通用语法：

1.SQL可单行或多行书写，**分号结尾**

2.可使用空格和缩进增强可读性

3.SQL语句不分区大小写：**建议关键词大写**

tips:**只有mysql>目录下的命令才是SQL语言**

4.**注释：**

单行注释： -- 内容或 #内容(**-- 后必须加个空格；#随便**)

多行注释：/\* 内容\*/

SQL分类：

DDL数据库定义语言：操作数据库和表：对数据类，表的修改（宏观）

DML数据库操作语言：操作表中的记录：增删改

DQL数据查询语言：操作表中的记录查询

DCL数据控制语言：权限（主要关于授权与安全访问）

##### DDL：

1.操作数据库：(CRUD)

\*创建：

1.创建数据库：**create database** 新建数据库名**;**

**create database if not exists** 新建数据库 > 先判断是否存在，不存在再创建 >(安全性高)

**create database** 新建数据库 **character set** 字符集名 >创建数据库时指定编码

综合：

create database db1 if not exists character set GBK;

tips:创建数据库默认编码是utf -8

\*查询：

1.显示所有数据库：**show databases;**

2.查看某个数据库的创建语句与字符集：**show create database** 数据库名

可以查看创建数据的命令(包括编码)

\*修改：

1.修改数据库字符集：**alter database** 数据库 **character set** 字符集**；**

\*删除：

1.删除数据库：**drop database** 数据库**；**

**drop database if exists** 数据库；>判断数据库存在？存在再删除

\*使用：

1.查询当前正在使用的数据库：**select database();**

2.使用数据库: **use** 数据库**；**

2.操作表：

\*C:

**1.创建表：**

create table 表名**（**

列名1 数据类型1**，**

列名2 数据类型2**，**

。。。

列名n 数据类型n **-- 最后一行不用加逗号**

**）；**

**数据类型：**

int 整数

double （最大位，小数位）小数

date 日期 >只包含年月日

datetime 日期 >年月日时分秒

timestamp 时间戳

tips：**如果不给赋值或赋值null，默认使用系统当前时间**

varchar(字符长度) 可变字符串 **一个汉字一个中文都只占一个字符**

**不常用：**

TEXT 大型文本：一般把磁盘路径存入数据库，利用IO流读取

BINARY(字节) 大型图片

2.**复制表：create table** 表名 **like** 被复制的表名；

\*R:

1. **查询某个数据库所有表**：**show tables；**

**tips：需要先进入某个数据库**

2. **查询表结构：**

**desc 表名；**

\*U

1.**修改表名**： **alter table** 表名 **rename to** 新表名**；**

2.**修改数据库字符集**:**alter** **table** 表名 **character set** 字符集**；**

3.**增加列**：alter **table** tablename **add** 列名 数据类型**；**

4.修改列名称，和类型

修改：**alter table tablename** **change** **旧列名** **新列名** **新数据类型**；  **>列和类型都改**

**alter table 表名** **modify** **列名 新数据类型**； **>只改类型**

5.删除列：**alter table** 表名 **drop** 列名**;**

\*D

1.删除表：**drop table** 表名**；**

**drop** **table if exists** 表名**；**

##### DML：

**1.插入：**

给指定列添加值: Insert into 表名(列1，列2...) values(值1，值2...)

给所有列添加值：insert into 表名values();//不写列，默认添加所有

**tips:**

\*列值对应

\*如果表名后没写列名，默认给该行所有列增加值；不全给，会报错。

insert into 表 values (值1，值2...)；

\*单双引都行：除了数字类型，其他类型需要使用引号括起来

**2.删除：**

**Delete from 表名 [where 条件]**//

Tips：【】内容可省，默认删除整条记录---***慎用***

**Truncate table 表名**：先删除表，再创建一个一模一样的空表

**3.修改：**

**Update**

1**.update 表名 set 列1 = 新值1， 列2 = 新值2...【where 条件】；**

tips：如果不加条件，会默认修改表中所有记录

##### DQL：

###### 单表查询：

查询表中的记录：

**完整语法：**

**select**

**字段列表**

**from**

**表名列表**

**where**

**条件列表**

**group by**

**分组字段**

**having**

**分组之后的条件**

**order by**

**排序**

**limit**

**分页限定**

**1.基础查询：**

1.查询某表所有记录（所有字段）：select \* from 表

2.查询指定字段：select 要查询的列1，列2， from 表名；

3.**结果去重：select distinct 列1，列2... from 表名；**

4.**计算列值：**

计算一些列的值（通常为数值型）

select 列1，列2。。。，列1 运算符 列2 from 表名；

**tips：如果某一个值为null，结果也为null；解决：ifnull(可能为null的列,0)**

select 列1，列2。。。，列1 运算符 ifnull(列2,0) [[as] 结果别名】 from 表名；

\*哪个列对应的值可能为null，就用这个函数

5. 别名：两种形式：

1.as 别名；

2.别名（省略as）,

tips :不管计算的结果可以使用别名，前面的列也可以使用别名

\*\*\*（数据库中实际名并没有修改，只是显示效果）

**2.条件查询：**

where 条件语句

**运算符：**

关系运算符:>,<,>=,<=,=,<>

逻辑运算符:推荐使用英文

*and --&&*

*or --||*

*not --!*

is null

in(集合)：可替代or，**in（值1，值2...）**

\*like 模糊查询

占位符：

\* \_：单个任意字符

\* %：多个任意字符

示例：

select \* from student where name like '马%'; 查询以'马'开头的姓名

select \* from student where name like '—化%'；查询第一个字符任意，第二个字符为'化',后面字符任意的姓名

select \* from student where name like '\_\_\_';查询姓名为三个字的

select \* from student where name like '%马%'；查询姓名中包含‘马’的人

**tips：**

1.or 的简洁用法:-->in()

where age =22 or age =18 or age =25;

where age in (22,18,25)

两种写法功能相同，但第二种更简洁；

2.null：

**null值不能使用=（！=）判断；可使用is(is not) null**

3.简化写法：

between ... and 在..之间 ：-->...and ...或...&&...

4.<> \*不等于

**3.排序查询：**

**order by 排序字子句**

语法：

order by 字段1 排序规则1 ， 字段2，排序规则2...；

\*可根据多字段排序，**优先使用第一个字段的排序规则，只有按排序规则1排序遇到相等时，启动排序规则2....**

\*没有写排序规则，默认升序

**ASC：升序**

**DESC：降序**

**4.聚合函数**

将一列数据作为一个整体，进行纵向计算

**select 函数（字段） from 表名；**

1.count()计算个数

1.一般选择非空列：主键

2.count(\*)，只要该行有数据，就计入个数。

2.min()

3.max()

4.avg()

5.sum()

tips:聚合函数的计算，排除null值

1.选择不包含空的列进行计算，

2.IFNULL函数()：比如math列中有一条记录为null，avg(ifnull(math,0))

**5.分组查询：**

**grounp by 分组字段**

**select 分组字段 math ,eng from stu group by 分组字段；**

tips：

\*分组之后查询的字段应该是:分组字段，聚合函数,其他没有意义

\*tips：

分组中where与having 的区别

WHERE 条件子句： **\*先筛选，再分组;** **\*后不可跟聚合函数**

HAVING 分组后条件子句 ：**先分组，后筛选；** **\*后可跟聚合函数**

SELECT sex 性别,math 数学,count(id) 人数 FROM table WHERE math<70 GROUP BY sex HAVING count（id）>4;

筛选出数学成绩小于70的，统计人数，按照性别进行分组，分组之后，显示人数大于4的组。

**6.分页查询：**

**limit 开始索引 ，每页显示记录**

\*公式：开始索引=（当前页-1）\*每页显示页数

\*limit是MySQL的方言，其他数据库不能用（其他数据库有自己的分页方式）

示例：

SELECT \* FROM stutable LIMIT 0,3; --第一页

SELECT \* FROM stutable LIMIT 3,3; --第二页

SELECT \* FROM stutable LIMIT 6,3; --第三页

###### 多表查询：

**内连接：去除无用的数据**

**隐示内连接： 语法：select 字段/\* from 表1 ， 表2 where 条件；**

**显式内连接**

**语法：select 字段/\* from 表1 [inner] join 表2 on 条件**

**Tips：如果超过两张表，表与表之间仍然用on连接**

**内连接tips\_1**

**1.需要查询那些表**

**2.查询条件是什么**

**3.需要查询哪些字段：实际中，并不需要所有字段**

**内连接tips\_2**

**1.条件的引号可有可无**

**2.两个表中重名列可用：表名.列来区分**

**3.建议给表起表名使用，更简洁**

**外链接：**

**左外链接：**

**语法：select 字段/\* from 左表 left [outer] join 右表 on 条件**

**功能：查询左表所有及交集**

**Tips：1.交集实际上是右表中的一个字段，由where条件判断**

**右外链接：**

**语法：select 字段/\* from 左表 right [outer] join 右表 on 条件**

**功能：查询右表所有及交集**

**Tips：**

**别名.\*为该别名代表的表的所有字段**

**子查询:**

**一个查询语句中嵌套另一个查询语句，嵌套查询称为子查询**

1.子查询的结果是单行单列

**子查询在where后做条件，使用运算符参与判断**

2.子查询的结果是多行单列

**结果相当于一个数组；子查询可以作为条件，使用in判断**

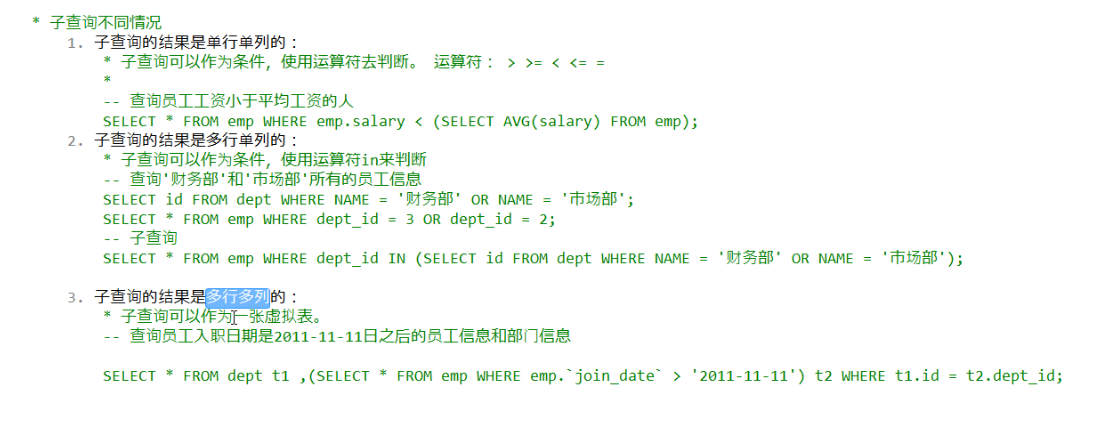
3.子查询的结果是多行多列

**子查询可以作为一张虚拟表参与查询**

**Tips：**

**1.子查询作为表需要取别名，否则这张表没有名称则无法访问表中的字段**

**2.该虚拟表使用时需要用（）括起来**



**内连接和外链接的关系：**

**外链接可以看做是在内连接基础上将X表所有内容（包括null）都显示**

##### DCL

数据库控制语言：主要用于用户管理，权限控制

MySQL中mysql数据库里的user表存放用户信息

用户管理：

增加用户：

**create user ‘用户名’@’主机名’identified by ‘密码’**

删除用户：

**drop user ‘用户名’@‘主机名’；**

查询用户

1**.需要先使用mysql数据库**

2.select \* from user；

Tips：%是通配符，代表任意主机

修改用户：

两种修改用户密码的方式：

**1.Update user set password = password(‘新密码’) where user =‘用户名’;**

**2.set password for ‘用户名’@‘主机名’= password(‘新密码’)；**

**Tips：1.mysql5.7后没有password字段，换成authentication\_string**

**2.FLUSH PRIVILEGES;刷新**

**3.set方式仍可用**

\*忘记密码：

1.管理员权限运行cmd:

2.停止MySQL服务

3.使用无验证方式启动服务 **mysqld –skip-grant-tables**

4.新打开一个cmd窗口：输入mysql，便可登录

5.登录后修改密码

6.任务管理器结束mysqld 服务后，新开cmd启动服务便可正常使用

权限控制：

查询

**show grants for ‘用户名’@’主机名’；**

授予

**grant 权限列表 on 数据库名.表名 to ‘用户名’@’主机名’；**

**授予全部权限：**

**Grant all on \*.\* to ‘用户名’@’主机名’**;

撤销

**Revoke 权限列表 on 数据库名.表名 from ‘用户名’@’主机名’;**

**Tips:**

**如果之前授予用户某种权限后，撤销全部权限方式，无法撤销第一次的权限**

#### 约束：

**概念：**对表中的数据进行限定，保证数据的正确性，有效性，完整性。

**分类：**

1.主键约束：primary key

2.非空约束：not null

3.唯一约束：unique

4.外键约束:foregin key

##### 非空约束：

概念：

使用：

1.创建表时添加：create...（

name varchar(8) nut null, -- 添加非空约束

）

2.创建表后，添加：

ALTER TABLE 表名 modify 被约束的列 类型 not null;

3.删除：ALTER TABLE 表 modify 列 类型

##### 唯一约束：

-1-.MySQL中唯一约束限定的列的值可以有多个NULL；

-2-.删除唯一约束：**ALTER TABLE 表名 DROP INDEX 唯一约束限定的列**；

-3-.创建表后，添加唯一约束：ALTER TABLE 表名 MODIFY 将被约束的列 数据类型 UNIQUE;

-4-.创建表时添加唯一约束：create table stu(id int,

name varchar(8) UNIQUE -- 添加了唯一约束

)

##### 主键约束：

主键是表中记录的唯一标识。

**\*非空且唯一**

**\*能够唯一标识一条记录**

\*一张表只能有一个字段为主键：如：ID

使用：

-1-.创建表时添加：create table stu(id int PRIMARY KEY, -- 添加了主键约束

name varchar(8) UNIQUE -- 添加了唯一约束

-2-.删除主键：**ALTER TABLE 表名 DROP PRIMARY KEY**;

-3-.创建表后，添加主键：ALTER TABLE 表名 MODIFY 将被约束的列 数据类型 PRIMARY KEY;

扩展：

**自动增长：**

**概念：**如果某一列的主键是数值型的，可以通过auto\_incream自动增长

auto\_increment

例：id INT PRIMARY KEY AUTO\_INCREAM,

**\*tips :**

1.设置主键自动增长后，在插入记录时可以给主键插入null，因为自动增长，

2.自动增长是根据上一条记录的值，变化

3.自动增长一般和主键一起使用

**删除自动增长：**

ALTER TABLE 表名 MODIFY 主键 数据类型；-- 只删除了自动增长，没有删除主键

**添加自动增长：**

ALTER TABLE 表名 MODIFY 主键 数据类型 AUTO\_INCREAM；

##### 外键约束：

让表与表产生关系，从而保证数据的正确性

**使用：**

**1.创建表时添外键；**(

dep\_id, -- 注意分号

【CONSTRAINT 外键名】 FOREIGN KEY (本表的外键列) REFERENCES 表2（表2的主键或者唯一约束的列）

)

**2.删除外键：**ALTER TABLE 表名 DROP FOREIGN KEY 外键名;

**3.创建表后，添加外键：**

ALTER TABLE 表1 ADD 【CONSTRAINT 外键名】 FOREIGN KEY (本表的外键字段) REFERENCES 表2（表2的主键或者唯一约束的列）

-------------------------------------------------------------------

**\*\*\*总结：**

modify添加:非空约束，主键约束，自动增长，唯一约束

add 添加:外键约束

modify删除:非空约束， 自动增长

drop 删除:唯一约束，主键约束，外键约束

-------------------------------------------------------------------

**\*扩展：**

**级联更新：**

**用途：**

对副表的操作，会自动更新到主表的：

比如，employee员工表 和 department部门表，员工表的dep\_id是外键，他关联department的主键id：

如果需要对部门表的主键ID进行操作，希望操作的结果能自动跟新到员工表的外键，可以使用级联更新：

**\*添加级联操作：**

alter table employee add CONSTRAINT fk FOREIGN KEY (dep\_id) REFERENCES department (id) ON UNDATE CASCADE ,ON DELETE CASCADE;

**\*分类：**

**级联跟新：** ON UPDATE CASCADE

**\*级联删除：** ON DELETE CASCADE

**慎重**使用级联删除：因为可能多个表互相关联，使用级联删除一个表的内容时，与之相关联的表的内容也会被删除

**多表操作：**

**多表关系：**

1.一对一

2.一对多

3.多对多

**实现关系：**

1.实现一对多：

**在多关系的一方建立外键关联1关系一方的主键**

2.实现多对多：

**需要借助第三张表来实现**

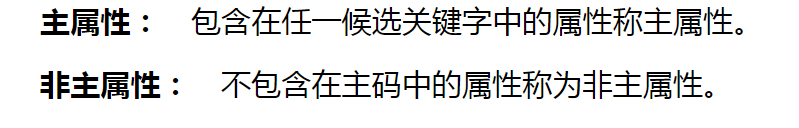
**中间表中至少两个字段**：m表的主键，n表的主键，他们作为第三张表的外键分别关联到m,n表；

联合主键： primary(m,n)

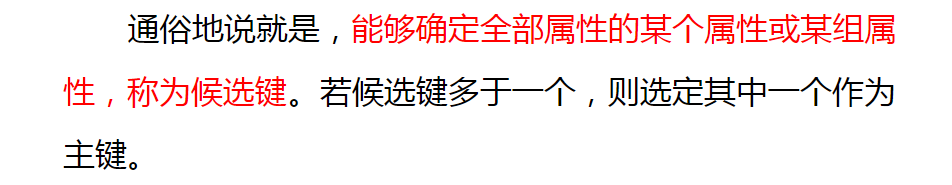
3.实现一对一：

可在任意一方添加唯一外键（unique限定）关联另一方的主键

#### 数据库设计：



**候选码**

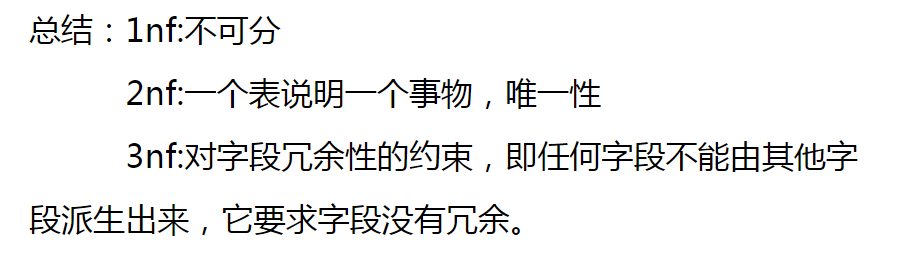


**范式**：

第一范式：具有**原子性**  
第二范式：主键列与非主键列遵循**完全函数依赖关系**（非主键和主键相关，**特别是联合主键的情况**）  
第三范式：非主键列之间没有**传递函数依赖关系**

Tips：

1NF:字段不可分;   
2NF:有主键，非主键字段完全依赖主键;   
3NF:非主键字段不能相互依赖;



#### 备份和还原：

方式：

1.命令行

2.图形化工具

命令行：

备份：

**Mysqldump –u用户名 –p 密码 要备份的DB > 存储地址**

**TIPS:备份不需要登录mysql**

还原：

①新建一个空数据库： create database 新DB；

②使用该数据库： use 新DB；

③还原： **source 要还原的数据库地址；**

图形化工具：以sqlyog为例

备份：

右键选择要备份的DB----备份数据库----选择备份地址----确定

还原：

执行SQL脚本---选择之前备份的地址----执行

#### 数据库常用设置：

1.修改密码

2.修改字符集

3....

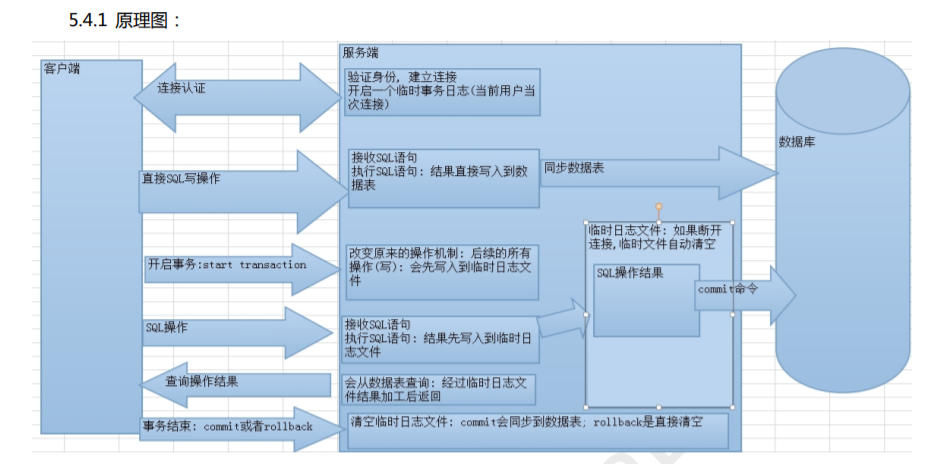
#### 事务：

概念：

事务执行是一个整体，所有的 SQL 语句都必须执行成功。如果其中有 1 条 SQL 语句出现异常，则所有的 SQL 语句都要回滚，整个业务执行失败。

简单说：**一个业务有多个步骤，被事务管理，则这些步骤要么同时成功，要么同时失败**

##### 事务原理：



事务的步骤：

1) 客户端连接数据库服务器，创建连接时创建此用户临时日志文件

2) 开启事务以后，所有的操作都会先写入到临时日志文件中

3) 所有的查询操作从表中查询，但会经过日志文件加工后才返回

4) 如果事务提交则将日志文件中的数据写到表中，否则清空日志文件。

扩展：

回滚点：

##### 提交方式：

自动提交：

\*MySQL是自动提交

\*一条**DML**（增删改）语句会**自动提交**一次事务

**\*\*\*->Tips:使用start transaction开启事务后，是手动提交**

-----------------------------------------------------------------------------

手动提交：

\*Oracle 是手动提交

\*手动提交下，使用一条DML语句后，需要**commit手动**提交事务，将修改持久保存DB； 否则，只是暂时改变数据。

设置默认方式：

查看默认方式：**show @@autocommit;** **1->自动提交 ； 0->手动提交**

修改默认方式：**set @@autocommit = 0/1 ;**

##### 事物特征：

1.原子性：事务的操作要么同时成功，要么同时失败

2.持久性：一旦事务执行成功，对数据库的修改是持久的。就算关机，也是保存下来的。 3.隔离性：多个事务之间，相互独立（实际上，互相关联，可通过隔离控制来保证）

4.一致性：事务操作前后，总量不变

##### 事务隔离：

###### 概念：

事务在操作时的**理想状态**：所有的事务之间保持隔离，互不影响。因为并发操作，多个用户同时访问同一个数据，可能引发并发访问的问题：

简述：**多个事务访问同一个数据引发并发访问的问题**

###### 问题：

|  |  |
| --- | --- |
| 脏读 | **一个事务**读取到了**另一个事务**中**尚未提交**的数据 |
| 不可重复读 | **同一个事务中**，两次读取的**数据内容不一致**， （幻读是数量不一致） |
| 幻读 | 一事务DML一个表，另一事务**增，删数据**，导致第一个事务无法查询修改 |

**解决：设置隔离级别**

###### 隔离级别：

1**.read uncommitted 读未提交** |**问题**：脏读、虚读、幻读

2.**read committed**  **读已提交** |**问题**： 虚读、幻读 |Oracle默认

3.**repeatable read** **可重读** |**问题**： 幻读 |MySQL默认

4.**serializable** **串行化** |**问题**：无

**Tips：级别从低到高，安全性越高，效率也越慢**。

1.查看隔离级别：

**select @@tx\_isolation;**

2.设置隔离级别

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

# 九、JDBC

### 概念：

Java数据库连接

JDBC是sun 公司提供的操作所有关系数据库的一种规范，是一种接口，由各大数据库厂商去提供实现类（驱动包）。

### 一般步骤：

导包 ：项目新建一文件夹-将数据库驱动 jar包复制进来—右键—add labray;

注册驱动: Class.*forName*(**"com.mysql.jdbc.Driver"**);

连接对像 Connection conn=DriverManager.*getConnection(url,user,pwd)*

创建执行对像：Statement st = conn.createStatement();

SQL语句 String sql =”...”

执行SQL语句st.executeUpdate()/executeQuerey()

处理结果 :根据需求进行操作:

释放资源:Statement,Connection,ResultSet

相关类与方法：

1.Class类的

getClassLoader()不管能够加载字节码文件，而且能够加载项目中的文件

URL getResourec(String name);返回一个资源的统一资源路径

GetResourceAsStream(String name)以流的形式获取文件

**---Java.sql 包---**

2.DriverManager --类

包： java.sql.DriverManager

方法：

1.s v registerDriver([Driver](mk:@MSITStore:E:\\黑马下载文件\\002%20Java基础第二阶段（2018版）\\01\\资料\\jdk%20api%201.8_google.CHM::/java/sql/Driver.html" \o "interface in java.sql) driver)注册给定的驱动器DriverManager

2.s Connection getConnection(String url)尝试建立与给定数据库的连接

s Connection getConnection(String url,String user,String pwd)

参数：

url :jdbc:mysql://IP地址（主机名）:端口号/数据库名

user:数据库用户名

pwd :密码

3.Connection –接口

获取SQL语句执行对象

Statement createStatement(); //无参

PrapareStatement PrapareStatement(String ...)//有参

4.Statement --接口

常用方法：

Boolean Execute(String sql)执行范围最广

Int ExecuteUpdate(String sql)

执行DML（返回受影响行数）和DDL语句（返回0或无结果）

ResultSet ExecuteQuery(String sql)

返回一个结果集

5.PrepareStatement --接口

它是StateMent接口子接口

6.ResultSet –接口

表示数据库结果集的数据表，通常通过执行查询数据库的语句生成。

**Boolean next()**判断是否还有下一行：false末尾，true还有下一行

While(rt.next){...}

**package** wxssl.jtools;  
  
**import** java.io.FileReader;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.net.URL;  
**import** java.sql.\*;  
**import** java.util.Properties;  
  
*/\*  
\* JDBC工具类  
\* 1.将注册驱动获取连接对象封装  
\* 2.封装释放资源方法  
\* \*/***public class** JdbcTools {  
 **private static** String *url*;  
 **private static** String *user*;  
 **private static** String *password*;  
 **private static** String *driver*;  
 **private static** Connection *conn*;  
  
 *//读取配置文件* **static**{  
 *loadPro*();  
  
 *//注册驱动* **try** {  
 Class.*forName*(*driver*);  
 *conn* = DriverManager.*getConnection*(*url*, *user*, *password*);  
 } **catch** (ClassNotFoundException e) {  
 e.printStackTrace();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **private static void** loadPro() {  
 *//创建属性集集合* Properties p = **new** Properties();  
  
 *//以流的形式获取配置文件* InputStream is = JdbcTools.**class**.getClassLoader().getResourceAsStream(**"jdbc.properties"**);  
 **try** {  
 p.load(is);  
 *//赋值  
 url* = p.getProperty(**"url"**);  
 *user* = p.getProperty(**"user"**);  
 *password* = p.getProperty(**"password"**);  
 *driver* = p.getProperty(**"driver"**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }**finally** {  
 **try** {  
 *//释放字节流* is.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 */\*\*  
 \* 获取连接对象  
 \** ***@return*** *连接对象  
 \** ***@throws*** *SQLException  
 \*/* **public static** Connection getConnection() **throws** SQLException {  
 *//返回连接对象* **return** *conn*;  
 }  
  
 */\*\*  
 \* 释放资源  
 \** ***@param st*** *\** ***@param conn*** *\** ***@param rt*** *\*/* **public static void** close(Statement st, Connection conn, ResultSet rt){  
 **if**(rt!=**null**){  
 **try** {  
 rt.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **if**(st!=**null**){  
 **try** {  
 st.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **if**(conn!=**null**) {  
 **try** {  
 conn.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

### JDBC连接池:

**概念**：连接池是一个容器（集合），存放数据库连接对象

**本质：**在系统初始化完成后，申请连接数，当访问数据库时取出连接对象，使用结束后归还

数据库连接池工具：

#### C3P0

**步骤：**

**1.导入jar包：**

C3po-0.9.5.2.jar 和machange-commons-java-0.2.12.jar

数据库驱动包

**2.定义配置文件：**

C3p0.properties或c3p0-config.xml

\***路径：直接将文件存放在src目录,名称不变**

**3.创建核心对象**：数据库连接池对象

**DataSource** ds = **new CombpooledDataSource（【配置名】）；**

**Config—配置名，不写，使用默认的。**

**4.创建连接对象: Connection conn = ds.getConnection();**

Close();当连接对象是从连接池中获取的，close方法是归还资源而不是释放资源

**【5.归还资源】**：conn.close();池子里的是归还，不是释放

Tips :

1.C3P0的配置文件名只能是C3p0.properties或c3p0-config.xml；而且必须位于类路径classpath下（src下的文件会放在class内），否则，会找不到配置文件

2.C3P0的配置文件会自动加载。

配置文件：

一个配置文件中config：使用时不指定名字，就使用默认配置

ds.getConnection(“配置文件的名字”)

-配置文件名

-初始连接数

-最大连接数

-超时时间

#### Druid

数据库连接池实现技术

步骤:

1.导入jar包:druid-1.0.9.jar

2.定义配置文件

Properties形式； 名称任意；位置任意

3.加载配置文件：

Properties p = new Properties();

InputStream is= XXX.class.getClassLoader().getSourceAsStream(“配置文件名”)

p.load(is);

**4.获取连接池对象：通过工厂类：**

**DataSource ds = DruidDataSourceFactory.createDataSource(p)；**

5.获取连接：

Connection = ds.getConnection

#### 工具类

1. 定义一个类 JDBCUtils  
  2. 提供静态代码块加载配置文件，初始化连接池对象  
  3. 提供方法

1. 获取连接方法：通过数据库连接池获取连接  
  2. 释放资源

      3. 获取连接池的方法

#### Spring Jdbc

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

步骤：

1. **导入jar包**

2. **创建JdbcTemplate对象**。**依赖于数据源DataSource**

\* **JdbcTemplate** template = new JdbcTemplate(**ds**);

3. **调用JdbcTemplate的方法来完成CRUD的操作**

**\*update**(String sql,Object ...args):执行**DML语句**。增、删、改语句

**\*queryForMap():**

将结果集封装为**map集合**，**列名为key**，**值为value** 将这条记录封装为一个map集合

参数：

**1.String sql, [Object ...args]**

**2.String sql**

**\* 注意：这个方法查询的结果集长度只能是1**

**\* queryForList():**

查询结果将结果集封装为**list集合**

**参数：**

**1.String sql, Class<T> elemType**

**2.String sql,Object ...args**

**3.String sql**

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

**\* query(String sql,RowMapper<T> row)**:查询结果，将结果封装为**JavaBean对象**

\***参数：RowMapper**

1.常用**BeanPropertyRowMapper实现类**。可以完成数据到JavaBean的自动封装

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

**2.也可自定义匿名内部类**

**\* queryForObject(Strin sql,Class<T> requiredType)**：

查询结果，将结果封装为**对象**

\* **一般用于聚合函数的查询**

\*参数：

String :字符串形式的sql语句

Class<T>：需要的字节码类型

# 十、Html

# Css

**1. 概念：** *Cascading Style Sheets* **层叠样式表**

\* **用于页面美化和布局控制**

\* 层叠：多个样式可以作用在同一个html的元素上，同时生效

**2. 好处：**

1. 功能强大

2. 将内容展示和样式控制分离

\* 降低耦合度。解耦

\* 让分工协作更容易

\* 提高开发效率

3. CSS与html结合方式

1. 内联样式

\* 在**标签内使用style属性**指定css代码

\* 如：<div style="color:red;">hello css</div>

2. 内部样式

\* **在head标签内，定义style标签**，style标签的标签体内容就是css代码

\* 如：

<style>

div{color:blue;}

</style>

<div>hello css</div>

3. 外部样式

1. **定义css资源文件**。

2. 在head标签内，定义link标签，引入外部的资源文件

\* 如：

**\* a.css文件：**

div{color:green;}

**\*html文件内引用：**

**<link rel="stylesheet" href="css/a.css">**

<div>hello css</div>

<div>hello css</div>

\* 注意：

\* 1,2,3种方式 css作用范围越来越大

\* **优先级（范围）不同，高优先级会覆盖低优先级的样式**

\* 1方式不常用，后期常用2,3

\* 第3种格式可以写为：

<style>

@import "css/a.css";

</style>

4. css语法：

\* 格式：

**选择器** {

属性名1:属性值1**;**

属性名2:属性值2;

...

}

\* 选择器:筛选具有相似特征的元素

\* 注意：

\* 每一对属性需要使用；隔开，最后一对属性可以不加；

5. 选择器：**筛选具有相似特征的元素**

\* 分类：

**1. 基础选择器**

1. id选择器：选择具体的id属性值的元素.建议在一个html页面中id值唯一

**#id属性值{} ---#号**

2. 元素选择器：选择具有相同标签名称的元素

**标签名称{}**

\* 注意：**id选择器优先级高于元素选择器**

3. 类选择器：选择具有相同的class属性值的元素。

**.class属性值{} ---“.”号**

\* 注意：**类选择器优先级高于元素选择器**

2. 扩展选择器：

1. 选择所有元素：

**\*{}** ---**“\*”号**

2. 并集选择器：

**选择器1,选择器2{}** ---**逗号“，”连接**

3. 子选择器：筛选选择器1元素下***所有的***选择器2元素

**选择器1 选择器2{}**

4. 父选择器：筛选选择器2的父元素选择器1 **---儿子级**

**选择器1 > 选择器2{}**

5. 属性选择器：选择元素名称，属性名=属性值的元素

**元素名称[属性名="属性值"]{}**

6. 伪类选择器：选择一些元素具有的状态

元**素:状态{}**

\* 如： <a>

\* 状态：

\* link：初始化的状态

\* visited：被访问过的状态

\* active：正在访问状态

\* hover：鼠标悬浮状态

**6. 属性**

**1. 字体、文本**

\* font-size：字体大小

\* color：文本颜色

\* text-align：对其方式

\* line-height：行高

**2. 背景**

\* background：

**3. 边框**

\* border：设置边框，符合属性

**4. 尺寸**

\* width：宽度

\* height：高度

**5. 盒子模型：控制布局**

\* **margin**：外边距

\* **padding**：内边距

\* **默认情况下内边距会影响整个盒子的大小**

\* box-sizing: border-box; 设置盒子的属性，让width和height就是最终盒子的大小

\* **float：浮动**

\* left

\* right

# JavaScript

**概念：** 一门**客户端脚本语言**

**特点：**

\* **运行在客户端浏览器中的**。每一个浏览器都有JavaScript的解析引擎

\* 脚本语言：不需要编译，直接就可以被浏览器解析执行了

**功能：**

\* 可以来**增强用户和html页面的交互过程**，可以来**控制html元素**，让页面有一些动态的效果，**增强用户的体验。**

**JavaScript = ECMAScript + JavaScript自己特有的东西(BOM+DOM)**

# Jqurey

jQuery是一个快速、简洁的JavaScript框架它封装JavaScript常用的功能代码，提供一种简便的JavaScript设计模式，优    化HTML文档操作、事件处理、动画设计和Ajax交互。

## JQ基本使用：

1.导包：项目中（模块）导入Jquery.XXX.js包

2.资源（网页文档）中引入外部资源：

<**script src="js/jquery-3.3.1.min.js"**></**script**>

3.使用

## \*JS与JQ转换

        1. JQuery对象在操作时，更加方便。  
        2. JQuery对象和js对象方法不通用的.  
        3. 两者相互转换  
            \* jq -- > js : jq对象[索引] 或者 jq对象.get(索引)  
            \* js -- > jq : $(js对象)

## \*选择器

1. 基本操作学习：  
            1. 事件绑定  
                //1.获取b1按钮  
                $("#b1").click(function(){  
                    alert("abc");

                });

            2. 入口函数  
                 $(function () { });

window.onload  和 $(function) 区别  
          \* window.onload 只能定义一次,如果定义多次，后边的会将前边的覆盖掉  
          \* $(function)可以定义多次的。  
3. 样式控制：css方法  
                 // $("#div1").css("background-color","red");  
                  $("#div1").css("backgroundColor","pink");

2. 分类  
       1. **基本选择器**  
              1. 标签选择器（元素选择器）  
                    \* 语法： $("html标签名") 获得所有匹配标签名称的元素  
              2. id选择器   
                    \* 语法： $("#id的属性值") 获得与指定id属性值匹配的元素  
              3. 类选择器  
                   \* 语法： $(".class的属性值") 获得与指定的class属性值匹配的元素  
              4. 并集选择器：  
                    \* 语法： $("选择器1,选择器2....") 获取多个选择器选中的所有元素  
         2.**层级选择器**  
               1. 后代选择器  
                    \* 语法： $("A B ") 选择A元素内部的所有B元素          
               2. 子选择器  
                    \* 语法： $("A > B") 选择A元素内部的所有B子元素  
         3. **属性选择器**               1. 属性名称选择器   
                    \* 语法： $("A[属性名]") 包含指定属性的选择器  
               2. 属性选择器  
                    \* 语法： $("A[属性名='值']") 包含指定属性等于指定值的选择器  
               3. 复合属性选择器  
                    \* 语法： $("A[属性名='值'][]...") 包含多个属性条件的选择器  
        4. **过滤选择器**  
                1. 首元素选择器   
                    \* 语法： :first 获得选择的元素中的第一个元素  
                2. 尾元素选择器   
                    \* 语法： :last 获得选择的元素中的最后一个元素  
                3. 非元素选择器  
                    \* 语法： :not(selector) 不包括指定内容的元素  
                4. 偶数选择器  
                    \* 语法： :even 偶数，从 0 开始计数  
                5. 奇数选择器  
                    \* 语法： :odd 奇数，从 0 开始计数  
                6. 等于索引选择器  
                    \* 语法： :eq(index) 指定索引元素  
                7. 大于索引选择器   
                    \* 语法： :gt(index) 大于指定索引元素  
                8. 小于索引选择器   
                    \* 语法： :lt(index) 小于指定索引元素  
                9. 标题选择器  
                    \* 语法： :header 获得标题（h1~h6）元素，固定写法  
            5. **表单过滤选择器**                1. 可用元素选择器   
                    \* 语法： :enabled 获得可用元素  
                2. 不可用元素选择器   
                    \* 语法： :disabled 获得不可用元素  
                3. 选中选择器   
                    \* 语法： :checked 获得单选/复选框选中的元素  
                4. 选中选择器   
                    \* 语法： :selected 获得下拉框选中的元素

## \*内容操作：

Text() ：获取与设置文本内容

Html() ：获取与设置标签体内容

Val() ：获取与设置 value属性

## \*属性操作：

1.通用属性操作：

1)attr()

一个参数：获取属性 $(“#id”).attr(“color”)

两个参数：设置属性 $(“#id”).attr(“color”,”red”)

removeAttr()

删除指定的属性: removeAttr(“被删除属性名”)

2) prop()

一个参数：获取属性

两个参数：设置属性

removePop()

删除属性

**区别**：

1.如果操作的是元素的**固有属性**，则**建议使用prop**

2.如果操作的是元素**自定义的属性**，则**建议使用attr**

**基本上都是attr()**

**固有属性**：html文档中标签自带属性

自定义属性：

单选和多选框的**checked**属性，下拉列表的**selected,disabled**,**abled**属性**不能用attr**

2.class属性操作

AddClass() 给选中的元素添加class属性值,从而设置其样式

RemoveClass() 删除class属性值

**ToggleClass**() 切换class属性值： 有-无

## \*样式控制：

**Css():**

1个参数：获取css属性值

**2个参数：设置元素的css属性**

**1）css写法： “background-color” “red”**

**2）js 写法： “backGroundColor” ”red”**

**只有多单词写法不同**

## \*CRUD

1. append():父元素将子元素追加到末尾  
       \* 对象1.append(对象2): 将对象2添加到对象1元素内部，并且在末尾

2. prepend():父元素将子元素追加到开头  
       \* 对象1.prepend(对象2):将对象2添加到对象1元素内部，并且在开头  
3. appendTo():  
       \* 对象1.appendTo(对象2):将对象1添加到对象2内部，并且在末尾  
4. prependTo()：  
       \* 对象1.prependTo(对象2):将对象1添加到对象2内部，并且在开头

5. after():添加元素到元素后边  
       \* 对象1.after(对象2)： 将对象2添加到对象1后边。对象1和对象2是兄弟关系  
6. before():添加元素到元素前边  
     \* 对象1.before(对象2)： 将对象2添加到对象1前边。对象1和对象2是兄弟关系

 7. insertAfter()  
     \* 对象1.insertAfter(对象2)：将对象2添加到对象1后边。对象1和对象2是兄弟关系  
8. insertBefore()  
    \* 对象1.insertBefore(对象2)： 将对象2添加到对象1前边。对象1和对象2是兄弟关系

9. remove():移除元素  
        \* 对象.remove():将对象删除掉  
10. empty():清空元素的所有后代元素。  
         \* 对象.empty():将对象的后代元素全部清空，但是保留当前对象以及其属性节点

## 动画的显示和隐藏：

三种方法：**参数语法相同：三个参数都可写可不写**

**Speed**:动画的速度，**预定义**的三个字符串（**slow,normal,fast**）或者**指定的毫秒**

**Easing**:指定切换效果：默认**swing**;**变速显示**(慢-块-慢) ；linear：**匀速显示**

**Fu:** 动画完成时，执行的**函数**：function(){};，每个元素只能执行一次

***默认：normal，swing***

1)默认显示/隐藏

**Show(“speed”,[easiing],[fu])**

**Hide()**

**Tagglle()**

2)上下滑动显示/隐藏

solidDown())

solidUp()

solidTaggle ()

3)淡入淡出显示/隐藏

fadeOut()

fadeIn()

fadeTaggle ()

## \*遍历:

JS遍历：for 循环

JQ遍历：

**CallBack为回调函数**

1.**JQ对象 .**each( **callBack** )

**2.$.each**( **Object** ,callBack )

Object:既可以是JS对象，也可以是JQ对象

CallBack：回调函数

citys.each(**function** () {

*alert*(**this**.**innerHTML**); *//* ***this是DOM对象****，可以使用innerHtml  
 alert*($(**this**).html()); *//* ***将this转换成JQ对象****，使用html()*})

带参数的回调函数：

citys.each(**function** (index,element) {*//index是索引,element是DOM对象  
 alert*(index+**":"**+element.**innerHTML**);  
 *alert*(index+**":"**+$(element).html());  
})

回调函数：

**返回值：**

1）true 🡪 continue **跳过本次循环**

2）false 🡪 break **结束循环**

citys.each(**function** (index,element) {

**if**(**"上海"**==$(element).html()){  
 **return true**; *// continue* }**else** {  
 **return false**; *// break* }  
 *alert*(index+**":"**+$(element).html());  
})

1,2用法相似

For...of : **Jquery 3.0版本及以上才能用**

for(元素对象 of 容器对象)

**for** (s **of** citys){  
 *alert*(s.**innerHTML**); *// li是JS对象*

}

**Tips:3.0版本才能使用**

**For in： ？？？**

**For (var key in 数组（对象）){**

**}**

## \*事件：

**1.*一般：***

1.JQ中给一个元素绑定多个事件，可使用**链式编程**（但不亦太多）

$(**"#name"**).mouseover(**function** () {  
 *alert*(**"你走了"**);  
}) **.** mouseout(**function** () {  
 *alert*(**"你来了"**);  
})

2.如果绑定事件，未传递回调函数。***会触发浏览器默认行为****：*

*如：focus,submit事件*

**2. *on、off 解除和绑定事件***

**on** (**“绑定的事件名”，回调函数**)

**off** (**“被解除的事件名”**)

tips:off()无参时，解除所有事件

$(**function** () {

$(**"#btn"**).on(**"click"**,**function** () {

*alert*(**"使用on绑定点击事件"**);  
 })

$(**"#btn2"**).on(**"click"**,**function** () {

$(**"#btn"**).off(**"click"**);

})

});

3.事件切换

Toggle(f1,f2...)

参数：函数，可定义多个，**循环切换**：f1-f2-...-fn-f1..

**高版本的JQuery需要引入一个插件**jQuery Migrate（迁移）插件可以恢复此功

<script src="../js/jquery-migrate-1.0.0.js" type="text/javascript" charset="utf-8"></script>

# BootStrap

# XML

可扩展标记语言

# Tomcat

## 安装目录：



**重要目录：**

1）Bin:存放可执行二进制文件（window和linux都有）

2）<conf:配置文件：可以在其子目录localhost中自定义XML文件；从非webApps目录> 中引入Web项目

Tips:可以被注解：@WebServlet代替

**3**）**lib:**存放jar包：如JDBC。**名称固定为lib**，在IDEA项目中，**在WEB-INF新建lib**

**web服务器**： (servlet容器)

**功能**：监听端口，读取浏览器请求，返回请求的数据；如：**tomcat;**

## 常见问题：

1）**窗口闪关**：java\_home配置错误：

2）**端口占用**：

1.修改端口（麻烦）

2.**关闭占用端口的进程**

**netstat -nto**

## 部署项目

**共三种方式：**

1）. 直接将项目放到webapps目录下即可。

\* /hello：项目的访问路径-->虚拟目录

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

\* war包会自动解压缩



2）. 配置**conf/server.xm**l文件(了解)

在<Host>标签体中配置

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

\* docBase:项目存放的路径

\* path：虚拟目录

**Tips：不推荐**，因为易导致web.xml出错，导致所有web项目无效

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

如：aaa.xml:

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

\* D:\hello：

\* **虚拟目录**：**xml文件的名称**

配置完成后，可通过浏览器访问服务器中的资源：

**Localhost:8080/虚拟目录/资源名**：**localhost:8080/aaa/index.html**

**Tips：浏览器访问服务器资源：**

**IP:端口/虚拟目录/资源名**

## IDEA配置Tomcat:

1.IDEA:-->Run-->Edit Configures -->点击加号，选择Tomcat

2.配置虚拟目录：

Edit Configures-–Tomcat--Deployment:--Application context

“/”代表项目就在Tomcat根目录，浏览器访问时，可不用虚拟目录

Tips：

一个服务器中有多个web项目，若都是用“/”会引起冲突，所以可为不同的模块（项目）

设置不同的虚拟目录；

## IDEA创建JavaEE项目

new JavaEE --

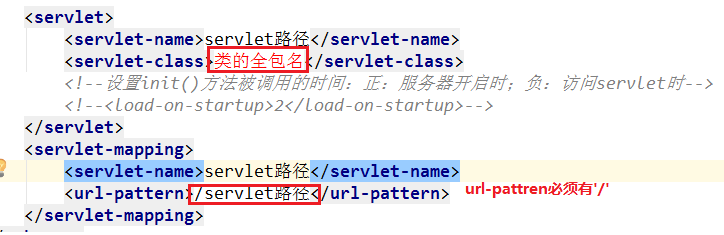
选择：javaee 7即可

选中：web application

选中：web.xml

创建实现servlet接口的实现类，实现接口五个抽象方法

在**web.xml**中配置servlet: [servlet路径改为自定义名称]



## 原理：

Tomcat服务器根据浏览器输入的url路径获取（解析）servlet路径；

在web.xml中匹配servlet路径，将相应的类的字节码加载进内存

创建该类的对象，调用方法

**利用注解配置servlet**



利用**@WebServlet()**注解**可替代在web.xml中配置**；属性值是：**url-pattern**

## 小技巧：

1. IDEA会为每一个tomcat部署的项目单独建立一份配置文件

UsingCATALINA\_BASE "C:\Users\fqy\.IntelliJIdea2018.1\system\tomcat\\_itcast"

2. 工作空间项目和tomcat部署的web项目

\* tomcat真正访问的是“tomcat部署的web项目”，

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

\* WEB-INF目录下的资源不能被浏览器直接访问。

**Web项目必须在WEB-INF目录外，否则无法访问**

**WEB-INF下可创建子目录lib存放各种jar包;**

**Html包可与web目录平级**

3. 断点调试：使用"小虫子"启动 dubug 启动

# Servlet

## 概述：

Javax.servlet包

他是一个接口，定义了java类被浏览器访问的规则

5个抽象方法：

1).init()

初始化，多用于加载资源；

\*只能执行一次

当servlet被访问时调用;也可设置调用为服务器开机时启动：

web.xml:

**值为正，服务器开启时调用**

**值为负，servlet被访问时创建**

2).service()

提供服务，

\*可多次执行

3).destroy()

销毁：服务器关闭前执行

**只有正常关闭服务器才会被调用**

---------------------------------------------------

## servlet3.0:

**使用servlet3.0可不创建web.xml，因为可通过注解代替配置servlet**

给方法添加注解：@webServlet

重要属性：url-pattern:servlet的资源路径

\* 步骤：

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

2. 定义一个类，实现Servlet接口

3. 复写方法

4. 在类上使用@WebServlet注解，进行配置

\* @WebServlet("资源路径")

---------------------------------------------------

## servlet体系结构：

**servlet接口 ：定义了5个抽象方法**

**|继承**

**GenericServlet接口**：实现了servlet4个抽象方法，**留下一个service方法**

**|继承**

**HttpServlet接口 :**实现了剩下的service方法，又定义了许多新方法：

**|** 为解决不同浏览器对代码的**兼容性**：根据提交方式：定义了许多新方法：

**|**   **如：doGet(resquest,response)**

**|继承**  **DoPost(...)**

**|** 二者功能相同，故可以只重写其中一个，然后在另一个中调用另一个

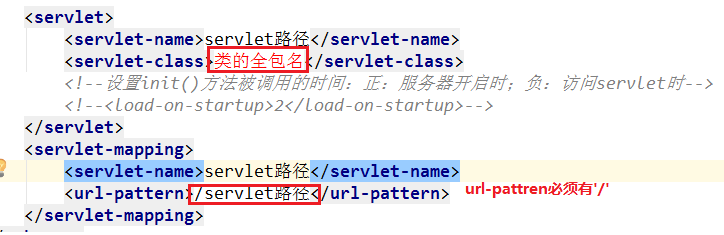
**| 注意post方法的乱码问题**:requset.setCharacterEncoding(“utf-8”)

**自定义servlet实现类：根据需要重写相应的方法，完成不同的需求**

---------------------------------------------------

servlet相关配置：

**1.xml配置：**



**2.注解**

**servlet访问路径： -------非 / 即 \***

urlpattern配置：\* @WebServlet("资源路径")

**1.单层目录（完全路径匹配）：**

1）"/路径名"

2）可为一个servlet配置多个urlpattern:{"/路径1"，"路径2"，"..."}

**2.多层目录（目录结构）：**

1）'/父路径/子路径'

2）"/\*" -->任意路径都可以 ---**优先级最低**

**3.\*通配符 （扩展名匹配）：**

1）\*.XX -->1）**不能加'/'**

2)代表任意以XX为尾的路径

-------------------------------------------------------------------------------

# Http

-----------------------------------------------------

## 概述：

超文本传输协议

**规定客服端和服务端通信的数据的格式**

**特点：**

重要协议：TCP/IP

默认端口：80

基于请求响应模型：一次请求对应一个响应

无状态：不同请求相互独立，不会发生数据交互

如：请求一个某网页后，服务器响应，返回该网页，但是当该网页在我们获取服务器响应后，发生了更新，我们只有刷新（再次请求）才能获取最新的内容

**版本：**

1.0：每一次请求都会建立一个连接

**\*1.1：复用连接**

## 双R原理：

## Request

### 请求消息数据格式：

**1.请求行:**

格式： **请求格式 请求url 请求协议/版本**

如：GET /login.html HTTP/1.1

**2.请求头:**

1·**浏览器告诉服务器的相关信息**

2·格式： 请求头：请求头值

如：Host : locahost

3·请求头介绍：

Host: 主机名  
**User-Agent**: 浏览器基本资料  
Accept: 浏览器能够识别的响应类型  
Accept-Language: 浏览器默认语言  
Accept-Encoding: 浏览器能够识别的压缩方式

Connecton：是否保持连接

Referer: 来路页面，

**Referer:**

告诉服务器，请求的来源

1).防盗链：判断请求来源是否合法，防止其他网站盗取链接（不是法律）

2).统计工作：如打广告时，统计不同来源广告吸引来的流量

**3.请求空行**

就是一个空行，分割请求头和请求体

**4.请求体**

**封装post请求的参数**

如：

<form action="..." method="POST">

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

<button type="sumbit">提交</button>

</form>

我们在输入框中输入的数据，就会成为请求体中的内容:

如：输入王以，请求体就是：

username=王以

\*tips：

1），

**GET:方法没有请求体**

**POST:方法有**

2），字符串格式：

### 体系结构：

ServletRequest ---接口

HttpServletRequest ---子接口

org.apache.catalina.connector.RequsetFacade -- 实现类（Tomcat中的）

### 功能：

获取请求消息内容

**1.获取请求行:**

**请求行： GET /requestDemo/demo01?name = zhangsan HTTP/1.1**

1）获取表单提交方式：String getMethod() ---GET

2）**获取 虚拟目录**： String getContextPath() ---/requestDemo

如果没有配置，值为空字符串

3) 获取 servlet路径：String getServletPath() ---/demo01

4) **获取GET方式请求参数**：String getQueryString() ---name = zhangsan

5) 获取请求uri:

String getRequest**URI**() --- **/requestDemo/demo01**

StringBuffer getRequest**URL**() - **-http://localhost/requestDemo/demo01**

6) 获取协议和版本： String getProtocol() --- HTTP/1.1

tips :

获取用户IP： String getRemoteAddr()

示例：



**2.获取请求头：**

1）**String getHeader(String name)** ---获取**指定请求头**的**请求值**

2）**Enumeration<String>** getHeaders() ---**获取所有请求头的名称**

tips：

**1）获取的只是名称，而不是值，仍需要第一个方法获取**

2）**可当做迭代器使用,方法：**

**3.获取请求体（参数）**

只有POST方式有请求体，在其中封装了POST请求的参数

步骤：

**1.获取流对象:**

1）BufferedReader getReader()

2)ServletInputStream getInputStream()

**2.再从流中获取数据：**

**详细：**

**BufferedReader br = request.getReader();**

**String line = null;**

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

**System.out.println(line);**

**}**

**输出的内容即为post方法的请求参数**

\*tips：

**get方式的请求参数在请求行中： String getQueryString()**

**通用获取请求参数的方法：**

1)， **String getParameter(String name)** --根据参数名称获取参数值

2)， **String[] getParameterValues(String name)**--根据参数名称获取参数值的数组；

tips：（同键不同值）;多用于获取多个相同name的参数，如：多选框

3)， **Enumeration<String> getParameterNames()**-- 获取所有请求的参数名称

tips：获取的是参数名称额，而不是参数值 可当做**迭代器使用**

4)， **Map<String ,String[]> getParameter** -- 获取所有参数的map集合

Map参数：

String:请求参数名

Strng[]:请求参数值---为避免一名多值，采用数组

\* **表单POST中文乱码问题：**

tips:GET方法没有这种问题,因为get方法已经解决乱码问题而post没有

解决：在获取请求内容前：**request.setCharacterEncoding(“utf-8”)**

**4.请求转发：**

**服务器中跳转； -----不用加虚拟路径；因为是在内部转发**

**特点：**

**1).只能跳转到服务器中的已有内部资源**

**2).浏览器地址栏不会变化**

**3).转发是一次请求 –可共享request域**

**操作：**

1）**通过request对象获取请求转发器对象**：

**RequestDispatcher getRequestDispatcher(String path)**

参数：path ：转发的目的地(servlet路径)

2）**使用转发器对象进行转发：**

**forward(ServletRequest reg,ServletResponse res)**

**详细：**

**\*request.getRequestDispatcher("/demo2").forward(request,response)**

**5.共享数据：**

**域对象：**一个有作用范围的对象，可以在该范围内共享数据

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

**方法：**

**void setArrtibute(String name,Object obj) --存储数据**

**Object getArrtibute(String name) 通过键获取值**

**void removeAttribute（String name) 根据建删除值**

**6.ServletContext:**

获取：

**ServletContext sc = request.getServletContext();**

**作用：**

**1.域对象：范围最大，慎用**

**2.获取资源的真实路径（物理路径）：getRealPath(“资源”)**

**3.获取指定资源的mime类型：getMIME(“”)**

## Response:

### 概述：

服务端发送给客户端数据

### 组成：

**响应消息数据格式：**

**1.组成：**

**响应行**：协议 响应状态码 状态码描述

\*状态码：

**响应头**： 头：值

\*常用响应头：

**contentType**: 响应的数据的格式和编码；--

如：text/html

Condent-disposition:

**响应体**：

-服务器-发送给-浏览器-的数据：

如：html,jsp，css,js文件

### 功能：

**1).设置响应消息**

1).设置响应行：

**setStatus(int num)**

2).设置响应头：

**setHeader(String name, String value)**

3).设置响应体

\* 使用步骤：

1. 获取输出流

\* 字符输出流：**PrintWriter getWriter()**

\* 字节输出流：**ServletOutputStream getOutputStream()**

**获取流的默认编码是ISO8859-1**，注意乱码问题，详情---下面乱码部分

2. 使用输出流，将数据输出到客户端浏览器

\*\*\*详细：

响应字符串： **response.getWriter().write(String str);**

响应二进制： **response.getOutputStream().write(byte[]);**

response.getOutputStream().write(“哈哈哈”.getBytes(“字符编码”);

**\*\*\*2).乱码：**

原因：

编解码使用的字符集不一致

中文GBK/utf-8 ---Tomcat：ISO8859-1

解决：

①.获取流对象前使用：

**response.setCharacterEncoding("字符集")**

②.设置响应头：

**response.setHeader("content-type","text/html;charset=字符集")**

tips：两个字符集要相同

**tomcat获取的流是ISO-8859-1,而浏览器默认解码是GBK,所以要设置服务器编码和浏览器解码一致才不会乱码**

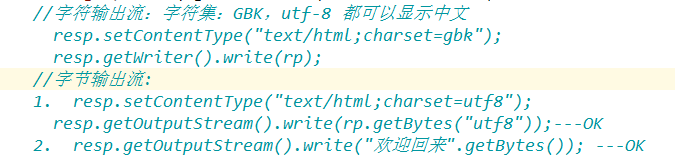
**\*简化：（推荐）**

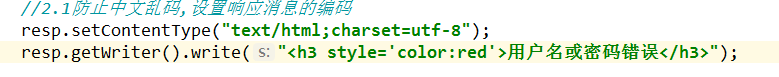
**Response.setContentType("text/html;charset=字符集")**

作用：

1).设置response对响应体使用的码表

2).告诉浏览器使用指定的码表解码





**重定向：**

不同于请求转发，**路径需要加虚拟目录**

二次请求： ---**重定向不能访问共享资源**

步骤：

1).

**1.response.setStatus(302)；**

**2.response.setHeader("location","路径")**

参数:

location:响应头（固定）

路径：/虚拟目录/servlet路径

\*\*\*简化：（推荐）

**response.sendRedict("路径")**

**tips：重定向与转发的区别：**

1).请求次数： 1-------2

1).地址栏 ： 变------不变

1).路径写法： 加虚拟目录-------不加

1).访问共享资源： no --------yes

1).跳转外网： yes-------no



**\*\*\*扩展：**

\*\*\*1).路径写法：

1. 路径分类

1. **相对路径**：通过相对路径不可以确定唯一资源

\* 如：./index.html

\* 不以/开头，以.开头路径

\* 规则：找到当前资源和目标资源之间的相对位置关系

\* ./：当前目录

\* ../:后退一级目录

2. **绝对路径**：通过绝对路径可以确定唯一资源

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

/day15/responseDemo2

\* 以/开头的路径

\* 规则：判断定义的路径是给谁用的？判断请求将来从哪儿发出

\* **给客户端浏览器使用**：需要加虚拟目录(项目的访问路径)

\* 如：**<a> , <form> 重定向...**

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

\* **给服务器使用**：不需要加虚拟目录 ----如：转发

除了请求转发不需要写工程名（虚拟路径）,其他都需要

-----------------------------------------------------

## 会话技术：

### Cookie：

客户端会话技术：

一次会话，多个请求间共享数据

1.**创建**：**Cookie** cookie = **new** Cookie(**"password"**,**"123456"**);

2.方法：

1).设置共享范围：cookie.setPath(**"/"**);

2).设置存活时间：cookie.setMaxAge

3).发送cookie ：response.add(cookie);

4).接收cookie ：**Cookie[]** cookies = request.getCookies();

获取键与值：遍历数组获取：

String getName()

String getValue();

5).设置cookie的值：cookie.setValue(String Value);

3.生命范围：

### Session:

服务端会话技术

1.**创建**: **HttpSession** session = **request.getSession()**;

2.方法：

1).设置session值：**void** setAttribute(**"键"，"值"**)

2).获取session值：**Object** msg = session.getAttribute(**"键"**);

Tips：

SetAttribute(**"键"，"值"**)

键 ： String

值 ： **Object**

**3.生命范围：**

# JSP

## El表达式:

JSP默认支持el表达式，不需要导入资源

## JSTL标签:

JSP需要导入jstl标签库才能使用jstl标签

# Fliter过滤器:

作用：

特点：

1. 多个过滤器构成一个完整的过滤器链

2. 多个过滤器串行；

3. 只要有一个过滤器不放行，请求就不能访问目标资源

过滤器链：

执行流程：

生命周期：

拦截路径配置：

1.注解配置：

2.web.xml配置

拦截方式配置：

1.注解配置

2.web.xml配置

# Listener

# 综合案例

# Ajax:

*概念：异步的javascript and xml*

*原理：异步-客户端请求资源时，无需等待，可以做其他事情，同步相反...*

*作用：提高用户体验*

*使用：*

## 1.原始JS方式

## 2.Ajax方式：

*$.ajax(url,[settings]);*

*参数：*

**url** : 请求资源的路径，相对路径或者绝对路径

***[settings]****:* *AJAX请求设置。所有选项都是****可选的****。*

***参数的值都是键值对***

*常用：*

*data: 发送到服务器的数据。*

*1) 将自动转换为****请求字符串格式****。*

*2) 必须为* ***Key/Value*** *格式*

*3) 如果为数组，jQuery 将自动为不同值对应同一个名称。*

*如 {foo:["bar1", "bar2"]转换为 "&foo=bar1&foo=bar2"。*

*4) GET 请求中将附加在 URL 后*

*dataType***:**

*1)***预期服务器返回的数据类型，**

*2)*如果不指定，jQuery 将**自动根据 HTTP 包 MIME 信息来智能判断**

*3)*随后服务器端返回的数据会根据这个值解析后，**传递给回调函数。**

*Type:请求方式：****默认GET***

***...***

**function** *fun*() {  
 ***$***.ajax({  
 **url**:**"AjaxServlet"**, // 请求路径  
 **type**:**"post"**, *// 请求方式：默认get* **data**:{**"username"**:**"忘掉我发完"**},// 请求参数  
 success:**function** (data) { // 请求成功后执行的函数  
 *alert*(data);  
 },  
 error:**function** () { // 请求失败后执行的函数  
 *alert*(**"出错了"**);  
 }, *// 结尾不建议加逗号* **dataType**:**"text" // 服务器响应结果（请求结果）的类型** });  
}

**text**: 返回纯文本字符串

**xml**: 返回 XML 文档，可用 jQuery 处理。

**html**: 返回纯文本 HTML 信息；包含的script标签会在插入dom时执行。

script: 返回纯文本 JavaScript 代码。不会自动缓存结果。除非设置了"cache"参数

注意：在远程请求时(不在同一个域下)，所有POST请求都将转为GET请求。(因为将使用DOM的script标签来加载)

**json**: 返回 JSON 数据 。

jsonp: [JSONP](http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/" \o "http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/) 格式。使用 [JSONP](http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/" \o "http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/) 形式调用函数时，如 "myurl?callback=?" jQuery 将自动替换 ? 为正确的函数名，以执行回调函数。

## 3.$.get()/post();

***参数：***

***URL：请求路径***

***[data]:请求参数---get:***

***[callback]：回调函数***

***[type]：服务器返回的数据类型 与$.ajax不同***

**function** *fun1*(){

***$***.post(**"AjaxServlet"**,**"age=11"**,**function**(data){*...* },**"text"**);

}

***二者用法相同，区别在于请求参数位置不同***

# Json:

1.JSON 指的是 JavaScript 对象表示法（JavaScript Object Notation）  
2.JSON 是轻量级的文本数据交换格式： ①保存，②传输

3.JSON 独立于语言

4.JSON 具有自我描述性，更易理解

## 语法：

***Js默认支持json语法，不需要引入jquery***

*基本规则：*

*1.键值对形式表示：*

***键：值***

*2.字符串（键，值）需要单双引号，或者什么都不用 ---统一风格*

JSON 规定字符集是UTF-8，字符串必须使用""，Object 的键也必须使用""

*3.数据由* ***逗号*** *隔开*

*4.****数组：【】***

*5.****对象：{}***

*Tips：****数组和对象可以互相嵌套：***

*[{},{}...{}]*

*{[{}],[],...[]}*

**数组或对象的最后一个成员，不能加 逗号**

*值类型：*

*字符串： 需要双引号 ---统一风格*

*逻辑值： true 或 false ---小写*

***数组****: []括起来*

***对象****： {括起来}*

*null：使用很少 ---小写*

***获取值****：*

***JSON对象.键名***

***JSON对象[“键名”]***

***数组对象[索引]*** *,****可用于jquery遍历中获取值***

*遍历：*

**var *person*** = {**"name"**: **"张三"**, **age**: 23, **'gender'**: **true**};  
**var *ps*** = [{**"name"**: **"张三"**, **"age"**: 23, **"gender"**: **true**},  
 {**"name"**: **"李四"**, **"age"**: 24, **"gender"**: **true**},  
 {**"name"**: **"王五"**, **"age"**: 25, **"gender"**: **false**}];  
***// 遍历对象*****function** *f1*() {  
 **for** (**var** key **in *person***){  
 *alert*(***person***[key]);  
 }  
}  
***// 遍历数组*****function** *f2*() {  
 **for**(**var** i=0;i<***ps***.length;i++ ){  
 **for** (**var** key **in *ps***[i]){ *// person对象  
 alert*(***ps***[i][key]);  
 }  
 }  
}

## Json解析器：

### Java对象转换Json字符串

**导包**

创建核心对象：ObjectMapper om = new ObjectMapper();

调用方法 :

1.**writeValue(参数1，Object obj)**

多种重载方式：

参数1：

**File**:将obj对象转换成Json字符串，并保存在指定文件 **Writer**：将obj对象转换成Json字符串，并填充到字符输出流中

**OutputStream**将obj对象转换成Json字符串，并填充到字节输出流中

2**.writeValueAsString(Object obj)**

将对象转换成Json字符串

Tips：

小技巧：

转换成json并，响应客户端：

**objectMapper.writeValue(response.getWriter(),map);**

**注解：**

@JsonIgnore排除属性，被修饰的属性会被忽略

@JsonFormat（pattrn=””）属性格式化 ，将属性格式化指定格式：如日期类属性

### 集合转Json字符串

**用法和对象相同**

格式对应：

**List 对应 Json里的 数组[]**

**Map 对应 Json里的 对象{}**

### Json字符串转换 Java对象

**导包**

创建核心对象

调用方法

ReadValue(String，Class )

参数：

String ：字符串形式的Json数据

Class ：希望转换成的java对象的类型

## 案例：

用户注册验证：

Html:

***$***.get(**"/UserServlet"**,**"username="**+value,**function** (data) {  
 **if**(data.userExist) { // value 输入的请求参数

*// 用户存在* span.css(**"color"**,**"red"**);  
 span.html(data.**msg**); // data 响应的json数据  
 }**else** { // **msg与**userExist是data的键

*// 用户不存在* span.css(**"color"**,**"green"**);  
 span.html(data.**msg**);  
 } } *// "json")*

*// 设置请求编码* request.setCharacterEncoding(**"utf-8"**);  
 *//设置响应编码为 json* response.setContentType(**"application/json;charset=utf-8"**);  
 *// 获取请求参数* String username = request.getParameter(**"username"**);  
 *// 查询数据库* JdbcTemplate template = **new**

JdbcTemplate(JDBCUtils.*getDataSource*());  
 String sql = **"select** *\** **from user where username = ?"**;  
 List<User> query = template.query(sql, **new** BeanPropertyRowMapper<User>(User.**class**), username);  
 System.***out***.println(**"数据库查询结果:"**+query);  
 *// 创建map，存放响应的数据* Map<String,Object> map = **new** HashMap<String, Object>();  
 *// 判断查询结果* **if**(query!=**null** && query.size()>0){  
 *// 有该用户* map.put(**"userExist"**,**true**);  
 map.put(**"msg"**,**"用户已存在"**);  
 }**else** {  
 map.put(**"userExist"**,**false**);  
 map.put(**"msg"**,**"用户名可用"**);  
 }  
 Set<String> keySet = map.keySet();  
 **for** (String s : keySet) {  
 System.***out***.println(**"响应数据:"**+s+map.get(s));  
 }

*// 将Map转换成Json，响应客户端  
 // 导jackSon包  
 // 创建核心对象* ObjectMapper objectMapper = **new** ObjectMapper();  
 *// 调用方法,并响应客户端*  **objectMapper.writeValue(response.getWriter(),map);**

**Tips：**

1). 使用Jquery发送异步请求时，如果$.get()/post()/$.ajax()方法，如果没有指定响应数据类型， jQuery 将**自动根据 (**servlet**)HTTP 包 MIME 信息来智能判断**

2). 如果发送请求时想要指定：

**$.get()/post() ：type参数设置为 json**

**$.ajax() ：dataType参数设置为json**

面试：

override 与 overload

redirect 与 forward

uri 与 url

# Redis

## 概念

用c语言编写的高性能开源非关系型数据库

Tips：

与关系型数据库互补而不是对立的关系

优缺点：

## 安装：

## 应用场景：

• 缓存（数据查询、短连接、新闻内容、商品内容等等）

• 聊天室的在线好友列表

• 任务队列。（秒杀、抢购、12306等等）

• 应用排行榜

• 网站访问统计

• 数据过期处理（可以精确到毫秒

• 分布式集群架构中的session分离

## Redis数据结构：

Redis中数据是以键值对形式存储。且key固定为String类型，value可为下列类型：

1) 字符串类型 string

2) 哈希类型 hash ： map格式

3) 列表类型 list ： linkedlist格式。 支持重复元素

4) 集合类型 set ： 不允许重复元素

5) 有序集合类型 sortedset： 不允许重复元素，且元素有顺序

## 命令操作：

**以下的key都是自定义的**

**String:**

设置：set key value

获取：get key

删除：del key

**Hash:**

设置：hset key field value

获取：hget key field  获取指定的field对应的值

hgetall key： 获取所有的field和value

删除: hdel key field

**list:**

设置：lpush key value 左插 rpush key value 右插

获取：lrange key start end 范围获取[0,-1]

删除: lpop key： 删除列表最左边的元素，并将元素返回

       rpop key： 删除列表最右边的元素，并将元素返回

**set:**

设置：sadd key value

获取：smembers key 获取set集合中所有元素

删除: srem key value 删除某个值

**sortedset:** 有序set集合，增加一个double类型的分数

设置：zadd key score value

获取：zrange key start end [withscores]

删除: zrem key value

**通用命令:**

1. keys \* : 查询所有的键  
  2. type key ： 获取键对应的value的类型  
  3. del key：删除指定的key value

## redis持久化

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

2. redis持久化机制：

1). **RDB**：默认方式，不需要进行配置，默认就使用这种机制

\* 在一定的间隔时间中，检测key的变化情况，然后持久化数据

1. 编辑redis.windwos.conf文件

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

**save 900 1**

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

save 300 10

2. 重新启动redis服务器，并指定配置文件名称

进入到安装目录---此处打开命令窗口：

**redis-server.exe redis.windows.conf**

2). **AOF**：日志记录的方式，可以记录每一条命令的操作。可以每一次命令操作后， 1. 编辑redis.windwos.conf文件

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

appendfsync always ： 每一次操作都进行持久化

appendfsync everysec ： 每隔一秒进行一次持久化

appendfsync no ： 不进行持久化

## Java客户端 Jedis

\* Jedis: 一款java操作redis数据库的工具.

\* 使用步骤：

1. 下载jedis的jar包

2. 使用

//1. 获取连接

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

//2. 操作

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

//3. 关闭连接

jedis.close();

## jedis连接池： JedisPool

\* 使用：

1. 创建JedisPool连接池对象

2. 调用方法 getResource()方法获取Jedis连接

//0.创建一个配置对象

JedisPoolConfig config = new JedisPoolConfig();

config.setMaxTotal(50);

config.setMaxIdle(10);

//1.创建Jedis连接池对象

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

//2.获取连接

Jedis jedis = **jedisPool.getResource()**;

//3. 使用

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

//4. 关闭 归还到连接池中

jedis.close();

连接池工具类

## Spring data redis:

Spring data redis 存储key乱码原因：

因为默认使用RedisTemplate默认使用的是JDK序列化器：JdkSerializationRedisSerialize;而它使用的编码是ISO-8859-1

### Question:

Question:

1.大key乱码，

2.小key乱码和value乱码

3.value为中文时，乱码

方式1：

代码解决：

设置key序列化方式：

redisTemplate.setKeySerializer(new StringRedisSerializer());



但是在获取value时，仍然是乱码：



Tips：下面的没测试：



方式2：**配置文件** 

**Key不在乱码，但是value为中文时，会乱码，解决：**



**解决中文乱码：**

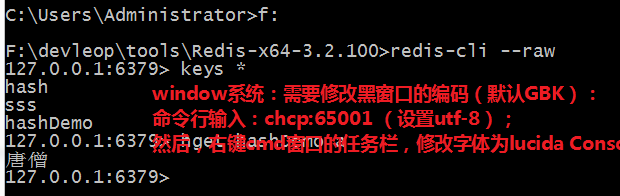
**Linux系统：**

Redis-cli.exe –raw:启动redis客户端

**Windows系统：**

需要先修改cmd窗口的编码：chcp 65001 设置字体编码为utf-8

然后Redis-cli.exe –raw:启动redis客户端，即可



# Maven:

## 简介：

**概念：**

**作用：**

**一键架构**

**Jar包管理**

## 安装：

直接下载解压到非中文且无空格目录中即可

## 配置:

系统环境变量配置：

**%MAVEN\_HOME%** ---安装目录

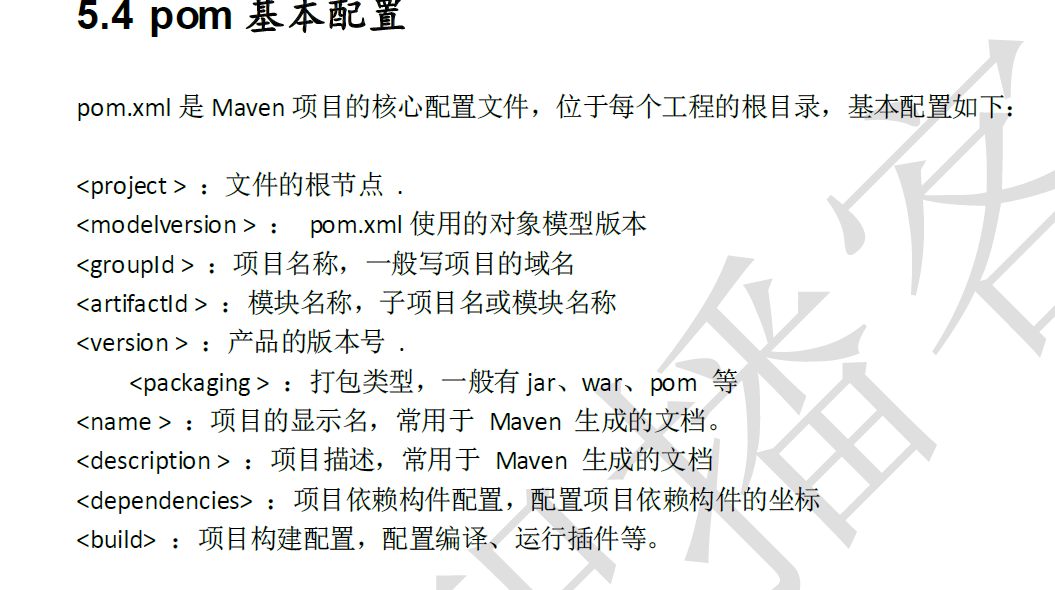
**Path：** ---**%MAVEN\_HOME%\bin**

Tips:

Maven的使用依赖%JAVA\_HOME%,需要正确配置%JAVA\_HOME%

**验证**是否成功：

**mvn-v:查看安装版本（及JDK版本）**



*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>cn.itcast</**groupId**>  
 <**artifactId**>travel</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 <**packaging**>war</**packaging**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>3.8.1</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 *<!--servlet-->* <**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>javax.servlet-api</**artifactId**>  
 <**version**>3.1.0</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
  
  
 *<!--mysql驱动-->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>5.1.26</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 *<!--druid连接池-->* <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>druid</**artifactId**>  
 <**version**>1.0.9</**version**>  
 </**dependency**>  
 *<!--jdbcTemplate-->* <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-core</**artifactId**>  
 <**version**>4.1.2.RELEASE</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jdbc</**artifactId**>  
 <**version**>4.1.2.RELEASE</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-tx</**artifactId**>  
 <**version**>4.1.2.RELEASE</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-beans</**artifactId**>  
 <**version**>4.1.2.RELEASE</**version**>  
 <**scope**>compile</**scope**>  
 </**dependency**>  
 ...其他依赖jar

</**dependencies**>

<**build**>  
 *<!--maven插件-->* <**plugins**>  
 *<!--jdk编译插件-->* <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**configuration**>  
 <**source**>1.8</**source**>  
 <**target**>1.8</**target**>  
 <**encoding**>utf-8</**encoding**>  
 </**configuration**>  
 </**plugin**>  
 *<!--tomcat插件-->* <**plugin**>  
 <**groupId**>org.apache.tomcat.maven</**groupId**>  
 *<!-- tomcat7的插件， 不同tomcat版本这个也不一样 -->* <**artifactId**>tomcat7-maven-plugin</**artifactId**>  
 <**version**>2.1</**version**>  
 <**configuration**>  
 <**port**>80</**port**>  
 *<!-- 项目访问路径🡪*

<**path**>/travel</**path**>  
</**configuration**>  
 </**plugin**>  
  
 </**plugins**>  
 </**build**>

**</project>**

## maven安装目录：

**使用：**

**命令：**

查看路径：set java\_home

## maven项目：

**maven项目目录：**

**maven工程 ：**

**1.-个人：首先去本地仓库找；没有： 联网状态下—去中央仓库下载**

**2.-公司：首先去本地-no-远程-no-中央仓库下载**

**坐标定义：**

**GroupID**

**ArtifiedID**

**Version**

**Maven会根据坐标去寻找jar包**

**仓库：**

**本地仓库：本地的jar包仓库，可离线使用:**

**默认在：c:user/.m2/repository；可修改**

**远程仓库：公司内部设置的（私服）；jar包来源：从中央仓库下载；本地上传**

**中央仓库：网络中jar包大本营：需要联网才能访问下载**

**仓库配置：**

修改本地仓库：

默认在c

修改：

修改中央仓库：

默认中央仓库在外国，访问困难，需要修改为阿里巴巴的

**命令**：

Clean :清理

Compile：编译

Run ：运行

Test ：测试

Package :打包

Install :安装

**问题：**

1.Tomcat6.0不支持1.8++

所以不能用mvn tomcat:run启动项目；可以使用mvn tomcat7:run来运行

**可以修改模块的pom.xml文件 配置插件**

**2.项目jar包缺少解决：**

**1.项目中配置pox.xml文件：**

**2.①.alt+enter:--add；②import class**

**3.jar包冲突问题：**

**设置scope作用域**

# 框架

## Mybatis:

框架：

作用：

三大架构：

表现层

业务层

持久层

### 简介

是一个优秀的基于java的持久层框架，它封装了jdbc,使开发者只需要关注sql语句本身，而不用费力处理加载驱动，创建连接，创建Statement等繁杂的过程

它通过注解或xml的方式将要执行的各种statement配置起来，通过java对象和statement中的sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回为java对象返回

ORM思想解决实体和数据库映射的问题，对jdbc进行封装，屏蔽了jdbc api 底层访问细节，使我们不用与jdbc api打交道，就可以完成数据库的持久化操作。

根据接口全路径+方法名匹配，找到对应的xml文件或注解，去完成任务，返回结果

mybatis:

基于java的持久层框架，封装了jdbc细节，使用者只需考虑sql语句的书写；通过java对象和statement的参数动态生成最终执行的sql语句

orm思想，实体类映射数据库表

1.实体类成员变量 名称 和 数据库表中字段 相同

2.不一致： ResultMap(推荐) ; 起别名(不推荐)；

？？？为什么要一致？不一致可以么？？？

### 相关概念：

映射器：

由一个java接口和xml文件（或注解组成）

代理对象：

根据接口全路径+方法名匹配，找到对应的xml文件或注解，去完成任务，返回结果

映射：

映射就是将两个对象对应起来。对应的对象叫象，被对应的对象叫原象

事务：

mybatis手动提交事务

OGNL表达式

ORM:

POJO:

### 环境搭建：

1.简单环境搭建 ---快速入门

\* 创建maven工程，并导入坐标

\* 创建实体类和dao 的 接口

\* 创建mybatis的主配置文件 --- SqlMapConfig.xml

\* 创建映射配置文件 --- 如：UserDaoInter.xml

1) 一般把持久层的操作接口和映射文件 叫做 ：Mapper （映射器）

2）IDEA中创建的包和目录，结构不同

3）mybatis的映射配置文件位置必须和dao接口的包结构相同

4）映射配置文件的mapper标签namespace属性值必须是dao接口的全限定类名

5）映射配置文件的操作配置，id属性的取值必须是dao接口的方法名

### Xml配置：

核心配置文件：Resources 一级目录

映射文件：①和接口在结构相同 ；②命名和接口一致

#### 核心配置文件：

sqlMapConfig.xml：--->(名称任意)

作用：

组成：

**properties**: 可在标签内部定义连接数据库信息

也可导入外部配置文件（推荐）

常用：resource:指定配置文件的位置，按照类路径的写法来写，且必须存在于类路径之下

URl： 按照url写法 :统一资源定位符： 可唯一标识一个文件的位置 全网

协议 主机 端口 uri

注意中文乱码问题

tips :

URL:统一资源标识符： 可在应用中唯一定位一个资源： 如项目中

设置数据源

原来的数据源的value属性: value = "${键名}"

**typeAlieases:**

**package**：指定要配置别名的包，当指定后，该名下的实体类都会注册别名，**且默认 类名就是别名** ， 不再区分大小写 **---多个：一次设置多个**

**typeAlieas**: 当指定别名时，使用时不再区分大小写 **---单个：一次设置一个**

**environments:**

环境配置

配置环境：

1.配置数据库环境（有一个默认的）

// 事务管理器

// 数据库链接

2.配置映射器：

// 指定映射文件的位置

**MAPPERS**: 核心配置中的映射配置

package name="包路径": 指定接口所在的包，当指定了，

就不再需要写<mapper resource或 class>

tips：

1.当使用resouce属性，配置mapper,由地址分隔符连接；而不是包名 ...

2.使用class属性时，...连接

映射配置： 每一个接口都有一个 ：

3.也可使用packging:代替mapper: 代表某个包内所有的接口

#### 映射文件：

作用：

\*1.提供SQL

\*2.SQL对POJO的映射规则定义

基本组成：

**nameSpace** : **值为 接口的 全包名**

**id** :值为实体类的方法

sql语句 （标点不影响）

**\*parameterType**: 接收的参数类型：

基本类型

对象：全路径

#{对象的属性}

POJO包装对象：全路径

包装对象：对实体类对象的包装

#{包装对象的属性.被包装对象的属性}

Tips：只传入一个参数，且类型为基本类型和String时， #{值}中的值可以随意写

**ResultType**: 封装数据的类型（返回值的类型）

\***ResultMap**: 当自定义映射规则时可以使用：

**一条sql语句中，ResultType与ResultMap冲突**

**自定义映射规则**

解决实体类属性 名 和 数据库表中 字段 不一致（对应） 的问题

**Id**: 相当于这个ResultMap的索引： sql中通过id指定resultMap

**type**: 封装对象模型 ---**实体类全路径**

**映射规则**：

主键字段：

<id =property"实体属性" column="表中字段" > </id>

非主键字段

<result property="实体属性" column="表中字段"> </result>

property <---> column

实体类属性 表中字段

**tips：**

**mysql在window系统不区分大小写**

linux 严格区分大小写

java严格大小写

**可定义多个ResultMap**

**OGNL表达式解析**

解析实体类属性： 通过getXXX实体类的属性: ①去除get②第一个大写换小写

**占位符**：

#{解析后的实体属性} ---预编译

????

'${value}' 固定为value：适用于只有一个参数的情况 ---sql注入问题

------------------------------------------------

**解决实体属性和表中字段不对应问题：**

1.配置方式: ResultMap ---**推荐**

2.别名： 给 sql语句 中 各字段 起别名 ： username as userName,...

效率高，sql层面作用

mybatis会读取映射文件，生成映射器

1.xml配置：

映射文件中配置：一级目录

<**resultMap id="person" type="com.wxss.domain.Person"**>  
 *<!--主键字段-->*

<**id property="uid" column="id"**></**id**>  
 *<!--非主键字段-->*

<**result property="userName" column="username"**></**result**>  
 <**result property="userSex" column="sex"**></**result**>  
 <**result property="userAddress" column="address"**></**result**>  
 <**result property="userBirthday" column="birthday"**></**result**>  
</**resultMap**>

2.起别名： 字段 as 属性名

映射文件中配置：

SELECT id as uid,username as userName,sex as userSex,address as userAddress FROM user

**实体类实现序列化接口 起标识作用**

### mybatis的具体使用：

#### 1）XML方式：

1.读取核心配置文件

2.创建SqlSessionFactory接口对象

利用SqlSessionFactoryBuilder 对象，调用build()方法创建

3.创建SqlSession对象：

SqlSessionFactory 接口对象调用 openSession()

4.创建代理对象

SqlSession对象调用getMapper()

#### 2）注解方式：

与xml基本类似

不同：

1.在dao接口里使用注解：在对应方法前，使用相应的注解

2.移除 映射配置文件

3. 修改核心配置文件:

修改映射器配置的mapper：设置属性为class

属性名：使用class属性名代替resource属性名

属性值：dao接口的全类名

其他不变

#### 3）实现类方式：

1.也不需要使用映射配置文件

2. 创建实现接口的实现类：

构造方法： 传递一个SqlSessionFactory对象

功能方法： 继承于接口： 利用创建实现类对象，传递的SqlSessionFactory对象,实现具体的功能

statement:**不能直接使用，防止多个相同方法时，出错**：

**全包名.方法名（推荐）**

List<User>users = session.selectList("com.itheima.dao.IUserDao.**findAll**");

#### 4）无接口方式：

不利于整合框架

设计模式:

构建者模式

工厂模式

代理模式

-----------------

### API分析:

Configuration 存储解析后的xml配置信息 ，可以读取它获得 mybatis配置信息

\* Configuration **单例**：只有一个，可重复使用，且占空间小

SqlSessionFactory 可由xml配置 或者 代码创建

推荐xml文件创建： （**避免硬代码**）

\*方便配置的修改

\*避免代码重复编译

创建：由SqlSessionFactoryBuilder构造器创建：

首先，mybatis解析xml文件，将配置信息存储到Configuration对象中，

然后，由SqlSessionFactoryBuilder读取Configuration对象，创建SqlSessionFactory

tips：

**单例**创建：可**自定义工具类** （静态代码块）

**多例**创建：**源码(默认方式)**---耗时间

SqlSession：

线程不安全

多例创建

### Mybatis连接池

#### 作用

**项目在启动**的时候，**初始化多个连接**，放到连接池中，每次使用的时候，直接去连接池中获取。**从而降低获取连接的时间，提高效率。**

#### mybatis中的连接池3种

POOLED：使用连接池技术



UNPOOLED：不使用连接池（不推荐使用）

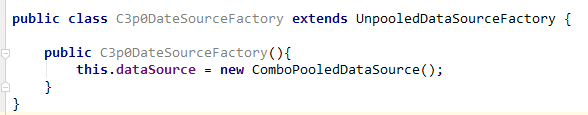
JNDI：使用服务器的JNDI技术来实现，必须是web项目（该技术比较老，现在基本不用）

#### 使用第三方连接池（C3p0）

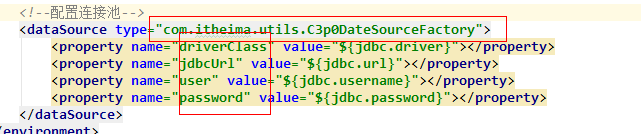
第一步：导坐标

<**dependency**>  
 <**groupId**>c3p0</**groupId**>  
 <**artifactId**>c3p0</**artifactId**>  
 <**version**>0.9.1.2</**version**>  
</**dependency**>

第二步：编写一个工具类，继承UnpooledDataSourceFactory



第三步：配置使用c3p0连接池



### Mybatis事务：

1、回顾事务知识点：

定义：包含多个步骤的逻辑操作，要么一起成功，要么一起失败，这就是事务控制

事务的四大特性：原子性、持久性、隔离性、一致性

如果不考虑隔离性，会导致的问题：脏读、不可重复读、幻读

事务的隔离级别：

1. read uncommitted：读未提交

\* 产生的问题：脏读、不可重复读、幻读

2. read committed：读已提交 （Oracle）

\* 产生的问题：不可重复读、幻读

3. repeatable read：可重复读 （MySQL默认）

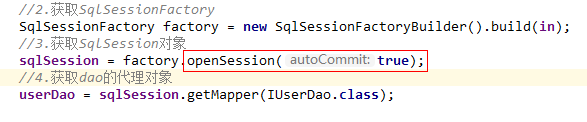
\* 产生的问题：幻读

4. serializable：串行化

\* 可以解决所有的问题

\* 注意：隔离级别从小到大安全性越来越高，但是效率越来越低

2、mybatis中的事务是**默认手动提交**，可以在获取SQLSession时设置自动提交



### 动态sql：

通过 if, choose, when, otherwise, trim, where, set, foreach等标签，可组合成非常灵活的SQL语句，从而在提高 SQL 语句的准确性的同时，也大大提高了开发人员的效率。

#### 1) if 语句：

Select \* from user where

<**if test="username != null and username != ''"**>// 逻辑表达式  
 username=#{username} // 拼接的字符串

</**if**>

<**if test="sex != null and sex != ''"**>// 逻辑表达式  
 **and** sex=#{sex} // 拼接的字符串

</**if**>

如果test返回true，就将if标签体内容拼接到基本sql语句中,但是如果第一个逻辑表达式false;sql语句就成为：Select \* from user where **and** sex=#{sex} **语法错误**

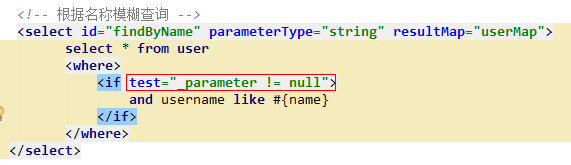
可以通过 **if+where标签优化**

**Tips:**

注意：在传入的参数是**基本数据类型，**的时候，而且**只有一个参数**时，使用<if>标签，会默认的使用OGNL表达式解析

此时，<if>标签中的test使用固定的写法：test=”\_parameter != null”

两者都有？？？



#### 2) if + where

|  |
| --- |
| **Select \* from user**  <**where**>  <**if test="username != null and username != ''"**>// 逻辑表达式  username=#{username} // 拼接的字符串  </**if**>  <**if test="sex != null and sex != ''"**>// 逻辑表达式  **and** sex=#{sex} // 拼接的字符串  </**if**>  </where> |

这个“where”标签会知道如果它包含的标签中有返回值的话，

它就**插入一个‘where’**。

此外，如果**标签返回的内容是以AND 或OR 开头的，则它会剔除掉**

#### 3) if+set

|  |
| --- |
| **update** user u  <**set**>  <**if test="username != null and username != ''"**>  u.username = #{username},  </**if**>  <**if test="sex != null and sex != ''"**>  u.sex = #{sex}  </**if**> </**set**> where id=#{id} |

和 普通if 一样用法；要注意 多个字段间的 逗号 **','**

#### 4) choose(when,otherwise)

类似 java 中的switch语句（更像jstl标签的）

多个查询条件，只想选择其中的一个，查询条件有一个满足即结束拼接字符串

|  |
| --- |
| select \* from user // 基本sql语句  <**where**>  <**choose**>  <**when test="id !='' and id != null"**>  id=#{id}  </**when**>  <**when test="username !='' and username != null"**>  and username=#{username}  </**when**>  <**otherwise**>  and sex=#{sex}  </**otherwise**>  </**choose**> </**where**> |

#### 5) trim语句

trim标记是一个格式化的标记，可以完成set或者是where标记的功能

替代if + where语句

|  |
| --- |
| <**trim prefix="where" prefixOverrides="and | or"**>  <**if test="username != null"**>  and username=#{username}  </**if**>  <**if test="sex != null"**>  and sex=#{sex}  </**if**> </**trim**> |

**替代set语句：**

|  |
| --- |
| <**trim prefix="set" suffixOverrides=","**>  <**if test="username != null and username != ''"**>  u.username = #{username},  </**if**>  <**if test="sex != null and sex != ''"**>  u.sex = #{sex},  </**if**> </**trim**> |

**Prefix：前缀**

**prefixOverrides： 剔除第一个 and 或者 or**

**suffixOverrides：剔除最后一个标记（, and,or）**

**suffix：**后缀

#### 6) sql片段

|  |
| --- |
| <**sql id="defaultSelect"**> // mapper内 ,与sql标签一级  SELECT *\** FROM user WHERE </**sql**>  *<!—在****sql标签内引用*** *sql 片段，如果refid 指定的****不在本文件****中，那么需要在前面加上* ***namespace*** *-->* <**include refid="selectUserByUserNameAndSexSQL"**></**include**> |

将一些重复的代码抽取，提高复用性

#### 7） foreach语句

**作用：遍历传递参数是一个集合或者是数组**

|  |
| --- |
| <**where**> <**if test="ids != null and ids.size > 0"**>  *ognl表达式解析*  <**foreach collection="ids" open="id in(" close=")" item="uuid" separator=","**>  #{uuid} *<!--uuid只是循环中的一个占位符，名称随意-->* </**foreach**> </**if**> </**where**> |

collection:指定输入对象中的集合或者数据

                item:每次遍历生成的对象

                open:开始遍历时的拼接字符串

                close:结束时拼接的字符串

                separator:遍历对象之间需要拼接的字符串

               例： select \* from user where 1=1 and id in (1,2,3)

动态sql注意事项：

标记（， and or ）

**换行时，不要忘记留空格**

### Mybatis表关系：

单表查询：

一对一

1.在主表对应的pojo类中设置副表对应的pojo类对象

2.在主表的映射文件中设置ass\

3.调用主表的方法

多对一

Mybatis中将多对一看做一对一

一对多：

1.在主表中创建从表对应的list<>集合

2.在主表的映射文件中设置：collection

3.调用主表的方法

多对多

多对多可以拆分为两个一对多

分别在主，副表对应的pojo类中，增加对方的list<>集合，再分别设置映射文件，调用各自的方法即可

Tips：

二者的sql些许差异

### 延迟加载和立即加载

代理方式：

1.在核心配置文件中设置：

Ag :true :立即加载：false：按需加载

3.4.1前默认true :之后默认false

2.映射配置：

子查询

注解方式：

Fetch:枚举

### Mybatis缓存：

一级缓存：

1).Mybatis**默认开启一级缓存**：

2).一级缓存是**SqlSession层面**的：

①.相同参数和sql语句，**同一个SqlSession**调用Mappen，只会查询一次：

第一次执行查询后，会把结果保存到：**SqlSession内部的一个空间内：Map结构**

之后，如果没有声明要刷新，或者缓存未过期，都是从内存中取数据

②**不同SqlSession对象是相互隔离**的，会再查询数据库

为了克制不同SqlSession相互隔离，需要配置二级缓存（SqlSessionFactory层面）

3)**一级缓存的清除**：

SqlSession.close()

SqlSession.clearCache()

两次查询间，有删除，修改，增加，操作时

二级缓存：

1).**二级缓存默认关闭**：可在**映射xml中配置**

2).二级缓存是**SqlSessionFactory**层面的：

①.由同一个SqlSessionFactory创建的**多个SqlSession可共享数据**

**②**.二级缓存，要求返回的**pojo必须可序列化**，即实现序列化接口:**Serializable**

③.二级缓存中存储的是**数据**而不是对象（地址不等）

3).使用步骤：

①核心配置中开启 ---开启当前框架

②映射配置中开启 ---开启当前映射文件

③sqL语句中使用二级缓存（**默认已使用**） ---开启当前操作支持

Tips：

实体类必须实现序列化接口

存储数据的形式：？？？

**Mybatis的缓存跨服务器性能差，多用redis缓存**

Mybatis将多对一看做一对一

### 自定义框架：

本质还是调用JDBC,但是其中使用其他知识（设计模式，反射，）

mybatis原理

？？？

key: String

nameSpace . 方法名

value:Mapper对象

String sql ---sql语句

String returnType ---全限定类名

----------------------------------------

解析配置文件：

dpm4j+xpath

动态代理分析：

Proxy.newProxyInstance(类加载器，代理对象要实现的接口，如何代理)

类加载器：使用的和被代理对象是相同的类加载器

代理对象要实现的接口：和被代理对象实现相同的接口

如何代理: 增强的方法,自己提供

此处为

前两个参数固定写法：

类.getClass()

类.getInterface()

构建者模式分析：

工厂模式分析

--------------------------------------------------------------------

自定义基本流程：

XMLConfiguration解析配置文件生成两个对象：Configuration 和mapper

创建SqlSessionFactory对象：构造器+实现类

创建SqlSession会话对象：实现类

动态代理：执行方法：

工具类Executor：执行sql，封装数据

实现：

Class Resource 读取配置文件

配置对象，创建工厂对象

动态代理，执行方法

反射，封装数据

-----------------------------------------------------

select last\_insert\_id()

增加前User{id=0, username='老许', birthday=Sun Oct 28 20:04:32 CST 2018, sex='女', address='王楼'}

增加后User{id=51, username='老许', birthday=Sun Oct 28 20:04:32 CST 2018, sex='女', address='王楼'

# Spring

**体系**：

利用Spring创建的Bean对象是单例的

Spring将自定义的BeanFactory中的加载配置文件，创建对象后存储到map集合中，实现了

***ApplicationContext的三个常用实现类：*** *ClassPathXmlApplicationContext：*

*它可以加载类路径下的配置文件，要求配置文件必须在类路径下。FileSystemXmlApplicationContext：*

*它可以加载磁盘任意路径下的配置文件(必须有访问权限）*

*ationConfigApplicationContext：*

*它是用于读取注解创建容器的*

***核心容器的两个接口引发出的问题：***

*\* ApplicationContext: 单例对象适用 （默认单例）*

*它在构建核心容器时，创建对象采取的策略是采用立即加载的方式。*

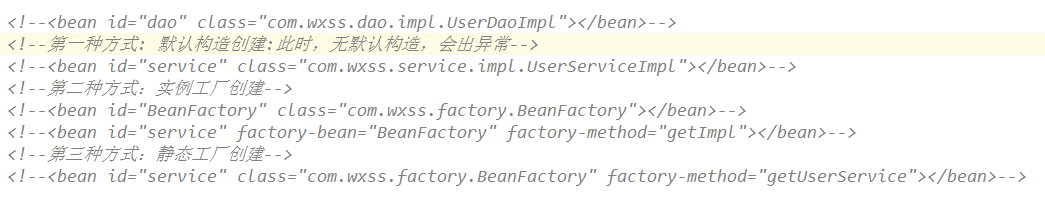
*也就是说，只要一读取完配置文件马上就创建配置文件中配置的对象。  
\* BeanFactory: 多例对象使用 （默认单例）*

*它在构建核心容器时，创建对象采取的策略是采用延迟加载的方式。*

*也就是说，什么时候根据id获取对象了，什么时候才真正的创建对象。*

## Bean:

***三种创建Bean对象的方式****：*



***Tips:***

***Spring默认单例创建Bean对象***

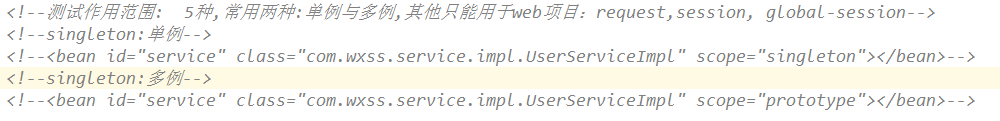
***单例创建时，采用立即加载策略***

***多例创建时，采用延迟加载策略***

***作用范围：***

***Bean标签的scope属性用来指定Bean对象的使用范围***

*Spring创建Bean对象时，****默认单例创建***



***生命周期：***

*单例创建：*

*容器器在，Bean在，；器亡人亡 ----器->spring*

*单例模式需要手动关闭容器*

*多例创建：*

*使用对象时，spring帮我们创建*

*对象使用过程中，就一直存在*

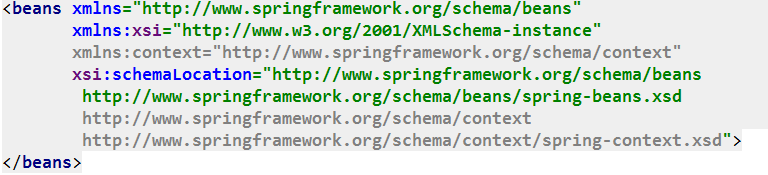
*当对象长时间不使用或没有引用关系，JVM垃圾回收机制销毁*

*Xml配置*

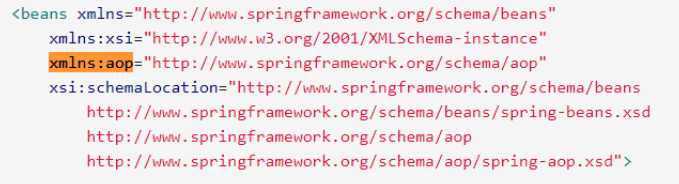
*1.xml约束：*

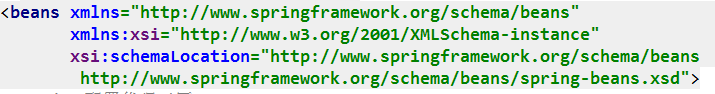
*可以一起使用*

*注解：*



*AOP:*





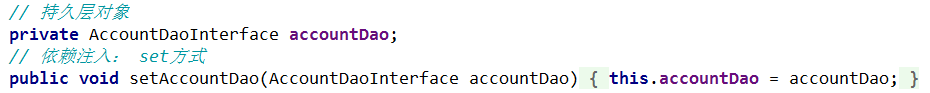
*事务管理类需要通过连接工具类获取当前线程的连接对象，来进行事务控制*

*基于AOP的事务控制：*

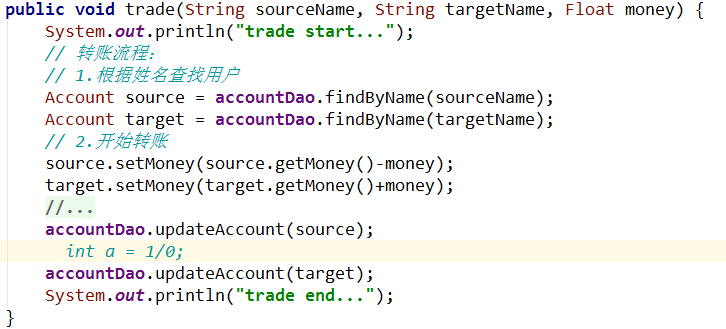
*Xml方式:*

Service:

成员变量：

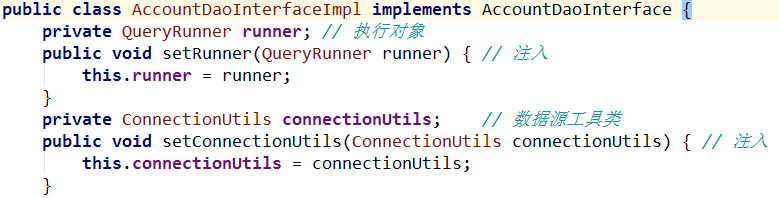


业务逻辑：



持久层：

成员变量：



*Spring：*

## IOC：

*创建Bean对象：*

*Xml:*

*无参构造函数：*

*实例工厂：一行代码*

*静态工厂：两行代码*

*工厂法：一般用于调用方法创建对象，返回想要创建的对象*

*注解：*

*Compoent*

*Service ：业务层*

*Controller ：表现层*

*Reponsity ：持久层*

*四个注解用法一模一样：*

*Value：唯一标识；不写，默认类名，且首字母小写*

## DI依赖注入：

*Xml:*

*构造函数：构造函数中所有参数都需要注入*

*Set方法：自由注入；****只于setXXX(...)方法有关****；和成员变量无关*

*注解：*

*Autowired:自动根据类型注入，且只有一个该类想的时候成功注入*

*IOC容器中没有该类型，报错；*

*IOC容器中有多个该类型，根据变量名查找，如果没有匹配的名称，报错*

@Qualifier:根据定义的 value属性值注入：

给类成员变量注入时，必须和@Autowired搭配使用，不能单独使用

给方法参数时可以单独使用

*@Resource*

*根据name属性值注入；可单独使用*

*@Bean*

*将方法的返回值存放到容器中*

*@ Value*

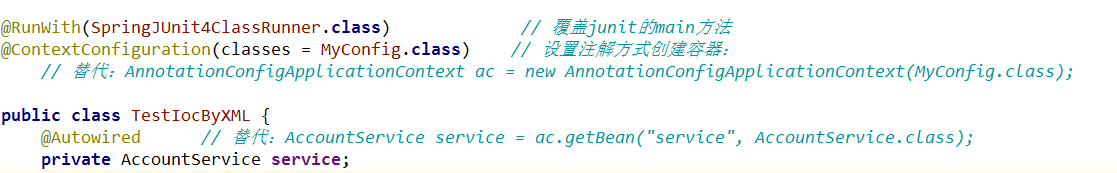
*通过value属性，给变量设置值；可使用spring的EL表达式：${值}*

*其他标签：*

@RunWith(SpringJUnit4ClassRunner.**class**)

用于spring整合jutit单元测试：**需要导入spring-text的jar包**

使用spring提高的main代替测试类的main方法  
@ContextConfiguration(locations = **"classpath:bean.xml"**)



*设置IOC容器创建方式：*

*Locations:xml方式创建*

*关键字:classpath:指定当前类路径下的bean.xml*

*Classes:注解方式创建*

*Classes = 配置类的.class*

*这种方式下：通常设置配置文件类，来代替配置文件，如bean.xml*

*@Configuration*

*指定当前类是配置文件类---用于注解方式*

*@import()*

*用于在父配置（主配置）导入子配置文件；*

*子配置文件：必须被@Configuration修饰，才能被spring识别为配置文件*

*值：{配置文件1.class，配置文件2.class } ---数组，可导入多个子配置文件*

@PropertySource(“jdbc.properties”)

导入属性集文件：如存放数据源信息的jdbc.properties文件



使用注解配置方法时，如果方法有参数，spring会去容器中查找是否有可用Bean对象，查找方式和@Autowired一样：类型—名称

## SpringAOP相关注解：

用于通知类的注解：

**以上注解都应用于通知类，以上注解可以被Spring事务控制API替代**

Spring事务控制API：

## 基于XML的spring声明式事务控制

1，将通知类交给spring管理：

2.使用Aop:config标签表明开始AOP的配置

3.在Aop:config标签内部使用：

aop:aspect:标签配置切面---设置切面和通知的关系

id属性设置切面的唯一标识

ref属性：通过ID关联通知类—

在aop:aspect:内部使用 ：

1）aop:通知类型 method=“” printCut

配置通知的类型，和建立通知方法和切入点方法的关联

属性：method:指定通知类中哪个方法是对应的通知

PrintCut: 设置对哪些方法进行增强

切入点表达式：execution() 无引号

PointCut-ref：引用通用切入点

2）aop:pointcut:配置通用切入点、

Id:

Expression =”exexution(...)”

Tips：

aop:pointcut标签可为多个通知设置切入点，使用时，通过PointCut-ref属性，引用通用切入点表达式的ID

tips：

通用切入点配置的作用范围：



在外面时，要根据XML约束，选择位置



基于注解的spring事务控制

## 基于注解的springAOP 事务控制

Spring配置：

配置类注解：

通知类注解:

1.@Aspect表明当前类是一个切面：功能类似xml中的AOP：confi

2. 通用切入点配置:

@Pointcut(**"execution(\* com.wxss.service.impl.\*.\*(..))"**)  
**private void pt1**(){}

3.设置不同的通知：引用通用切入点

@After(“**pt1()”**) 最终通知------括号不能省略

@Before() 前置通知

@AfterReturning() 后置通知

@AfterThrowing() 异常通知

后置通知和异常通知互斥，不能同时使用

以上四个注解一起进行事务控制时，由于执行顺序问题，会出错，无法进行事务控制，所以多利用环绕增强进行事务配置

@Around() ---环绕通知

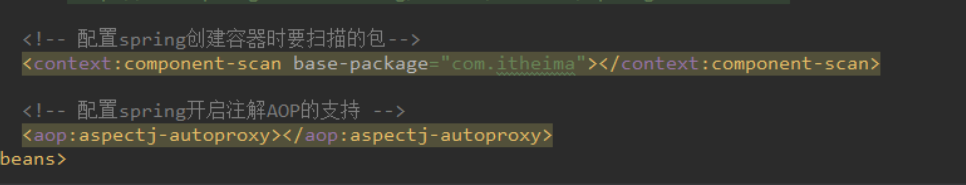
值为通用切入点注解修饰的方法名，---**带括号**

在通知类中多用环绕增强进行事务控制：  
@Around(**"pt1()"**)  
**public** Object aroundAdvice(ProceedingJoinPoint pjp){  
 Object rtValue = **null**;  
 **try** {  
 *//1.获取参数* Object[] args = pjp.**getArgs**();  
 *//2.开启事务* **this**.beginTransaction();  
 *//3.执行方法* rtValue = pjp.**proceed**(args);  
 *//4.提交事务* **this**.commit();  
 *//返回结果* **return** rtValue;  
 }**catch** (Throwable e){  
 *//5.回滚事务* **this**.rollback();  
 **throw new** RuntimeException(e);  
 }**finally** {  
 *//6.释放资源* **this**.release();  
 }  
}

注解替代：



替代下图开启注解支持



## 优化业务层代码：

1.JdbcSupport类：

自定义：业务层继承创建的JdbcSupport类，将重复的代码放到该类中

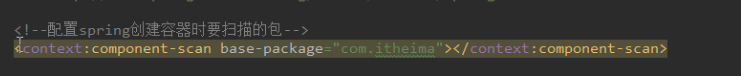
其实不用创建，因为spring提供的有该类：只需要业务层继承spring 提供的JdbcSupport类即可

基于xml的AOP实现事务控制

配置切面，关联通知类

基于注解的的AOP实现事务控制

配置创建对象要扫描的包：可使用@component-scan注解替代



开启spring对注解事务的支持：可使用@E



## Spring提供的事务控制API：；

Spring-tx...jar包内 ---maven需要引入

PlatformTransactionManager接口提供实务操作的方法：

获取事务状态信息：

TransactionStatus GetTransaction（TransactionDefinition definition）

完全基于注解的spring声明式事务控制

提交事务：

Void commit(TransactionStatus status)

回滚事务：

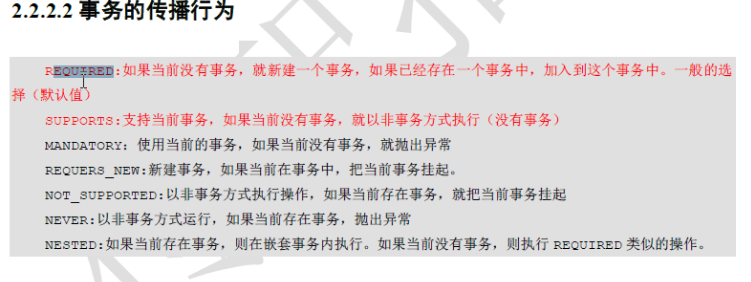
Void rollback(TransactionStatus status)

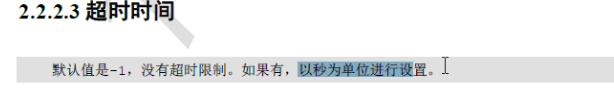
因为它是接口，所有真正发挥作用的是DataSourceTransactionManager实现类

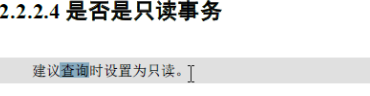
TransactionDefinition：事务的定义信息：隔离级别，传播行为，超时时间，是否可读



传播行为









## spring中基于XML的声明式事务控制配置步骤

1、配置事务管理器

2、配置事务的通知

此时我们需要导入事务的约束 tx名称空间和约束，同时也需要aop的

使用tx:advice标签配置事务通知

属性：

id：给事务通知起一个唯一标识

transaction-manager：给事务通知提供一个事务管理器引用

3、配置AOP中的通用切入点表达式

4、建立事务通知和切入点表达式的对应关系

5、配置事务的属性

是在事务的通知tx:advice标签的内部的<tx:attributes>标签 isolation：用于指定事务的隔离级别。默认值是DEFAULT，表示使用数据库的默认隔离级别。

propagation：用于指定事务的传播行为。默认值是REQUIRED，表示一定会有事务，增删改的选择。查询方法可以选择SUPPORTS。

read-only：用于指定事务是否只读。只有查询方法才能设置为true。默认值是false，表示读写。

timeout：用于指定事务的超时时间，默认值是-1，表示永不超时。如果指定了数值，以秒为单位。

rollback-for：用于指定一个异常，当产生该异常时，事务回滚，产生其他异常时，事务不回滚。没有默认值。表示任何异常都回滚。

no-rollback-for：用于指定一个异常，当产生该异常时，事务不回滚，产生其他异常时事务回滚。没有默认值。表示任何异常都回滚。

<!-- 配置事务管理器 -->

<bean id="transactionManager" class="**org.springframework.jdbc.datasource.DataSourceTransactionManager**">

<property name="dataSource" ref="dataSource"></property>

</bean>

<!-- 配置事务的通知-->

<tx:advice id="txAdvice" transaction-manager="transactionManager">

<**tx:attributes**>

<tx:**method name=**"\*" propagation="REQUIRED" read-only="false"/>

<tx:method name="find\*" propagation="SUPPORTS" read-only="true"></tx:method>

</tx:attributes>

</tx:advice>

<!-- 配置aop-->

<aop:config>

<!-- 配置切入点表达式-->

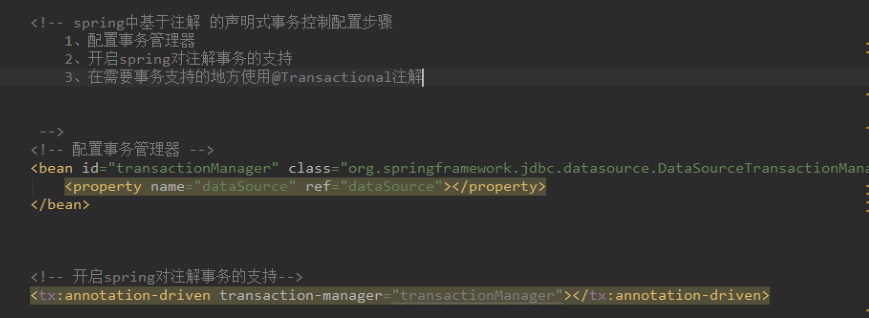
<aop:pointcut id="pt1" expression="execution(\* com.itheima.service.impl.\*.\*(..))"></aop:pointcut>

<!--建立切入点表达式和事务通知的对应关系 -->

<aop:advisor advice-ref="txAdvice" pointcut-ref="pt1"></aop:advisor>

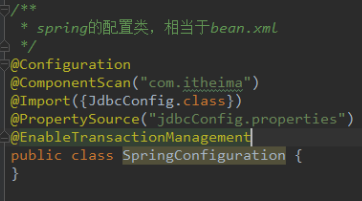
</aop:config>

## Spring基于注解的声明式事务控制



可以加属性也可以不加





第五个注解代替spring开启注解事务的支持

事务管理器：

# 全文检索Lucene：

全文检索流程：

原始文档--->文档--->创建文档对象

获取文档：

文档对象：

为每一个文档创建一个文档对象。

作用：创建文档对象，将非结构化数据结构化的过程

组成：

Filed（域）:全文检索中，存储数据的字段

特点：

一个文档对象可以有多个域（key-value）

可以有相同的域（key和value都相同）

分析文档：

就是分词的过程

一般：

拆分

去除标点

去除停用词

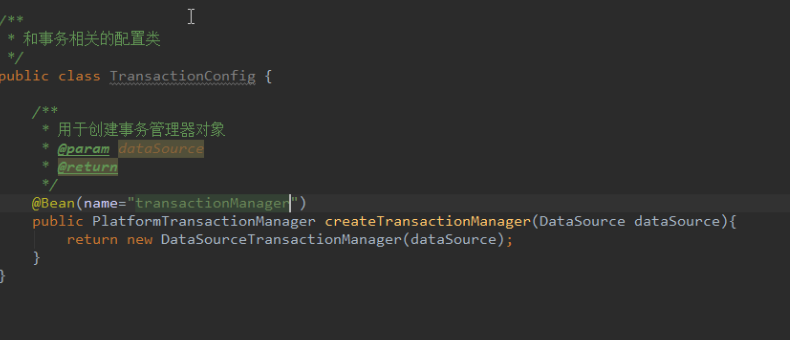
生成关键字对象（Term）:一field多value

创建索引：

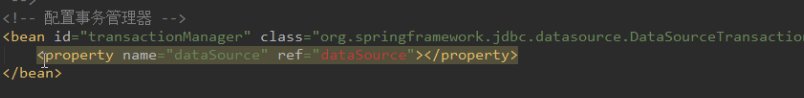
查询索引：

先从

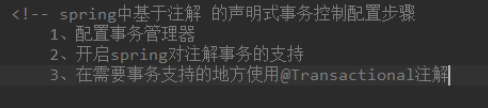
搜索-->索引-->关键字-->域-->文档对象-->原始文档



上图替代下图



编程式事务控制



Spring security 安全框架

1.概念

2.核心思想

3.主要内容

应用场景

a.用户登录认证

b.服务端方法级别权限控制（页面访问权限控制）

在服务端，可通过Spring security中三种类型的注解，进行权限控制：

JSR-250

@Securd

基于表达式的注解

Spring security默认关闭这三种注解，需要在配置文件中开启

1.JSR-250 ：

a.导入依赖: –--jsr250.api

b.spring-security.xml开启注解 :

c.Controller中的方法上使用 ：@RoleAllowed

外来依赖，需要导入依赖

2.@Securd:spring security包中的，不需要再导入依赖

3.支持表达式的注解：可以使用SPEL表达式

Authorise：

use-expression为true时，可以 需要设置use-expression为true

对应的access需要使用hasRole()

c.页面端权限控制

基本步骤：

a.导入jar包

b.编写配置文件

web.xml

spring-security.xml ---开启注解

Demo1: 用户登录验证

Demo2:页面权限控制

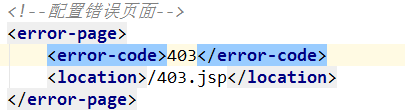
基本步骤：

a.pom.xml中导入jar包

b.在spring-security.xml中开启

c.使用注解

d.设置错误页面（提高用户体验）



# 常用类：

## System类：

**常用静态方法：**

获取系统相关信息或系统操作

System.exit(int status); 退出JVM虚拟机 0正常退出，非0异常退出

System.currentTimeMilliS();//获取当前系统时间

System.arrayCopy(**arr1,arr1Index,arr2,arr2Index,intlength**);

从数组arr1中索引为arr1Index起复制length个元素到目标数组arr2中arr2Index索引起

## Arrays类：

构造方法私有

**常用两静态方法：**

1.public static String **toString**(**数组**)

将参数数组按默认形式转换成字符串[,,,]

2.public static void **sort**(**数组**)

默认升序排序

tips：如果是数值，默认升序

字符串,按字母升序

自定义类型，自定义类型需要有Comparable或Comparator接口的支持

3.扩展：String类的toCharArray()可将字符串转换成字符数组

String str ="safgagag";

char[] ch = str.toCharArray();

## Colloections类：

包：java.util.Collections;

功能：

1.public static <T> boolean addAll(Collection<T> c , T ...elements)往集合中添加一些元素

参数1：集合类； 参数2：可变参数 ； 可用于一次性往集合中添加多个元素

2.p s void shuffle(List<?> list);打乱List集合中元素的顺序；

3.p s <T> void sort(List<T> list)将List集合中元素按照默认规则排序（升序）

使用前提：被排序集合中存储的元素的类**必须实现Comparable<T>接口**，**重写接口中的compareTo方法**，来自定义排序的规则：

自己（this） - 参数：升序

参数 - 自己：降序

基本类型和String已重写，自定义类型一定要实现接口，重写方法

4.p s <T> void sort(List<T> list ， Comparator<? super T>):将List集合中元素按照指定规则排序

**注意：与3不同:**

Comparable:自己与参数比较，需要实现接口，重写compareTo方法

Comparator:相当于第三方判断：两个第三方参数比较，需要创建一个Compartor接口匿名内部类，重写compare方法;而不需要元素所在类实现接口

第一个参数 - 第二个-->升序

第二个参数 - 第一个-->降序

扩展：组合排序：

首先按数字排序，相等的再按首字母排序

----------------------------------------------------------------------------

## Math类：

**静态方法：**

double abs(double num)：绝对值

double ceil(double num):向上取整

double floor(double num)：向下取整

long round(double num)：四舍五入

double Math.PI：圆周率

## Random类：

生成随机数

Random rd = new Random();

int number = rd.nextInt()；//无参，整个范围

int number = rd.nextInt(10)；//有参，指定范围【0，10）

int number = rd.nextInt(10)+n;//范围[0+n,10+n)如+1：[1,11)

...

## Object类：

所有类的根类；子类可使用其所有方法：

常用方法：

1.**String toString**（）：

(1)、功能：返回对象的字符串形式

a.如果重写，返回是toString内的内容，否则，返回的是对象的地址

对象类型@地（getClass().getName()+'@'Integer.toHexString(hashCode())）

b.**直接打印对象名，默认调用toString()方法打印的是地址值：**

(2)、作用:测试，查看对象的属性

tips：一般，判断一个类的对象是否重写toString()方法：直接打印对象名，结果为地址，没重写；否则，重写

特殊情况，直接打印一些对象，结果不是地址，但也没有重写，比较的仍是地址，如：StringBuilder类

2**.bolean equlas(Object obj)**方法：

源码：

public boolean equlas(Object obj){

return this == obj;

}

(1)、比较俩个对象(地址)是否相等

tips：“==”比较

**基本数据类型，比较的是值；引用数据类型比较的是地址值**

(2)、重写equlas方法：

**存在问题：**

隐含一个多态，无法使用子类的内容，需要向下转型；

Object类型，传递参数多样

**优化：**

a.向下转型，判断

b.传递的参数为调用对象本身；null及与调用对象类型不同的判断

public boolean equals(Object o) {

// 如果对象地址一样，则认为相同

if (this == o)

return true;

// 如果参数为空，或者类型信息不一样，则认为不同

if (o == null || getClass() != o.getClass())

return false;

// 转换为当前类型

Person person = (Person) o;

// 要求基本类型相等，并且将引用类型交给java.util.Objects类的equals静态方法取用结果

return age == person.age && Objects.equals(name, person.name);

3.public **static <T> T requireNonNull**(T obj):

查看指定引用对象不是null。

## Date类：

所属包：java.util

**构造方法：**

Date()；**获取当前系统日期与时间**

Date(long date);设置毫秒值，转换为date日期，格林尼治时间

**成员方法：**

**long** **getTime**(long date);获取当前时间距格林尼治时间的毫秒

与System.currentTimeMillis功能相同

void **setTime**(long date);将毫秒值转换为日期 与时间

getTime与setTime互逆

String tolocalString():已过时

## DateFormat类：

**该类是一个抽象类，无法创建对象，内含两抽象方法**，**使用其子类SimpleDateFormat**

抽象方法:

String formt(Date date)将日期按指定的模式转换为字符串

Date parse(String dateStr) thtows parseException ;将字符串转成日期 ,抛出异常，需要处理捕获

## SimpleDateFormat类

DateFormat抽象类的子类

**构造方法：**

SimpleDateFormat(parent:"指定的模式")如yyyy-MM-dd hh:mm:ss -> 2018-8-7 19:07:00

**tips:同一个对象的格式化和解析模式必须相同**

## calender类：

**抽象类**，无法直接创建对象

利用静态方法**static Calender getInstance()创建对象**:使用默认时区和语言环境创建一个日历

**成员字段：**包含大量静态成员变量

**成员方法：**

int get(int field):返回给定日历字段的值

void set (int field ,int value):设置给定日历字段的值

abstract void add(int field,int amount):为给定的日历字段添加或减去指定的时间量

tips：Calender与Date转换：

**Date getTime()**:返回此Calender时间值的Date对象

**void setTime(Date date)**：将指定的Date类日期转换成Calender日历; 使用给定的 Date 设置此 Calendar 的时间。

## String:类

**创建后不能修改和变动;原理是byte[]数组**

在循环中+=效率低（因为生成了很多对象）

创建方法：

1.直接创建：

String s = "abcde";

2.常用构造方法:

String s = new String(...);

三构造方法：

public String()：创建一个空字符串

public String(char[] array)：跟根据char数组内容创建

public String(byte[] array)：跟根据byte数组内容创建

字符串常量池：

只有利用""直接创建的字符串才在字符串常量池中；new的不在

创建两个对象，如果二者值相同，常量池中存储的字符串值重复利用

方法：

String方法：

concat连接 substring子串toLowerCase小写 toUpperCase大写

trim去空格 toString转换成字符串

查找：endsWith和startWith以什么开头或结尾；indexOf 查找字符串出现位置lastIndexOf

比较：equals 判断相等不equalsIgnoreCase **忽略大小写的相等**

代替：replace() replaceAll()

字符及长度：charAt;length

**格式函数：format将一个整数，实数等替换成某种格式**

Tips:字面量相同的字符串相等，但不等于new的

“abc”==”abc”//相等

“abc”！=new String(“abc”)//不等

可共享

String str = "abc";

等效于：

char data[] = {'a', 'b', 'c'};

String str = new String(data);

Tips:

如果常量与变量比较,避免空指针

推荐常量在前面："hello".equals(str1);

比较：equals;equalsIgnoreCase

切割：根据正则表达式规则

返回一个字符串数组

字符串.split（规则）

.;+;\*不能直接当做规则，需要[\\转义成普通字符](file:///\\\\转义成普通字符)

## 字符串缓冲区：

**StringBuilder:多线程，不安全，但效率高**

**StringBuffer:单线程，同步的，安全，但效率低**

**二者方法类似**；存放临时数据的空间，

功能：

提高字符串操作效率；底层也是byte[]数组，但没被final修饰；

长度可变，容量不足时，自动扩充

**常用构造函数：**

无参：默认长度16的

有参（String str）创建一个字符串生成器，并用指定字符串初始化

StringBuffer（）//无参

StringBuffer（int capacity）容量

StringBuffer（String initialString）原始字符串

**成员方法：**

insert(插入位置，内容)//重载,位置后移插入

StringBuilder append(Object o)在末尾加添加内容//多种重载形式，可添加任意类型内容；返回的是this对象，可链式编程

delete(int start, int end)//删除子串

deleteCharAt(int index)//删除指定索引出元素

String toString()

reverse()//倒置

**转换：**

**String与StringBuilder互换**

String--StringBuilder

append(String str)

有参构造方法

StringBuilder--String

toString()

**转换：**

**基本类型--->字符串**

1.+""

2.包装类的静态方法toString()

3.String类的静态方法valueOf()

**字符串 --->基本类型**

1.包装类的静态方法：

public static 包裹类型 parseXxxx（String s）

如： parseInt

tips：字符串参数如果不对，会NumberFormatException异常

# 系统类库：

# 问题与解疑：

# Java单词：

Initialized 初始化

Variable 变量

Exception 异常

Malformed ：难看的，畸形的

Encountered： 遇到，