## 一、Java基础

### 简述&&和&运算符的区别

&&与&都表示两边表达式为true，才返回true

&&前面为false后面就不执行了

&前面为false后面也执行

||前面为true，后面就不执行了

|前面为true后面也执行

### error和exception有什么区别？

Error类一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢等。对于这类错误的导致的应用程序中断，仅靠程序本身无法恢复和和预防，遇到这样的错误，建议让程序终止。

Exception类表示程序可以处理的异常，可以捕获且可能恢复。遇到这类异常，应该尽可能处理异常，使程序恢复运行，而不应该随意终止异常。

Exception类又分为运行时异常（Runtime Exception）和受检查的异常(Checked Exception )，运行时异常;ArithmaticException,IllegalArgumentException，编译能通过，但是一运行就终止了，程序不会处理运行时异常，出现这类异常，程序会终止。而受检查的异常，要么用try。。。catch捕获，要么用throws字句声明抛出，交给它的父类处理，否则编译不会通过。

### String、StringBuffer、StringBuilder有什么区别？

在这方面运行速度快慢为：StringBuilder > StringBuffer > String

String为字符串常量，而StringBuilder和StringBuffer均为字符串变量，即String对象一旦创建之后该对象是不可更改的，但后两者的对象是变量，是可以更改的。

在线程安全上，StringBuilder是线程不安全的，而StringBuffer是线程安全的

String：适用于少量的字符串操作的情况

　　StringBuilder：适用于单线程下在字符缓冲区进行大量操作的情况

　　StringBuffer：适用多线程下在字符缓冲区进行大量操作的情况

### 4）hashMap、hashTable区别

1. 两者最主要的区别在于Hashtable是线程安全，而HashMap则非线程安全。

2.HashMap可以使用null作为key，不过建议还是尽量避免这样使用。HashMap 以null作为key时，总是存储在table数组的第一个节点上。而Hashtable则不允许null作为key。

3.HashMap和Hashtable的底层实现都是数组+链表结构实现。

### linkendList、arrayList、Vector区别

Arraylist基于数组实现，有下标，查询块，增删慢

linkendList 基于链表实现，没有下标，查询慢，增删块（只需要改变指针指向）

Vector与arraylist类似，区别为vector线程安全，arraylist线程不安全

1. 接口与抽象类区别

接口更多的是在系统架构设计方法发挥作用，主要用于定义模块之间的通信契约(制定规则)。而抽象类在代码实现方面发挥作用，可以实现代码的重用

1. 抽象类可以有抽象方法与普通方法、接口全是public abstract修饰的抽象方法
2. 抽象类可以有常量与普通变量，接口中全是public static final修饰的静态常量
3. 抽象类只能被单继承，接口可以多实现
4. 抽象类中可以包含静态方法(static)，接口中不能包含静态方法.
5. 抽象类和接口都不能直接实例化
6. 有抽象方法，abstract修饰的类为抽象类，抽象类里可以没有抽象方法

### 重载与重写的区别

重载是指一个类里，方法名相同，参数不同的多个方法之前的关系。不能通过返回值与权限来判断

参数不同：

1、参数个数不同

2、参数类型不同

3、参数顺序不同

重写：

在继承、实现中当父类方法的功能不能满足子类需求的时候就可以方法重写

1.子类方法覆盖父类方法，必须要保证权限大于等于父类权限

权限修饰符：public >protected >default（不写的情况下）>private

2.方法的返回值类型必须一样

3.方法名必须一致

4.参数必须一致

5.重写的方法抛出的异常只能小于等于父类方法抛出的异常(重写的方法可以不抛异常)

### Final、finally、finalize

（1）如果一个类被声明为final的话，则此类不能被继承；

         如果一个变量声明为final的，变为常量，只能赋值一次；  
         如果一个方法被声明为final的，则此方法只能被使用，不能被重写

（2）在异常处理时，使用finally块来进行必要的清理工作（垃圾回收，关闭资源等），不管是否发生异常都会执行

（finally前面有返回值则返回前面的，如果finally中有返回值则覆盖前面的）  
（3）在垃圾回收器将内存中的对象进行清空之前，允许使用finalize()方法做清理工作

### List、set区别

List:有序，可重复

Set：无序不重复

### 说说你对面向对象的理解

在我理解,面向对象是向现实世界模型的自然延伸，这是一种“万物皆对象”的编程思想

面向对象有三大特性，封装、继承和多态

封装:封装对象的属性和实现细节，对外提供公共的访问方式

* + 1、使代码模块化,提高了代码的复用性。
  + 2、隐藏了实现细节，还要对外提供可以访问的方式。便于调用者的使用。这是核心之一，也可以理解为就是封装的概念。
  + 3、提高了安全性。

继承：从已有类获得信息派生新类的过程

继承的好处：

1.抽取出了重复的代码，减少代码冗余，但是并未减少代码

2.为多态提供的条件

缺点：使代码耦合性增加

### 进程与线程的区别

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.

线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源

.一个进程由一个或多个线程组成，线程是一个进程中代码的不同执行路线；

调度和切换：线程上下文切换比进程上下文切换要快得多。

### 在进行数据库编程时，连接池有什么作用？

1、资源复用

　　　由于数据库连接得到重用，避免了频繁创建、释放连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增进了系统运行环境的平稳性（减少内存碎片以及数据库临时进程/线程的数量）。

　　2、更快的系统响应速度

　　　　数据库连接池在初始化过程中，往往已经创建了若干数据库连接至于池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接，避免了数据库连接初始化和释放过程的时间，从而缩减了系统整体响应时间。

　　3、统一的连接管理，避免数据库连接泄漏

　　　　在较为完备的数据库连接池实现中，可根据预先的连接占用超时设定，强制收回被占用连接。从而避免了常规数据库连接操作中可能出现的资源泄漏。

### Lock与synchronized区别

Synchronized：

1）获取锁的线程执行完了该代码块，然后线程释放对锁的占有；

　 　2）使用Synchronized线程执行发生异常，此时JVM会让线程自动释放锁。

Lock线程发生异常不会自动释放锁 需要unlock()手动释放

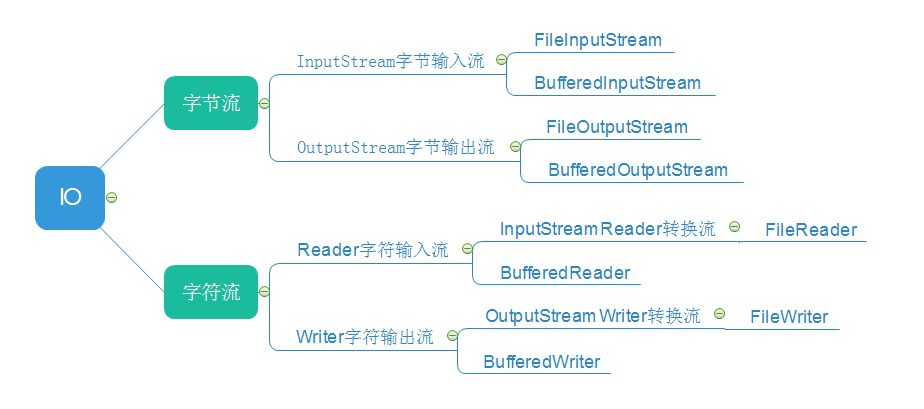
Lock和synchronized有一点非常大的不同，采用synchronized不需要用户去手动释放锁， 当synchronized方法或者synchronized代码块执行完之后，系统会自动让线程释放对锁的 占用；而Lock则必须要用户去手动释放锁，如果没有主动释放锁，就有可能导致出现死锁现象。

### 序列化的意义(方便网络传输)

序列化的作用就是为了不同jvm之间共享实例对象的一种解决方案.

把对象转换为字节序列的过程称为对象的序列化。  
　　把字节序列恢复为对象的过程称为对象的反序列化。  
　　对象的序列化主要有两种用途：  
　　1） 把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中；  
　　2） 在网络上传送对象的字节序列。

### 说出常用io流



### map的三种遍历方式

Entryset,keset,values

### Udp与tcp的区别

1.基于连接与无连接

  2.TCP要求系统资源较多，UDP较少；   
  3.UDP程序结构较简单   
  4.流模式（TCP）与数据报模式(UDP);   
  5.TCP保证数据正确性，UDP可能丢包   
  6.TCP保证数据顺序，UDP不保证

### 18）Java的对象反射是什么？它有什么用？

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；

对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制  
Java反射机制主要提供了以下功能： 在运行时判断任意一个对象所属的类；在运行时构造任意一个类的对象；在运行时判断任意一个类所具有的成员变量和方法；在运行时调用任意一个对象的方法；生成动态代理。

### ） 同步方法和同步代码块的区别是什么？

同步方法默认用this或者当前类class对象作为锁。

同步代码可以选择以什么来加锁，比同步方法更细颗粒化，同步代码可以同步有同步问题的部分代码而不是整个方法。

同步方法用关键字synchronized修饰方法，同步代码主要修饰需要进行同步的代码块，用synchronized（object）｛代码内容｝进行修饰。

### 20.） 什么是死锁(deadlock)？

线程间相互等待对方释放锁

### . Java中的两种异常类型是什么？他们有什么区别？

**运行时异常：**都是RuntimeException类及其子类异常，如NullPointerException(空指针异常)、IndexOutOfBoundsException(下标越界异常)等，这些异常是不检查异常，程序中可以选择捕获处理，也可以不处理。这些异常一般是由程序逻辑错误引起的，程序应该从逻辑角度尽可能避免这类异常的发生。

      运行时异常的特点是Java编译器不会检查它，也就是说，当程序中可能出现这类异常，即使没有用try-catch语句捕获它，也没有用throws子句声明抛出它，也会编译通过。  
       **非运行时异常 （编译异常）：**是RuntimeException以外的异常，类型上都属于Exception类及其子类。从程序语法角度讲是必须进行处理的异常，如果不处理，程序就不能编译通过。如IOException、SQLException等以及用户自定义的Exception异常，一般情况下不自定义检查异常。

### 数组的三种创建方式

1. // 1.方式一  声明 分配空间并赋值
2. **int**[] arr1 = {1,2,3};
3. // 2.方式二 显示初始化
4. **int**[] arr2;
5. arr2 = **new** **int**[]{1,2,3};
6. // 3.方式三 显示初始化()
7. **int**[] arr3;
8. arr3 = **new** **int**[3];

### String str = new String("hello");创建了几个对象 tell me why

1个或者两个，如果常量池中存在了hello就是一个，如果不存在hello的常量池就是两个

1. Math.round(11.5) 等于多少？Math.round(-11.5)等于多少？

12 -11

### switch 是否能作用在byte 上，是否能作用在long 上，是否能作用在String上？

不能作用在long上面

### 用最有效率的方法计算2乘以8？

因为将一个数左移n 位，就相当于乘以了2 的n 次方，那么，一个数乘以8 只要将其左移3 位  
即可，而位运算cpu 直接支持的，效率最高，所以，2 乘以8 等於几的最效率的方法是2 << 3。

### 数组有没有length()方法？String有没有length()方法？

数组是length属性 string是length()方法

### 是否可以继承String类？

不可以继承，string是final修饰的

为什么设计为final类

1. 、确保它们不会在子类中改变语义。String类是final类，这意味着不允许任何人定义String的子类。  
   换言之，  
   如果有一个String的引用，它引用的一定是一个String对象，而不可能是其他类的对象。

2、从效率上讲：   
 1)、设计成final，JVM才不用对相关方法在虚函数表中查询，而直接定位到String类的相关方法上，提高了执行效率。   
2)、Java设计者认为共享带来的效率更高。

1. 总而言之，就是要保证 java.lang.String 引用的对象一定是 java.lang.String的对象，而不是引用它的子孙类，这样才能保证它的效率和安全。

### 、char 型变量中能不能存贮一个中文汉字，为什么？

能

### 列出10个你常见的运行时异常，简单解释？

ja

的length，以免出现这个异常。

java.lang.NoSuchMethodError

方法不存在错误。当应用试图调用某类的某个方法，而该类的定义中没有该方法的定义时抛出该错误。

java.lang.IndexOutOfBoundsException

索引越界异常。当访问某个序列的索引值小于0或大于等于序列大小时，抛出该异常。

java.lang.NumberFormatException

数字格式异常。当试图将一个String转换为指定的数字类型，而该字符串确不满足数字类型要求的格式时，抛出该异常。va.lang.NullPointerException

这个异常的解释是 "程序遇上了空指针 "，简单地说就是调用了未经初始化的对象或者是不存在的对象，这个错误经常出现在创建图片，调用数组这些操作中，比如图片未经初始化，或者图片创建时的路径错误等等。对数组操作中出现空指针，即把数组的初始化和数组元素的初始化混淆起来了。数组的初始化是对数组分配需要的空间，而初始化后的数组，其中的元素并没有实例化，依然是空的，所以还需要对每个元素都进行初始化（如果要调用的话）。

java.lang.ClassNotFoundException

异常的解释是"指定的类不存在"，这里主要考虑一下类的名称和路径是否正确即可

java.lang.ArrayIndexOutOfBoundsException

这个异常的解释是"数组下标越界"，现在程序中大多都有对数组的操作，因此在调用数组的时候一定要认真检查，看自己调用的下标是不是超出了数组的范围，一般来说，显示（即直接用常数当下标）调用不太容易出这样的错，但隐式（即用变量表示下标）调用就经常出错了，还有一种情况，是程序中定义的数组的长度是通过某些特定方法决定的，不是事先声明的，这个时候，最好先查看一下数组

java.sql.SQLException

Sql语句执行异常

java.io.IOException

输入输出异常

java.lang.IllegalArgumentException

方法参数错误

java.lang.IllegalAccessException

无访问权限异常

### 31）Sleep wait yield三个方法的区别

· wait只能在同步（synchronize）环境中被调用，而sleep不需要。详见[Why to wait and notify needs to call from synchronized method](https://link.jianshu.com?t=http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html" \t "_blank)

· 进入wait状态的线程能够被notify和notifyAll线程唤醒，但是进入sleeping状态的线程不能被notify方法唤醒。

· wait通常有条件地执行，线程会一直处于wait状态，直到某个条件变为真。但是sleep仅仅让你的线程进入睡眠状态。

· wait方法在进入wait状态的时候会释放对象的锁，但是sleep方法不会。

· wait方法是针对一个被同步代码块加锁的对象，而sleep是针对一个线程

yield和sleep的主要是，yield方法会临时暂停当前正在执行的线程，来让有同样优先级的正在等待的线程有机会执行。如果没有正在等待的线程，或者所有正在等待的线程的优先级都比较低，那么该线程会继续运行。执行了yield方法的线程什么时候会继续运行由线程调度器来决定，不同的厂商可能有不同的行为。yield方法不保证当前的线程会暂停或者停止，但是可以保证当前线程在调用yield方法时会放弃CPU。

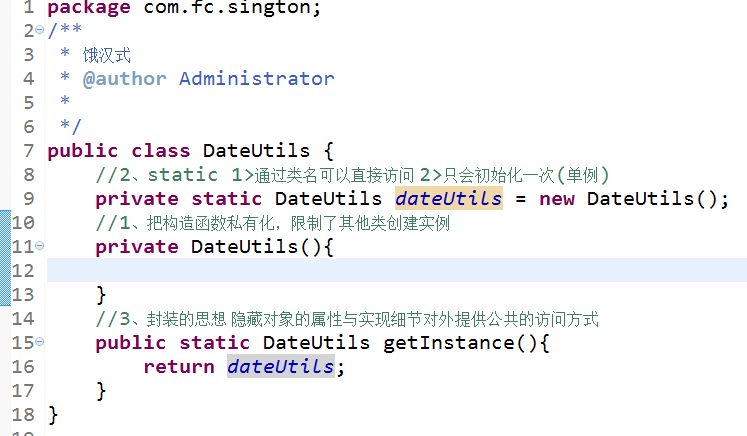
### 32）short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？

有错，表达式自动向大类型转换 题中1是int类型

没错，java自动强制转换

### 单例模式

#### 饿汉式



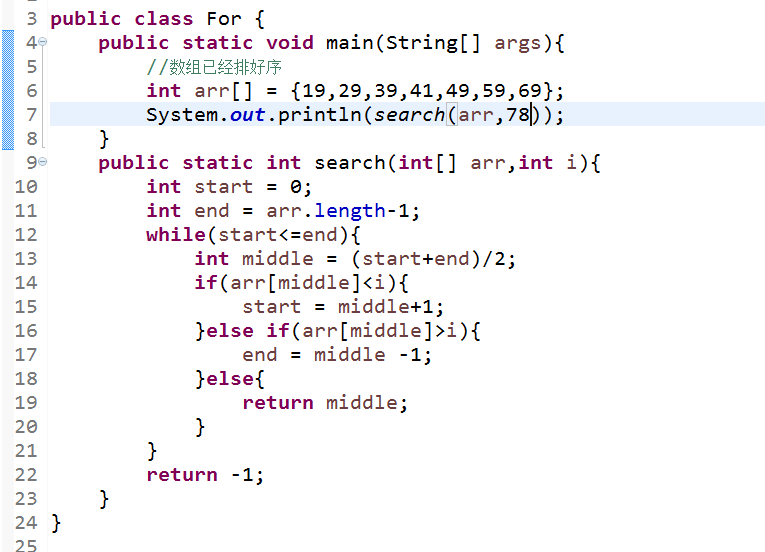
#### 懒汉式:



加强版



### 34）二分查找



### 35）链表反转



### 36）线程生产者消费者模式

|  |
| --- |
| Object objects[] = **new** Object[5];  **int** index = 0;  **boolean** a = **true**;  **public** **synchronized** **void** push(Object obj){//synchronized 相当于synchronized(this)  **if**(index == 5){  **try** {  System.***out***.println("放满了在等待");  a=**false**;  wait();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  objects[index] = obj;  System.***out***.println("放入了"+obj);  System.***out***.println(index++);  notify();  }  **public** **synchronized** Object pop(){  **if**(index==0){  **try** {  wait();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  index --;  Object objj = objects[index];    System.***out***.println("取出了"+objj);  **if**(!a){  System.***out***.println("叫醒队友");  a = **true**;  }  notify();  **return** objj;  }  **public** **static** **void** main(String[] args) {  **final** ThreadDemo threadDemo = **new** ThreadDemo();  **new** Thread(**new** Runnable() {  @Override  **public** **void** run() {  **int** j = 0;  **while**(**true**){  threadDemo.push("放入"+j++);  **if**(j==5){  j=0;  }  }  }  }).start();  **new** Thread(**new** Runnable() {  @Override  **public** **void** run() {  **while**(**true**){  threadDemo.pop();  }  }  }).start();  } |

### 37）计算字符串中字符的个数

|  |
| --- |
| String str = "abcd";  String str2 = "";  **while**(str.length()>0){  str2 = str.substring(0, 1);  System.***out***.println(str2+"的个数为"+(str.length()-str.replace(str2, "").length()));  str = str.replace(str2, "");  } |

### 38)递归实现字符串反转

|  |
| --- |
| **public** **static** String reverse(String str){  **if**(str.length()==1)  **return** str;    **return** *reverse*(str.substring(1))+str.charAt(0);  } |

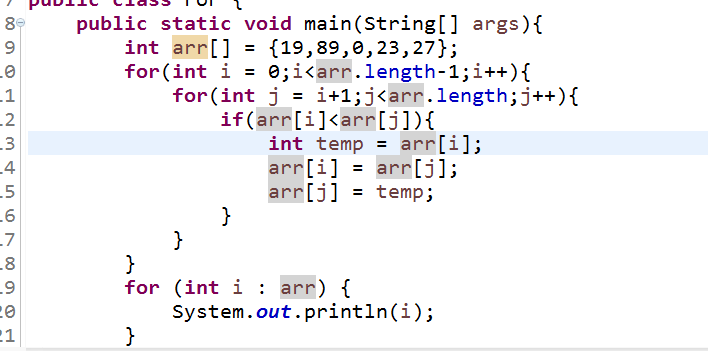
### 线程实现死锁

|  |
| --- |
| **static** **final** String ***a***="lock1";  **static** **final** String ***c***="lock2";  @Override  **public** **void** run() {  **while**(**true**){  **synchronized** (***a***) {  System.***out***.println(Thread.*currentThread*().getName()+"进入a");  **synchronized** (***c***) {  System.***out***.println(Thread.*currentThread*().getName()+"进入c");  }  }  **synchronized** (***c***) {  System.***out***.println(Thread.*currentThread*().getName()+"进入c");  **synchronized** (***a***) {  System.***out***.println(Thread.*currentThread*().getName()+"进入a");  }  }  }  }  **public** **static** **void** main(String[] args) {  **new** Thread(**new** DeadLock()).start();  **new** Thread(**new** DeadLock()).start();  } |

### 40）获取指定字符串中，大写字母、小写字母、数字的个数。

### 2017-12-15_112523

### 选择排序:



### 冒泡排序(过于简单)

### 快速排序（能看懂的看没有写注释）

|  |
| --- |
| **public** **static** **void** fastSort(**int**[] arr,**int** low,**int** up){  **int** begin = low;  **int** end = up;  **int** key = arr[begin];  **while**(begin<end){  **while**(begin<end && arr[end]>=key){  end --;  }  **if**(begin<end && arr[end]<key){  **int** temp = arr[end];  arr[end] = arr[begin];  arr[begin] = temp;  }  **while**(begin<end&& arr[begin]<=key){  begin ++;  }  **if**(begin<end && arr[begin]>key){  **int** temp = arr[end];  arr[end] = arr[begin];  arr[begin] = temp;  }  }  **if**(begin-1>=low)*fastSort*(arr,low,begin-1);  **if**(begin+1<=up)*fastSort*(arr, begin+1, up); |

### 不借助变量交换值

|  |
| --- |
| **int** a = 10;  **int** b = 20;  a = a+b;  b = a-b;  a = a-b;  System.***out***.println(a);  System.***out***.println(b); |

### 工厂模式

### 装饰者模式

## 二、Javaweb

### get与post的区别

get会显示在地址栏、post不会

Post相对于get安全

Post传输的数据量大，get小，post效率相对于get低

Post与get乱码后台处理不一样

增删改用post，查询用get

Get是像服务器请求数据，post是提交数据

### Servlet的生命周期

void init(ServletConfig config):初始化

\* 初始化方法

\* 执行者:服务器

\* 执行次数:一次

\* 执行时机:默认第一次访问的时候

void service(ServletRequest request,ServletResponse response):服务 处理业务逻辑

\* 服务

\* 执行者:服务器

\* 执行次数:请求一次执行一次

\* 执行时机:请求来的时候

void destroy():销毁

\* 销毁

\* 执行者:服务器

\* 执行次数:只执行一次

\* 执行时机:当servlet被移除的时候或者服务器正常关闭的时候

### User表 写出增删改查的sql语句

### 说一下join的用法 左外链接 右外链接 内链接

内连接：INNER JOIN

内连接INNER JOIN是最常用的连接操作。从数学的角度讲就是求两个表的交集，从笛卡尔积的角度讲就是从笛卡尔积中挑出ON子句条件成立的记录。有INNER JOIN，WHERE（等值连接），STRAIGHT\_JOIN,JOIN(省略INNER)

左连接：LEFT JOIN

左连接LEFT JOIN的含义就是求两个表的交集外加左表剩下的数据。

右连接：RIGHT JOIN

同理右连接RIGHT JOIN就是求两个表的交集外加右表剩下的数据。

外连接：OUTER JOIN

外连接就是求两个集合的并集。从笛卡尔积的角度讲就是从笛卡尔积中挑出ON子句条件成立的记录，然后加上左表中剩余的记录，最后加上右表中剩余的记录。另外MySQL不支持OUTER JOIN，但是我们可以对左连接和右连接的结果做UNION操作来实现。

### 重定向与跳转的区别

1. 请求转发url不会改变，重定向url会改变
2. 重定向是response的方法,请求转发是request的方法
3. 重定向效率低，请求转发效率高
4. 重定向不可以传递request作用域的数据，请求转发可以
5. 重定向是发生的浏览器端,跳转是发生服务器端
6. 需要在其它项目中或者站外获得数据必须用重定向

### 给user表增加一列sex varchar 删除一列address

* 查看列：desc 表名;
* 修改表名：alter table t\_book rename to bbb;
* 添加列：alter table 表名 add column 列名 varchar(30);
* 删除列：alter table 表名 drop column 列名;
* 修改列名MySQL： alter table bbb change nnnnn hh int;
* 修改列名SQLServer：exec sp\_rename't\_student.name','nn','column';
* 修改列名Oracle：lter table bbb rename column nnnnn to hh int;
* 修改列属性：alter table t\_book modify name varchar(22);

### Cookie与session的区别

@1:cookie不是很安全,别人可以分析存放在本地的cookie并进行cookie 欺骗,如果主要考虑到安全应当使用session

@2:session会在一定时间内保存在服务器上.当访问增多,会比较占用你服务器的性能,如果主要考虑到减轻服务器性能方面,应当使用cookie

2:单个cookie在客户端的限制是3K,就是说一个站点在客户端存放的cookie不能3K

@3:将登陆信息等重要信息存放为session;其他信息如果需要保留,可以放在 cookie 中

3:session不能区分路径,同一个用户在访问一个网站期间,所有的session在任何一个地方都可以访问到.而cookie中如果设置了路径参数,那么同一个网站中不同路径下的cookie互相是访问不到的.cookie只能是子路径访问父路径设置的cookie

4.session依赖cookie的jsesssionid

### 服务器的三个域对象描述

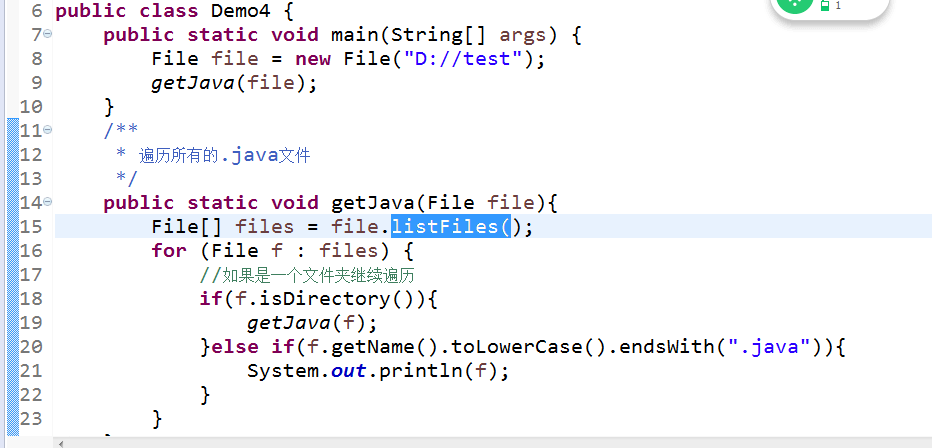
ApplicationContext :整个项目

Session：一次会话

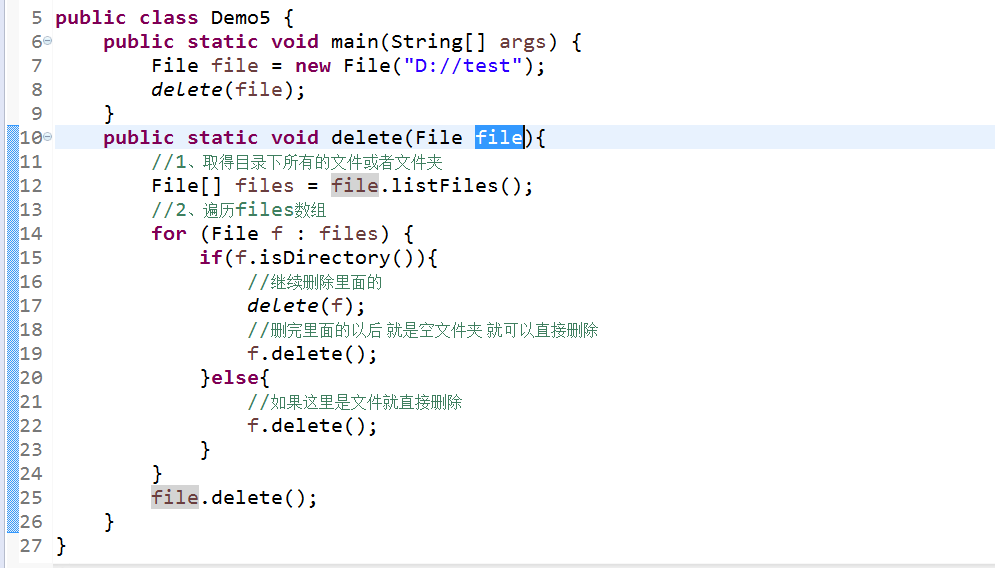
Request：一次请求

### 遍历D:a文件夹下面所有文件

遍历目录下面所有的.java



递归删除文件



### Jsp四大作用域

pageContext（pageContext）：相当于this

request（HttpServletRequest）：一次请求

session（HttpSession）：一次会话

application（ServletContext）：整个项目

### 九大内置对象，并简单描述其作用

page、config、application、request、response、session、out、exception、pageContext

　page：page对象代表当前JSP页面，是当前JSP编译后的Servlet类的对象。相当于this。

　　　　　　config：标识Servlet配置，类型：ServletConfig，api跟Servlet中的ServletConfig对象是一样的，能获取该servlet的一些配置信息，能够获取ServletContext

　　　　　　application：标识web应用上下文，类型：ServletContext，详情就看Servlet中的ServletContext的使用

　　　　　　request:请求对象，　　类型：httpServletRequest

　　　　　　response:响应对象　　类型：httpServletResponse

　　　　　　session：表示一次会话，在服务器端记录用户状信息的技术

　　　　　　out：输出响应体 类型：JspWriter

　　　　　　exception 表示发生异常对象，类型 Throwable，在上面我们介绍page指令中的一个errorPage属性时就有说到他

　　　　　　pageContext：表示 jsp页面上下文（jsp管理者） 类型：PageContext

### 写出静态包含与动态包含的代码，解释区别

动态包含<jsp :include page=“a.jsp”/>与静态包含<%@include  file=“fileurl”%>的区别

1.动态包含用的元素是page，而且有两种形式。静态包含用的是file,只有一种形式。

2.生成的文件不同，静态的包含是将两个jsp文件二合一，生成一个以包含页面命名的servlet和class文件，动态包含的两个jsp文件各自生成自己的servlet和class文件。

3. 动态包含传参方式一：<jsp:include page=“a.jsp?param=123”/>时被包含的jsp页面是可以访问该参数的。

4. 传参方式二：

            <jsp:include page=“a.jsp”>

                     <jsp:param name=“” value=“”>

                    <jsp:param name=“” value=“”>

           </ jsp:include >

5.在客户端访问包含页面时地址栏后面直接加上参数后传递，这种形式的传参是客户端送来的，但是这两个页面的request对象不是同一个，因为3中已经说了包含的页面可以向被包含的页面传递参数，所以被包含的request对象含的参数个数应该大于等于包含页面的参数个数的。所以它们各有各的request对象。而且被包含的jsp页面可以访问传到包含页面的参数。

6.动态包含只有在执行到它的时候才加载，所以它才叫动态包含。

### Mysql limit 的用法

limit翻页两种用法 limit a,b 从a+1开始 显示b条 也就是从第一条开始 a就是0

limit a 从第一条开始 显示a条

### 解释事务

就是一个完整的业务,包含多个操作单元,这些操作要么全部成功,要么全部失败.

### 解释隔离级别与脏读、不可重复读、幻读

事务的特性:★★★

ACID

原子性:事务里面的操作单元不可切割,要么全部成功,要么全部失败

一致性:事务执行前后,业务状态和其他业务状态保持一致. 成不成功都是1600

隔离性:一个事务执行的时候最好不要受到其他事务的影响

持久性:一旦事务提交或者回滚.这个状态都要持久化到数据库中

不考虑隔离性会出现的读问题★★

脏读:在一个事务中读取到另一个事务没有提交的数据

不可重复读:在一个事务中,两次查询的结果不一致(针对的update操作)

虚读(幻读):在一个事务中,两次查询的结果不一致(针对的insert操作)

通过设置数据库的隔离级别来避免上面的问题(理解)

read uncommitted 读未提交 上面的三个问题都会出现

read committed 读已提交 可以避免脏读的发生

repeatable read 可重复读 可以避免脏读和不可重复读的发生

serializable 串行化 可以避免所有的问题

了解

演示脏读的发生:

将数据库的隔离级别设置成 读未提交

set session transaction isolation level read uncommitted;

查看数据库的隔离级别

select @@tx\_isolation;

避免脏读的发生,将隔离级别设置成 读已提交

set session transaction isolation level read committed;

不可避免不可重复读的发生.

避免不可重复读的发生 经隔离级别设置成 可重复读

set session transaction isolation level repeatable read;

演示串行化 可以避免所有的问题

set session transaction isolation level serializable;

锁表的操作.

四种隔离级别的效率

read uncommitted>read committed>repeatable read>serializable

四种隔离级别的安全性

read uncommitted<read committed<repeatable read<serializable

### 简述你所知道的事务传播特性

1. PROPAGATION\_REQUIRED: 如果存在一个事务，则支持当前事务。如果没有事务则开启；

2. PROPAGATION\_SUPPORTS: 如果存在一个事务，支持当前事务。如果没有事务，则非事务的执行；

3. PROPAGATION\_MANDATORY: 如果已经存在一个事务，支持当前事务。如果没有一个活动的事务，则抛出异常；

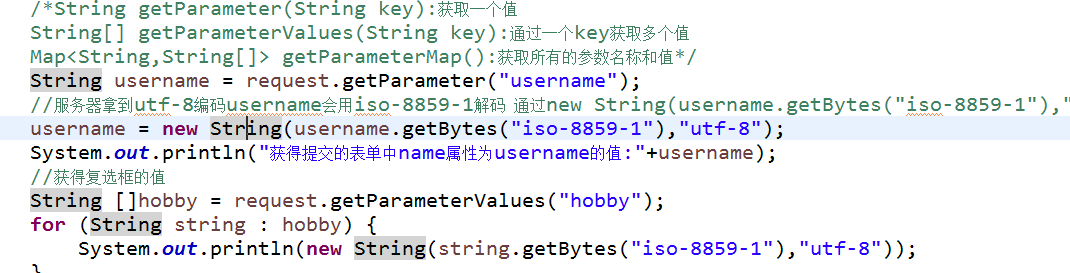
4. PROPAGATION\_REQUIRES\_NEW: 总是开启一个新的事务。如果一个事务已经存在，则将这个存在的事务挂起；

5. PROPAGATION\_NOT\_SUPPORTED: 总是非事务地执行，并挂起任何存在的事务；

6. PROPAGATION\_NEVER: 总是非事务地执行，如果存在一个活动事务，则抛出异常；

7. PROPAGATION\_NESTED：如果一个活动的事务存在，则运行在一个嵌套的事务中. 如果没有活动事务, 则按TransactionDefinition.PROPAGATION\_REQUIRED 属性执行。

### post与get乱码处理



New String(value.getBytes("iso-8859-1")

,"utf-8")(post也可以用这种方式解码)

Post专有解码方式：

request.setCharacterEncoding("utf-8");

### 简述jdbc的步骤

 (1).注册驱动  
        Class.forName("oracle.jdbc.OracleDriver");  
    (2).连接数据库  
        String url = "jdbc:oracle:thin:@localhost:1521:xe";//其中xe为sid  
        String user = "XXX";  
        String password = "XXX";  
        Connection conn = DriverManager.getConnection(url,name,password);  
    (3)获得执行平台statement 为了防止sql注入这里用预编译平台preparedStatement  
        Statement state = conn.createStatement();  
    (4).,执行SQL语句  
        String sql = "select id,name from s\_emp";   //"insert into s\_emp(id,name) values(12,'zhangsan')";  
        ResultSet rs = state.executeQuery(sql);  
    (5).处理结果集  
        while(rs.next()){  
            int id = rs.getInt("id");  
            String name = rs.getString(2);  
            System.out.println(id+" "+name);  
        }  
    (6).关闭连接  
        rs.close();  
        state.close();  
        conn.close();

### 什么是sql注入,jdbc中如何避免

字符串拼接加上恒成立条件就可能造成sql注入

采用预编译

### 监听器有哪些作用和用法？

Java Web开发中的监听器（listener）就是application、session、request三个对象创建、销毁或者往其中添加修改删除属性时自动执行代码的功能组件，如下所示：  
①ServletContextListener：对Servlet上下文的创建和销毁进行监听。  
②ServletContextAttributeListener：监听Servlet上下文属性的添加、删除和替换。  
③HttpSessionListener：对Session的创建和销毁进行监听。

④HttpSessionAttributeListener：对Session对象中属性的添加、删除和替换进行监听。  
⑤ServletRequestListener：对请求对象的初始化和销毁进行监听。  
⑥ServletRequestAttributeListener：对请求对象属性的添加、删除和替换进行监听。

### Session失效的做法

1. 在web容器中设置（此处以tomcat为例）

 <session-config>

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

</session-config>

1. 在工程的web.xml中设置

<session-config>

      <session-timeout>15</session-timeout>

</session-config>

1. 通过java代码设置

session.setMaxInactiveInterval（30\*60）;//以秒为单位

或者session.invalidate()方法

### Jquery的常用选择器

https://blog.csdn.net/pseudonym\_/article/details/76093261博客

1. 基本选择器
2. 层级选择器
3. 过滤选择器
4. 属性过滤选择器
5. 子元素过滤选择器
6. 表单对象属性过滤选择器
7. 表单选择器

### 原生js获得对象的方式

1. id ：  document.getElementById('id')

2. 标签 : document.getElementsByTagName('标签') //获得的是一个标签数组

3. Name :document.getElementsByName（'name名'） // 该方法与 getElementById() 方法相似，但是它查询元素的 name 属性，而不是 id 属性

4. className:document.getElementsByClassName('class') //获得一个class的数组

### 写出jquery ajax三种方式 写出参数并解释

1. $.ajax({
2. url : "AjaxJson",   //请求url
3. type : "POST",  //请求类型  post|get
4. // data : "key=value&key1=value2", //后台用 request.getParameter("key");
5. dataType : "json",  //返回数据的 类型 text|json|html--
6. success : **function**(users){  //回调函数 和 后台返回的 数据
8. },
9. Error:function(){
   * + 1. }
10. }
11. });
12. });
13. // get 方式 发送请求
14. // $.get(url,data,callback,type); data为 发送的数据  {key:value}形式 ，
15. //  callback成功时的回调函数 function(result){}  result 后台返回的数据  ，
16. //type  返回的数据类型  默认为 text ,可以设为 json text
17. //post 方式 发送请求
18. // $.post(url,data,callback,type);

### 描述ajax get与post的区别

Get方式:

用get方式可传送简单数据，但大小一般限制在1KB下，数据追加到url中发送（http的header传送），也就是说，浏览器将各个表单字段元素及其数据按照URL参数的格式附加在请求行中的资源路径后面。另外最重要的一点是，它会被客户端的浏览器缓存起来，那么，别人就可以从浏览器的历史记录中，读取到此客户的数据，比如帐号和密码等。因此，在某些情况下，get方法会带来严重的安全性问题。

Post方式:

当使用POST方式时，浏览器把各表单字段元素及其数据作为HTTP消息的实体内容发送给Web服务器，而不是作为URL地址的参数进行传递，使用POST方式传递的数据量要比使用GET方式传送的数据量大的多。

总之，GET方式传送数据量小，处理效率高，安全性低，会被缓存，而POST反之。

### 说出你所知道会话跟踪技术

1. cookie：
2. Session：
3. Url重写
4. 隐藏域

### 过滤器的生命周期及作用

fiter:init()---启动服务  
fiter:service()---请求

fiter:destroy()---关闭服务

servlet:init()  
servlet:service()  
servlet:destroy() 

作用：乱码处理、登录拦截、权限拦截等

### 什么是mvc模式

MVC 的全称为model-view-controller(模型-视图-控制器)。MVC 是一种开发应用程序  
的模式，这个模式已经具有了很好的框架架构，并且十分容易维护。使用MVC 开发出来的应  
用程序一般包括以下几块内容：  
· 控制器(Controller)：控制器类处理客户端向Web 应用程序发出的请求，获取数据，  
并指定返回给客户端，用来显示处理结果的视图。  
· 模型（Model）：模型类代表了应用程序的数据，这些数据通常具有一个数据验证逻  
辑，用来使得这些数据必须符合业务逻辑。  
· 视图（View）:视图类是Web 应用程序中用来生成并显示HTML 格式的服务器端对客  
户端请求的响应结果的模板文件

### 什么是tcp

TCP(Transmission Control Protocol [传输控制协议](https://www.baidu.com/s?wd=%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE&tn=SE_PcZhidaonwhc_ngpagmjz&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnWR1PH63uAFWuyDdn1bd0ZwV5Hcvrjm3rH6sPfKWUMw85NP7gdKWyh9GuA7EUMICmdq8uLK9uvdJX6K1TL0qnfK1TL0z5HD0IgF_5y9YIZ0lQzqlpA-bmyt8mh7GuZR8mvqVQL7dugPYpyq8Q1DzP1m1P1Tz" \t "https://zhidao.baidu.com/question/_blank))是一种面向连接的、可靠的、基于字节流的传输层通信协议

### Tcp与udp的区别是什么

1。TCP是基于连接的，UDP是基于无连接 2。对系统资源的要求（TCP较多，UDP少） 3。UDP程序结构较简单 4。流模式与数据报模式 5。TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证

### 什么tcp的三次握手

第一次握手：建立连接时,客户端发送syn包(syn=j)到服务器,并进入SYN\_SEND状态,等待服务器确认；   
SYN：同步序列编号(Synchronize Sequence Numbers)  
第二次握手：服务器收到syn包,必须确认客户的SYN（ack=j+1）,同时自己也发送一个SYN包（syn=k）,即SYN+ACK包,此时服务器进入[SYN\_RECV](https://www.baidu.com/s?wd=SYN_RECV&tn=SE_PcZhidaonwhc_ngpagmjz&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnHbvPHR1rAF9uHT4ryRs0ZwV5Hcvrjm3rH6sPfKWUMw85NP7gdKWyh9GuA7EUMICmdq8uLK9uvdJX6K1TL0qnfK1TL0z5HD0IgF_5y9YIZ0lQzqlpA-bmyt8mh7GuZR8mvqVQL7dugPYpyq8Q1f1rHRLnjfsn0" \t "https://zhidao.baidu.com/question/_blank)状态；   
第三次握手：客户端收到服务器的SYN＋ACK包,向服务器发送确认包ACK(ack=k+1),此包发送完毕,客户端和服务器进入ESTABLISHED状态,完成三次握手.  
完成三次握手,客户端与服务器开始传送数据

### List<User>:user 属性有name，age 运用标准的json格式描述该集合

[

{“name”:”张三”,”age”:20,book:{name:1111,money:12}},

{“name”:”李四”,”age”:20},

{“name”:”王五”,”age”:20}

]

### jquery循环的方式

1、选择器+遍历

$('div').each(function (i){

   i就是索引值

   this 表示获取遍历每一个dom对象

});

2、选择器+遍历

$('div').each(function (index,domEle){

   index就是索引值

  domEle 表示获取遍历每一个dom对象

});

3、更适用的遍历方法

1）先获取某个集合对象

2）遍历集合对象的每一个元素

var d=$("div");

$.each(d,function (index,domEle){

  d是要遍历的集合

  index就是索引值

  domEle 表示获取遍历每一个dom对

});

### Js定义数组的方式

1. /\*
2. 数组：
3. 1、数组内可以存放任意类型的数据
4. 2、数组元素不赋值，则为undefined
5. 3、打印数组时，如果某个元素没有赋值，则为“”
6. 4、访问数组范围以外的元素时，不会出现越界异常，为undefined
7. 5、定义的数组大小，依然可以添加更多的元素
8. \*/
10. //第一种
11. var arr = new Array();
12. arr[0] = 123
13. arr[1] = "abc";
14. arr[5] = false;
15. alert(arr[3]);//undefined
16. alert(arr);//123,abc,,,,false
17. alert(arr[10]);//undefined，不会出现越界异常
19. //for in 遍历，未定义的不遍历
20. for(var i in arr){
21. alert(arr[i]);
22. }

25. //第二种
26. var arr2 = new Array(2);//规定了数组的长度为2
27. arr2[0] = "ddd";
28. arr2[1] = "www";
29. alert(arr2);//ddd,www
30. arr2[2] = "fff";//可以再次添加元素，定义的数组大小对此没有影响
31. alert(arr2);//ddd,www,fff
33. //第三种
34. var arr3 = new Array(true,123,false);
35. alert(arr3);//true,123,false
37. //第四种,推荐使用
38. var arr4 = [false,"aaa",123];
39. alert(arr4);//false,aaa,123

### ==与===的区别

”==”与”===”是不同的,

==是判断值是否相等,

===是判断值及类型是否完全相等

### Jsp常用指令有哪些

1、①page  指定编码   <%@ page language="java" contenType="text/html;charset=utf-8" pageEncoding="utf-8" >

                        ②page 导包  <%@ page import="java.util.\* "%>

  2、include 静态包含  <%@ include file="include.jsp"%>

3.taglib   是用来在当前jsp页面中导入第三方的标签库 <%@ taglib prefix="c" url="http://java.sun.com/jsp/jstl/core" %>    prefix：指定标签前缀 （可以随意起名） url：指定第三方标签库的url（唯一标识） 当然 需要先把第三方标签库所需jar包放到类路径中。

### 37 jsp常用动作：

forward动作:  
跳转页面:<jsp:forward page="login.jsp" />

param动作(传递参数)):  
到达跳转页面可以通过 request.getParameter(“参数名”)方式取出参数值  
<jsp:include page="转向页面的url" >  
           <jsp:param   name="参数名1" value=http://www.hake.cc/kf/201109/"参数值1">

include 动态包含(分别编译):  
用jsp:include动作实现<jsp: include page="included.jsp" flush="true" />

### 38自动登录实现

|  |
| --- |
| @Override  **public** **void** doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2)  **throws** IOException, ServletException {  HttpServletRequest request1 = (HttpServletRequest)arg0;  MyRequest request = **new** MyRequest(request1);  //从作用域中取session之前就需要先判断是否是在访问需要登录才能访问资源  //注册与登录 register、login  //System.out.println(request.getRequestURI());  String url = request.getRequestURL().toString();  //自己写的项目知道 登录与注册 包含了什么请求地址 把包含了这两个请求地址的放过  **if**(url.contains("loginA")||url.contains("register")){  arg2.doFilter(request, arg1);  }**else**{  //因为登录成功以后把登录信息放在了session 现在就从session中取出来  HttpSession session = request.getSession(**true**);  User user = (User) session.getAttribute("user");  **if**(user==**null**){//作用域session中取不到 就没登录  //需要判断是否选择了自动登录  //那么从cookie中就能取出用户名与密码  //拿到所有的cookies  Cookie[] cookies = request.getCookies();  **if**(cookies!=**null**&&cookies.length>0){  //遍历数组拿到保存用户名与密码的cookie  **for** (Cookie cookie : cookies) {  **if**(cookie.getName().equals("userCookie")){  //拿到用户名跟密码  String[] str = cookie.getValue().split("-");//admin-1234  //需要登录 把用户名跟密码送到控制层LoginController  //拿到用户名跟密码  request.getRequestDispatcher("loginA?username="+str[0]+"&password="+str[1]).forward(request, arg1);  **return**;  }  }  }  request.setAttribute("msg", "请先登录");  request.getRequestDispatcher("login.jsp").forward(request, arg1);  }**else**{//session中能取到 说明才登录  arg2.doFilter(request, arg1);  }  } |
| **else**{//登录成功  //把用户信息放入session中  request.getSession().setAttribute("user", user);  String auto = request.getParameter("auto");  **if**(auto!=**null**){//说明用户选择了自动登录  //把用户跟密码写入cookie中才能完成自动登录  Cookie cookie = **new** Cookie("userCookie",username+"-"+password);  cookie.setMaxAge(600);  response.addCookie(cookie);  } |

## 三、框架部分

### 简述悲观锁、乐观锁

悲观锁：是认为自己在处理数据的时候一定会有其它事务老干扰，所以强行上锁，其它事务不能处理数据，效率低。

乐观锁：是认为自己在处理数据的时候不会有其它事务来干扰，不上锁，但是为了数据正确，加一个版本列version，读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据

### MyBatis中使用#和$书写占位符有什么区别？

1. ＃{}将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号。如：order by #{id}，如果传入的值是111,那么解析成sql时的值为order by “111”, 如果传入的值是id，则解析成的sql为order by “id”。
2. ${}将传入的数据直接显示生成在sql中。如：order by   
   ${id}，如果传入的值是111,那么解析成sql时的值为order by 111, 如果传入的值是id，则解析成的sql为order   
   by id。
3. #方式能够很大程度防止sql注入。
4. $方式无法防止Sql注入。
5. $方式一般用于传入数据库对象，例如传入表名.
6. 一般能用#的就别用$.

### MyBatis中的动态SQL是什么意思？列举出常用的标签

* if
* where
* set
* choose（when，otherwise）
* trim
* Foreach

简单举例

|  |
| --- |
| <sql id=*"queryColumn"*>  UID,UNAME,PASSWORD  </sql>  <insert id=*"insert"* parameterType=*"user"*>  insert into user  <!--prefix：前缀 suffix:后缀 -->  <trim prefix=*"("* suffix=*")"* suffixOverrides=*","*>  <if test=*"uname != null"*>  uname,  </if>  <if test=*"password != null"*>  password,  </if>  </trim>  <trim prefix=*"values ("* suffix=*")"* suffixOverrides=*","*>  <if test=*"uname != null"*>  #{uname},  </if>  <if test=*"password != null"*>  #{password},  </if>  </trim>  </insert>  <update id=*"update"* parameterType=*"user"*>  <!-- update user set uname=?,password=? where uid=? -->  update User  <!-- 自动会去掉最后一个, -->  <set>  <if test=*"uname!=null"*>  uname=#{uname},  </if>  <if test=*"password!=null"*>  password=#{password},  </if>  </set>  <![CDATA[where uid<#{uid}]]>  </update>  <delete id=*"deleteByIds"*>  <!-- DELETE FROM USER WHERE uid IN(1,2,3) -->  <!-- 集合 collection="list" index:下标0开始 item每个元素-->  delete from user where uid in  <foreach collection=*"list"* open=*"("* close=*")"* index=*"index"* item=*"a"* separator=*","*>  #{a},#{index}  </foreach>  </delete>  <delete id=*"deleteByIds2"*>  <!-- DELETE FROM USER WHERE uid IN(1,2,3) -->  <!-- 数组 collection="array"必须是array index:下标0开始 item每个元素-->  delete from user where uid in  <foreach collection=*"array"* open=*"("* close=*")"* index=*"index"* item=*"a"* separator=*","*>  #{a},#{index}  </foreach>  </delete>  <!-- 传入多个参数  1、用下标第一个参数0 不要写paramterType  2、可以通过@param取名字  -->  <select id=*"query"* resultType=*"user"*>  select \* from user where uname=#{u } and uid=#{1}  </select> |

### 什么是IoC和DI？DI是如何实现的？

IoC叫控制反转，是Inversion of Control的缩写，控制反转是把传统上由程序代码直接操控的对象的调用权交给容器，通过容器来实现对象组件的装配和管理。所谓的"控制反转"就是对组件对象控制权的转移，从程序代码本身转移到了外部容器，由容器来创建对象并管理对象之间的依赖关系。

控制反转——Spring通过一种称作控制反转（IoC）的技术促进了松耦合。当应用了IoC，一个对象依赖的其它对象会通过被动的方式传递进来，而不是这个对象自己创建或者查找依赖对象，是容器在对象初始化时不等对象请求就主动将依赖传递给它。通过IOC反转控制DI依赖注入完成各个层之间的注入，使得层与层之间实现解耦合，增加运行效率利于维护。

例子：造车

### 简述mybatis延迟加载步骤

mybatis一对一，一对多的时候可以采用延迟加载的技术

在sqlMapConfig中配置

<settings>

<!-- 打开延迟加载的开关 -->

<setting name=*"lazyLoadingEnabled"* value=*"true"* />

<!-- 将积极加载改为消极加载，即延迟加载 -->

<setting name=*"aggressiveLazyLoading"* value=*"false"* />

</settings>

### Mybatis如何实现翻页

导入jar

|  |
| --- |
| <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>${pagehelper.version}</version>  </dependency> |

在sqlMapConfig中配置插件

|  |
| --- |
| <plugins>  <plugin interceptor=*"com.github.pagehelper.PageHelper"*>  <property name=*"dialect"* value=*"mysql"* />  <!-- <property name="offsetAsPageNum" value="false"/> <property name="rowBoundsWithCount"  value="false"/> <property name="pageSizeZero" value="true"/> <property name="reasonable"  value="false"/> <property name="supportMethodsArguments" value="false"/>  <property name="returnPageInfo" value="none"/> -->  </plugin>  </plugins> |

运用

|  |
| --- |
| PageHelper.*startPage*(curPage, pageUtils.getPageSize());  Page<UserInfo> page = (Page<UserInfo>)userInfoMapper.queryAll(userInfo);  //设置查询到的数据  pageUtils.setDataList(page.getResult());  //设置总条数  pageUtils.setTotalCount((**int**)page.getTotal()); |

### Spring中Bean的作用域有哪些？

**1、[singleton](http://student.csdn.net/source/csdn/fckeditor/editor/ch03s04.html" \l "beans-factory-scopes-singleton" \t "https://blog.csdn.net/fengruifang/article/details/_blank)**:默认配置表示单例

<bean id="empServiceImpl" class="cn.csdn.service.EmpServiceImpl" scope="singleton">

1. **[prototype](http://student.csdn.net/source/csdn/fckeditor/editor/ch03s04.html" \l "beans-factory-scopes-prototype" \t "https://blog.csdn.net/fengruifang/article/details/_blank)**：多例

**3、[request](http://student.csdn.net/source/csdn/fckeditor/editor/ch03s04.html" \l "beans-factory-scopes-request" \t "https://blog.csdn.net/fengruifang/article/details/_blank)**：

**4、[session](http://student.csdn.net/source/csdn/fckeditor/editor/ch03s04.html" \l "beans-factory-scopes-global-session" \t "https://blog.csdn.net/fengruifang/article/details/_blank)**：

**5、[global session](http://student.csdn.net/source/csdn/fckeditor/editor/ch03s04.html" \l "beans-factory-scopes-global-session" \t "https://blog.csdn.net/fengruifang/article/details/_blank)：**在一个全局的HTTP Session中，一个bean定义对应一个实例。典型情况下，仅在使用portlet context的时候有效。该作用域仅在基于web的Spring ApplicationContext情形下有效。

### 解释一下什么叫AOP（面向切面编程）？

Aop是面向切面编程，基于动态代理实现，主要作用是解除耦合

Aop的功能是在不阻碍业务逻辑代码纵向运行的情况下，横向的切入比如，日志，权限，事务等逻辑。我在项目中的运用是通过spring配置事务，从而完成在不改变service代码的基础上通过aop对其中的方法增加事务，从而完成了对业务逻辑与事务的解耦

Spring配置实现：

|  |
| --- |
| <!-- 事务管理器 -->  <bean id=*"transactionManager"*  class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 数据源 -->  <property name=*"dataSource"* ref=*"dataSource"* />  </bean>  <!-- 通知 根据不同的方法做不同处理 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <!-- 传播行为 -->  <tx:method name=*"save\*"* propagation=*"REQUIRED"*/>  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"add\*"* propagation=*"REQUIRED"* />  <tx:method name=*"create\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"login\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <!-- 切面 -->  <aop:config>  <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* com.feicui.service.\*.\*(..))"* />  </aop:config> |



代理：找我做事情，我把他交给一个跟我有一定联系的类来做叫代理

这里的我是委托类（被代理类），找的那个类就代理类

优点一：可以隐藏委托类的实现;

优点二：可以实现客户与委托类间的解耦，在不修改委托类代码的情况下能够做一些额外的处理（在方法中增加额外功能）。

静态代理：若代理类在程序运行前就已经存在，那么这种代理方式被成为 静态代理

缺点：

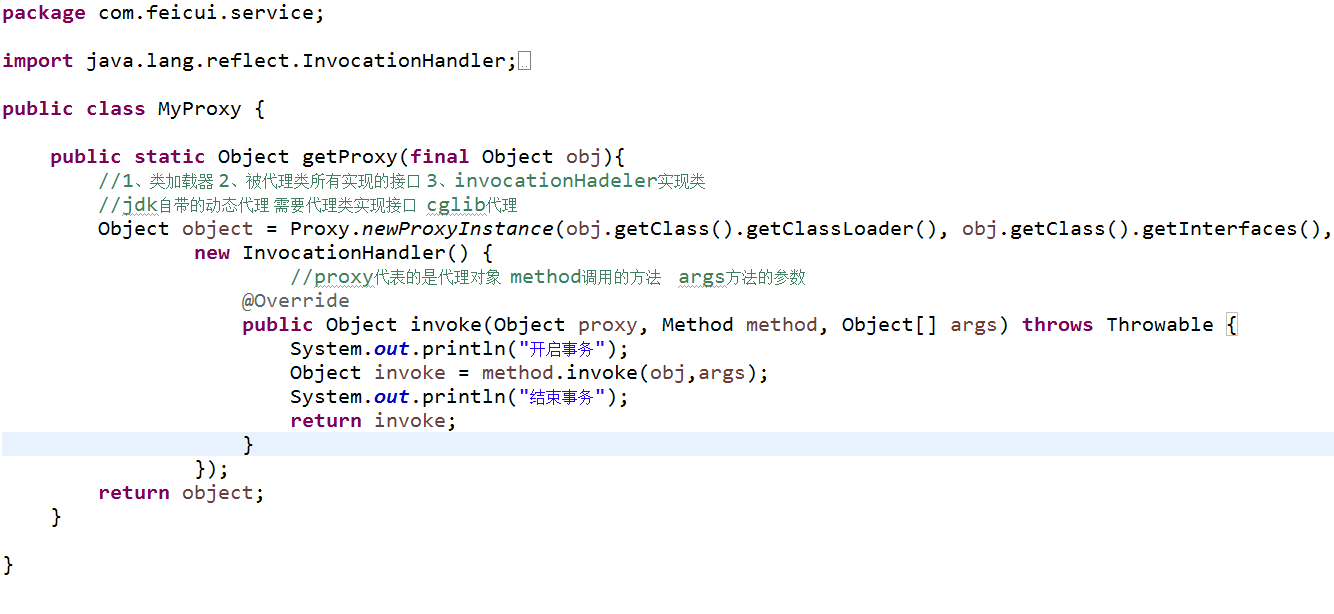
静态代理的局限在于运行前必须编写好代理类

会造成类的爆炸，过于麻烦，冗余代码太多，扩展麻烦

动态代理：代理类在程序运行时动态创建的代理方式被成为 动态代理

相对于静态代理，这儿只需要写一个类或者一个方法就能对任意类实现代理，任意方法进行增强

Jdk动态代理实现：final object obj传入需要代理的类



### 你如何理解AOP中的连接点（Joinpoint）、切点（Pointcut）、增强（Advice）、引介（Introduction）、织入（Weaving）、切面（Aspect）这些概念？

|  |
| --- |
| Joinpoint(连接点):需要被代理的类中的所有方法  所谓连接点是指那些被拦截到的点。在spring中,这些点指的是方法,因为spring只支持方法类型的连接点.  Pointcut(切入点):需要被增强的方法  所谓切入点是指我们要对哪些Joinpoint进行拦截的定义.  Advice(通知/增强):在原有方法上加强的一些代码  所谓通知是指拦截到Joinpoint之后所要做的事情就是通知.通知分为前置通知,后置通知,异常通知,最终通知,环绕通知(切面要完成的功能)  Introduction(引介):引介是一种特殊的通知在不修改类代码的前提下, Introduction可以在运行期为类动态地添加一些方法或Field.  Target(目标对象):被代理的类  代理的目标对象  Weaving(织入):生成代理对象过程，  是指把增强应用到目标对象来创建新的代理对象的过程.  spring采用动态代理织入，而AspectJ采用编译期织入和类装在期织入  Proxy（代理）:代理类  一个类被AOP织入增强后，就产生一个结果代理类  Aspect(切面):  是切入点和通知（引介）的结合 |

### Spring中如何使用注解来配置Bean？有哪些相关的注解？

在配置中扫描包：

|  |
| --- |
| <context:component-scan base-package=*"com.feicui.service,com.feicui.utils"*></context:component-scan> |

常用注解：在类中使用

@Component:包括下面三个（自定义类等使用）

@Repository:告诉别人这个类是dao层

@service：告诉别人这个类是业务层

@controller：告诉别人这个类是控制层

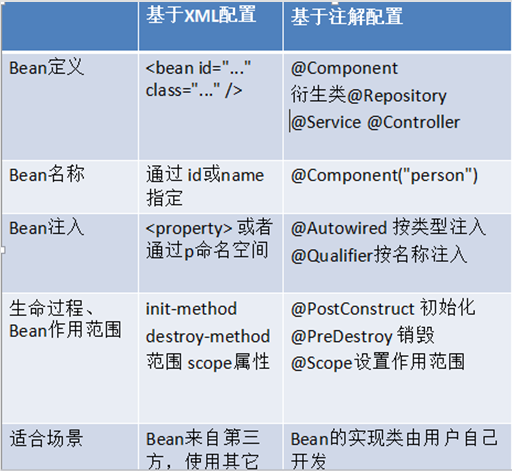
//@Autowired 自动装配

//@Qualifier("dao")//强制要求指定名字

//只写一个@Resource 跟@Autowired

//@Resource(name="adao")相当于Autowired+Qualifier 必须得根据name的value

@Resource(name="dao")//注解是java自带的 name



### 如何在Web项目中配置Spring的IoC容器？

|  |
| --- |
| <!-- 1、通过id或者name配置唯一标识  //加载spring容器  ClassPathXmlApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext2.xml");  //根据id或者name取  Car car = (Car) ac.getBean("car");  2、scope指定作用域  3、init-method、destroy-method指定初始化与销毁bean执行的方法  -->  <bean id=*"car"* class=*"com.feicui.bean.Car"* scope=*"singleton"* init-method=*"init"* destroy-method=*"destroy"*></bean> |

### 如何在Web项目中配置Spring MVC？

|  |
| --- |
| <!-- springMVC的配置 -->  <servlet>  <servlet-name>spring</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>spring</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping> |

### 依赖注入的三种方式

1. 构造方法注入
2. setter注入
3. 接口注入

|  |
| --- |
| bean id=*"car"* class=*"com.feicui.bean.Car"*>  <!-- 构造注入对象  把这个看成构造方法  <constructor-arg ref="cheShen"></constructor-arg>  引用了车身 首先得有车身 所以第一步初始化车身  -->  <!-- <constructor-arg ref="cheShen"></constructor-arg> -->  <!-- 初始化属性  <constructor-arg name="name" value="郑勇"></constructor-arg>  <constructor-arg name="i" value="3"></constructor-arg>  -->  <!-- set注入属性 -->  <property name=*"name"* value=*"郑勇"*></property>  <property name=*"i"* value=*"2"*></property>  <property name=*"cheShen"* ref=*"cheShen"*></property>  <property name=*"list"*>  <list>  <value>点点点</value>  <value>ffff</value>  </list>  </property>  </bean>  <bean id=*"cheShen"* class=*"com.feicui.bean.CheShen"*> |

### Spring MVC的工作原理、流程是怎样的？

启动配置文件就注册：HandlerMapping (处理映射器)

map key---value

1. url找到DispatcherServlet
2. DispatcherServlet通知HandlerMapping 来了一个url
3. HandlerMapper 取得对应value 处理器的信息（controller）
4. HandlerMapper 处理器的信息及处理器拦截器(如果有的话)返回给DispatcherServlet
5. DispatcherServlet 把获得的类信息给对应handlerAdapter(处理适配器)
6. handlerAdapter 通过反射初始化类 调用方法 运行处理器(controller)
7. 执行结束返回一个ModelAndView给DispatcherServlet
8. DispatcherServlet 把获得的ModelAndView交给ViewReslover视图解析器

9、ViewReslover解析后返回具体View

10、DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。

11、DispatcherServlet响应用户

### 简述springmvc拦截器实现步骤

1. 自定义类实现implements HandlerInterceptor接口

|  |
| --- |
| package com.feicui.interceptor;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import org.springframework.web.servlet.HandlerInterceptor;  import org.springframework.web.servlet.ModelAndView;  public class MyInterceptor implements HandlerInterceptor{  /\*\*  \*  \* controller执行后且视图返回后调用此方法  \* 这里可得到执行controller时的异常信息  \* 这里可记录操作日志，资源清理等  \* 后  \*/    @Override  public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)  throws Exception {  System.out.println("afterCompletion1...");    }  /\*\*  \* controller执行后但未返回视图前调用此方法  \* 这里可在返回用户前对模型数据进行加工处理，比如这里加入公用信息以便页面显示  \* 中  \*/  @Override  public void postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3)  throws Exception {  System.out.println("postHandle...");    }  /\*\*  \* 执行目标之前(date.do)执行  \* http://localhost:8877/springmvc\_last01/test/date.do  \* 如果返回true就是放过 如果返回false就不执行目标方法  \*/  @Override  public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object arg2) throws Exception {  //request.getRequestURI() 请求地址包含了 login或者register 直接return true  if(request.getSession().getAttribute("user")==null){  //response重定向到登录  //return false  }  System.out.println("拦截器1....");  return true;  }  } |

1. 在springmvc容器中配置拦截器信息

|  |
| --- |
| <mvc:interceptors>  <!--多个拦截器,顺序执行 -->  <mvc:interceptor>  <mvc:mapping path=*"/\*\*"*/>  <bean class=*"com.feicui.interceptor.MyInterceptor"*></bean>  </mvc:interceptor>  <mvc:interceptor>  <mvc:mapping path=*"/\*\*"*/>  <bean class=*"com.feicui.interceptor.MyInterceptor2"*></bean>  </mvc:interceptor>  </mvc:interceptors> |

### 简述springmvc 返回json步骤

1. 加入json jar

2018-04-24_123155

|  |
| --- |
| <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.46</version>  </dependency> |

1. 在xml中配置json转换器

|  |
| --- |
| <mvc:annotation-driven>  <mvc:message-converters register-defaults=*"true"*>  <!-- 配置Fastjson支持 -->  <bean class=*"com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter"*>  <property name=*"supportedMediaTypes"*>  <list>  <value>text/html;charset=UTF-8</value>  <value>application/json</value>  </list>  </property>  <property name=*"features"*>  <list>  <value>WriteMapNullValue</value>  <value>QuoteFieldNames</value>  </list>  </property>  </bean>  </mvc:message-converters>  </mvc:annotation-driven> |

1. 后台接收前台的json数据用@RequestBody(Map接收)



4.后台传递到前台用@ResponseBody

|  |
| --- |
| @RequestMapping("/roleListAjax")  **public** @ResponseBody Map<String,Object> roleListAjax() |

### 简述springmvc 文件上传步骤

1. 导入jar

2018-04-24_112816

|  |
| --- |
| <!-- 文件上传组件 -->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>${commons-fileupload.version}</version>  </dependency> |

1. springmvc中配置文件解析器

|  |
| --- |
| <!-- 文件上传解析器 -->  <bean id=*"multipartResolver"*  class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <!-- 设置上传文件的最大尺寸为5MB -->  <property name=*"maxUploadSize"*>  <value>5242880</value>  </property>  <property name=*"defaultEncoding"*>  <value>utf-8</value>  </property>  </bean> |
|  |

1. 前后台代码编写与对应

如果name=file 与后台的multipartFile 的名字不对应

@requestParam(“写前台的name属性的值”)MultipartFile file



### 简述springmvc文件下载步骤

基于springmvc自带的ResponseEntity

|  |
| --- |
| @RequestMapping("/testHttpMessageDown")  public ResponseEntity<byte[]> download(HttpServletRequest request) throws IOException {  File file = new File("E://123.jpg");  byte[] body = null;  InputStream is = new FileInputStream(file);  body = new byte[is.available()]; is.read(body);  HttpHeaders headers= new HttpHeaders();  headers.add("Content-Disposition","attchement;filename="+file.getName());  HttpStatus statusCode = HttpStatus.OK;  ResponseEntity<byte[]> entity = new ResponseEntity<byte[]>(body, headers, statusCode);  return entity;  } |

2java通用实现

|  |
| --- |
| //设置响应头和客户端保存文件名  response.setCharacterEncoding("utf-8"); response.setContentType("multipart/form-data"); response.setHeader("Content-Disposition","attachment;fileName="+fileName);  //用于记录以完成的下载的数据量，单位是byte long downloadedLength = 0l;  try { //打开本地文件流  InputStream inputStream = new FileInputStream(filePath);  //激活下载操作 OutputStream os = response.getOutputStream();  //循环写入输出流  byte[] b = new byte[2048];  int length=0;  while ((length = inputStream.read(b)) > 0) { os.write(b, 0, length);  downloadedLength += b.length; }  // 这里主要关闭。  os.close();  inputStream.close();  } catch (Exception e){  throw e; } |

### 写出springmvc常用注解

太多了自己看博客

https://www.cnblogs.com/yjq520/p/6734422.html

### SpringMVC怎么样设定重定向和转发的

1. 可以通过形参的方式获得request与response
2. 可以通过返回值

|  |
| --- |
| **return** "forward:/system/roleList.jsp";  **return** "redirect:/system/roleList.jsp"; |

### Spring如何开启事务

见9题

### 为什么要使用为什么要用Spring

主要是三大思想：ioc、aop、di

Spring就是一个大工厂，可以将所有对象创建和依赖关系维护，交给Spring管理

2.AOP编程的支持

Spring提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能

声明式事务的支持

只需要通过配置就可以完成对事务的管理，而无需手动编程

3.方便集成各种优秀框架

Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如：Struts、Hibernate、MyBatis等）的直接支持

4.降低JavaEE API的使用难度

Spring 对JavaEE开发中非常难用的一些API（JDBC、JavaMail、远程调用等），都提供了封装，使这些API应用难度大大降低

### MyBatis怎么配置一对多、一对一？

一对一

|  |
| --- |
| <association property="menu" javaType="com.tieasy.model.MenuModel" >  <id property="id" column="id" />  <result property="menuName" column="menu\_name" />  <result property="url" column="url" />  </association> |

一对一

|  |
| --- |
| <association property="menu" column="menu\_id" select="glrolemenu\_getMenuById" /> |

一对多

|  |
| --- |
| <!-- 加入临时字段 -->  <result column=*"headImg"* property=*"headImg"* jdbcType=*"VARCHAR"* />  <!-- 加入了角色的临时字段 类型为list  property:属性名  ofType:指定类型  select:集合的数据在哪去找 指定了 namespace查找的roleMapper.xml中的  column:指定的通过用户的id的去查  -->  <collection property=*"roleList"* ofType=*"role"* select=*"com.feicui.mapper.RoleMapper.queryRoleByUserId"* column=*"id"*>  </collection> |

### Mybatis如何获得返回的主键

|  |
| --- |
| <!-- keyProperty:把查询到的主键放在那个属性中  order:定义在主sql语句之后还是之前执行SELECT LAST\_INSERT\_ID()  1、BEFORE:  2、AFTER:  -->  <selectKey keyProperty=*"uid"* order=*"AFTER"*  resultType=*"java.lang.Integer"*>  SELECT LAST\_INSERT\_ID()  </selectKey>  <!-- <selectKey keyProperty="uid" order="BEFORE"  resultType="java.lang.String">  SELECT UUID()  </selectKey> 了解--> |

### Mybatis调用存储过程的过程

|  |
| --- |
| <parameterMap type=*"Map"* id=*"insertNoticeMap"*>  <!-- property属性跟map中的key对应 jdbcType数据库类型 需要大写 mode是指定输出或者输入参数-->  <parameter property=*"noticeId"* jdbcType=*"VARCHAR"* javaType=*"string"* mode=*"IN"*/>  <parameter property=*"user\_id"* jdbcType=*"VARCHAR"* javaType=*"string"* mode=*"IN"*/>  <parameter property=*"title"* jdbcType=*"VARCHAR"* javaType=*"string"* mode=*"IN"*/>  <parameter property=*"content"* jdbcType=*"VARCHAR"* javaType=*"string"* mode=*"IN"*/>  <parameter property=*"is\_accessory"* jdbcType=*"INTEGER"* javaType=*"integer"* mode=*"IN"*/>  <parameter property=*"acc\_id"* jdbcType=*"VARCHAR"* javaType=*"string"* mode=*"IN"*/>  <!-- <parameter property="t\_id" jdbcType="VARCHAR" javaType="string" mode="IN"/> -->  <parameter property=*"acc\_url"* jdbcType=*"VARCHAR"* javaType=*"string"* mode=*"IN"*/>  <parameter property=*"msg"* jdbcType=*"VARCHAR"* javaType=*"string"* mode=*"OUT"*/>  </parameterMap>  <!-- 指定执行类型 为callable -->  <select id=*"callAddNotice"* parameterMap=*"insertNoticeMap"* statementType=*"CALLABLE"*>  {call oa.addNotice(  ?,?,?,?,?,?,?,?  ) }  </select> |

设置参数：

|  |
| --- |
| //调用存储过程 存储附件及通知公告表  Map<String,Object> map = **new** HashMap<>();  map.put("msg", "success");  //生成通知公告表的id  String noticeId = UUIDUtils.*getUUID*();  map.put("noticeId", noticeId);  map.put("user\_id", uid);  map.put("content", content);  map.put("title", title);  map.put("is\_accessory", 0);  **if**(fileName!=**null**){ //说明有附件  map.put("is\_accessory",1);  //put附件表  map.put("acc\_id", UUIDUtils.*getUUID*());  map.put("t\_id", noticeId);  map.put("acc\_url", fileName);  } |

简单的存储过程：

|  |
| --- |
| DELIMITER $$  USE `oa`$$  DROP PROCEDURE IF EXISTS `addNotice`$$  CREATE DEFINER=`root`@`localhost` PROCEDURE `addNotice`(IN v\_id VARCHAR(50),IN v\_user\_id VARCHAR(50),IN v\_title VARCHAR(50)  ,IN v\_content VARCHAR(2000),IN v\_is\_accessory INT,IN v\_acc\_id VARCHAR(50),IN v\_acc\_url VARCHAR(50)  ,OUT msg VARCHAR(10)  )  BEGIN  DECLARE error\_code INT;#定义错误编号  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET error\_code=1;#定出现SQLEXCEPTION异常 程序继续  START TRANSACTION;#开启事务  INSERT INTO notice(id,user\_id,title,content,is\_accessory)  VALUES (v\_id,v\_user\_id,v\_title,v\_content,v\_is\_accessory);  IF ROW\_COUNT()>0 THEN #row\_count 刚才执行的代码影响了多少条  INSERT INTO accessory(id,t\_id,acc\_url,upload\_time)  VALUES(v\_acc\_id,v\_id,v\_acc\_url,SYSDATE());  IF ROW\_COUNT()>0 THEN #说明两条插入全部成功  COMMIT;  ELSE #表示第二条没有成功  SET msg:='插入附件表失败,公告发布失败';  ROLLBACK;  END IF;  ELSE  SET msg:='插入公告表失败,公告发布失败';  ROLLBACK;#第一条没有插入成功直接回滚  END IF;  END$$  DELIMITER ; |

### 简述orm的含义，简述mybatis与hibernate的区别

Object relationShip mapping对象关系映射

1.mybatis是一个不完全的orm框架，因为mybatis需要程序员自己写大量的sql，需要程序员对sql的掌握比较高，不过mybatis可以通过xml文件可以灵活的配置要运行的sql语句，将sql与Java对象做了很好的关系映射。

2、Mybatis的学习门槛较低，可严格控制sql的执行性能，灵活度高，适合于对关系数据模型要求不高的软件开发，比如互联网软件、企业运营类软件等。因为这类软件的需求变化快而且多。灵活的前提是无法做到数据库的无关系，如果要实现支持多种数据库的软件则需要自定义多套sql映射文件，工作量大。

3、Hibernate是一套完全的orm框架，数据库无关性好，适用于关系数据模型较高的软件，可以节省很多代码，提高工作效率。而且hibernate的性能调优需要很强的经验和能力

### spring框架中的事务处理有几种。请分别阐述两者的区别

实现方式共有两种：编码方式；[声明式事务](https://www.baidu.com/s?wd=%E5%A3%B0%E6%98%8E%E5%BC%8F%E4%BA%8B%E5%8A%A1&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)管理方式。

基于AOP技术实现的[声明式事务](https://www.baidu.com/s?wd=%E5%A3%B0%E6%98%8E%E5%BC%8F%E4%BA%8B%E5%8A%A1&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)管理，实质就是：在方法执行前后进行拦截，然后在目标方法开始之前创建并加入事务，执行完目标方法后根据执行情况提交或回滚事务。

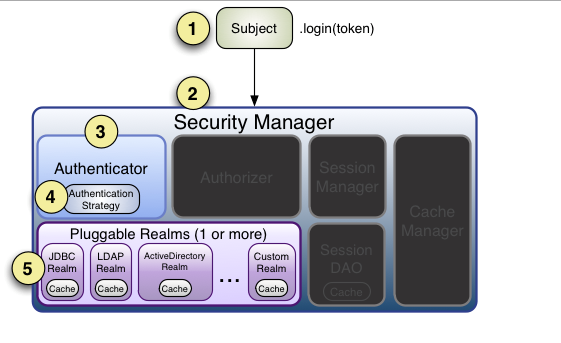
[声明式事务](https://www.baidu.com/s?wd=%E5%A3%B0%E6%98%8E%E5%BC%8F%E4%BA%8B%E5%8A%A1&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)管理又有两种方式：基于XML配置文件的方式；另一个是在业务方法上进行@Transactional注解，将事务规则应用到[业务逻辑](https://www.baidu.com/s?wd=%E4%B8%9A%E5%8A%A1%E9%80%BB%E8%BE%91&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)中。

### Mybatis如何处理转义字符

<![CDATA[where uid<#{uid}]]>

### 简述 Shiro 的核心组件

### Shiro认证过程



1. subject.login(token);
2. Realm中

|  |
| --- |
| /\*\*  \*认证 、登录  \*AuthenticationToken token controller层放入的用户名密码  \*执行时间:控制层调用login方法  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  UsernamePasswordToken upToken = (UsernamePasswordToken) token;  //获得用户名  String username = upToken.getUsername();  UserInfo userInfo = userInfoService.login(username);  //去比较密码 principal第一个参数是身份 就是用户信息（其他地方会用）  //credentials 第二个参数 数据库查询出来的密码  //realmName第三个参数 放入一个名字 待会可以根据这个名字来取  **if**(userInfo!=**null**)//用户名存在  **return** **new** SimpleAuthenticationInfo(userInfo, userInfo.getPassword(), **this**.getName());  **return** **null**;  } |

1. 自定义密码比较器

|  |
| --- |
| **public** **class** CustomCredentialsMatcher **extends** SimpleCredentialsMatcher{  /\*\*  \* AuthenticationToken 前台输入 usernamepasswordtoken传递过来用户名密码  \*  \* AuthenticationInfo realm中传递过来的信息  \* return new SimpleAuthenticationInfo(userInfo, userInfo.getPassword(), this.getName());  \*/  //SimpleAuthenticationInfo的三个参数  //去比较密码 principal第一个参数是身份 就是用户信息（其他地方会用）  //credentials 第二个参数 数据库查询出来的密码  //realmName第三个参数 放入一个名字 待会可以根据这个名字来取  @Override  **public** **boolean** doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) {  //页面密码 还未加密 首先对页面密码进行加密  UsernamePasswordToken upToken = (UsernamePasswordToken) token;  **char**[] password = upToken.getPassword();  //char数组转换string  String md5 = EncryptUtils.*md5*(**new** String(password), upToken.getUsername());  //数据库中的密码 在realm中返回了new SimpleAuthenticationInfo 就把密码带过来了  Object credentials = info.getCredentials();  **return** equals(md5, credentials);  }  } |

### Shiro授权过程

Realm中授权方法

|  |
| --- |
| **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  //这个类就是用来存储权限与角色  SimpleAuthorizationInfo simpleAuthorizationInfo = **new** SimpleAuthorizationInfo();  //1。查询到角色  UserInfo userInfo = (UserInfo) principals.fromRealm(**this**.getName()).iterator().next();//在认证中指定的名字 返回的是一个身份集合  List<Role> roleList = userInfo.getRoleList();//取到当前用户所有的角色  **for** (Role role : roleList) {  //把角色保存在shiro中 告诉shiro当前用户有哪些角色  simpleAuthorizationInfo.addRole(role.getRoleName());//获取角色名字 放入shiro  //放权限  List<Menu> menuList = role.getMenuList();  **for** (Menu menu : menuList) {//这儿不去重 不增加学习难度  **if**(menu.getMenuUrl()!=**null**){  simpleAuthorizationInfo.addStringPermission(menu.getMenuUrl());  }//添加权限  System.***out***.println("进入");  }  }    **return** simpleAuthorizationInfo;  } |

然后通过注解、标签、配置等实现权限拦截

例如@RequiresPermissions <shiro:hasRole name=*"管理员"*>（需要引入标签库） 等等

### 解释quartz的三大核心组件及其作用

1.job：(生命周期)

Jobdetail:绑定job，携带当前job的信息

重要属性: 用来判断现在该执行哪个job,能记录日志等

Name:必须的 job的名称

Group:必须的 job的分组 默认值为DEFAULT

Jobclass:必须的 指定绑定的job

Jobdatamap:可以装载任何可序列化的数据对象，可以把这些参数传递给job

是map的子类

2.trigger：线程池运行（calendar）

1>SimpleTrigger:

2>CronTrigger:

3.scheduler

### Quartz与spring集成配置

|  |
| --- |
| <!-- 定时器 -->  <bean id=*"jobDetail"*  class=*"org.springframework.scheduling.quartz.JobDetailFactoryBean"*>  <!-- 指定自定义的job -->  <property name=*"jobClass"* value=*"com.feicui.quartz.UserInfoQuartz"*></property>  <property name=*"durability"* value=*"true"* />  </bean>  <bean id=*"cronTriggerBean"*  class=*"org.springframework.scheduling.quartz.CronTriggerFactoryBean"*>  <property name=*"jobDetail"* ref=*"jobDetail"*></property>  <!-- 每天0点执行 -->  <!-- <property name="cronExpression" value="0 0 0 \* \* ?"></property> -->  <!-- 每天10点开始执行一次 -->  <!-- <property name="cronExpression" value="0 0 10 \* \* ?"></property> -->  <!-- 这里为了测试是否成功改成两分钟一次 -->  <!-- cron表达式 指定时间-->  <property name=*"cronExpression"* value=*"30 \* \* \* \* ? 2019"*></property>  </bean>  <!-- 启动定时器 -->  <bean id=*"trigger"*  class=*"org.springframework.scheduling.quartz.SchedulerFactoryBean"*>  <property name=*"triggers"*>  <list>  <ref bean=*"cronTriggerBean"* />  </list>  </property>  </bean> |

### 详细quartz的cron表达式

cron表达式:

格式:

[秒][分][小时][日][月][周][年](1/20 \* \* \* \* 4#3 \*)

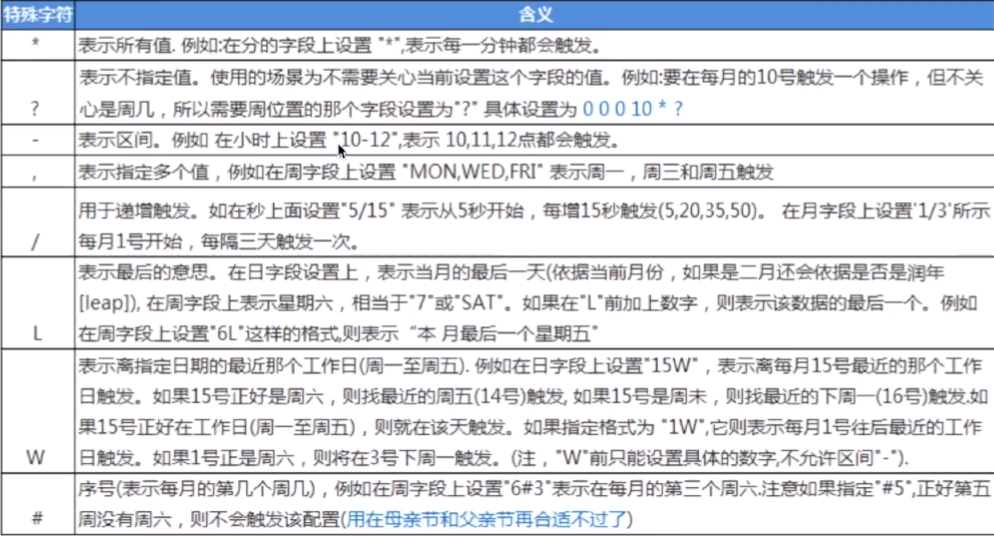


,是或 0/5表示0开始 每个5触发 6#3表示第三周的星期5

6L 表示最后一周的星期5

示例





### Poi 操作2003与2007的区别分别用到了什么类、

2003 xls结尾XSSFWorkbook 数据量65535

2007 xlsx结尾HSSFWorkbook

### 简述poi导出的流程

例如：

|  |
| --- |
| /\*\*  \* 导出xls为后缀名的方法  \* **@param** title 标题  \* **@param** headers 头信息  \* **@param** dataList 数据  \* **@throws** IOException  \*/  **private** **static** **void** export(Workbook wb,String title,String[] headers,List<List<String>> dataList,OutputStream os) {  //创建工作簿new HSSFWorkbook() Excel2003以前（包括2003）的版本,扩展名是.xls 的  //是操作Excel2007的版本，扩展名是.xlsx的  //创建sheet  Sheet sheet = wb.createSheet();  //创建行 标题 合并行列  CellRangeAddress cellRangeAddress = **new** CellRangeAddress(0, 1, 0, headers.length-1);  sheet.addMergedRegion(cellRangeAddress);  //获得第标题行  Row titleRow = sheet.createRow(0);    Cell titleCell = titleRow.createCell(0);  //设置标题  titleCell.setCellValue(title);  //设置样式  CellStyle titleStyle = wb.createCellStyle();  //设置字体  Font titleFont = wb.createFont();  titleFont.setBold(**true**);  titleFont.setFontHeightInPoints((**short**)20);  titleStyle.setFont(titleFont);  titleStyle.setAlignment(HorizontalAlignment.***CENTER\_SELECTION***);  titleCell.setCellStyle(titleStyle);  //建立header的行  Row headerRow=sheet.createRow(2);//前两行已经被title占了  Cell headerCell=**null**;  //设置header样式  CellStyle headerStyle = wb.createCellStyle();  Font headerFont = wb.createFont();  headerFont.setBold(**true**);  headerFont.setFontHeightInPoints((**short**)12);  headerStyle.setFont(headerFont);  headerStyle.setAlignment(HorizontalAlignment.***CENTER\_SELECTION***);  //设置列名  **for**(**int** i = 0;i<headers.length;i++){  headerCell = headerRow.createCell(i);//创建列  //给列设置值  headerCell.setCellValue(headers[i]);  headerCell.setCellStyle(headerStyle);  //设置列宽  sheet.setColumnWidth(i, 5000);  }  **for**(**int** i = 0;i<dataList.size();i++){  Row dataRow = sheet.createRow(i+3);  **for**(**int** j = 0;j<dataList.get(i).size();j++){  Cell dateCell = dataRow.createCell(j);  dateCell.setCellValue(dataList.get(i).get(j));  }  }  **try** {  wb.write(os);  } **catch** (IOException e) {  System.***out***.println("导出失败");  e.printStackTrace();  }**finally**{  **try** {  os.close();  wb.close();  } **catch** (IOException e) {  System.***out***.println("流为null");  }  }  }  /\*\*  \* Excel2003以前（包括2003）的版本,扩展名是.xls 的  \* **@param** title  \* **@param** headers  \* **@param** dataList  \* **@param** os  \* **@throws** IOException  \*/  **public** **static** **void** exportXls(String title,String[] headers,List<List<String>> dataList,OutputStream os) **throws** IOException{  *export*(**new** HSSFWorkbook(),title,headers,dataList,os);  }  /\*\*  \* 是操作Excel2007的版本，扩展名是.xlsx的  \* **@param** title  \* **@param** headers  \* **@param** dataList  \* **@param** os  \* **@throws** IOException  \*/  **public** **static** **void** exportXlsx(String title,String[] headers,List<List<String>> dataList,OutputStream os) **throws** IOException{  *export*(**new** XSSFWorkbook(),title,headers,dataList,os);  } |
| **public** **void** export(HttpServletResponse response) **throws** UnsupportedEncodingException{  List<List<String>> dataList = **new** ArrayList<>();  List<UserInfo> userList = userInfoService.queryAll();  response.reset();  response.setContentType("application/vnd.ms-excel;charset=UTF-8");  List<String> list=**null**;  **for**(UserInfo userInfo:userList){  list = **new** ArrayList<>();  list.add(userInfo.getRealName());  list.add(userInfo.getMobile());  list.add(userInfo.getEmail());  dataList.add(list);  }  String[] headers = {"姓名","手机","邮箱"};  response.setHeader("Content-disposition", "attachment; filename="+**new** String("用户通讯录.xls".getBytes(),"iso-8859-1"));  **try** {  ExcelExportUtil.*exportXls*("通讯录", headers, dataList, response.getOutputStream());  //return "success";  } **catch** (IOException e) {  e.printStackTrace();  //return "error";  }  } |

### 简述poi导入的流程

|  |
| --- |
| /\*\*  \* 只做了读取演示  \* **@param** file  \* **@throws** IOException  \*/  @RequestMapping("import")  **public** **void** importExcel(MultipartFile file) **throws** IOException{  List<UserInfo> list = **new** ArrayList<>();  //拿到file的名字 可以获得后缀名 根据.xls或者是.xlsx判断 对应的xls对应HSSF。。。如果是xlsx对应XSSF。。。  String originalFilename = file.getOriginalFilename();  InputStream inputStream = file.getInputStream();  HSSFWorkbook wb = **new** HSSFWorkbook(inputStream);  HSSFSheet sheetAt = wb.getSheetAt(0);  **int** countRowNum = sheetAt.getLastRowNum();//获得最后行数的下标  System.***out***.println(countRowNum);  HSSFRow row =**null**;  UserInfo userInfo =**null**;  **for**(**int** i = 3;i<=countRowNum;i++){  userInfo = **new** UserInfo();  row = sheetAt.getRow(i);  //row.getLastCellNum()获得最后一列的下标  /\*for(int j = 0;j<=row.getLastCellNum();j++){  userInfo.setRealName(row.getCell(j).getStringCellValue());;  }\*/  userInfo.setRealName(row.getCell(0).getStringCellValue());  userInfo.setMobile(row.getCell(1).getStringCellValue());  userInfo.setEmail(row.getCell(2).getStringCellValue());  list.add(userInfo);  }  System.***out***.println(list);  } |

### springbean的生命周期

1.容器寻找Bean的定义信息并且将其实例化。

2.受用依赖注入，Spring按照Bean定义信息配置Bean的所有属性。

3.如果Bean实现了BeanNameAware接口，工厂调用Bean的setBeanName()方法传递Bean的ID。

4.如果Bean实现了BeanFactoryAware接口，工厂调用setBeanFactory()方法传入工厂自身。

5.如果BeanPostProcessor和Bean关联，那么它们的postProcessBeforeInitialzation()方法将被调用。

6.如果Bean指定了init-method方法，它将被调用。

7.最后，如果有BeanPsotProcessor和Bean关联，那么它们的postProcessAfterInitialization()方法将被调用。   
到这个时候，Bean已经可以被应用系统使用了，并且将被保留在Bean Factory中知道它不再需要。

有两种方法可以把它从Bean Factory中删除掉。

1.如果Bean实现了DisposableBean接口，destory()方法被调用。

2.如果指定了订制的销毁方法，就调用这个方法。

Bean在Spring应用上下文的生命周期与在Bean工厂中的生命周期只有一点不同，   
唯一不同的是，如果Bean实现了ApplicationContextAwre接口，setApplicationContext()方法被调用。

只有singleton行为的bean接受容器管理生命周期。   
non-singleton行为的bean，Spring容器仅仅是new的替代，容器只负责创建。

### spring与mybatis整合

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 数据库连接池 -->  <!-- 加载配置文件 -->  <context:property-placeholder location=*"classpath:properties/\*.properties"* ignore-unresolvable=*"true"*/>  <!-- 数据库连接池 -->  <bean id=*"dataSource"* class=*"com.alibaba.druid.pool.DruidDataSource"*  destroy-method=*"close"*>  <property name=*"url"* value=*"${jdbc.url}"* />  <property name=*"username"* value=*"${jdbc.username}"* />  <property name=*"password"* value=*"${jdbc.password}"* />  <property name=*"driverClassName"* value=*"${jdbc.driver}"* />  <property name=*"maxActive"* value=*"10"* />  <property name=*"minIdle"* value=*"5"* />  </bean>  <!-- 配置sqlsessionFactory -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"configLocation"* value=*"classpath:mybatis/sqlMapConfig.xml"*></property>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- 配置扫描包，加载mapper代理对象 -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.feicui.mapper"*></property>  </bean>  </beans> |

### Spring与javamail整合

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <description>mail</description>  <context:property-placeholder location=*"classpath:properties/mail.properties"*  ignore-unresolvable=*"true"* />  <!-- 邮箱信息 包括发件，收件人，内容.... -->  <bean id=*"mailMessage"* class=*"org.springframework.mail.SimpleMailMessage"*>  <property name=*"from"*>  <value>${mailFrom}</value> <!-- 发件人 -->  </property>  <!-- 查看SimpleMailMessage源码还可以注入标题,内容等 -->  </bean>  <!-- 申明JavaMailSenderImpl对象 配置发送邮箱的信息，要发送邮件需要登录 所以需要配置用户名、密码、用什么服务器发送等信息 -->  <!-- <bean id="dddUserMailUtils" class="com.feicui.utils.AddUserMailUtils">  <property name="mailSender" ref="mailSender"></property> </bean> -->  <bean id=*"mailSender"* class=*"org.springframework.mail.javamail.JavaMailSenderImpl"*>  <property name=*"defaultEncoding"* value=*"UTF-8"* />  <property name=*"host"* value=*"${mail.host}"* /> <!-- smtp.qq.com smtp.163.com -->  <property name=*"username"* value=*"${mail.username}"* />  <property name=*"password"* value=*"${mail.password}"* />  <property name=*"javaMailProperties"*>  <props>  <!-- 设置认证开关 -->  <prop key=*"mail.smtp.auth"*>true</prop>  <!-- 启动调试开关 -->  <prop key=*"mail.debug"*>true</prop>  <!-- 设置发送延时 -->  <prop key=*"mail.smtp.timeout"*>0</prop>  </props>  </property>  </bean>  </beans> |

### Spring与shiro整合

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop.xsd*  *http://www.springframework.org/schema/tx*  *http://www.springframework.org/schema/tx/spring-tx.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>    <description>Shiro的配置</description>    <!-- SecurityManager配置 -->  <!-- 配置Realm域 -->  <!-- 密码比较器 -->  <!-- 代理如何生成？ 用工厂来生成Shiro的相关过滤器-->  <!-- 配置缓存：ehcache缓存 -->  <!-- 安全管理 -->  <bean id=*"securityManager"* class=*"org.apache.shiro.web.mgt.DefaultWebSecurityManager"*>  <!-- Single realm app. If you have multiple realms, use the 'realms' property instead. -->  <property name=*"realm"* ref=*"authRealm"*/><!-- 引用自定义的realm -->  <!-- 缓存 -->  <!-- <property name="cacheManager" ref="shiroEhcacheManager"/> -->  </bean>  <!-- 自定义权限认证 -->  <bean id=*"authRealm"* class=*"com.feicui.shiro.MyRealm"*>  <!-- <property name="userService" ref="userInfoService"/> -->  <!-- 自定义密码加密算法 -->  <property name=*"credentialsMatcher"* ref=*"passwordMatcher"*/>  </bean>    <!-- 设置密码加密策略 md5hash -->  <bean id=*"passwordMatcher"* class=*"com.feicui.shiro.CustomCredentialsMatcher"*/>  <!-- filter-name这个名字的值来自于web.xml中filter的名字 -->  <bean id=*"shiroFilter"* class=*"org.apache.shiro.spring.web.ShiroFilterFactoryBean"*>  <!-- 安全管理器 -->  <property name=*"securityManager"* ref=*"securityManager"*/>  <!--登录页面 如果没有登录 自动跳转-->  <property name=*"loginUrl"* value=*"/login.jsp"*></property>  <!-- 登录成功后 -->  <property name=*"successUrl"* value=*"/system/index/index.jsp"*></property>  <property name=*"unauthorizedUrl"* value=*"/auth.jsp"*></property>  <property name=*"filterChainDefinitions"*>  <!-- /\*\*代表下面的多级目录也过滤 -->  <value>  <!-- /login.jsp = anon  /logout\* = anon  /login/\*\* = anon  /bootstrap/\*\* = anon  /css/\*\* = anon  /img/\*\* = anon  /js/\*\* = anon  /My97DatePicker/\*\* = anon  /\*\* = authc  /\*.\* = authc -->  <!-- /userInfo/\*\*= roles["管理员,总经理"]  /userInfo/addUser.do = perms["用户管理-增加"] 配置shiro访问/userInfo/addUser.do 需要这个权限"用户管理-增加"  -->  </value>  </property>  </bean>  <!-- 用户授权/认证信息Cache, 采用EhCache 缓存 -->  <!-- <bean id="shiroEhcacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">  <property name="cacheManagerConfigFile" value="classpath:ehcache-shiro.xml"/>  </bean> -->  <!-- 保证实现了Shiro内部lifecycle函数的bean执行 -->  <bean id=*"lifecycleBeanPostProcessor"* class=*"org.apache.shiro.spring.LifecycleBeanPostProcessor"*/>  </beans> |

### 42springmvc配置

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>  <!-- 开启shiro注解 可以在springmvc中配置下面两个bean -->  <bean  class=*"org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"*  depends-on=*"lifecycleBeanPostProcessor"* />  <bean  class=*"org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor"*>  <property name=*"securityManager"* ref=*"securityManager"* />  </bean>  <!-- 第一步: 创建自定义日期转换规则 -->  <bean id=*"dateConvert"* class=*"com.feicui.utils.DateConvert"* />  <!-- 第二步： 创建convertion-Service ，并注入dateConvert -->  <bean id=*"conversion"*  class=*"org.springframework.format.support.FormattingConversionServiceFactoryBean"*>  <property name=*"converters"*>  <set>  <ref bean=*"dateConvert"* />  </set>  </property>  </bean>  <context:component-scan base-package=*"com.feicui.controller"* />  <mvc:annotation-driven conversion-service=*"conversion"*>  <!--设置不使用默认的消息转换器 -->  <mvc:message-converters register-defaults=*"false"*>  <!--配置spring的转换器 -->  <!-- <bean class="org.springframework.http.converter.StringHttpMessageConverter"  /> <bean class="org.springframework.http.converter.support.AllEncompassingFormHttpMessageConverter"  /> <bean class="org.springframework.http.converter.ByteArrayHttpMessageConverter"  /> <bean class="org.springframework.http.converter.BufferedImageHttpMessageConverter"  /> -->  <!--配置fastjson中实现HttpMessageConverter接口的转换器 -->  <bean id=*"fastJsonHttpMessageConverter"*  class=*"com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter4"*>  <!--加入支持的媒体类型，返回contentType -->  <property name=*"supportedMediaTypes"*>  <list>  <!--这里顺序不能反，一定要先写text/html，不然IE下会出现下载提示 -->  <value>text/html;charset=UTF-8</value>  <value>application/json;charset=UTF-8</value>  </list>  </property>  </bean>  </mvc:message-converters>  </mvc:annotation-driven>  <!-- 资源映射 -->  <mvc:resources location=*"/css/"* mapping=*"/css/\*\*"* />  <mvc:resources location=*"/js/"* mapping=*"/js/\*\*"* />  <mvc:resources location=*"/img/"* mapping=*"/img/\*\*"* />  <mvc:resources location=*"/bootstrap/"* mapping=*"/bootstrap/\*\*"* />  <!-- 前缀 后缀 -->  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <!-- <property name="prefix" value="/WEB-INF/pages/" /> -->  <!-- <property name="suffix" value=".jsp" /> -->  </bean>  <!-- 配置文件上传 -->  <bean id=*"multipartResolver"*  class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <property name=*"defaultEncoding"* value=*"utf-8"*></property>  <property name=*"maxUploadSize"* value=*"10485760000"*></property>  <property name=*"maxInMemorySize"* value=*"40960"*></property>  </bean>  <!--拦截器 -->  <!-- <mvc:interceptors> 多个拦截器,顺序执行 <mvc:interceptor> <mvc:mapping path="/\*\*"/>登录拦截  <bean class="com.feicui.interceptor.LoginInterceptor"></bean> </mvc:interceptor>  <mvc:interceptor>拦截权限 <mvc:mapping path="/\*\*"/> <bean class="com.feicui.interceptor.MenuInterceptor"></bean>  </mvc:interceptor> </mvc:interceptors> -->  </beans> |

### 43webxml配置

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*  id=*"WebApp\_ID"* version=*"3.0"*>  <display-name>feicui-oa</display-name>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  <!-- Shiro Security filter filter-name这个名字的值将来还会在spring中用到 -->  <filter>  <filter-name>shiroFilter</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  <init-param>  <param-name>targetFilterLifecycle</param-name>  <param-value>true</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>shiroFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- 配置监听spring -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 指定spring bean 的配置文件所在目录 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext-\*.xml</param-value>  </context-param>  <!-- 配置spring字符编码为utf-8 post乱码 -->  <filter>  <filter-name>encodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  <init-param>  <param-name>forceEncoding</param-name>  <param-value>true</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>encodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- springMVC的配置 -->  <servlet>  <servlet-name>spring</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>spring</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping>  <!-- log4j配置 -->  <context-param>  <param-name>log4jConfigLocation</param-name>  <param-value>classpath:properties/log4j.properties</param-value>  </context-param>  <context-param>  <param-name>webAppRootKey</param-name>  <param-value>SLSaleSystem.root</param-value>  </context-param>  <!-- spring加载log4j的监听 -->  <listener>  <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>  </listener>  </web-app> |

## 四、数据库部分

### 1.Sql语句部分

#### 新建用户

CREATE USER 'username'@'host' IDENTIFIED BY 'password';

说明：

username：你将创建的用户名

host：指定该用户在哪个主机上可以登陆，如果是本地用户可用localhost，如果想让该用户可以从任意远程主机登陆，可以使用通配符%

password：该用户的登陆密码，密码可以为空，如果为空则该用户可以不需要密码登陆服务器

#### 授权

GRANT privileges ON databasename.tablename TO 'username'@'host'

说明:

privileges：用户的操作权限，如SELECT，INSERT，UPDATE等，如果要授予所的权限则使用ALL

databasename：数据库名

tablename：表名，如果要授予该用户对所有数据库和表的相应操作权限则可用\*表示，如\*.\*

Flush privileges ;

#### 设置更改密码

SET PASSWORD FOR 'username'@'host' = PASSWORD('newpassword');

如果是当前登陆用户用:

SET PASSWORD = PASSWORD("newpassword");

#### 删除用户

DROP USER 'username'@'host';

#### 增加/删除/修改列

* 查看列：desc 表名;
* 修改表名：alter table t\_book rename to bbb;
* 添加列：alter table 表名 add column 列名 varchar(30) default null;
* 删除列：alter table 表名 drop column 列名;
* 修改列名： alter table bbb change nnnnn hh int;
* 修改列属性：alter table t\_book modify name varchar(22);

#### 基本sql

查看表的各种数据类型

describe `student`

查看表的已存数据

select \* from 表名 (select id,name,number,numberid from 表名)

添加数据

insert into 表名 (ID,name,number,numberid) values(1,'Logic',220,1),(2,'HTML',160,1),(3,'Java OOP',230,1);

删除数据

delete from 表名 where 条件;

修改数据

update 表名 set numberid=2 where 条件

降序排列

select id,name,number,numberid from 表名 order by id desc;

#### 复制表

1.只复制表结构到新表

CREATE TABLE 新表 SELECT \* FROM 旧表 WHERE 1=2;

或

CREATE TABLE 新表 LIKE 旧表 ;

注意上面两种方式，前一种方式是不会复制时的主键类型和自增方式是不会复制过去的，而后一种方式是把旧表的所有字段类型都复制到新表。

2.复制表结构及数据到新表

CREATE TABLE 新表 SELECT \* FROM 旧表

3.复制旧表的数据到新表(假设两个表结构一样)

INSERT INTO 新表 SELECT \* FROM 旧表

4.复制旧表的数据到新表(假设两个表结构不一样)

INSERT INTO 新表(字段1,字段2,.......) SELECT 字段1,字段2,...... FROM 旧表

#### 删除表/表数据 delete/trucate/drop区别

Delete语句（可以回滚不影响表结构）

delete from 表名 where 条件

DELETE语句执行删除的过程是每次从表中删除一行，并且同时将该行的删除操作作为事务记录在日志中保存以便进行进行回滚操作。

Trucate语法（不能带where条件，不删除表结构，不能回滚）

TRUNCATE TABLE  table\_name

Drop语法：(drop语句将表所占用的空间全释放掉。)

drop table table\_name

区别：

1、在速度上，一般来说，drop> truncate > delete。

2、在使用drop和truncate时一定要注意，虽然可以恢复，但为了减少麻烦，还是要慎重。

3、如果想删除部分数据用delete，注意带上where子句，回滚段要足够大；

4、TRUNCATE 和DELETE只删除数据， DROP则删除整个表（结构和数据）。

#### 9.时间函数

获得当前日期+时间（date + time）函数：now()

获得当前日期+时间（date + time）函数：sysdate()

Curdate()只是获得日期 不包括时间

暂时只记住TIMESTAMPDIFF

先记简单的

<https://blog.csdn.net/fanxl10/article/details/44172569>

不够用了在看

<https://www.cnblogs.com/ggjucheng/p/3352280.html>

#### 10.limit用法

1.limit m 表示显示1-m（包括1，m） 例如：

select \* from Customer LIMIT 10;--检索前10行数据，显示1-10条数据

1. limit m,n 表示从m+1开始 显示n条  
   select \* from Customer LIMIT 1,10;--检索从第2行开始，累加10条id记录，共显示id为2....11

#### null值(简单列举几个)

1. 查询带null的条件需要用 列名 is null/is not null 不能用 列名=null、!=null
2. 除了count(\*) 不忽略null值外 其他聚合函数会忽略null值

例如：

三条数据 10 null 20

使用select count(\*) from table 结果3

使用select count(列名) from table 结果为2

使用select avg(列名) 结果为15

所以在对有null值进行聚合函数时 需要通过ifnull(列名,如果为null则显示)进行处理

1. 用order by ... asc时，null值会被放在最前面，而用order by ... desc时，null时会被放在最后面，相当于null是一个无穷小的值。
2. 看例子

select \* from t1;

+------+-------+

| id | name |

+------+-------+

| 1 | chen |

| 2 | zhang |

| 3 | NULL |

+------+-------+

3 rows in set (0.00 sec)

当查询名字不为chen的人时

select \* from t1 where name!='chen';

+------+-------+

| id | name |

+------+-------+

| 2 | zhang |

+------+-------+

1 row in set (0.00 sec)

但是结果好像不太如我们所愿，id为1的那条记录没有被查出来，null肯定和'chen'不相等呀，为什么会查不出呢？

应该使用

select \* from t1 where name!='chen' or name is null;

#### order by 用法

Order by后面可以跟列名 可以跟别名 可以跟序号

Select sex,name from student order by name;

Select sex s,name from student order by s;

Select sex,name from student order by 1;(1代表sex)

mysql中，我们可以使用 ASC 或 DESC 关键字来设置查询结果是按升序或降序排列。 默认情况下，它是按升序排列。

order by 后可加2个字段，用英文逗号隔开，

如A用升序， B降序，SQL该这样写，order by A ASC,  B DESC;也可以这样写：order by A, B DESC;  
如果都用降序，必须用两个desc ，order by A字段 DESC, B字段 DESC;

#### distinct：

1. 在count计算不重复的记录的时候能用到  
   比如SELECT COUNT( DISTINCT player\_id ) FROM task;  
   就是计算talbebname表中id不同的记录有多少条

b, 在需要返回记录不同的id的具体值的时候可以用  
比如SELECT DISTINCT player\_id FROM task;  
返回talbebname表中不同的id的具体的值

c. 上面的情况2对于需要返回mysql表中2列以上的结果时会有歧义  
比如SELECT DISTINCT player\_id, task\_id FROM task;  
实际上返回的是player\_id与task\_id同时不相同的结果,也就是DISTINCT同时作用了两个字段，必须得player\_id与task\_id都相同的才被排除了,与我们期望的结果不一样,我们期望的是player\_id不同被过滤

  在这种情况下，distinct同时作用了两个字段，player\_id,task\_id

这时候可以考虑使用group\_concat函数来进行排除,不过这个mysql函数是在mysql4.1以上才支持的

group\_concat用法看 https://blog.csdn.net/mary19920410/article/details/76545053

或者使用group by

select player\_id,task\_id from task group by player\_id

注意：order by 与distinct 同时使用

如果order by的字段不在select中，执行sql语句时首先执行distinct，之后产生的虚拟临时表中没有order by的字段，所以再执行order by时会报错。

例如：evt\_id不在\_evt\_tim中

SELECT DISTINCT evt\_tim FROM evt\_etl ORDER BY evt\_id ASC; 报错

原因：

首先，在mysql中distinct 的执行顺序高于order by。

第二，distinct执行时会对查询的记录进行去重，产生一张虚拟的临时表；

第三，order by执行时对查询的虚拟临时表进行排序，产生新的虚拟临时表。

#### group by 太多了 看博客

<http://www.cnblogs.com/rainman/archive/2013/05/01/3053703.html#m7>

#### join用法 同样看博客

<https://www.cnblogs.com/fudashi/p/7491039.html>

#### mysql交并差 （先简单了解，以后在去看复杂的)

<https://blog.csdn.net/qq_39023116/article/details/79008085>

#### mysql常用聚合函数

<https://www.cnblogs.com/geaozhang/p/6745147.html>

#### case when then end 用法

|  |
| --- |
| --简单Case函数  CASE sex           WHEN '1' THEN '男'           WHEN '2' THEN '女'  ELSE '其他' END  --Case搜索函数  CASE WHEN sex = '1' THEN '男'           WHEN sex = '2' THEN '女'  ELSE '其他' END   1. 已知数据按照另外一种方式进行分组，分析。   有如下数据:(为了看得更清楚，我并没有使用国家代码，而是直接用国家名作为Primary Key)  国家（country） 人口（population）  中国 600  美国 100  加拿大 100  英国 200  法国 300  日本 250  德国 200  墨西哥 50  印度 250   根据这个国家人口数据，统计亚洲和北美洲的人口数量。应该得到下面这个结果。  洲 人口  亚洲 1100  北美洲 250  其他 700   想要解决这个问题，你会怎么做？生成一个带有洲Code的View，是一个解决方法，但是这样很难动态的改变统计的方式。  如果使用Case函数，SQL代码如下:  SELECT  SUM(population),          CASE country                  WHEN '中国'     THEN '亚洲'                  WHEN '印度'     THEN '亚洲'                  WHEN '日本'     THEN '亚洲'                  WHEN '美国'     THEN '北美洲'                  WHEN '加拿大'  THEN '北美洲'                  WHEN '墨西哥'  THEN '北美洲'          ELSE '其他' END  FROM    Table\_A  GROUP BY CASE country                  WHEN '中国'     THEN '亚洲'                  WHEN '印度'     THEN '亚洲'                  WHEN '日本'     THEN '亚洲'                  WHEN '美国'     THEN '北美洲'                  WHEN '加拿大'  THEN '北美洲'                  WHEN '墨西哥'  THEN '北美洲'          ELSE '其他' END;   同样的，我们也可以用这个方法来判断工资的等级，并统计每一等级的人数。SQL代码如下；   SELECT          CASE WHEN salary <= 500 THEN '1'               WHEN salary > 500 AND salary <= 600  THEN '2'               WHEN salary > 600 AND salary <= 800  THEN '3'               WHEN salary > 800 AND salary <= 1000 THEN '4'          ELSE NULL END salary\_class,          COUNT(\*)  FROM    Table\_A  GROUP BY          CASE WHEN salary <= 500 THEN '1'               WHEN salary > 500 AND salary <= 600  THEN '2'               WHEN salary > 600 AND salary <= 800  THEN '3'               WHEN salary > 800 AND salary <= 1000 THEN '4'          ELSE NULL END;   二，用一个SQL语句完成不同条件的分组。   有如下数据  国家（country） 性别（sex） 人口（population）  中国 1 340  中国 2 260  美国 1 45  美国 2 55  加拿大 1 51  加拿大 2 49  英国 1 40  英国 2 60   按照国家和性别进行分组，得出结果如下  国家 男 女  中国 340 260  美国 45 55  加拿大 51 49  英国 40 60   SELECT country,         SUM( CASE WHEN sex = '1' THEN                        population ELSE 0 END),  --男性人口         SUM( CASE WHEN sex = '2' THEN                        population ELSE 0 END)   --女性人口  FROM  Table\_A  GROUP BY country; |

#### 随机查询5条数据

SELECT \* FROM `table` ORDER BY RAND() LIMIT 5

### 2.Trigger (需要掌握作用,简单的创建与删除)

作用必须知道

https://www.cnblogs.com/duodushu/p/5446384.html

### View(需要掌握作用,简单的创建与删除)

作用必须知道：

https://www.cnblogs.com/sker/p/5765042.html

### 索引(需要掌握优缺点,简单的创建与删除)

优缺点使用场景必须知道

索引：可以理解为“排好序的快速查找数据结构”

　　优点：

　　　　　　　1、所有的MySql列类型(字段类型)都可以被索引，也就是可以给任意字段 设置索引

　　　　　　　2、大大加快数据的查询速度

　　缺点：

　　　　　　　1、创建索引和维护索引要耗费时间，并且随着数据量的增加所耗费的时间 也会增加

　　　　　　　2、索引也需要占空间，我们知道数据表中的数据也会有最大上线设置的， 如果我们有大量的索引，索引文件可能会比数据文件更快达到上线值

　　　　　　　3、当对表中的数据进行增加、删除、修改时，索引也需要动态的维护，降 低了数据的维护速度。

　　使用原则：

　　　　　　      通过上面说的优点和缺点，我们应该可以知道，并不是每个字段度设置索 引就好，也不是索引越多越好，而是需要自己合理的使用。

　　　　　　　1、对经常更新的表就避免对其进行过多的索引，对经常用于查询的字段应 该创建索引，

　　　　　　　2、数据量小的表最好不要使用索引，因为由于数据较少，可能查询全部数 据花费的时间比遍历索引的时间还要短，索引就可能不会产生优化效果。

　　　　　　　3、在一同值少的列上(字段上)不要建立索引，比如在学生表的"性别"字段上 只有男，女两个不同值。相反的，在一个字段上不同值较多可是建立索引。

2.  创建索引

在执行CREATE TABLE语句时可以创建索引，也可以单独用CREATE INDEX或ALTER TABLE来为表增加索引。

1．ALTER TABLE

ALTER TABLE用来创建普通索引、UNIQUE索引或PRIMARY KEY索引。

ALTER TABLE table\_name ADD INDEX index\_name (column\_list)

ALTER TABLE table\_name ADD UNIQUE (column\_list)

ALTER TABLE table\_name ADD PRIMARY KEY (column\_list)

其中table\_name是要增加索引的表名，column\_list指出对哪些列进行索引，多列时各列之间用逗号分隔。索引名index\_name可选，缺省时，MySQL将根据第一个索引列赋一个名称。另外，ALTER TABLE允许在单个语句中更改多个表，因此可以在同时创建多个索引。

2．CREATE INDEX

CREATE INDEX可对表增加普通索引或UNIQUE索引。

CREATE INDEX index\_name ON table\_name (column\_list)

CREATE UNIQUE INDEX index\_name ON table\_name (column\_list)

table\_name、index\_name和column\_list具有与ALTER TABLE语句中相同的含义，索引名不可选。另外，不能用CREATE INDEX语句创建PRIMARY KEY索引。

3．索引类型

在创建索引时，可以规定索引能否包含重复值。如果不包含，则索引应该创建为PRIMARY KEY或UNIQUE索引。对于单列惟一性索引，这保证单列不包含重复的值。对于多列惟一性索引，保证多个值的组合不重复。

PRIMARY KEY索引和UNIQUE索引非常类似。事实上，PRIMARY KEY索引仅是一个具有名称PRIMARY的UNIQUE索引。这表示一个表只能包含一个PRIMARY KEY，因为一个表中不可能具有两个同名的索引。

下面的SQL语句对students表在sid上添加PRIMARY KEY索引。

ALTER TABLE students ADD PRIMARY KEY (sid)

4.  删除索引

可利用ALTER TABLE或DROP INDEX语句来删除索引。类似于CREATE INDEX语句，DROP INDEX可以在ALTER TABLE内部作为一条语句处理，语法如下。

DROP INDEX index\_name ON talbe\_name

ALTER TABLE table\_name DROP INDEX index\_name

ALTER TABLE table\_name DROP PRIMARY KEY

其中，前两条语句是等价的，删除掉table\_name中的索引index\_name。

第3条语句只在删除PRIMARY KEY索引时使用，因为一个表只可能有一个PRIMARY KEY索引，因此不需要指定索引名。如果没有创建PRIMARY KEY索引，但表具有一个或多个UNIQUE索引，则MySQL将删除第一个UNIQUE索引。

如果从表中删除了某列，则索引会受到影响。对于多列组合的索引，如果删除其中的某列，则该列也会从索引中删除。如果删除组成索引的所有列，则整个索引将被删除。

了解：（不建议现在了解）

https://www.cnblogs.com/whgk/p/6179612.html

### 存储过程/函数补充

游标使用：

|  |
| --- |
| DELIMITER //  CREATE PROCEDURE java1707()  BEGIN  DECLARE DONE INT; /\*用于判断游标\*/  DECLARE my\_account VARCHAR(50);  DECLARE my\_password VARCHAR(50);  DECLARE myCursor CURSOR FOR SELECT account,PASSWORD FROM user\_info;  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET DONE = 1; /\*DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;\*/  OPEN myCursor;  my\_loop:LOOP  FETCH NEXT FROM myCursor INTO my\_account,my\_password;  IF DONE=1  THEN  LEAVE my\_loop;  END IF;  INSERT INTO test(tname) VALUES(my\_account);  INSERT INTO test(tname) VALUES(my\_password);  END LOOP;  CLOSE myCursor;  END//  DELIMITER ; |

下面的现在不讲 如果上班需要用到自己去看 现在讲了也记不住，需要知道都是可以实现的

mybatis调用mysql返回结果集非常方便

<https://blog.csdn.net/liuxiao723846/article/details/46819157>

Mybatis传递数组或者集合到mysql存储过程 太麻烦了 就说需求没遇到过

常问：

1. 存储过程的优缺点
2. 存储过程与函数的区别
3. In、out、inout区别
4. 异常处理

### sql优化

必须掌握：(冰山一脚，但是面试够了，如果以后需要用到再深入学习)

一、导致索引失效（应该尽量避免）

1.对索引列运算，运算包括（+、-、\*、/、！、<>、%、like'%\_'（%放在前面）、or、in、exist等），导致索引失效，应该建立聚合函数索引，或者去掉计算。

例如：查询工资大于5000的员工

select name from student where salary-5000>0 其中salary为索引,会导致失效,因为进行了运算

应该改为

select name from student where salary>5000

2.如果条件中有or，即使其中有条件带索引也不会使用(这也是为什么尽量少用or的原因)

3.　like查询是以%开头

4.隐式转换 （如果列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不使用索引）

5.条件中使用is null 索引失效；is not null Betree索引生效(如果可以应该使用默认值代替null字段)

如：

select id from t where num is null

可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：

select id from t where num=0

in 和 not in 也要慎用，否则会导致全表扫描，尽量使用between或者exits代替in

如：

select id from t where num in(1,2,3)

对于连续的数值，能用 between 就不要用 in 了：

select id from t where num between 1 and 3

如：

select num from a where num in(select num from b)

用下面的语句替换：

select num from a where exists(select 1 from b where num=a.num)

进行函数操作导致索引失效

select id from t where substring(name,1,3)='abc'--name以abc开头的id

select id from t where datediff(day,createdate,'2005-11-30')=0--'2005-11-30'生成的id

应改为:

select id from t where name like 'abc%'

select id from t where createdate>='2005-11-30' and createdate<'2005-12-

9.任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段