# 模拟面试题

## 组别1

### 1.HashMap初始值大小

HashMap 初始化大小是 16 ，扩容因子默认 0.75（可以指定初始化大小，和扩容因子）

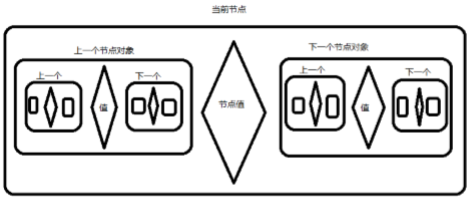
扩容机制:(当前大小 和 当前容量 的比例超过了 扩容因子，就会扩容，扩容后大小为一倍。例如：初始大小为 16 ，**扩容因子 0.75 ，当容量为 12 的时候，比例已经是满足 0.75 。触发扩容，扩容后的大小为 32**.)

### 2.ArrayList初始值大小

**ArrayList 初始化大小是10扩容大小为原来的1.5倍**



**linkedList 是一个双向链表，没有初始化大小，也没有扩容的机制**，就是一直在前面或者后面新增就好。下图为对于双向链表的理解



### 3.线程生命周期

当线程创建并启动后 它经历**五种状态:**

* **新建**
* **就绪**
* **运行**
* **阻塞**
* **死亡**
* **新建:** 当线程对象创建出来进入了新建状态，和其他java对象一样，仅仅是由java虚拟机为其分配了内存
* **就绪**:当调用start方法之后进入就绪状态。这里start后不是进入运行状态 其实线程内部还是依赖jvm的调度 当调用start方法之后 jvm会认为这个线程可以执行的 至于什么执行取决于jvm内部的调度
* **运行:**如果就绪状态的线程获得了cpu的执行权，这个线程就是运行状态，当这个线程运行的时候不会一直霸占cpu的执行权，线程在执行的过程中 会从cpu上调度下来 以便其他线程能获取执行的机会
* **阻塞:**线程进入停滞状态 ?

在以下情况下线程会由运行进行阻塞状态：

* 线程调用一个阻塞的方法 方法返回该线程之前 该线程一直阻塞
* 线程调用sleep方法
* 线程尝试获取同步监视器，但是该同步监视器被其他线程持有
* 线程调用suspend方法挂起

由阻塞进入运行状态?

* 调用阻塞的方法返回
* 调用sleep方法到期
* 线程成功获取同步监视器
* 调用resume方法恢复
* **死亡:** 就是线程的结束 ；

让线程结束的方法:

* Run方法执行完毕
* 线程抛出异常
* 直接调用stop方法结束线程

**判断线程死亡可以使用isAlive方法。**当线程处于就绪、运行和阻塞状态返回true，否则返回false

**不能对死亡的线程重写调用start方法重新启动**

另外 **suspend stop** 方法容易导致**死锁 不推荐使用**

### 4.数据库使用过哪几种？

我所接触过的数据库有Microsoft Sql Server数据库、Microsoft Access数据库、oracle数据库、mysql数据库，他们都是属于关系型数据库，此外还接触过NoSQL数据库例如redis，以键值形式存储数据

**关系型数据库应用广泛，能进行事务处理和表连接等复杂查询。相对地，NoSQL数据库只应用在特定领域，基本上不进行复杂的处理，但它恰恰弥补了之前所列举的关系型数据库的不足之处。**

非关系型数据库具有如下特点：

a.数据模型比较简单

b.需要灵活性更强的IT系统

c.对数据库性能要求较高

d.不需要高度的数据一致性

e.对于给定key，比较容易映射复杂值的环境

Redis提供了5种不同类型的数据结构，各式各样的问题都可以很自然地映射到这些数据结构上：Redis的数据结构致力于帮助用户解决问题，而不会像其他数据库那样，要求用户扭曲问题来适应数据库。除此之外，**通过复制、持久化（persistence）和客户端分片（client-side sharding）等特性，用户可以很方便地将Redis扩展成一个能够包含数百GB数据、每秒处理上百万次请求的系统。**

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。 它支持多种类型的数据结构，**如字符串（strings）， 散列（hashes）， 列表（lists）， 集合（sets）， 有序集合（sorted sets）**与范围查询,bitmaps，hyperloglogs和地理空间（geospatial） 索引半径查询。

### 5.JDBC获取流程

**（1）加载 JDBC 驱动程序：**

在连接数据库之前，首先要加载想要连接的数据库的驱动到 JVM（Java 虚拟机），这通过 java.lang.Class 类的静态方法 forName(String className)实现。成功加载后，会将 Driver 类的实例注册到 DriverManager 类中。

**（2）提供 JDBC 连接的 URL**

•连接 URL 定义了连接数据库时的协议、子协议、数据源标识。

•书写形式：协议：子协议：数据源标识

协议：在 JDBC 中总是以 jdbc 开始

子协议：是桥连接的驱动程序或是数据库管理系统名称。

数据源标识：标记找到数据库来源的地址与连接端口。

**（3）创建数据库的连接**

•要连接数据库，需要向 java.sql.DriverManager请求并获得Connection 对象， 该对象就代表一个数据库的连接。

•使用 DriverManager 的 getConnectin(String url , String username , String password )方法传入指定的欲连接的数据库的路径、数据库的用户名和密码来获得。

**（4）创建一个 Statemen**t

•要执行 SQL 语句，必须获得 java.sql.Statement 实例，Statement 实例分为以下 3 种类型：

执行静态 SQL 语句。通常通过 Statement 实例实现

执行动态 SQL 语句。通常通过 PreparedStatement 实例实现

执行数据库存储过程。通常通过 CallableStatement 实例实现

具体的实现方式：

Statement stmt = con.createStatement() ;

PreparedStatement pstmt = con.prepareStatement(sql) ;

CallableStatement cstmt = con.prepareCall("{CALL demoSp(? , ?)}") ;

**（5）执行 SQL 语句**

Statement 接口提供了三种执行 SQL 语句的方法：executeQuery 、executeUpdate 、execute

1.ResultSet executeQuery(String sqlString)：执行查询数据库的 SQL 语句，返回一个结果集（ResultSet）对象。

2.int executeUpdate(String sqlString)：用于执行insert、update或 delete语句以及 SQL DDL 语句，如：create table 和 drop table 等

3.execute(sqlString):用于执行返回多个结果集、多个更新计数或二者组合的 语句。

具体实现的代码：

**ResultSet rs = stmt.executeQuery("SELECT \* FROM ...") ;**

**int rows = stmt.executeUpdate("INSERT INTO ...") ;**

**boolean flag = stmt.execute(String sql) ;**

**（6）处理结果**

两种情况：

1、执行更新返回的是本次操作影响到的记录数。

2、执行查询返回的结果是一个 ResultSet 对象。

• ResultSet 包含符合 SQL 语句中条件的所有行，并且它通过一套 get 方法提供了对这些行中数据的访问。

• 使用结果集（ResultSet）对象的访问方法获取数据：

while(rs.next()){

String name = rs.getString("name") ;

String pass = rs.getString(1) ; // 此方法比较高效

}

（列是从左到右编号的，并且从列 1 开始）

**（7）关闭 JDBC 对象**

操作完成以后要把所有使用的 JDBC 对象全都关闭，以释放 JDBC 资源，关闭顺序和声明顺序相反：

a、关闭记录集

b、关闭声明

c、关闭连接对象

### 6.SpringMVC流程

**SpringMVC架构学习内容：**

* 什么是SpringMVC?
* SpringMVC框架原理(掌握)

**前端控制器、处理器映射器、处理器适配器、视图解析器**

* SpringMVC的入门程序

目的：对前端控制、处理器映射器、处理器适配器、视图解析器学习

非注解的处理器映射器、非注解的处理器适配器

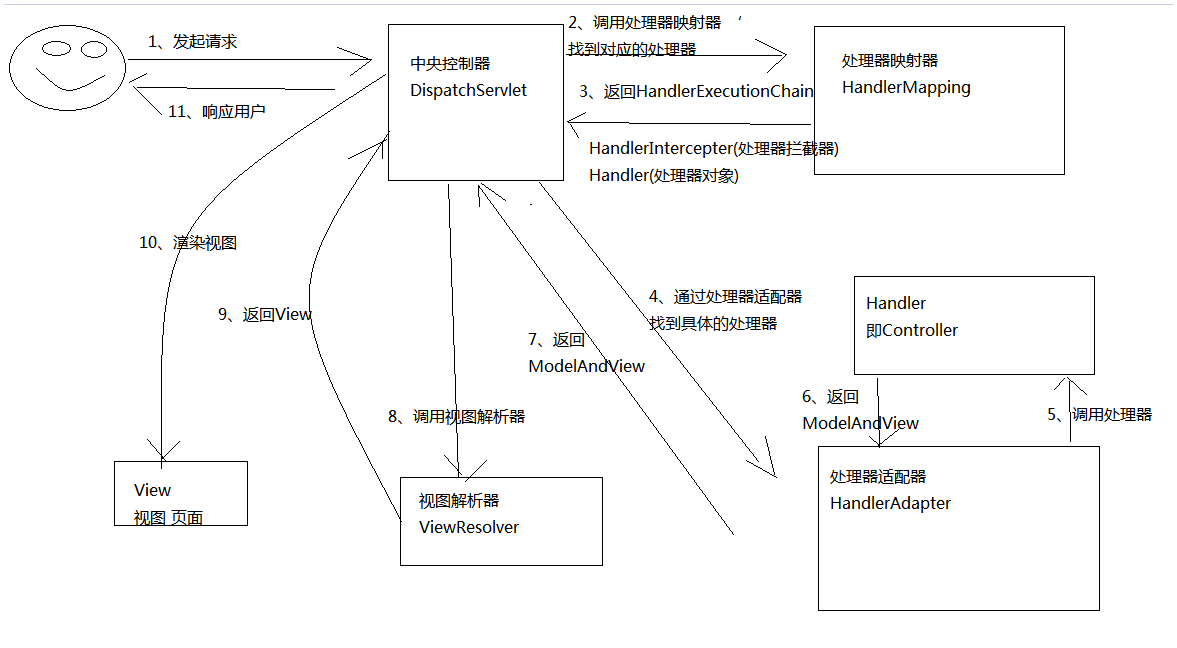
注解的处理器映射器、处理器适配器

* SpringMVC和MyBatis的整合
* SpringMVC注解开发(重点)

常用的注解学习

参数绑定(简单类型、POJO、集合类型)

自定义参数绑定



**架构流程：**

* 用户发送请求到前端控制器DispatchServlet
* DispatchServlet接受到请求调用HandlerMapping处理器映射器
* 处理器映射器根据请求的URL找到具体的处理器生成处理器对象以及拦截器并把所有的内容返回到DispatchServlet
* DispatchServlet通过HandlerAdapter处理器适配器调用处理器
* 执行处理器(Controller )
* Controller执行完成返回ModelAndView
* HandlerAdapter将Controller执行的结果ModelAndView返回给DispatchServlet
* DispatchServlet将ModelAndView传递给ViewReslover视图解析器
* 视图解析器会返回具体的View
* DispatchServlet对View进行渲染、将模型数据填充到视图
* DispatchServlet响应客户

### 7.面向对象的特性

面向对象的三大特性：继承、多态、封装

* 继承：

子类拥有父类的所有属性和方法（除了private修饰的属性不能拥有）从而实现了实现代码的复用；

**Java继承具有单继承的特点，每个类只能继承一个直接父类**

继承的特性：

* 在java中所有的类默认继承了**java.lang.object**这个类
* 继承具有传递性
* 继承是单继承，只能继承一个父类。
* 当初始化子类时先初始化父类，然后再初始化子类。
* 构造函数是不能被继承的，但是可以在子类中通过super关键字调用
* 多态：

Java引用变量有两个类型，一个是编译时的类型，一个是运行时的类型。其中编译时的类型由声明该变量时使用的类型决定，而运行时的类型由实际赋值给该变量的对象决定。

多态分为编译时多态（又称静态多态）运行时多态（又称动态多态）

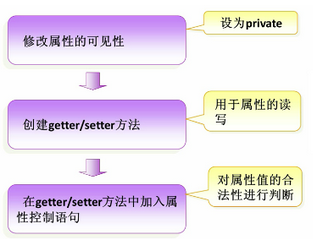
* 编译时多态有重载（overload），编译时多态在编译时就已经确定，运行时运行的时候调用的是确定的方法。
* 运行时多态有子类继承父类（extends）、类实现接口（implements）
* 封装：

将类的某些信息隐藏在类内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问。

只能通过规定的方法访问数据。

隐藏类的实例细节，方便修改和实现。

封装的实现步骤：



### 8.多态是什么？

Java引用变量有两个类型，一个是编译时的类型，一个是运行时的类型。其中编译时的类型由声明该变量时使用的类型决定，而运行时的类型由实际赋值给该变量的对象决定。

多态分为编译时多态（又称静态多态）运行时多态（又称动态多态）

* 编译时多态有重载（overload），编译时多态在编译时就已经确定，运行时运行的时候调用的是确定的方法。
* 运行时多态有子类继承父类（extends）、类实现接口（implements）

### 9.线程创建方式

线程的创建有两种方式

* 继承Thread类，重写run方法，调用start方法启动线程
* 实现Runnable接口，重写run方法，调用start方法启动线程

详细说明：

通过继承Thread类或者实现Runnable接口都可以实现多线程：

区别:

* 采用实现Runnable接口的方式：
* 线程仅仅是实现了Runnable接口，还可以继承其他类
* 在这种方式下，可以多个线程共享一个Target对象，非常适合多个线程共同处理同一份资源，从而将cpu 代码、数据分开，较好的体现了面向对象的思想
* 劣势:编程稍微复杂，如果要访问当前线程，必须使用Thread.currentThread()方法
* 采用继承Thread类
* 劣势：因为线程继承了Thread类，所有不能再继承其他父类
* 优势:编程比较简单，如果需要访问当前线程，直接使用this对象即可

推荐使用实现Runnable接口即可

### 10.UDP和TCP的区别

* TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接
* TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付
* TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的。UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）
* 每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信
* TCP首部开销20字节;UDP的首部开销小，只有8个字节
* TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

### 11.Zookeeper如何使用

**Zookeeper是一个高效的分布式协调服务**，它暴露了一些公用服务，比如命名，配置，管理，同步控制，群组服务，配置维护、域名服务、分布式同步、组服务等。我们可以使用zk来实现，比如达成共识，**集群管理， 选举操作**等。

**Zookeeper是一个高可用的分布式管理与协调框架**，是基于ZAB算法(原子消息广播协议)的实现。该框架能很好的保证分布式环境中数据的一致性，也正是基于这样的特性使用的Zookeeper成为解决分布式一致性问题的利器。

Zookeeper有三个角色：leader、Follower、Observer

Zookeeper应用场景

* 统一的命名服务
* 配置管理
* 集群管理
* 共享锁
* 队列管理

使用过程：

a.在linux系统中下载、安装zookeeper

b.配置zookeeper相关的环境变量

c.修改zookeeper的配置文件信息zoo.cfg

d.通过命令行zkServer.sh start启动测试zookeeper集群是否可用

e.可以在通过java操作zookeeper，需要导入相关的插件、jar包，编写代码进行测试，有以下两种使用方式：

基于zookeeper原生态的API的使用（Watch（zookeeper的状态和事件类型））

基于ZkClient第三方API的使用（ZkClient实现Watcher）

### 12.Maven有哪些命令，其使用流程是什么？

下载安装maven框架，并配置maven相关的环境变量

安装完成之后通过maven -version指令查看maven安装版本

常见命令：

compile：查看根目录变化 编译程序为.class文件

clean：清理target文件目录的所有的内容

clean compile ：先执行清理后执行编译

clean test：执行清理，然后执行测试，如果程序正常 则会提示执行时间等信息

clean package：把程序打成jar包，可以供其他项目使用，生成的保存目录参考控制台输出

install：通过install指令将相关项目部署到仓库下

我所了解的是maven在eclipse中的使用流程，如果是第一次使用可能需要配置相应的maven仓库，等待较长时间完成数据加载，通过修改pom.xml文件完成相关jar包的导入，也可引入其他部署完成的工程内容（抑或直接通过导入相关jar完成引用）。

Maven的继承与聚合：

继承：继承是为了消除重复，我们把很多相同的配置提取出来

聚合：如果想要执行每个子工程都需要：编辑---测试—打包—部署，可以在父工程中进行统一定义



### 13.数据库存储过程和函数的区别

存储过程(Stored Procedure)：一组可编程的函数，是为了完成特定功能的SQL语句集，经编译创建并保存在数据库中，用户可通过指定存储过程的名字并给定参数(需要时)来调用执行。

存储过程一般是作为一个独立的部分来执行（ EXECUTE 语句执行），而函数可以作为查询语句的一个部分来调用（SELECT调用），由于函数可以返回一个表对象，因此它可以在查询语句中位于FROM关键字的后面。 SQL语句中不可用存储过程，而可以使用函数。函数是可以嵌入在sql中使用的,可以在select中调用，而存储过程不行。

存储过程一旦调试完成通过后就能稳定运行，这与各个业务在一段时间内是相对稳定和确定是匹配的；存储过程大大地减少了业务系统与数据库的交互，一定程度降低了业务系统与数据库的耦合，例如即使业务系统与应用系统不在同一城市，对性能的影响也可控（100条SQL语句交互一次，即使延时由同城1ms增加到异地50ms，也只是增加49ms，如果交互100次，则增加4900ms）。

函数有只能返回一个变量的限制，而存储过程可以返回多个。

函数限制比较多，比如不能用临时表，只能用表变量．还有一些函数都不可用等等．而存储过程的限制相对就比较少。

一般存储过程是用于执行特定的操作 比如插入员工数据、比如定时任务、都可以使用存储过程实现，函数用于返回特定的值

## 组别2

### 1.Java中的常用类有哪些？

根据不同的内容可将Java中的常用类分为以下的几类：

**a.与系统相关的常用类：Scanner、System、Runtime**

<1>Scanner类用于数据读取（键盘、文件、控制结束）

<2>System类用于输出数据（err带缓冲输出、System.identityHashCode(obj)对对象进行唯一标识）

String重写了hashCode，只要满足两个字符串的字符序列相同则其hashCode也对应相同，但这样就无法用hashCode唯一标识相应对象，因此使用System类的identityHashCode方法对对象进行唯一标识

<3>Runtime类用于输出系统的相关信息或者是执行相关的指令

Runtime类代表的java程序的运行环境，每个java程序都有与之对应的Runtime实例，应用程序与对象的运行环境进行关联

**b.与字符串相关的常用类：Object、String、StringBuffer、StringBuilder、StringTokenizer**

<1>Object是所有类、数组的父类，也就说允许在任何类对象中调用Obejct的方法。

Object常用的方法：

* equals(Object obj)：判断字符串是否相同、可以通过重载判断是否引用同一个对象
* finalize()：初始化
* hashCode()：获取对象的内存地址（哈希码）
* getClass()：获取当前对象所属类的类名
* toString()：输出对象的内存地址，可通过重载实现输出对象的相关信息

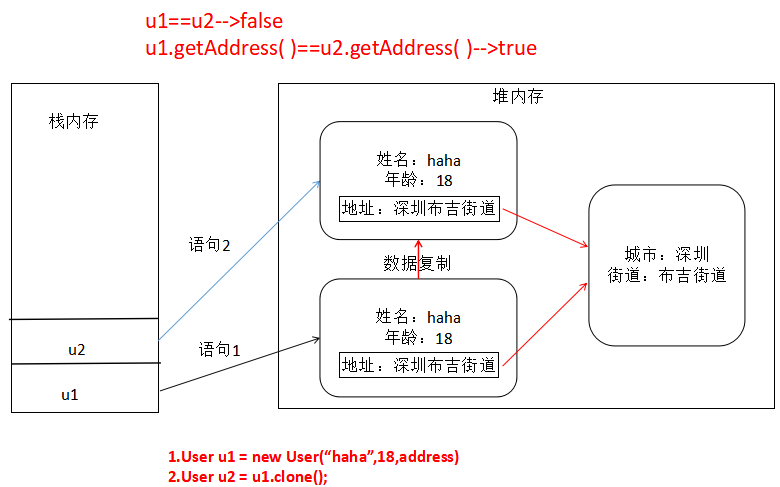
涉及内容：

**浅克隆**相关内容，如果一个类中要实现克隆，需要实现Cloneable接口

Java7新增了一个工具类**Objects**， 它提供了一些工具方法，这些工具方法大多数是针对安全性操作的。

Cloneable接口是个空接口，是**标志性接口**,如果实现该接口，需要重写clone方法（不需要作任何改动） 分析：此处定义User类实现Cloneable接口 ,u2是由u1对象克隆而来，因此u1、u2并不是指向同一个对象，但两者的信息完全相同（此处为姓名、年龄、地址），克隆的含义是指将数据复制一份，由u2指向这份数据，但这份数据中的，address指向还是原来的address，因此有u1.getAddress()==u1.getAddress()返回的结果为true，而u1==u2返回的结果为false

（标志性接口的含义：在java 中，RandomAccess和Cloneable 、Serializable一样，都是标志性接口，不需要任何实现，只是又来表明其实现类具体有某种特质的，实现了Cloneable表明可以拷贝，实现Serializable 接口表明被序列化了，实现RandomAccess接口则表明这个类可以随机存取。）



<2>String类是不可变类，就是一个String对象被创建出来以后，包含在这个字符串对象的字符序列的不可改变的，即直到这个对象被销毁，被垃圾回收都不会改变。

<3>StringBuffer对象代表的是一个可变字符序列，当一个StringBuffer创建出来以后，通过StringBuffer提供的append/insert/reverse/setCharAt等方法改变这个字符串对象的字符序列

<4>StringBulider和StringBuffer这两个类功能基本相似，用法基本相似，不同的是StringBuffer是线程安全的，StringBulider不是线程安全的。

<5> 对于特殊字符，无法直接通过String类型的split方法对指定字符串进行分隔，则可通过StringTokenizer能够实现以指定特殊字符分割字符串，之后还会介绍正则表达式相关的内容，借助正则表达式，实现指定格式的字符串分割

**c.与数学数字相关的常用类：Math、Random、BigDecimal（涉及大数运算与精确运算）**

Math类用于执行常见的数学运算操作

Random类用于获取不同数据类型的随机数

BigDecimal、BigInteger（存储整型数据）类用于实现大数运算和精确运算

**d.与日期相关的常用类：Date、Calendar、位符的格式化（NumberFormat、DateFormat、SimpleDateFormat）**

Date、Calendar类用于处理日期

NumberFormat用于处理数值格式

DateFormat、SimpleDateFormat用于处理日期格式

### 2.String常用的方法有哪些？equals与==的区别？

**<1>String类中常用的方法**

a.得到字符串中指定位置的字符(char)、得到指定字符的位置(int)

charAt、indexOf

b.两个字符串之间比较大小

compareTo、compareToIgnoreCase

c.链接字符串(可通过‘+’实现)

concat

d.看指定字符串是否包满足要求(是否为空、是否包含某些内容、是否包含指定的前后缀、是否匹配)

isEmpty、contains、startsWith、endsWith、matches

e.将字符串与指定的字符串数据进行比较

equals、equalsIgnoreCase

f.得到指定字符串的某些属性(信息)

hashCode、length、toString、trim()

g.用指定的内容替换原字符串数据

replace、replaceAll

h.根据指定要求拆分字符串、 截取字符串数据

split、substring

i.将指定字符串进行转化(转化为字符数组、全大写、全小写)

toCharArray、toLowerCase、toUpperCase

j.将基本的数据类型转化为String类型

valueOf(...)

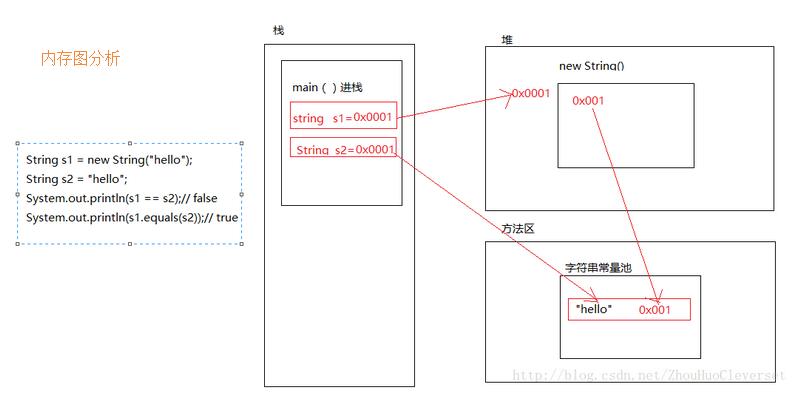
**<2>equals与==的区别**

Java程序中有两种方式是两个变量是否相等。一种是利用==判断，另外一种是使用equals方法来判断。

当使用==进行判断的时候，如果两个变量都是基本类型的变量，并且都是数值类型，则只要两个变量的值相同时，便返回true。但如果两个变量的类型是引用类型，则只有两个变量均指向相同的对象才返回true。

equals比较引用类型默认也是比较地址值是否相同，而String类重写了equals()方法，比较的是内容是否相同。使用equals方法进行判断时则以具体equals方法中实际比较的项目为准，可通过重写equals方法实现自定义的比较方式（equals 方法是用于比较两个独立对象的内容是否相同，就好比去比较两个人的长相是否相同，它比较的两个对象是独立的）

举例：String s1 = new String("hello"); String s2 = "hello";两者的区别



1.首先，通过main（）方法进栈。

2.然后再栈中定义一个对象s1,去堆中开辟一个内存空间，将内存空间的引用赋值给s1，“hello”是常量，然后去字符串常量池 查看是否有hello字符串对象，没有的话分配一个空间存放hello，并且将其空间地址存入堆中new出来的空间中。

3.在栈中定义一个对象s2，然后去字符串常量池中查看是否有”hello”字符串对象，有，直接把”hello”的地址赋值给s2.

4.即s1中存的是堆中分配的空间，堆中分配的空间中存的是字符串常量池中分配空间存放”hello”的空间的地址值。而s2中之间存的是字符串常量池中分配空间存放”hello”的空间的地址值。

5.由于s1与s2中存放的地址不同，所以输出false。因为，类String重写了equals()方法，它比较的是引用类型的值是否相等，所以输出true。即结果为false、true。

### 3.简单介绍Object类和其常见的方法？说说为什么要重写equals方法？

**<1>Object常用的方法：**

* equals(Object obj)：判断字符串是否相同、可以通过重载判断是否引用同一个对象
* finalize()：初始化
* hashCode()：获取对象的内存地址（哈希码）
* getClass()：获取当前对象所属类的类名
* toString()：输出对象的内存地址，可通过重载实现输出对象的相关信息

**<2>为什么要重写equals方法？**

默认equals方法在比较两个对象时判断其是否指向相同的地址，但在实际的开发中我们可能希望两个对象的某些属性相同就认为这两个对象相同，因此需要重写equals方法，用以适应不同的情景设计

### 4.IO中的设计模式是什么（装饰者模式）？简单介绍装饰者模式，并说明为什么不用继承而选择装饰者模式？

装饰者模式：动态的将职责附加到对象上，如果需要对类进行功能的扩展，装饰者模式能提供比继承更加良好的设计原则

多用组合，少用继承

类的设计是对扩展开放，对修改关闭

/\*\*

\* 装饰者模式：

\* 1.定义一个私有的装饰对象

\* 2.提供一个带参数的构造函数初始化该对象

\* 3.根据需求补充相应的功能

\*/

/\*\*

\* 装饰者模式是为了解决给类的功能增强而出现的

\* --对Writer根据不同的业务进行划分

\* Writer

\* |------TextWriter

\* |------MediaWriter

\* |------ImageWriter

\*

\* ---随着业务功能的提升 这时需要对Writer再次改写 要求提升性能 --利用缓冲

\* Writer

\* |------TextWriter

\* |----BufferedTextWriter

\* |------MediaWriter

\* |---BufferedMediaWriter

\* |------ImageWriter

\* |---BufferedImageWriter

\*

\* ---业务进一步提升需求 这时要求 对Writer 提高容错性和完整性 --- 利用差异传输...

\* Writer

\* |------TextWriter

\* |----BufferedTextWriter

\* |----InfoExceptionBufferedTextWrite

\* |------MediaWriter

\* |---BufferedMediaWriter

\* |---InfoExceptionBufferedMediaWriter

\* |------ImageWriter

\* |---BufferedImageWriter

\* |----InfoExceptionBufferedImageWriter

\*

\* 可以利用装饰者模式 解决继承来实现功能增强的这种行为

\* Writer

\* |---TextWriter

\* |---MediaWriter

\* |---ImageWriter

\* |---BufferedWriter

\* |--InfoExceptionWriter

\*

\* class BufferedWriter extends Writer{

\* private TextWriter textWriter;

\* public BufferedWriter( TextWriter textWriter){

\* this.textWriter=textWriter;

\* }

\* }

\*

**\* 使用装饰者模式的 特点:**

**\* 1.装饰者中的构造函数 必须接受被装饰的类**

**\* 2.装饰的类和被装饰的类应该属于同一个体系**

\*/

在java开发中，继承为单继承模式，一个类只能继承一个父类，需要通过实现接口扩展业务功能，作用比较局限

装饰模式比继承要灵活，避免了继承体系臃肿。而且降低了类于类之间的关系。(继承体系的时候，缓冲类，必须继承相应的类。)  
装饰类因为增强已有对象，具备的功能和已有的是相同的，只不过提供了更强功能。  
所以装饰类和被装饰类通常是都属于一个体系中的。(可以理解为实现共同的接口，或继承一样的父类)

### 5.Vector、ArrayList、LinkedList三者的区别?LinkedList如何查找数据？（查看源码）

个人解析：

* Java提供了一个线性接口。而ArrayList/Vector/LinkedList都是其子类。
* ArrayList和Vector底层都是数组结构，而LinkedList集合底层是链表结构。
* Vector是线程安全的，其他两个不是线程安全的。
* 如果需要经常执行插入、删除操作改变含有大量数据的List集合的大小 可以使用LinkedList。使用ArrayList和Vector会重新分配内部数组的大小和位置，所以性能相当于LinkedList较低。如果经常用到查找和修改操作，可以使用ArrayList和Vector。

参考解析：

a.ArrayList 和Vector底层实现是基于数组的实现，而LinkedList底层实现是基于链表的实现。

b.ArrayList和Vector 实现了List接口（List继承于Collection接口）、而LinkedList除了实现了List接口以外还实现了Deque接口。

c.他们都是有序集合，即存储在集合中的元素的位置都是有顺序的，我们以后可以按位置索引号取出某个元素，，并且其中的数据是允许重复的。

d.从同步性来说 Vector是属于线程安全的而ArrayList 和LinkedList不是线程安全的

e.从效率来说ArrayList和Vector底层属于数组 查找修改速度快 增加删除效率慢 LinkedList底层是链表 所以 增删比较快 查改 慢

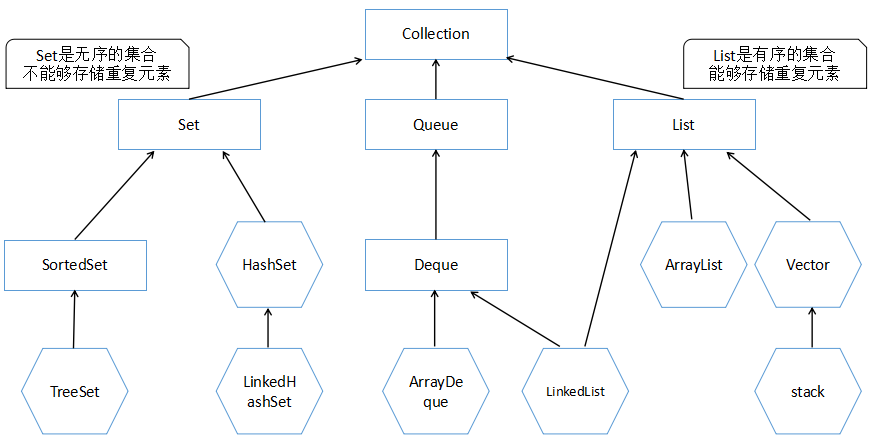
f.从数据增长来说：ArrayList与Vector都有一个初始的容量大小，当存储进它们里面的元素的个数超过了容量时。

在jdk1.7中 ArrayList底层实现增长的方式是

int newCapacity = oldCapacity + (oldCapacity >> 1);

Vector实现增长的方式是

int newCapacity = oldCapacity + ((capacityIncrement > 0) ? capacityIncrement : oldCapacity);



### 6.mysql是如何实现分页的？

根据不同的需求分析去实现：客户端通过传递start(页码)，limit(每页显示的条数)两个参数去分页查询数据库表中的数据，MySql数据库提供了分页的函数limit m,n，但是该函数的用法和我们的需求不一样，所以就需要我们根据实际情况去改写适合我们自己的分页语句，可以得出符合我们自己需求的分页sql格式是：select \* from table limit (start-1)\*limit,limit; 其中start是页码，limit是每页显示的条数。

<1>limit m,n分页语句，但如果在高并发的应用中，该方式查找效率非常dixia

select \* from dept order by deptno desc limit m,n;

<2>limit m语句：

  select \* from dept where deptno >10 order by deptno asc limit n;//下一页

  select \* from dept where deptno <60 order by deptno desc limit n//上一页

这种方式不管翻多少页只需要扫描n条数据，可以给这条语句加上一个条件限制。使得每次扫描不用从第一条开始。这样就能尽量减少扫描的数据量。

例如：每页10条数据，当前是第10页，当前条目ID的最大值是109，最小值是100.

那么跳到第9页：select \* from dept where deptno<100 order by desc limit 0,10;

那么跳到第8页：select \* from dept where deptno<100 order by desc limit 10,10;

那么跳到第11页：select \* from dept where deptno>109 order by asc limit 0,10;

那么跳到第11页：select \* from dept where deptno>109 order by asc limit 10,10;

### 7.单例模式的多种实现方式有哪些？

单例模式：解决保证内存中始终只有一个对象。

如何保证内存中只有一个对象呢？

* 不让其他程序创建该类的对象
* 在本类中创建一个自己的对象
* 对外提供一个获取该对象的方法

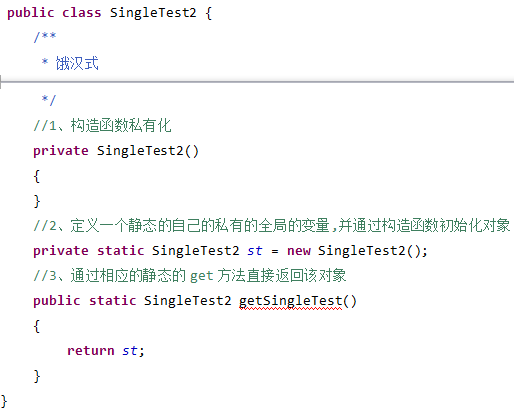
实现的步骤：

* + 构造函数私有化
  + 提供一个静态的自己的私有的全局的变量
  + 然后提供一个get方法用来获取该全局变量

单例模式分为有多重模式，此处简单介绍懒汉式、饿汉式、双重校验锁方式

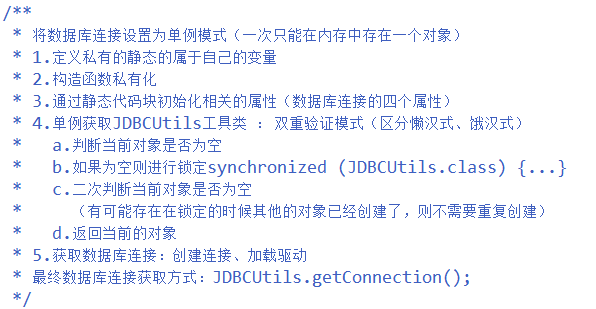
懒汉式指的是在提供静态的自己的私有的全局变量的时候初始化为null，即后续相应的get方法返回该对象时需进行判断，如果为null则进行初始化。下述懒汉式是线程不安全的，在多线程下不能工作。但在getInstance()方法中加入了**synchronized**锁。能够在多线程中很好的工作，而且看起来它也具备很好的lazy loading，但是效率很低（因为锁），并且大多数情况下不需要同步。

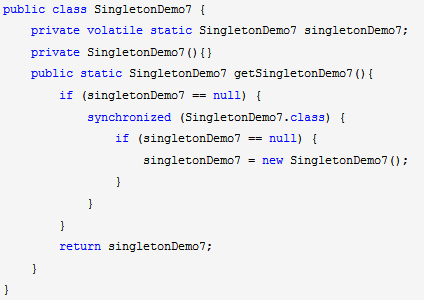
饿汉式指的是在提供静态的自己的私有的全局变量的时候直接创建对象，后续通过相应的get方法返回该对象



此外，在项目设计的过程中将数据库连接设置为单例模式（一次在内存中只能存在一个对象）

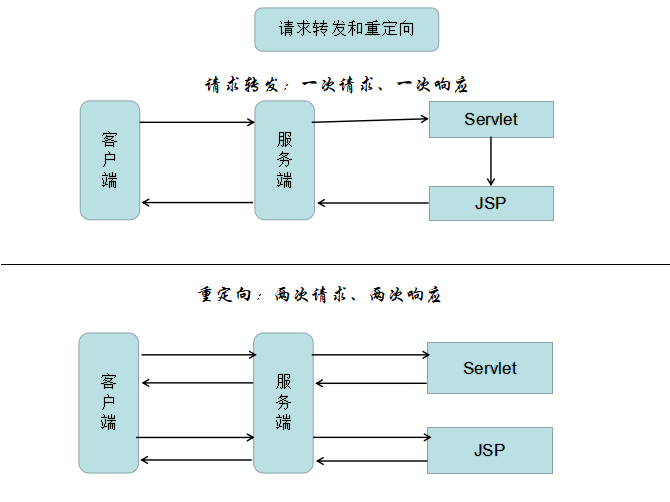
**双重校验锁方式**





### 8.请求转发和重定向的区别？

* 一个web资源接收到客户端请求后，通知浏览器去调用另外一个web资源进行处理，称之为请求转发
* 一个web资源接收到客户端请求后，通知浏览器去访问另外一个web资源，称之请求重定向



* RequestDispatcher.forward方法只能将请求转发给同一个WEB应用中的组件；而HttpServletResponse.sendRedirect 方法还可以重定向到同一个站点上的其他应用程序中的资源，甚至是使用绝对URL重定向到其他站点的资源。 **(范围的不同)**
* 如果传递给HttpServletResponse.sendRedirect 方法的相对URL以“/”开头，它是相对于整个WEB站点的根目录；如果创建RequestDispatcher对象时指定的相对URL以“/”开头，它是相对于当前WEB应用程序的根目录。 **(“/”的含义)**
* 调用HttpServletResponse.sendRedirect方法重定向的访问过程结束后，浏览器地址栏中显示的URL会发生改变，由初始的URL地址变成重定向的目标URL；调用RequestDispatcher.forward 方法的请求转发过程结束后，浏览器地址栏保持初始的URL地址不变。**（URL地址是否变化）**
* HttpServletResponse.sendRedirect方法对浏览器的请求直接作出响应，响应的结果就是告诉浏览器去重新发出对另外一个URL的访问请求；RequestDispatcher.forward方法在服务器端内部将请求转发给另外一个资源，浏览器只知道发出了请求并得到了响应结果，并不知道在服务器程序内部发生了转发行为。 **(请求次数和响应次数)**
* RequestDispatcher.forward方法的调用者与被调用者之间共享相同的request对象和response对象，它们属于同一个访问请求和响应过程；而HttpServletResponse.sendRedirect方法调用者与被调用者使用各自的request对象和response对象，它们属于两个独立的访问请求和响应过程。

### 9.get方式、post方式请求两者有何区别？

标准答案中：(地址栏、数据安全、提交位置)

GET在浏览器回退时是无害的，而POST会再次提交请求。

GET请求会被浏览器主动cache，而POST不会，除非手动设置。

GET请求只能进行url编码，而POST支持多种编码方式。

GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。

GET请求在URL中传送的参数是有长度限制的，而POST没有。实际上HTTP协议对URL长度是没有限制的；限制URL长度大多数是浏览器或者服务器的配置参数

对参数的数据类型，GET只接受ASCII字符，而POST没有限制。

GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。但实际上也可利用开发工具或者抓包获取POST传递的数据。

GET参数通过URL传递，POST放在Request body中。

我在一篇博客中看到有关get、post两种请求方式的详细介绍，对其有有着不同的介绍和理解

GET和POST是HTTP协议中的两种发送请求的方法（HTTP是基于TCP/IP的关于数据如何在万维网中如何通信的协议），两者的区别在于GET产生一个TCP数据包；POST产生两个TCP数据包。

详细来说，对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。

因为POST需要两步，时间上消耗的要多一点，看起来GET比POST更有效。但不能盲目地用get取代post

* GET与POST都有自己的语义，不能随便混用。
* 据研究，在网络环境好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。而在网络环境差的情况下，两次包的TCP在验证数据包完整性上，有非常大的优点。
* 并不是所有浏览器都会在POST中发送两次包，Firefox就只发送一次。

### 10.Servlet是单例实现还是多例实现？如果有1000个用户同时访问是否存在线程安全问题？如何避免这个问题？

Servlet容器默认是采用**单实例多线程**的方式处理多个请求的：

<1>当web服务器启动的时候（或客户端发送请求到服务器时），Servlet就被加载并实例化(只存在一个Servlet实例)；

<2>容器初始化化Servlet主要就是读取配置文件（例如tomcat,可以通过servlet.xml的<Connector>设置线程池中线程数目，初始化线程池通过web.xml,初始化每个参数值等等。

<3>当请求到达时，Servlet容器通过调度线程(Dispatchaer Thread) 调度它管理下线程池中等待执行的线程（Worker Thread）给请求者；

<4>线程执行Servlet的service方法（doGet、doPost方法）；

<5>请求结束，放回线程池，等待被调用；

（注意：避免使用实例变量（成员变量），因为如果存在成员变量，可能发生多线程同时访问该资源时，都来操作它，照成数据的不一致，因此产生线程安全问题）

如果要避免线程安全问题，最简单的方式是将全局变量变为局部变量（即避免全局变量的使用），也可通过加锁（synchronize）设计线程安全的Servlet

### 11.Servlet在执行的时候是如何区分get、post？Servlet的生命周期有哪些？

**<1>Servlet中get、post区别**

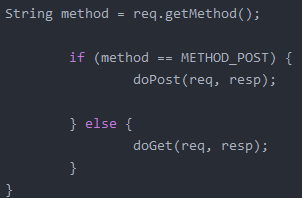
get安全性小于post因为get会把访问信息暴露在浏览器地址栏。

get提交内容信息一般不要大于2k，post提交内容一般不要大于64k。

访问速度get大于post。

get可以保留uri中的参数便于保留。

通过request对象调用getMethod( )方法判断传递数据的请求



**<2>Servlet的生命周期和工作原理**

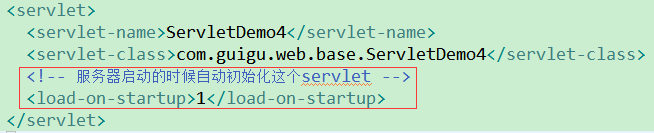
Servlet的的声明周期分为三个阶段：1.初始化阶段 2.响应客户阶段 3.终止阶段

这三个阶段分别对应init方法 service方法 和destory方法，这三个方法分别在不同的阶段进行调用。

**1）Servlet的初始化阶段**

在以下时刻servlet进行初始化 servlet初始化包含以下三种情况，这三种情况都会执行init方法

a.servlet容器启动后，自动加载某些配置信息。需要配置<load-on-startup></load-on-startup>



b.在servlet容器启动后，客户端第一次向servlet发起请求

c.servlet的文件被更改后 重新加载servlet

**注意： init方法只会在第一次初始化servlet的时候执行。**

**2）servlet响应客户端阶段**

Servlet当接受到客户端请求后，servlet会根据这个请求创建两个对象HttpServletRequest对象和HttpServletResponse对象，这个对象分别代表接受请求和响应请求，然后servlet调用service方法。Service方法会根据HttpRequest对象从请求中获取相关的数据，然后处理请求，最后通过HttpResponse对象把数据写出到客户端。

**3）servlet终止阶段**

* 当web应用被终止，或者servlet容器终止运行
* 或servlet容器被重新加载
* 当前终止的时候回调用destroy方法进行终止。

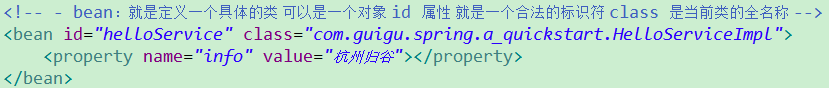
### 12.Spring核心是什么？（AOP、DI、IOC...）

**IOC控制反转、DI依赖注入和AOP面向切面的编程**

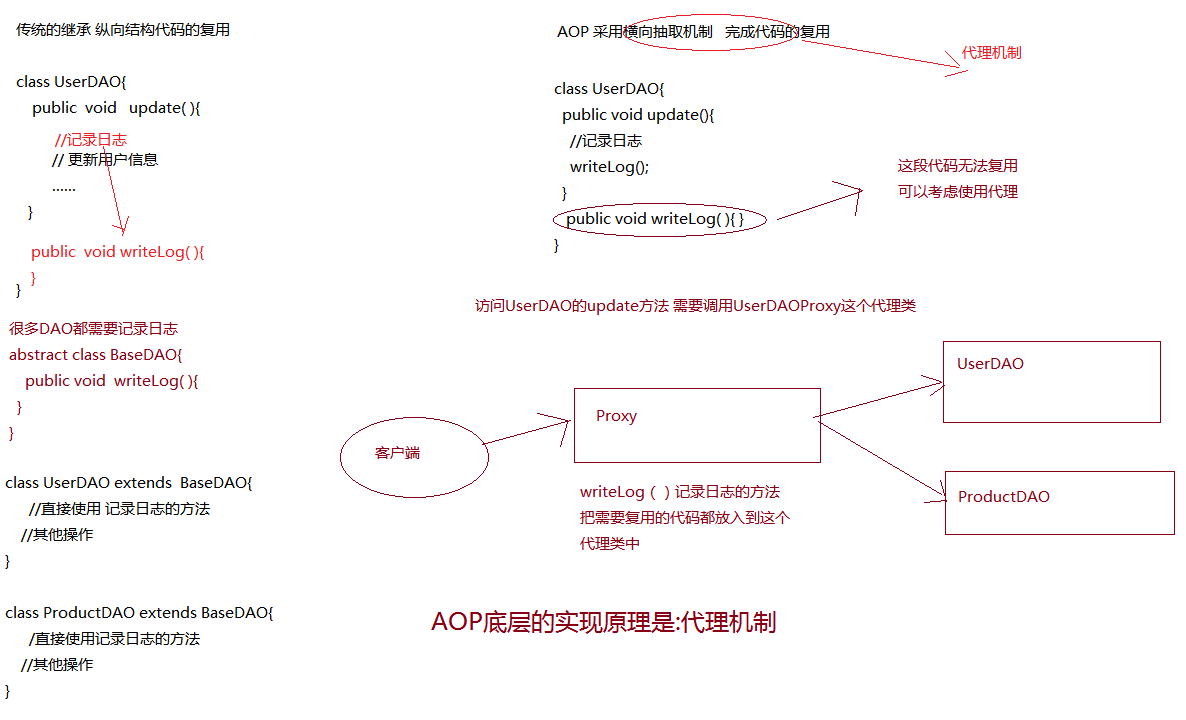
IOC容器 Inverser Of Control反转控制 就是把原来在程序中创建HelloService对象的控制权限交给Spring管理，简单的说 就是HelloService对象控制器被反转到Spring框架内



DI： 依赖注入 在Spring框架负责创建Bean对象的时候 动态的将数据注入到Bean的组件



* AOP Aspect Oriented Programing 面向切面编程
* **AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码**（性能监视、事务管理、安全检查、缓存）
* Spring AOP使用纯Java实现，不需要专门的编译过程和类加载器，在运行期通过代理方式向目标类织入增强代码
* AspectJ是一个基于Java语言的AOP框架，Spring2.0开始，Spring AOP引入对Aspect的支持，AspectJ扩展了Java语言，提供了一个专门的编译器，在编译时提供横向代码的织入



### 13.Spring事务管理方式有哪些？

Spring事务管理分为两种形式

1）编程式事务 (了解)-->基于配置文件的声明式事务

在代码中使用TransactionTemplate手动进行管理事务 (实际开发中几乎没有使用)

1. 声明式事务 (重点掌握)-->基于注解的声明式事务

@Transactional

a、定义一个事务管理器  
b、配置事务特性(相当于声明通知。一般在业务层的类的一些方法上定义事务)  
c、配置哪些类的哪些方法需要配置事务(相当于切入点。一般是业务类的方法上)

### 14.事务涉及的基本概念（特性、隔离性引发的问题、隔离级别、mysql/oracle数据库对应默认的隔离级别）

#### 事务的四个重要概念

##### 什么是事务？

[数据库事务](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E4%BA%8B%E5%8A%A1" \t "https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E4%BA%8B%E5%8A%A1/_blank)(Database Transaction) ，是指作为单个逻辑工作单元执行的一系列[操作](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C/33052" \t "https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E4%BA%8B%E5%8A%A1/_blank)，要么完全地执行，要么完全地不执行。事务的完成由若干个DML语句组成的。

在数据库中,所谓事务是指一组逻辑操作单元,使数据从一种状态变换到另一种状态

为确保数据库中数据的一致性,数据的操纵应当是离散的成组的逻辑单元:当它全部完成时,数据的一致性可以保持,而当这个单元中的一部分操作失败,整个事务应全部视为错误,所有从起始点以后的操作应全部回退到开始状态

事务的操作:先定义开始一个事务,然后对数据作修改操作,这时如果提交(COMMIT),这些修改就永久地保存下来,如果回退(ROLLBACK),数据库管理系统将放弃所作的所有修改而回到开始事务时的状态

##### 事务的四大特性？ACID

**数据库事务有四大特性: ACID --> 原子性 一致性 隔离性 持久性**

* **原子性(Atomicity)**

指事务在逻辑上是不可分割的操作单元，所有的语句要么都执行成功，要么都执行失败并进行撤销。

* **一致性(Consistency)**

从一个状态转换为另外一个状态

事务应确保数据库的状态从一个一致状态转变为另一个一致状态。一致状态的含义是数据库中的数据应满足完整性约束

* **隔离性(Isolation)**

隔离性是针对并发而言。所谓的并发是指**数据库服务器同时执行多个事务，如果在执行的过程中不采取有效的专门控制机制，并发事务之间会发生相互干扰**

隔离性就是隔离并发运行的多个事务避免产生相互影响

事务的隔离性是指一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰

* **持久性(Durability)**

事务一旦提交对数据的修改就是持久性的，数据已经从内存转移到了外部服务器上，并执行了固化的步骤

##### 在数据库中由并发导致的各种问题包含哪些?

**由于数据库的隔离性导致的并发问题包括以下内容**

对于同时运行的多个事务，当这些事务访问数据库中相同的数据，如果没有采取必要的隔离机制将会导致各种并发问题。

* **脏读**

对于两个事务T1,T2 。 T1读取了已经被T2更新但是还没有提交的字段之后，如果T2发生了回滚，T1读取的内容就是无效的。

* **不可重复读**

对于两个事务T1，T2。T1读了一个字段，然后T2更新了该字段之后，T1再次读取同一个字段，两次读取的值就是不同的。

* **幻读(虚度)**

对于两个事务T1,T2 T1从一个表中读取一个字段，然后T2在该表中插入一个新的行之后，如果T1再次读取这个表发现数据行数变多。

* **丢失更新**

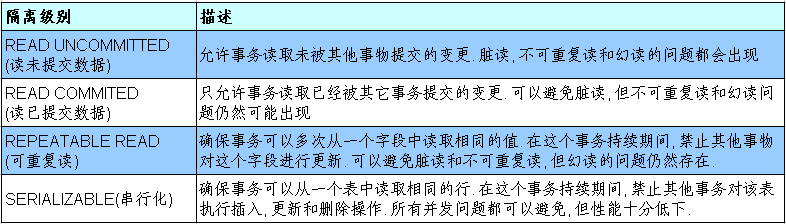
是第一次提取的内容被第二次提交的内容进行了覆盖

**数据库事务的隔离性: 数据库系统必须具有隔离并发运行各个事务的能力, 使它们不会相互影响, 避免各种并发问题.**

**一个事务与其他事务隔离的程度称为隔离级别. 数据库规定了多种事务隔离级别, 不同隔离级别对应不同的干扰程度, 隔离级别越高, 数据一致性就越好, 但并发性越弱**

##### 为了解决以上问题设计了哪些隔离级别详细解释?

为了解决并发问题 数据库定义了四种隔离级别。隔离级别越高效率越慢，根据业务选择一个合适的隔离级别，在性能和安全上做平衡。



* **read\_uncommitted：读未提交的数据**

允许事务读取未被其他事务提交的变更，脏读，不可重复读，（虚读）幻读问题都会出现

* **read\_commited ：读已提交的数据**

只允许事务读取已经被其他事务提交的变更，可以避免脏读，但是不可重复读和幻读依然存在

* **Repeatableread：可重复读**

确保事务可以多次从一个字段中读取相同的值，这个事务存在期间，禁止其他事务对这个字段进行更新，可以避免脏读，不可重复读但是幻读依然存在。

* **Serializable ：串行化**

确保可以从一个表中读取相同的行，这个事务存在期间禁止其他事务对该表进行插入、更新和删除操作。所有的并发问题都可以避免，但是性能十分低下。

* 丢失更新：主要是使用乐观锁和悲观锁解决丢失更新。

**并不是所有的数据库都支持这四种隔离级别**

* Oracle支持两种隔离级别 read\_commited 、Serializable 默认的隔离级别是**read\_commited**
* Mysql支持四种隔离级别 mysql的默认隔离级别是**repeatableread**

**1 read uncommited**

**2 read commited**

**4 repeatable read**

**8 serializable**

### 15.Ajax的优势?实现方式？

**Ajax工作原理：一种不用刷新整个页面也能保持和服务器通信的技术**  
 相当于在客户端与服务端之间加了一个抽象层(Ajax引擎)，使用户请求和服务器响应异步化，并不是所有的请求都提交给服务器，像一些数据验证和数据处理都交给Ajax引擎来完成，只有确认需要向服务器读取新数据时才有Ajax引擎向服务器提交请求。

**Ajax优缺点：**

优点：

1.无刷新更新数据（不更新整个页面的前提下实现局部刷新）

Ajax最大的优点就是能在不刷新整个页面的情况下维持与服务器通信

2.异步与服务器通信

使用异步的方式与服务器通信，不打断用户的操作

3.前端与后端负载均衡（在多个用户同时访问页面时能够减轻服务端的负担）

将一些后端的工作移到前端，减少服务器与带宽的负担

4.基于规范被广泛支持

不需要下载浏览器插件或者小程序，但需要客户允许JavaScript在浏览器上执行。

5.界面与应用分离

Ajax使得界面与应用分离，也就是数据与呈现分离

缺点：

1.Ajax干掉了Back与History功能，即对浏览器机制的破坏

在动态更新页面的情况下，用户无法回到前一页的页面状态，因为浏览器仅能记忆历史纪录中的静态页面

2.安全问题

AJAX技术给用户带来很好的用户体验的同时也对IT企业带来了新的安全威胁，Ajax技术就如同对企业数据建立了一个直接通道。这使得开发者在不经意间会暴露比以前更多的数据和服务器逻辑。

3.对搜索引擎支持较弱

4.破坏程序的异常处理机制

5.违背URL与资源定位的初衷

6.不能很好地支持移动设备

7.客户端肥大，太多客户段代码造成开发上的成本

**实现方式有两种：**

方式1：普通代码编程，首先是要掌握ajax的开发步骤，从而进行编码

\* a.创建XMLHttpRequest对象

\* b.打开与服务器的链接

\* c.发送数据

\* d.接收服务器的响应数据

404 没找到页面(not found)

403 禁止访问(forbidden)

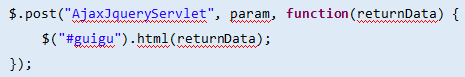
500 内部服务器出错(internal service error)

200 一切正常(ok)

304 没有被修改(not modified)(服务器返回304状态，表示源文件没有被修改 )

方式2：通过jquery框架提供的放ajax方法完成ajax请求





### 16.MVC模式的介绍（百度一下！！）

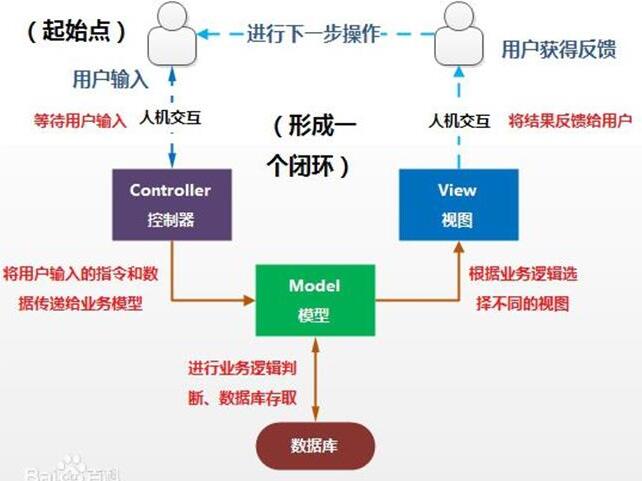
Model 模型：应用程序的主体部分，用于表示业务逻辑。

View 视图：应用程序中用户界面相关的部分，是用户看到并与之交互的界面。（接受用户输入，不作任何业务逻辑处理）

Controller 控制器：用于根据用户的输入，控制用户界面数据显示，更新 Model 对象状态。

MVC 模式的出现不仅实现了功能模块和显示模块的分离，同时还提够了应用系统的可维护、可扩展性、可移植性、和组建的可复用性。一个Controller 可以管理多个 View ，当View 发生改变时，可以随意 的更换。

MVC的处理过程：首先控制器接受用户的请求，调用相应的模型来进行业务处理，并返回数据给控制器。控制器调用相应的视图来显示处理的结果。并通过视图呈现给用户。



### 17.过滤器和拦截器两者的应用和区别？

拦截器 ：拦截器就是拦截 Action 在 Action 执行先执行拦截器 在 Action 执行之后再回到拦截器 中执行相关的代码，是在面向切面编程的就是在你的 service 或者一个方法，前调用一 个方法，或者在方法后调用一个方法比如动态代理就是拦截器的简单实现，在你调用方法前打印 出字符串（或者做其它业务逻辑的操作），也可以在你调用方法后打印出字符串，甚至在你抛出 异常的时候做业务逻辑的操作。

过滤器：是在 java web 中，你传入的 request,response 提前过滤掉一些信息，或者提前设置一 些参数，然后再传入 servlet 或者 struts 的 action 进行业务逻辑，比如过滤掉非法 url（不是 login.action 的地址请求，如果用户没有登陆都过滤掉）,或者在传入 servlet 或者 struts 的 action 前统一设置字符集，或者去除掉一些非法字符.

区别：

1、拦截器是基于 java 的反射机制的，而过滤器是基于函数回调

2、过滤器依赖于 servlet 容器，而拦截器不依赖于 servlet 容器

3、拦截器只能对 action 请求起作用，而过滤器则可以对几乎所有的请求起作用

4、拦截器可以访问 action 上下文、值栈里的对象，而过滤器不能

5、在 action 的生命周期中，拦截器可以多次被调用，而过滤器只在容器初始化时调用一次

两者的应用场景有所不同：

拦截器可以用于用户强制登录、用户权限管理、日志记录、系统性能监控

过滤器可以用于解决全站乱码问题、防盗链（图片、网址）、脏话过滤、html转义（防止js脚本攻击）

## 组别3

### 1.jdk1.8的新特性：函数式接口、Lambda表达式

Lambda表达式是类型也被称为目标类型，Lambda表达式目标类型必须是函数式接口，所谓的函数式接口是指只有一个抽象方法的接口。函数式接口代表只包含一个抽象方法的接口，函数式接口可以包含多个默认方法，类方法，但只能声明一个抽象方法。

Java8专门为函数式接口提供了一个注解**@FunctionalInterface**，该注解通常放在定义接口的前方，该注解对程序没有任何作用，它仅仅是告诉编译器更加严格的检查当前接口是否是函数式接口。

Lambda表达式实现的是匿名内部类的替代，只能实现特定的函数式接口， 也就意味着Lambda表达式遵循以下两个限制

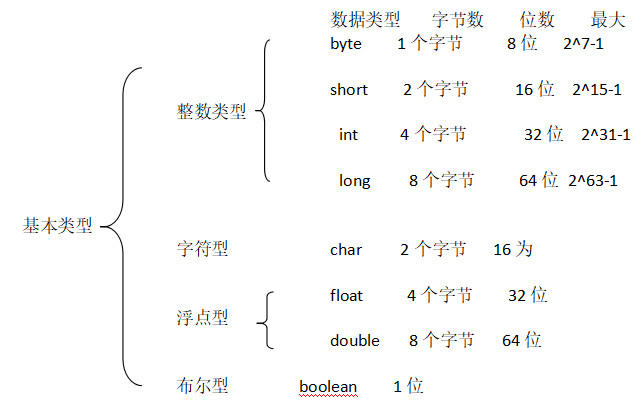
* **Lambda表达式目标类型必须是明确的函数式接口**
* **Lambda表达式只能为函数式接口创建对象**

为了保证Lambda表达式目标类型是一个明确的函数式接口，可以有三种常见的方式

* 将Lambda表达式赋值给函数式接口的变量
* 将Lambda表达式作为函数式接口的形参传递给某个方法
* 使用函数式接口对Lambda表达式进行强制类型转换

### 2.8种基本数据类型？自动装箱、自动拆箱？

答：整型（byte、short、int、long）、字符型（char）、浮点型（float、double）、布尔型（boolean）



Java是面向对象的编程语言包含了八种基本数据类型。但是八种基本数据类型不支持面向对象的编程机制。基本数据类型不具备对象的特征。基本数据类型没有成员变量也没有方法，所以不方便使用。我们基本数据类型不能使用面向对象的一些特征，为了解决这些问题，定义了包装类。

在jdk1.5之后提供了自动拆箱和自动装箱功能。

* 所谓自动装箱 就是把一个基本数据类型直接赋值给对应的包装类
* 所谓的自动拆箱，是指把包装类直接赋值给基本数据类型。

### 3.怎么理解面向对象

面向过程：是一种以过程为中心的编程思想，面向过程也称之为面向记录编程思想，他们不支持丰富的面向对象的特征(比如继承、多态),并且他们不允许混合持久化状态和域逻辑，就是分析出问题所需的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用这就是面向过程。 C Delphi

面向过程是以事件驱动开发为指导。

面向对象: 把涉及到的对象抽离出来，然后考虑对象之间的关系构建整个项目，面向对象的设计是一种提供符号设计系统的面向对象的实现过程。，面向对象是以领域对象开发为指导。

举例说明:

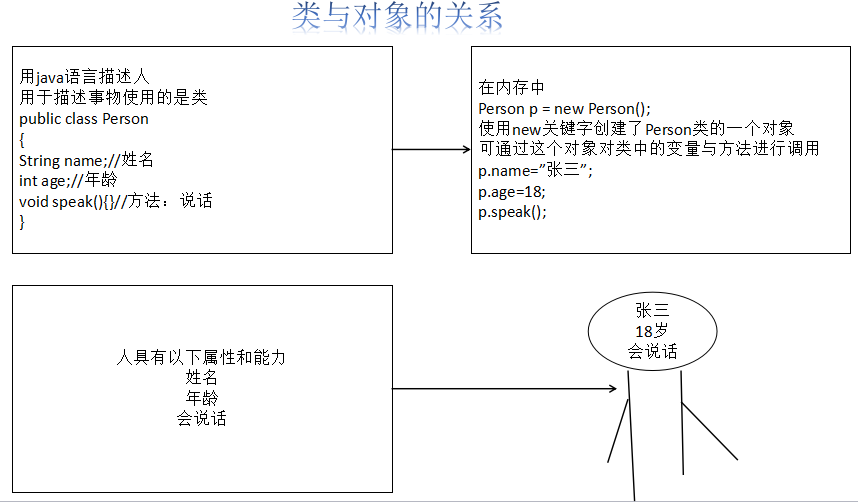
把大象装进冰箱

面向过程的思维:

1. 打开冰箱
2. 装进大象
3. 关闭冰箱

面向对象的思想：

1. 冰箱打开
2. 冰箱存储
3. 冰箱关闭



### 4.spring ioc注入，底层实现

IOC容器 Inverser Of Control反转控制 就是把原来在程序中创建HelloService对象的控制权限交给Spring管理，简单的说 就是HelloService对象控制器被反转到Spring框架内



IOC核心作用：将对象的创建交给spring进行管理

IOC操作：

IOC容器装配Bean的方式就是实例化该对象的方式，Spring提供了三种装配Bean的方式

* 使用类的构造函数进行初始化 (默认是无参的构造函数)
* 使用静态工厂方法：利用factory-method=属性指定静态工厂方法
* 使用实例化工厂方法 (工厂模式)

普通方式：通过上述三种方式之一装配Bean、编写核心配置文件applicationContext.xml、编写代码进行测试

注解方式：需要通过注解装配Bean、在核心配置文件中配置“自动扫描和相关的包”、编写代码进行测试

IOC底层原理实现使用技术：

* xml配置文件
* dom4j解析xml
* 工厂设计模式
* 反射



### 5.stream -- io相关

按照流的流向分为输入流和输出流

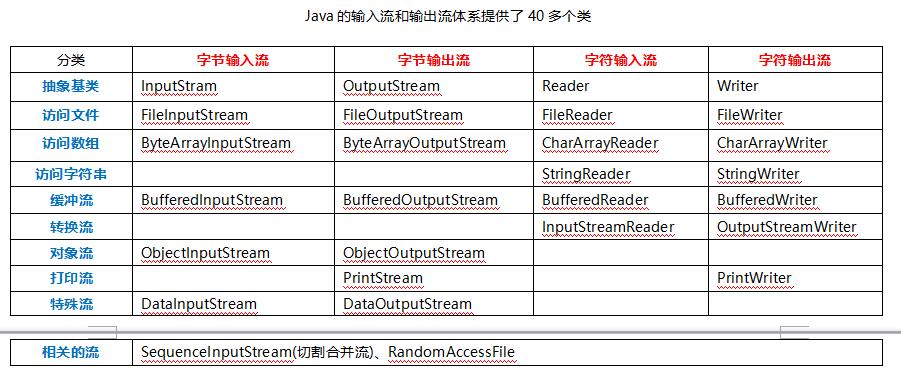
* 输入流：只能从流中读取数据，不能向其写入数据
* 输出流：只能向其写入数据，而不能从中读取数据

Java的输入流的基类主要是由**InputStream** 和**Reader**组成，输出流的基类主要是由**OutputStream**和**Writer**组成

字节流和字符流的用法基本相同，**区别在于字节流和字符流操作的数据单元不同**

**字节流操作的数据单元是8位字节，而字符流操作的数据单元是16位字节**

* 字节流主要是由InputStream和OutputStream组成
* 字符流主要是由Reader和Writer组成



### 6.数据库索引

索引是用来提高查询效率的数据库对象，**使用索引可以快速定位数据**，减少磁盘操作

* 一种独立于表模式的对象，可以存储在于表不同的磁盘表空间
* 索引被删除或者被破坏不会对表产生影响，只会影响查询速度
* 索引一旦创建完毕，oracle管理系统会对其自动维护，而且oracle管理系统决定何时使用什么索引，用户不需要在查询语句中指定使用哪个索引
* 在删除一个表的时候所有基于该表的索引都会被自动删除
* 通过指针加速oracle 服务器的查询速度
* 通过快速定位的方法检索磁盘I/O

索引分类

* 唯一性索引和非唯一性索引

按照索引字段是否允许出现重复

* 单字段索引和联合索引

按照字段基于索引的数目

* 普通索引和函数索引

按照索引锁基于的是普通字段还是复合表达式

* B树索引和位图索引

按照索引的数据结构划分B树索引(B\*Tree) 和位图索引(Bitmap)

索引使用的优点？效果？

*/\**

*创建索引的目的是加快查询速度，减少磁盘的操作*

*只需要创建相应的索引即可，系统会自行管理、调用相应的索引*

*\*/*

*/\**

*测试:*

*下述哪些约束会在创建的时候自动创建索引*

*a.primary key b.unique*

*c.not null d.foreign key*

*e.check*

*--预测结果：****primary key 、 unique***

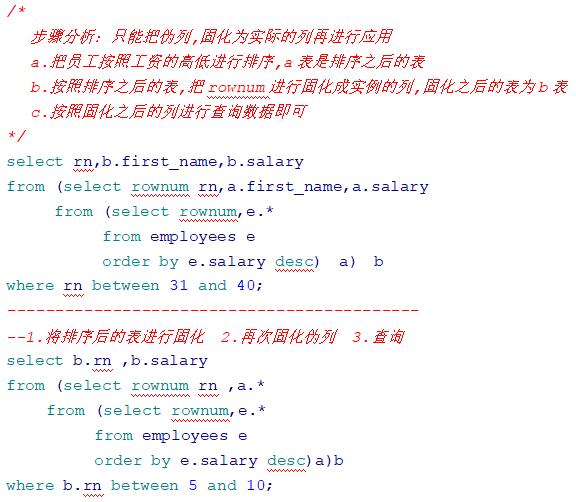
*\*/*

### 7.项目中的权限管理如何实现

如果是超市管理系统则是通过创建权限表记录每个用户对应角色所拥有的权限，从而在相关功能组件上添加权限判断，根据不同的权限开发不同的功能

在基于javaweb开发的电子商城项目中，主要是创建用户角色、用户权限、角色权限表用以控制用户权限，一个用户可以对应多个角色，一个角色又可以对应多个权限，通过角色权限表解决用户与权限之间多对多的关系。在用户角色中根据需求创建不同的用户角色，在用户权限中根据系统功能的分类划分等级，按照不同的模块提供相应的权限编号和url（用以访问该功能模块的url），在角色权限表中记录每个用户对应角色和权限信息，从而拦截器中通过判断语句完成用户的权限设置。

### 8.oracle数据库分页如何实现



### 9.谈谈对redis、activemq、zookeeper的简单了解

redis是一个非关系型数据库（nosql），以键值形式存储数据

非关系型数据库具有如下特点：

a.数据模型比较简单

b.需要灵活性更强的IT系统

c.对数据库性能要求较高

d.不需要高度的数据一致性

e.对于给定key，比较容易映射复杂值的环境

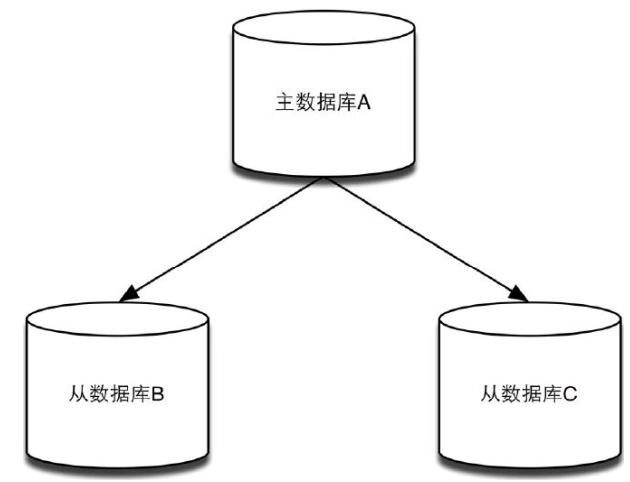
Redis提供了5种不同类型的数据结构，各式各样的问题都可以很自然地映射到这些数据结构上：Redis的数据结构致力于帮助用户解决问题，而不会像其他数据库那样，要求用户扭曲问题来适应数据库。除此之外，**通过复制、持久化（persistence）和客户端分片（client-side sharding）等特性，用户可以很方便地将Redis扩展成一个能够包含数百GB数据、每秒处理上百万次请求的系统。**

Redis 是一个开源（BSD许可）的内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。 它支持多种类型的数据结构，**如字符串（strings）， 散列（hashes）， 列表（lists）， 集合（sets）， 有序集合（sorted sets）**与范围查询,bitmaps，hyperloglogs和地理空间（geospatial） 索引半径查询。

Redis主要应用：**主从复制、哨兵监听**

通过持久化功能，Redis保证了即使在服务器重启的情况下也不会损失（或少量损失）数据。但是由于数据是存储在一台服务器上的，如果这台服务器的硬盘出现故障，也会导致数据丢失。为了避免单点故障，我们希望将数据库复制多个副本以部署在不同的服务器上，即使有一台服务器出现故障其他服务器依然可以继续提供服务。**这就要求当一台服务器上的数据库更新后，可以自动将更新的数据同步到其他服务器上，Redis提供了复制（replication）功能可以自动实现同步的过程。**

同步后的数据库分为两类，一类是主数据库（master），一类是从数据库（slave）。主数据库可以进行读写操作，当发生写操作时自动将数据同步给从数据库。而从数据库一般是只读的，并接受主数据库同步过来的数据。一个主数据库可以拥有多个从数据库，而一个从数据库只能拥有一个主数据库。如下图所示。



有了主从复制的实现 如果想对主服务器进行监控 ，在redis中提供了一个哨兵监控机制。

主服务器是用来写数据，同时把数据同步到其他的从服务器中

从服务器是用来进行读取数据

主服务器如果挂掉。通过哨兵监测机制可以监测到Redis系统运行的状态，**可以启动多个哨兵去监控Redis的运行状态。**

**监控主服务器的状态，如果挂掉，则通过选举机制选举出新的主服务器。可以把从服务器切换为主服务器，实现自动切换。**

Activemq是Apache出品的开源消息总线，即消息队列。其中有两个比较重要的概念，即生产者和消费者。生产者是消息的发送者，消费者是消息接收者。Activemq的消息模式有点对点模式（Queue）和消息订阅模式（Topic）。

Zookeeper是一个高效的分布式协调服务，它暴露了一些公用服务，比如命名，配置，管理，同步控制，群组服务，配置维护、域名服务、分布式同步、组服务等。我们可以使用zk来实现，比如达成共识，集群管理， 选举操作等。

Zookeeper是一个高可用的分布式管理与协调框架，是基于ZAB算法(原子消息广播协议)的实现。该框架能很好的保证分布式环境中数据的一致性，也正是基于这样的特性使用的Zookeeper成为解决分布式一致性问题的利器。

Zookeeper应用场景

* 统一的命名服务
* 配置管理
* 集群管理
* 共享锁
* 队列管理

**Activemq结合zookeeper搭建集群**

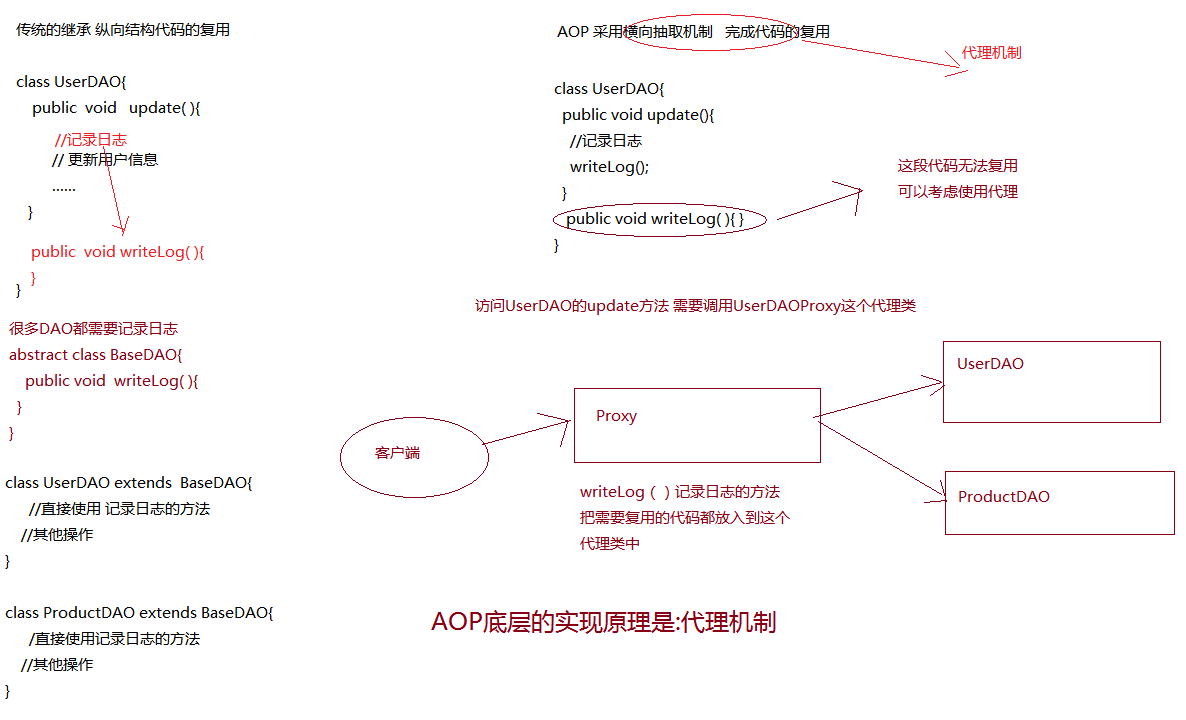
在测试连接过程中，断开当前activeMQ连接，可以看到其会自动连接其他的服务器发送数据。（如果还是失败，则可能是由于网络问题，因为页面都无法进入）

## 组别4

### 1.面向切面的编程AOP

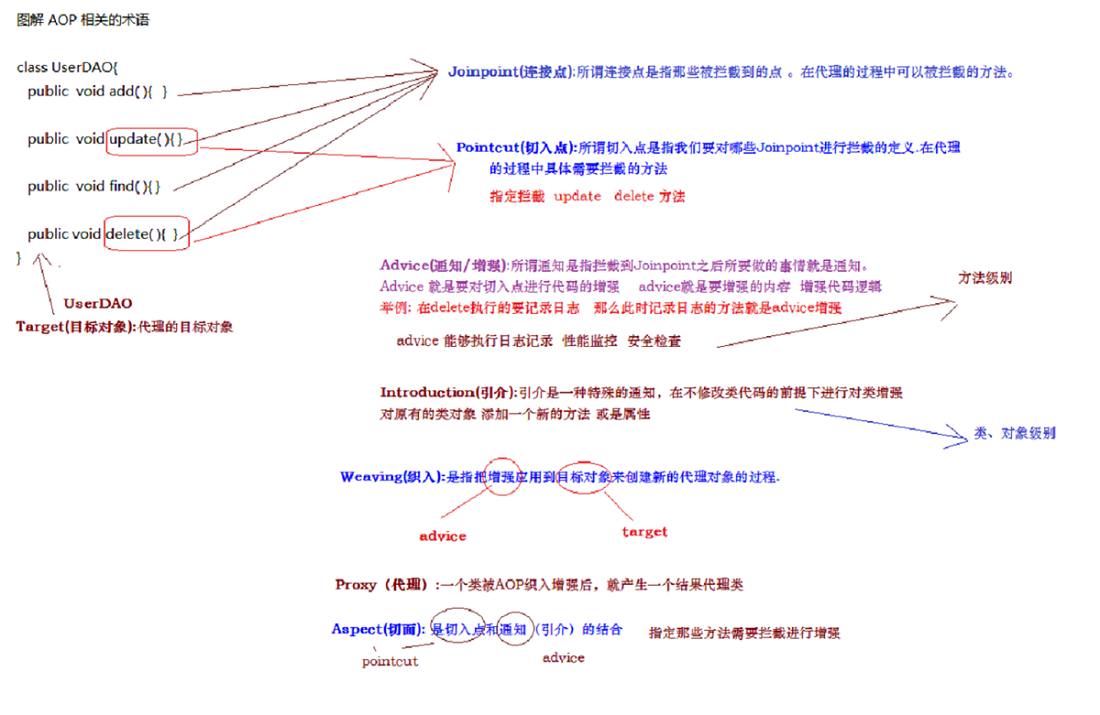
#### a.什么是AOP

* AOP Aspect Oriented Programing 面向切面编程
* **AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码**（性能监视、事务管理、安全检查、缓存）
* Spring AOP使用纯Java实现，不需要专门的编译过程和类加载器，在运行期通过代理方式向目标类织入增强代码
* AspectJ是一个基于Java语言的AOP框架，Spring2.0开始，Spring AOP引入对Aspect的支持，AspectJ扩展了Java语言，提供了一个专门的编译器，在编译时提供横向代码的织入



#### b.AOP相关的术语

* **Joinpoint(连接点)**: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):**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)**: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):**代理的目标对象
* **Weaving(织入):**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载期织入
* **Proxy（代理）:**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



#### c.AOP的五类Advice

AOP联盟为通知Advice定义了org.aopalliance.aop.Interface.Advice

Spring按照通知Advice在目标类方法的连接点位置，可以分为5类

* 前置通知 org.springframework.aop.**MethodBeforeAdvice**

在目标方法执行前实施增强

* 后置通知 org.springframework.aop.**AfterReturningAdvice**

在目标方法执行后实施增强

* 环绕通知 org.aopalliance.intercept.**MethodInterceptor**

在目标方法执行前后实施增强

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

在方法抛出异常后实施增强

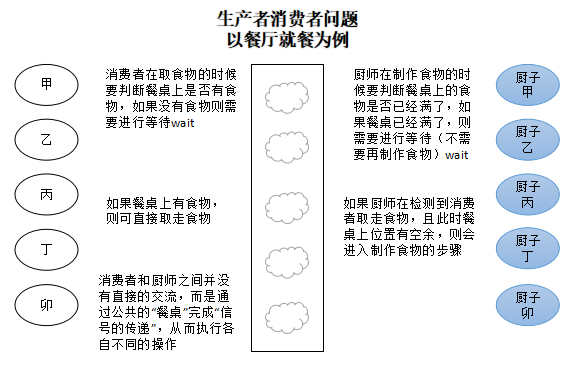
* 引介通知 org.springframework.aop**.IntroductionInterceptor**

在目标类中添加一些新的方法和属性

### 2.消费者和生产者如何实现？

生产者和消费者在同一时间段内共用同一存储空间，生产者向空间里生产数据，而消费者取走数据。

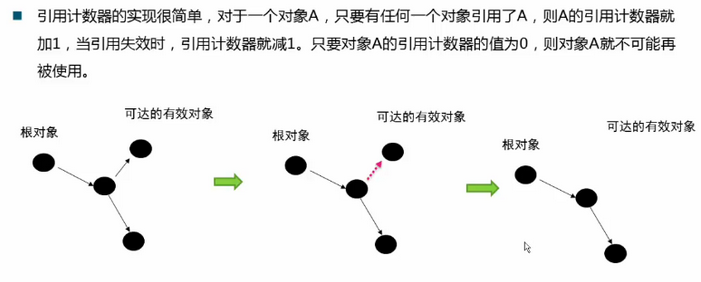
阻塞队列就相当于一个缓冲区，平衡了生产者和消费者的处理能力。这个阻塞队列就是用来给生产者和消费者解耦的。



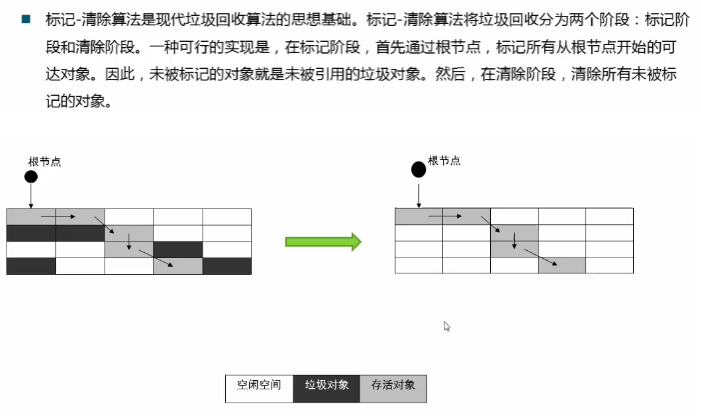
### 3.JVM垃圾回收是如何实现的？用了什么算法？

 Java的垃圾回收的java语言重要的功能之一，当程序创建对象，数组，或者引用对象，系统都会在内存中为其分配一个空间，对象就是保存在这个内存区域中，当这块内存区域不再被任何变量引用，这个内存就变为垃圾，等待垃圾回收机制进行回收。GC通过确定对象是否被活动对象引用来确定是否收集该对象。

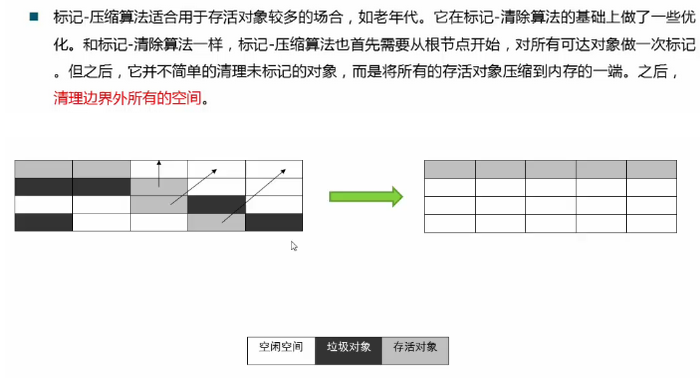
a.引用计数法（因为无法处理循环引用的情况，因此，在Java的垃圾回收器中没有使用这种算法）



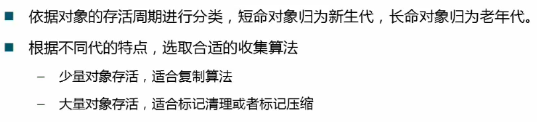
b.标记-清除法



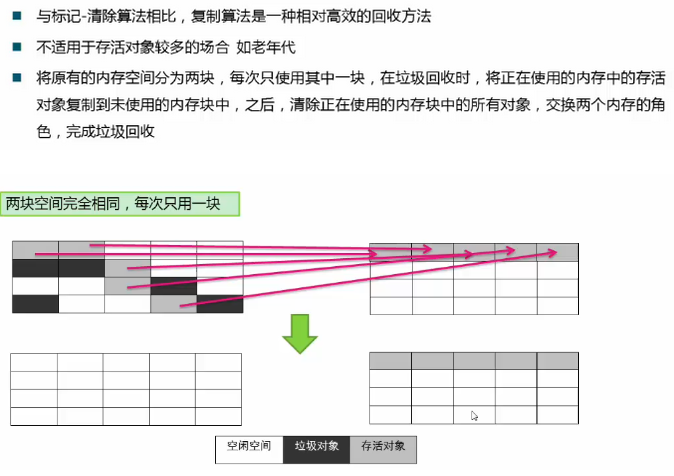
c.标记-压缩法



d.分代思想



e.复制算法



### 4.缓存的概念和实现

我是在mybatis学习时了解到有关缓存的相关概念

MyBatis 包含一个非常强大的查询缓存特性,它可以非常方便地配置和定制。MyBatis 3 中的缓存实现的很多改进都已经实现了,使得它更加强大而且易于配置。

默认情况下是没有开启缓存的,除了局部的 session 缓存,可以增强变现而且处理循环、依赖也是必须的。要开启二级缓存,你需要在你的 SQL 映射文件中添加一行代码: <cache/>



MyBatis提供查询缓存用于减轻数据压力，提供数据库性能。

MyBatis提供了两级缓存，一级缓存和二级缓存：

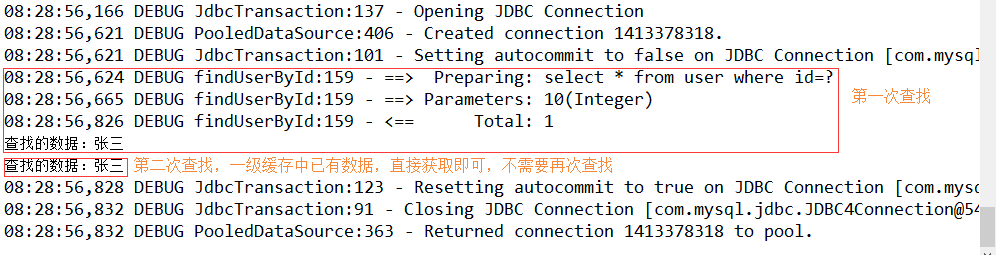
一级缓存是SqlSession级别的缓存，在操作数据库需要构建SqlSession对象。在这个对象中有一个数据结构HashMap，用于存储缓存数据，不同的sqlSession之间的缓存数据是互不影响的

二级缓存是mapper级别的缓存，多个sqlSession之间可以操作通过一个Mapper的sql语句，多个sqlSession可以共用二级缓存，二级缓存是跨SqlSession的

如果缓存中有的数据直接从缓存中操作，不需要从数据库获取，大大提高系统的性能

下述通过id查找用户的案例简单了解一级缓存和二级缓存

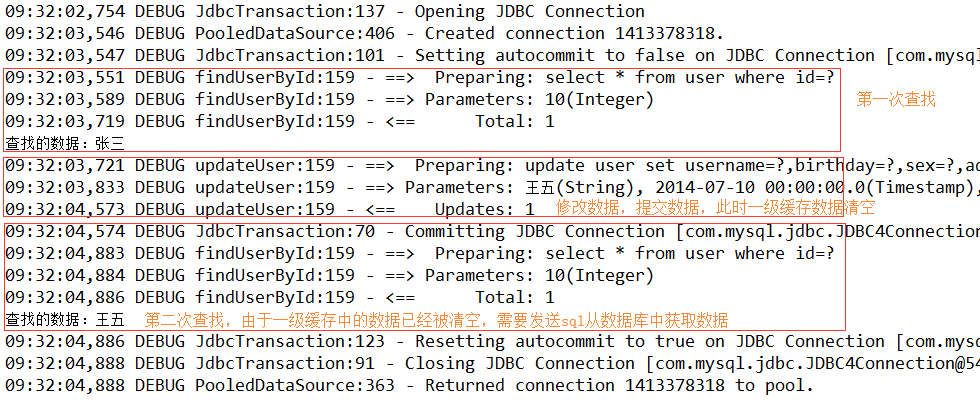
* **一级缓存测试：**



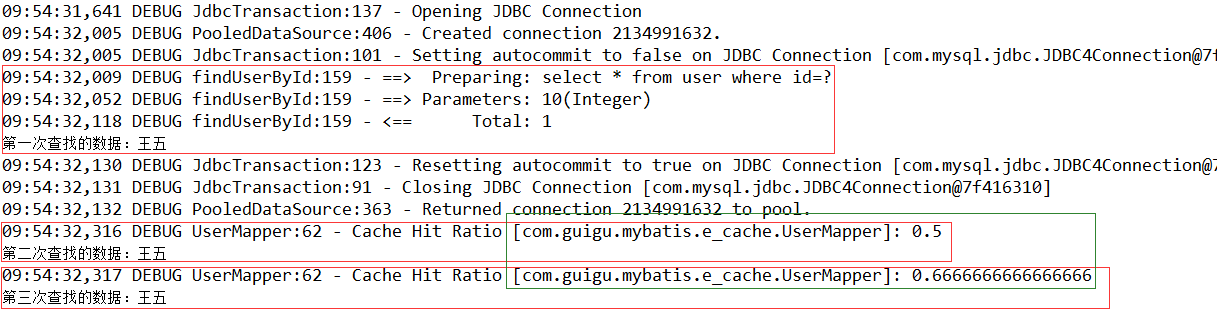
此处只考虑一级缓存，暂时不考虑二级缓存。第一次查找的时候，先查询缓存中的数据，一级缓存中没有相关的数据，需要发送sql从数据库中查找获取相应的用户信息，但完成查找之后会在一级缓存中存入已查找过的用户信息

在第二次查找的时候，从缓存中获取数据，由于查找的是相同的数据，此时一级缓存中已经存在相关的用户信息，因此可以直接获取，而不需要发送sql进行查找，从而显示上述测试结果

此外，在查询完数据之后对原有数据进行修改并提交，随后再次进行查询相同的数据，此时一级缓存中查询的数据已被清空，因此在第二次查找会发送sql语句从数据库中查找相关的数据信息，显示结果如下所示



* **二级缓存测试：**



第一次查找的时候先从一级缓存中获取数据，如果一级缓存中没有指定的数据，再从二级缓存中获取数据，二级缓存中如果没有指定的数据则发送sql从数据库中获取数据，查询完数据之后，随后关闭当前使用的sqlSession，此时会相应地清空一级缓存中的数据

第二次查找相同数据的时候，此时一级缓存中的数据已经被清空，开启新的sqlSession获取数据，此时会从二级缓存中获取数据，二级缓存中存在指定的数据则可直接获取，此时可看到指定的参数Cache Hit Ratio（缓存命中率）

### 5.sql注入是什么？如何防止sql注入？

/\*\*

\* sql注入攻击问题分析：

\* 用户登录问题：

\* 用户输入错误的用户名或密码也能够登录成功

\* 使用Statement对象执行sql语句 sql语句是通过拼接随后再执行的，

\* 因此数据库可能会辨识出不同的内容，从而导致二义性，即便是不存在的

\* 用户名或密码也能够成功通过检测，得到错误的信息

\* 问题解决：

\* 使用PreparedStatement进行优化 PreparedStatement会对sql语句进行预编译

\* 因此传入的username、password是直接作为整个内容传入，

\* 而非通过拼接进行操作，因此不会具有二义性

\*/

### 6.action是单实例还是多实例的，多用户访问同一个action时，全局变量怎么处理？

**action是单实例的。当多个用户访问一个请求的时候，服务器内存中只有一个与之对应的action类对象。**因为当服务器第一次加载struts的配置文件的时候,创建了一个Action后,每发送一个请求,服务器都会先去检索相应的范围内(request,session)是否存在这样一个action实例,如果存在,则使用这个实例,如果不存在,则创建一个action实例。

Spring下Controller全局变量引起的并发问题，多个用户同时请求访问该全局变量，该内容被所有请求共享，则会出现并发请求时变量内容被篡改的问题

解决方式：

方式1： 既然是全局变量惹的祸，那就将全局变量都编程局部变量，通过方法参数来传递。

方式2： jdk提供了java.lang.ThreadLocal,它为多线程并发提供了新思路。（当使用ThreadLocal维护变量时，ThreadLocal为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本）

### 7.线程的创建方式？有哪些线程池，有什么特点？

**线程有几种创建方式? 这几种创建方式有和优缺点有何区别?**

通过继承Thread类或者实现Runnable接口都可以实现多线程：

区别:

* 采用实现Runnable接口的方式：
* 线程仅仅是实现了Runnable接口，还可以继承其他类
* 在这种方式下，可以多个线程共享一个Target对象，非常适合多个线程共同处理同一份资源，从而将cpu 代码、数据分开，较好的体现了面向对象的思想
* 劣势:编程稍微复杂，如果要访问当前线程，必须使用Thread.currentThread()方法
* 采用继承Thread类
* 劣势：因为线程继承了Thread类，所有不能再继承其他父类
* 优势:编程比较简单，如果需要访问当前线程，直接使用this对象即可

推荐使用实现Runnable接口即可

线程池相关概念：

我对线程这一块的知识点比较薄弱，线程池作用：限制系统中执行线程的数量，减少资源的占用和浪费，至于线程池的特点我可能不太清楚

但在springmvc项目开发中了解了几种常见的数据库连接池的使用，Spring内置的数据库连接池、c3p0数据库连接池、DBCP数据库连接池、DRUID数据库连接池（阿里巴巴）（可以采用外部文件配置数据库，便于修改），完成四个参数的配置（用户名、密码、驱动driver、连接url）

## 实战演练（针对个人简历技术技能提出问题）

### 1.了解过的设计模式有哪些，分别简单叙述

在不同的项目应用情景中简单了解过多种设计模式的思想并加以运用，例如单例模式、装饰者模式、静态工厂方法、工厂模式、代理模式、观察者模式、适配器模式、策略模式

由于时间关系，我简单介绍其中几个模式的思想和应用：

* **单例模式**

单例模式（保证内存中只有一个对象）主要分为懒汉式、饿汉式、双重锁验证方式，

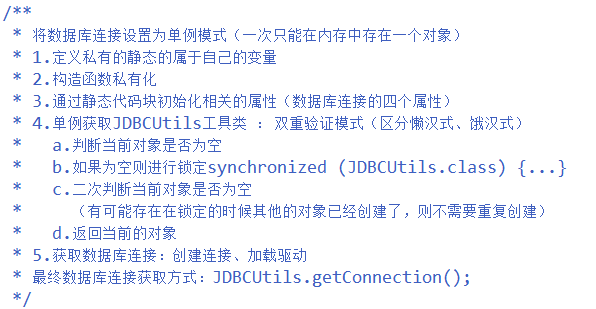
其实现的主演思想分为以下步骤：

a.创建一个静态的私有的全局的属于自己的变量

b.构造函数私有化

c.提供一个公有的方法返回该变量供外界使用

实际使用：



* **装饰者模式**

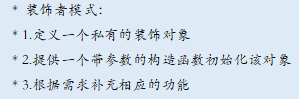
装饰者模式：动态的将职责附加到对象上，如果需要对类进行功能的扩展，装饰者模式能提供比继承更加良好的设计原则

应用场景：实际举例说明

在实际生活中，在最原始的开发模式中提供了最基础的功能，但随着业务的扩展，我们需要在原有功能的基础上去扩展其他的功能，因此需要借助装饰者模式完成这一操作

例如，最开始人们吃饭仅仅有“吃饭”，到后来的“边吃饭边聊天”，但随着生活水平的提高，人们开始追求更加卓越生活，进而发展到“吃饭、聊天、饭后活动”。这便是功能的一个扩展

但其实使用继承的方式也能够实现上述基本功能，但对比继承，我们选择用装饰者模式原因在于java中的继承是单继承模式，一个子类只能继承一个父类，如果需要完成其他的功能扩展可能需要通过实现接口进行操作，但装饰者模式有着良好的设计思想，能够很好地解决其中的不足，有着更强的扩展性（避免继承体系臃肿，降低了类与类之间的关系）



（装饰类与被装饰类需处于同一个体系，可以理解为实现共同的接口或继承同一个父类）

* **静态工厂方法、工厂模式**

在spring项目开发时，涉及到IOC容器装配Bean时使用到该思想

IOC容器装配Bean的方式就是实例化该对象的方式，Spring提供了三种装配Bean的方式

* 使用类的构造函数进行初始化 (默认是无参的构造函数)
* 使用静态工厂方法：利用factory-method=属性指定静态工厂方法
* 使用实例化工厂方法 (工厂模式)
* **代理模式**

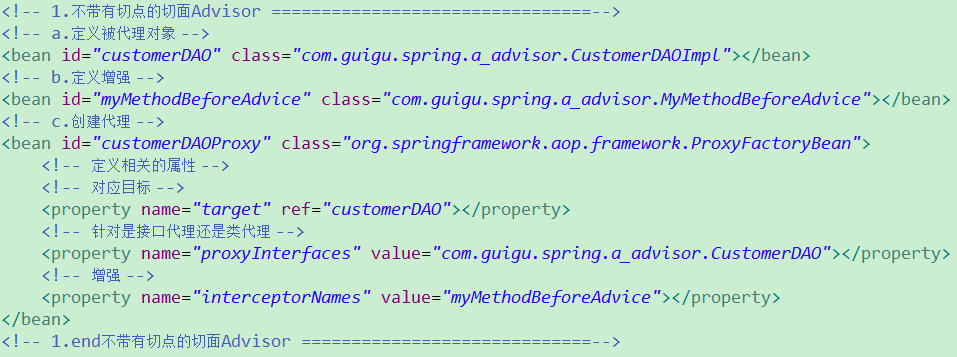
在spring项目开发中，涉及到AOP面向切面编程，在配置核心配置文件时采用了代理模式的思想：代理模式给某一个对象提供一个代理对象，并由代理对象控制对原对象的引用。通俗的来讲代理模式就是我们生活中常见的**中介**。

为什么要使用代理模式？

**中介隔离作用：**在某些情况下，一个客户类不想或者不能直接引用一个委托对象，而代理类对象可以在客户类和委托对象之间起到中介的作用，其特征是代理类和委托类实现相同的接口。

**开闭原则，增加功能：**代理类除了是客户类和委托类的中介之外，我们还可以通过给代理类增加额外的功能来扩展委托类的功能，这样做我们只需要修改代理类而不需要再修改委托类，符合代码设计的开闭原则。代理类主要负责为委托类预处理消息、过滤消息、把消息转发给委托类，以及事后对返回结果的处理等。代理类本身并不真正实现服务，而是同过调用委托类的相关方法，来提供特定的服务。真正的业务功能还是由委托类来实现，但是可以在业务功能执行的前后加入一些公共的服务。例如我们想给项目加入缓存、日志这些功能，我们就可以使用代理类来完成，而没必要打开已经封装好的委托类。

以下是简单参考：



* **观察者模式**

观察者模式，又可以称之为发布-订阅模式，观察者，顾名思义，就是一个监听者，类似监听器的存在，一旦被观察/监听的目标发生的情况，就会被监听者发现，这么想来目标发生情况到观察者知道情况，其实是由目标奖情况发送到观察者的。

观察者模式多用于实现**订阅功能**的场景，例如微博的订阅，当我们订阅了某个人的微博账号，当这个人发布了新的消息，就会通知我们。实际java应用中的**按钮监听**也应用了观察者模式思想

* **适配器模式（分为类适配器模式、对象适配器模式）**

适配器模式：将一个类的接口转换成客户希望的另外一个接口。Adapter 模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作，使用适配器模式是为了在面向接口编程中更好的复用。

在java基础中的GUI图形用户接口（swing、awt包）学习中，**如果需要实现某些监听，可以把多个接口定义用一个类来实现，然后在使用监听时，可以使用自定义的类，根据需求重写某些方法即可**

**即自定义适配器用一个类实现多个接口，在使用的时候只需要重写需要用到的方法即可**

* **策略模式**

策略模式定义了一系列的算法，并将每一个算法封装起来，而且使他们可以相互替换，让算法独立于使用它的客户而独立变化。

forEach 遍历、List、TreeSet 使用策略模式

### 2.在小组项目中承担什么角色？简单说明整个流程？

以近期小组项目为例进行叙述，在电子商城项目开发中主要完成基本的需求分析 、简单任务分配，此外完成数据库的基本设计以及应用系统的模块、功能设计并帮助完成基本模块（例如用户权限管理、用户登录注册、个人信息管理、订单管理、数据统计等），在最终项目整合中完成项目的基本整合和简单测试。

从最基本的主题确定到需求分析，我们主要是采用纸质、电子问卷调查的方式了解当下消费者的消费状态和消费心理，以及他们对电子商城的了解和建议，由于只是基于基本的设计，涉及的范围比较窄，可能某些方面还需要不断改进。此外，我们小组也通过浏览许多知名的购物网站并向老师咨询意见来进一步完善我们的项目设计。

在数据库设计方面，在基于之前项目的开发经验的基础上完成数据库的基本设计，在利用powerdesigner工具完成数据库建模。

确定主题、系统规划 -- 可行性分析、需求分析 -- 概念分析、设计 -- 逻辑和物理分析设计 -- 数据库建模

完成基本的项目设计，开始选定模板，根据组员的基本能力和建议完成初步的任务分配，便着手项目开发。在项目临检前几天完成项目的最终整合和基本测试

在测试这方面，除了在项目的开发过程中一些常用的测试技巧（例如控制台输出、debug模式），在软件测试课程上也了解了软件测试的一些基本内容（单元测试、继承测试、系统测试、回归测试），例如黑盒测试、白盒测试、功能测试等。

在这个项目开发过程中我们主要是基于MVC模式、SSM框架完成JAVA Web的项目开发，结合linux、redis、activeMQ、Zookeeper、FastDFS等互联网技术优化项目，并通过JXL完成数据的批量操作以及借助echarts完成数据的统计分析并显示数据视图。

由于条件的限制，在项目开发过程中会遇到各种各样的问题，还存在许多不足之处需要加以改进和完善！

**PS：在数据库设计会考虑什么？（数据表是如何设计的？）**

在数据表的设计方面我们会考虑在之前项目开发经验的基础上结合实际业务需求去设计数据表，需要分析业务模块及其相互间的关系，例如我们在设计商品表的时候除了考虑商品的基本属性，还有商品相应的分类信息、存储的仓库信息等内容，涉及到主外键相关联。但如果需要处理的数据过多，我们的这种设计方式稍显不合理，在之后有试着去了解有关“分库分表”的概念，但由于时间关系没有很深入的去进一步考究。

### 3.平时数据库优化做了哪些工作？

**查询优化**：可以使用索引加快查询速度

**数据批量操作进行优化**：

/\*\*

\* 批量sql操作处理：

\* 情景：向数据库中插入10万条数据(可以操作更多，时间、效率对比更加明显)

\* 1.普通方式：不使用任何批量操作，直接用Statement进行操作

\* 2.批量处理：使用批量处理，用Statement进行操作

\* 3.批量处理：使用批量处理，用PreparedStatement进行操作

\* 4.批量处理：使用批量处理，用PreparedStatement进行操作，且

\* 当积攒到一定的程度，统一执行一次批量处理操作

\* 测试的时候使用Junit工具进行测试

\* 在要测试的方法前加入@Test标识，点击方法名右键选择Run As JunitTest

\* 测试结果：效率--> 4>3>2>1

\*/

1) 用 PreparedStatement 一般来说比 Statement 性能高：一个 sql 发给服务器去执行，涉及 步骤：语法检查、语义分析，编译，缓存

2) 有外键约束会影响插入和删除性能，如果程序能够保证数据的完整性，那在设计数据库时就去掉外键。

3) 表中允许适当冗余，譬如，主题帖的回复数量和最后回复时间等

4) 使用子查询代替关联查询, 下面子查询语句要比第二条关联查询的效率高

1.select e.name,e.salarywhere e.managerid=(select id from employee where name='hzgg');

2. select e.name,e.salary,m.name,m.salary

fromemployees e,employees m where

e.managerid = m.id andm.name='hzgg';

5）sql 语句全部大写，特别是列名和表名都大写。特别是 sql 命令的缓存功能，更加需要统一大小写。

6）合理建立索引改善查询性能。ORACLE 使用了一个复杂的自平衡 B-tree 结构。一般通过索引查询数据比全表扫描要快。当ORACLE 找出执行查询和 Update语句的最佳路径时, ORACLE优化器将使用索引增加查询效率。

### 4.OOP编程

Java 是一个支持并发、基于类和面向对象的计算机编程语言。下面列出了面向对象软件开发的优点：

* 代码开发模块化，更易维护和修改
* 代码复用
* 增强代码的可靠性和灵活性
* 增加代码的可理解性
* 面向对象编程有很多重要的特性，比如：封装，继承，多态和抽象