# 专业知识：

常见语句：

Oracle分页：（开始位置，结束位置）

select \* from (select t.\*, rownum rn from (select \* from menu order by id desc) t where rownum < 10) where rn >=5

Mysql分页：（开始位置，每页条数）

select \* from music where id limit 5,5

删除重复记录：

delete from xin a where a.rowid != (

select max(b.rowid) from xin b

where a.name = b.name

);

## 单例：

### 懒汉式单例模式：

public class Singleton {

private static Singleton singleton = null;

private Singleton(){

}

synchronized public static Singleton.getInstance(){

if(singleton == null){

singleton = new Singleton();

return singleton;

}

}

### 饿汉式单例模式：

public class a{

private static final Singleton singleton = new Singleton();

public Singleton(){

}

public static Singleton.getInstance(){

return singleton ;

}

}

### 线程安全式单例模式：

public class Singleton{

private static Singleton singleton;

private Singleton(){

}

public static Singleton.getInstance(){

if(null == singleton){

synchronized(Singleton.class){

if(null == singleton){

singleton = new Singleton();

}

}

}

return singleton;

}

}

# 你是怎么搭建ssm框架的？

这个我们一般是先把spring和mabatis整合到一块，（1）先建立一个jdbc属性文件，一些像这个数据库的初始化连接大小，连接池的最大、最小空闲，获取连接最大等待时间写上。 然后再建一个applicationContext.xml文件进行整合配置，这里边我们先配置一下自动扫描配置文件，然后就是把jdbc属性文件引进来，然后去配置一下数据源basicDataSource,配置完数据源接着在配置会话工厂sqlSessionFactoryBean，配会话工厂的时候要把数据源引入到会话工厂里边，再有就是配置dao扫描器MapperScannerConfigurer，最后再配一个事务管理器DataSourceTransactionManager,（http://blog.csdn.net/chen\_pt/article/details/70159633事务配置详解）同样也要把数据源引进来，我们一般事务这一块用的注解，所以再配一下启用注解配置，（如果用的不是注解的形式的话，那就是把事务的通知织入到目标对象的连接点的相应位置）这样的话spring和mabatis整合基本上就配完了！



接下来我们该整合SpringMVC了，这个我们一般会建立一个spring-mvc.xml文件，然后先配置一下自动扫描控制器的配置，再配置一下SpringMVC启动注解的配置，再加上视图解析器InternalResourceViewResolver的配置，配上文件跳转的前后缀，其他的像这个文件上传CommonsMutipartResolver一般这个也配上能用到！这样的话SpringMVC基本上配完了！

最后的话就是配置web.xml了，这个的话基本上就是配置一下加载spring的配置文件、用监听器ContextLoaderListener去加载spring的这个配置文件，Spring字符编码过滤器CharacterEncodingFilter、SpringMVC的核心控制器DispatcherServlet,如果拦截请求配的是拦截所有请求的话，那就在配一下静态资源不被拦截的配置！这样的话SSM框架基本上就配置完成了！

# MySQL与Oracle的区别

1. Oracle是大型数据库而Mysql是中小型数据库，Oracle市场占有率达40%，Mysql只有20%左右，同时Mysql是开源的而Oracle价格非常高。

2. Oracle支持大并发，大访问量，是OLTP最好的工具。

3. 安装所用的空间差别也是很大的，Mysql安装完后才152M而Oracle有3G左右，且使用的时候Oracle占用特别大的内存空间和其他机器性能。

4.Oracle也Mysql操作上的一些区别

①主键 Mysql一般使用自动增长类型，在创建表时只要指定表的主键为auto increment,插入记录时，不需要再指定该记录的主键值，Mysql将自动增长；Oracle没有自动增长类型，主键一般使用的序列，插入记录时将序列号的下一个值付给该字段即可；只是ORM框架是只要是native主键生成策略即可。

②单引号的处理 MYSQL里可以用双引号包起字符串，ORACLE里只可以用单引号包起字符串。在插入和修改字符串前必须做单引号的替换：把所有出现的一个单引号替换成两个单引号。

③翻页的SQL语句的处理 MYSQL处理翻页的SQL语句比较简单，用LIMIT 开始位置, 记录个数；ORACLE处理翻页的SQL语句就比较繁琐了。每个结果集只有一个ROWNUM字段标明它的位置, 并且只能用ROWNUM<100, 不能用ROWNUM>80

④ 长字符串的处理 长字符串的处理ORACLE也有它特殊的地方。INSERT和UPDATE时最大可操作的字符串长度小于等于4000个单字节, 如果要插入更长的字符串, 请考虑字段用CLOB类型，方法借用ORACLE里自带的DBMS\_LOB程序包。插入修改记录前一定要做进行非空和长度判断，不能为空的字段值和超出长度字段值都应该提出警告,返回上次操作。

⑤空字符的处理 MYSQL的非空字段也有空的内容，ORACLE里定义了非空字段就不容许有空的内容。按MYSQL的NOT NULL来定义ORACLE表结构, 导数据的时候会产生错误。因此导数据时要对空字符进行判断，如果为NULL或空字符，需要把它改成一个空格的字符串。

⑥字符串的模糊比较 MYSQL里用 字段名 like '%字符串%',ORACLE里也可以用 字段名 like '%字符串%' 但这种方法不能使用索引, 速度不快。

⑦Oracle实现了ANSII SQL中大部分功能，如，事务的隔离级别、传播特性等而Mysql在这方面还是比较的若

# 谈谈你对Spring 的理解：

话术:

spring是一个开源框架一个对象容器，当项目启动读取web.xml自动通过监听器ContextLoaderListener 加载applicationContext.xml，读取spring的配置。

我认为spring实质上就是一个bean工厂，用来管理bean的生命周期和框架集成，当然了，spring的核心大家也都知道是IOC控制反转和AOP面向切面编程。

在我理解的IOC和AOP中。

我认为IOC就比如以前传统的java开发模式中，当我们需要一个对象时，我们会自己通过去new一个对象，而在Spring开发模式中，Spring容器使用了工厂模式为我们创建了所需要的对象，当我们使用的时候就不需要自己再去创建了，直接调用Spring为我们提供的对象即可，也就是IOC把对象的创建、初始化、销毁交给spring来管理，而不是由开发者控制，实现控制反转，IOC控制反转降低耦合性。

AOP面向切面编程 （对象是接口） ,在面向对象编程(OOP)思想中，我们将事物纵向抽象成一个个的对象 ,而在AOP面向切面编程中我们将一些具有横切逻辑的应用（，也就是将一个个对象某些类似的方面）横向抽象成一个切面，对这个切面进行一些权限验证，事务管理，记录日志这种操作的过程就是面向切面编程，

从Spring的角度看的话，AOP最大的用途就在于提供了事务管理的能力，Spring在你访问数据库之前，自动帮你开启事务，当你访问数据库结束之后，自动帮你提交或者回滚事务。（一句话概括：把切面中的通知织入到目标对象的连接点的相应位置，通过aop代理实现）

实现AOP功能采用的是代理技术（jdk针对接口代理；cglib针对类实现代理），客户端程序不再调用目标，而调用代理类，代理类与目标类对外具有相同的方法声明；

# Mybatis理解

## 1.mybatis是什么

mybatis是一个优秀的持久层框架，他对jdbc操作数据库的过程进行了封装，使开发着只用关注sql本身，不用去关注例如注册驱动，加载链接，得到statement，处理结果集等复杂的过程。

mybatis通过xml或者注解的方式，将要执行的各种sql语句配置起来，并通过Java对象和statement中的sql语句映射生成最终的sql语句，最后由mybatis框架执行sql语句，并将结果映射成Java对象返回。

## 2.工作原理

mybatis通过配置文件创建sqlsessionFactory，sqlsessionFactory根据配置文件(配置文件来源于两个方面:一个是xml，一个是Java中的注解)获取sqlSession。SQLSession包含了执行sql语句的所有方法，可以通过SQLSession直接运行映射的sql语句，完成对数据的增删改查和事物的提交工作，用完之后关闭SQLSession。

## 3.工作的流程

mapper接口：

接口的全类名是xml文件中namespace的值。

接口中的方法名是xml文件中mapperstatement的id值。

接口中方法的参数就是传递给sql的参数

mapper接口是没有实现类的，当调用一个方法时，接口的全类名定位一个配置文件，接口的方法名定位这个配置文件中的一个mapperStatment，所以说mapper的方法名是不能重载的，因为mapperStatment的保存和寻找策略。

mapper接口的工作原理是，mybatis会使用jdk动态代理方式为mapper接口创建proxy对象，代理对象会拦截接口中的方法，转而执行mapperStatment所代表的sql语句，然后将执行的结果封装返回。

## 4.mybatis解决的问题

1）.使用数据库连接池管理链接，避免了频繁创建了、关闭链接，浪费资源，影响性能的问题。

2）.用xml管理sql语句，让Java代码和sql语句分离，使得代码更易维护。

3）.解决了sql语句参数不定的问题。xml中可以通过where条件决定sql语句的条件参数。mybatis将Java对象映射到sql语句，通过statement的parameterType定义输入参数的类型。

4）.mybatis自动将结果集封装成Java对象， 通过statement的resultType定义输出的类型。避免了因sql变化，对结果集处理麻烦的问题。

## 5.mybatis配置延迟加载（懒加载）

概念:延迟加载即先从单表查询、需要时再从关联表去关联查询，大大提高数据库性能，因为查询单表要比关联查询多张表速度要快。

开发流程:

在mybatis的核心配置文件当中配置settings 设置懒加载属性,注意:在properties标签后面加入下面标签

<settings>

//打开延迟加载的开关

使用association标签完成一对一映射时，s<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>

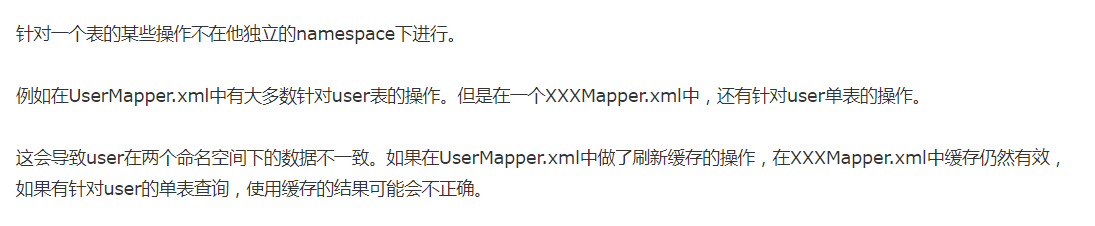
elect属性指定延迟加载需要执行的statement(必须指定该statement的唯一标识),这样即完成了延迟加载.

## 6.Mabatis的缓存

### MyBatis有两级缓存：

一级缓存：是SqlSession级别的缓存，是基于 HashMap 的本地缓存。不同的 SqlSession 之间的缓存数据区域互不影响。他的作用域是同一个SqlSession，当同一个 SqlSession 执行两次相同的 sql 语句时，第一次执行完后会将数据库中查询的数据写到缓存，第二次查询时直接从缓存获取不用去数据库查询。当 SqlSession 执行 insert、update、delete 操做并提交到数据库时，会清空缓存，保证缓存中的信息是最新的。

二级缓存：是mapper级别的缓存，作用域是mapper的同一个namespace，不同的sqlsession在同一个namespace下执行相同的sql语句，第一次查询的结果缓存到缓存中，第二次则可从缓存中获取数据。

二级缓存危害：

Mybatis默认开启一级缓存，没有开启二级缓存，二级缓存需要在xml配置中手动配置开启。

# Springmvc和struts2的对比？

|  |  |
| --- | --- |
| Struts2 | Springmvc |
| 基于Filter | 基于servlet |
| 传值通过定义私有属性，生成getter/setter方法 | 通过方法参数进行传值 |
| 一般使用xml配置 | 一般使用注解配置 |
| 一般一个请求就new一个action  因为定义私有属性是线程不安全的 | springmvc 一般使用一个单例的conroller  一般传值通过方法参数，不定义私有属性 |
| 自带的标签性能比较低，可以使用jstl替代 |  |
|  | springmvc可以支持rest风格的编程，传递路径参数，就是在url中反斜杠(/)后面定义参数  这时使用注解@PathVariable |
|  | 可以使用@ResponseBody注解返回json数据 |
| 有自己的拦截器机制 | 单独的AOP进行拦截 |
| 基于类 | 基于方法 |

# 谈谈对hibernate的理解

它是ORM思想的一个实现，对JDBC进行了很好的封装，它通过配置使JavaBean对象和数据库表之间进行映射，并提供对增、删、改、查便利的操作方法，同时支持事务处理，它对数据库记录还提供了缓存机制，提高效率，它可以使程序员不用书写SQL，也不用关心数据库SQL之间的差异，它通过方言对数据库进行了很好的适配。

1. Hiberante的主配置文件为hibernate.cfg.xml，其中定义了数据源、连接池、缓存、事务、表生成策略等配置

2. 通过\*.hbm.xml映射配置文件，定义了JavaBean对象和数据库表之间的映射关系，还定了对象和对象之间的映射关系，包括：一对一、多对一、一对多、多对多,hibernate.cfg.xml文件中需要登记每个hbm.xml文件。

3. 对于复杂查询和修改，Hiberante提供了Query对象进行支持，它可以使用HQL语句处理查询、修改、删除和分页。如果需要处理针对数据库相关的SQL，可以SQLQuery对象处理。

4. Hibernate可以集成到Spring中，通过Spring进行事务处理，事务的配置通常分两类，一种是AOP方式，一种是注释方式的声明式事务。

5. Hibernate对常用的缓存技术做了封装，可以轻松的调用各种缓存框架

6.使用Hibernate的基本流程是：配置Configuration对象、产生SessionFactory、创建session对象，启动事务，完成CRUD操作，提交事务，关闭session。

7.在应用Hibernate时，重点要了解Session的缓存原理，级联，延迟加载和hql查询。

4． 说下Hibernate的缓存机制

1. 内部缓存存在Hibernate中又叫一级缓存，属于应用事物级缓存

2. 二级缓存：

a) 应用及缓存

b) 分布式缓存

条件：数据不会被第三方修改、数据大小在可接受范围、数据更新频率低、同一数据被系统频繁使用、非 关键数据

c) 第三方缓存的实现

# 谈谈对Struts的理解

Struts框架是对MVC模式的封装，为开发者提供了MVC的3个逻辑组成部分，主要由ActionServlet · Action和struts-config.xml配置文件组成控制层，由ActionForm来承担模型层的功能，而struts 的视图层主要由jsp完成;

它处理一次请求的流程:由ActionServlet接受一个请求，然后依据配置文件来判断由哪个Action来处理和由哪个ActionForm来保存数据，再通过Action的返回值来判断应该由哪个jsp来展示数据

# 谈谈你对MVC的理解

MVC是Model—View—Controler的简称。即模型—视图—控制器。MVC是一种设计模式，它强制性的把应用程序的输入、处理和输出分开。

MVC中的模型、视图、控制器它们分别担负着不同的任务。

视图: 视图是用户看到并与之交互的界面。视图向用户显示相关的数据，并接受用户的输入。视图不进行任何业务逻辑处理。

模型: 模型表示业务数据和业务处理。相当于JavaBean。一个模型能为多个视图提供数据。这提高了应用程序的重用性

控制器: 当用户单击Web页面中的提交按钮时,控制器接受请求并调用相应的模型去处理请求。然后根据处理的结果调用相应的视图来显示处理的结果。

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。

# SSH和SSM框架区别

本人使用过SSH框架一段时间，最后在优化项目的时候决定使用SSM框架，这两种框架的一些浅显的区别我在这里说下：

1.SSH框架是Struts+Spring+Hibernate而SSM是指的Spring-MVC+Spring+MyBatis,光从这两个缩写就可以看出来，如果比较两者的区别的话主要比较Struct和Spring-MVC以及Hibernate和MyBatis：

1.Struts和Spring-MVC都是负责取值转发的，但是两者针对request的请求上面区别很大，Strust是针对一个Action类来进行请求的，即一个Action类对应于一个请求，所以类拦截，请求的数据类共享。而Spring-MVC则是针对于方法级别的请求的，也就是一个方法对应于一个请求，属于方法拦截，请求的数据方法不共享。

2.Spring-MVC的配置文件相对来说较为少，容易上手，可以加快软件开发的速度，亲身体验有效。

3.Spring-MVC的入口是Servlet级别的而Struct的级别是Filter级别的。

针对Hibernate和MyBitas对持久化类进行操作，他们之间的区别可以整理如下：

1.Hibernate是一种O/R关系型，即完成数据库表和持久化类之间的映射，而MyBitas是针对的SQL-Maping，个人理解是一种Hibernate把数据库给封装好以后，可以调用相应的数据库操作语句HQL，而MyBitas则是用的原始的数据库操作语句。

2.基于1中的原因，则Hibernate优化起来相对MyBitas较难。

3.MyBitas入门较快，而Hibernate掌握起来相对较难。

4.针对高级查询，Mybatis需要手动编写SQL语句，以及ResultMap。而Hibernate有良好的映射机制，开发者无需关心SQL的生成与结果映射，可以更专注于业务流程。

5.Hibernate数据库移植性很好，MyBatis的数据库移植性不好，不同的数据库需要写不同SQL。

## SpringMVC有哪些注解

@Controller声明此类是一个控制层类，后续dispatchServlet在分发请求时就可以请求到该类，@Controller的注解需要与 @RequestMapping结合使用才能访问到控制层的具体方法。

@Service声明此类是一个业务逻辑层类

@Repository用于标注数据访问组件

@RequestMapping声明方法的具体访问路径，@RequestMapping放在类上时相当于struts2的@nameSpace，就是在具体访问路径之前再加一层目录。

@RequestParam相当于request.getAttribute()方法，从request作用域对象中获取对应的参数值。

@PathVariable解析@RequestMapping路径中的变量参数，需要与@RequestMapping结合使用，整体使用感觉和@requestParam差别不大。

@ResponseBody 把返回的参数转换成xml或者json格式，主要与ajax请求结合使用。

@RequestBody 把request请求中的json参数转为对应的实体对象，要求必须在请求头信息中声明参数类型contentType:"application/json。

@Resource默认按照ByName（根据名字）自动注入，由J2EE提供，需要导入包javax.annotation.Resource。@Resource有两个重要的属性：name和type，而Spring将@Resource注解的name属性解析为bean的名字，而type属性则解析为bean的类型。所以，如果使用name属性，则使用byName的自动注入策略，而使用type属性时则使用byType自动注入策略。如果既不制定name也不制定type属性，这时将通过反射机制使用byName自动注入策略。

@Autowired为Spring提供的注解，需要导入包org.springframework.beans.factory.annotation.Autowired;只按照byType注入（根据类型注入）。@Autowired注解是按照类型（byType）装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许null值，可以设置它的required属性为false。如果我们想使用按照名称（byName）来装配，可以结合@Qualifier注解一起使用。

# 你对多线程是怎么理解的？

1、新建状态(New)：新创建了一个线程对象。

2、就绪状态(Runnable)：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态(Running)：就绪状态的线程获取了CPU，执行程序代码。

4、阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

(一)、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。

(二)、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

(三)、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

5、死亡状态(Dead)：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

实现线程的两种方式：

是继承Thread类或实现Runnable接口，但不管怎样，当new了这个对象后，线程就已经进入了初始状态

wait和sleep的区别：

线程访问：

锁池状态，之后等待锁释放，然后访问代码

wait

等待队列(释放资源)--->调用notify或者notifyall之后锁池状态--->( 等待锁释放)--->可运行状态--->运行状态---->访问代码

sleep,join

不释放资源-->结束后直接进入可运行状态--->运行状态---->访问代码

一个java控制台程序，默认运行两个线程，一个主线程，一个垃圾回收线程。

线程与进程的区别：

1.线程(Thread)与进程(Process)

进程定义的是应用程序与应用程序之间的边界，通常来说一个进程就代表一个与之对应的应用程序。不同的进程之间不能共享代码和数据空间，而同一进程的不同线程可以共享代码和数据空间。

2.一个进程可以包括若干个线程，同时创建多个线程来完成某项任务，便是多线程。

# 分布式+负载均衡话术

# Nginx知识点

在项目初期参与了项目架构的设计，项目需求文档的编写。项目中我们使用的是分布式框架ssm，在做这个项目的时候，考虑到网站前台的性能，承受更大的并发，前端采用的是nginx+tomcat集群进行的部署。

## 1、动静分离

这块主要是考虑到高并发的情况，以及减缓我们服务器的压力。

使用nginx做动静分离，静态资源通过绝对路径去访问，放在nginx服务器中；

动态资源通过url拼接字符串的方式去访问tomcat服务器，首先将静态内容文件复制到nginx的html目录下，然后修改nginx.conf文件，对*location*中的内容进行修改，

静态资源的内容使用一些通配符进行配置，动态资源使用location/转发到对应的服务器。

## 2、单点故障

我们知道使用服务器的过程中经常存在的一个问题就是单点故障，就是这台机器挂掉后，然后连接会一直连着，我们为了避免这种连接的发生，通常会设置超时时间，然后会避免单点故障，keepalived搭建双Nginx server集群，keepalived会隔一段时间向主服务器发送一个请求，查看主服务器的状态，超过时间没有得到响应，我们就会将接下来的连接发送到备用服务器上，同时通知相关人员去解决问题，修复好之后再将连接转发到住服务器上。

此外nginx还可以用来做反向代理、负载均衡以及解决session共享的问题，

## 3、反向代理

Nginx反向代理，将用户的请求转发到内部网络上的服务器上，然后将结果返回给Internet上请求连接的客户端，我们通过location使用proxy\_pass指定服务器的域名或者ip地址。

## 4、负载均衡

Nginx负载均衡，简单说就是配置三台nginx服务器，当用户访问nginx时刷新页面会出现三个不同的页面，nginx的upstream目前支持五种分配：第一种是轮询（默认的）按照请求时间逐一分配到不同的服务器上，如果后端服务器出现down，会自动剔除；第二种是权重（weight），配置权重值，主要是用在后端服务器性能不均的情况；第三种是ip\_hash，每个请求按照ip进行hash结果分配，这样可以使得每次的请求都在同一个服务器上，可以用来解决session共享的问题；第四种是url\_hash，按访问的url进行hash，使得每个url都重定向到统一个服务器。第五种是fair，是根据服务器响应时间进行分配的。

## 5、session共享

我们还用nginx解决session共享的问题，就是我们访问nginx的时候，可能会分配到不同的服务器上，造成session失效，这块的话有三种解决方案：第一种是session或凭据缓存到独立的服务器中，第二种是session或凭据保存到数据库中，第三种是使用nginx的ip\_hash，使得同一ip的请求都指定到同一个服务器上。保存到数据库的这种方式虽然可行，但是需要对数据库进行一些维护等等，比较麻烦，还会涉及到数据的增删改以及同步的问题，使用nginx的ip\_hash，同一ip指定同一个服务器，如果这台服务器出现故障或者问题，就会导致请求失败，此外我们一些文件的上传下载也会遇到这样的问题，可能我们上传的服务器在一个服务器上，下载的请求发送到另外的服务器上，就会出现类似session共享的问题，解决办法有两种：搭建独立的文件服务器或者使用文件压缩数据库。

使用主-从服务器的架构，当用户在主服务器上登录后，通过脚本或者守护进程的方式，将session信息传递到各个从服务器中，这样，用户访问其它的从服务器时，就可以读到session信息。

缺点：比如速度慢、不稳定等，另外，如果session信息传递是主->从单向的，会有一些风险，比如主服务器down了，其它服务器无法获得session信息

因此我们选择是nginx+redis+tomcat的方式，使用redis的缓存进行存储session，解决session共享的问题。

# Nginx+Tomcat+Apache

配置Nginx,打开conf下的nginx.conf文件进行配置,首先对tomcat和apache分配权重,分配方法是服务器的IP地址+tomcat/apache的端口号+权重比.然后配置tomcat和apache的位置,根据刚才分配权重比是自己所填写的负载均衡规则名找到服务器下的资源.

配置apache服务器,因为要实现nginx的负载均衡,所以要准备至少两个apache服务器,这里只说一个,另一个配置是一样的,首先配置conf文件夹下的httpd.conf文件,先把sevroot的路径写成自己apache所存放服务器的路径,然后把listen和本地的端口号改成一样的,例如:8089.最后把所存放的静态资源的路径填写进去就好了.然后打开conf文件夹下的extra文件夹下的httpd-ahssl.conf文件和httpd-ssl.conf文件,把里面的443端口改成和另外一台apache服务器下的这两个文件的端口改成不一样的,防止冲突.要注意的是两个服务器里面的所有端口都不能冲突.

配置tomcat服务器,因为要实现nginx的负载均衡,所以要准备至少两个tomcat服务器,这里只说一个,另一个配置是一样的,打开conf目录下的server.xml文件进行配置,首先配置服务器的SHUTDOWN的端口,再配置服务器的端口以及Connector标签的端口号,最后在host标签中把所要访问的项目的路径写进去就可以了,这里需要注意的是,两台服务器的端口号是不一致的,防止端口号冲突.

# Session和cookie：

## cookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、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，考虑到安全应当使用session。

3、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，考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议：  
   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

## Session和cookie的概念：

### Cookie：

Cookie是客户端技术，程序把每个用户的数据以cookie的形式写给用户各自的浏览器。当用户使用浏览器再去访问服务器中的web资源时，就会带着各自的数据去。这样，web资源处理的就是用户各自的数据了。

### Session：

Session是服务器端技术，利用这个技术，服务器在运行时可以为每一个用户的浏览器创建一个其独享的session对象，由于session为用户浏览器独享，所以用户在访问服务器的web资源时，可以把各自的数据放在各自的session中，当用户再去访问服务器中的其它web资源时，其它web资源再从用户各自的session中取出数据为用户服务。

# Java基础

# Map整理

#### **1、“你知道HashMap的工作原理吗？”** “你知道HashMap的get()方法的工作原理吗？”

　　HashMap是基于hashing的原理，我们使用put(key, value)存储对象到HashMap中，使用get(key)从HashMap中获取对象。当我们给put()方法传递键和值时，我们先对键调用hashCode()(public native int hashCode();

)方法，返回的hashCode用于找到bucket位置来储存Entry对象。”这里关键点在于指出，HashMap是在bucket中储存键对象和值对象，作为Map.Entry。

## 2、“当两个对象的hashcode相同会发生什么？”

　　因为hashcode相同，所以它们的bucket位置相同，‘碰撞’会发生。因为HashMap使用LinkedList存储对象，这个Entry(包含有键值对的Map.Entry对象)会存储在LinkedList中。(当向 HashMap 中添加 key-value 对，由其 key 的 hashCode() 返回值决定该 key-value 对（就是 Entry 对象）的存储位置。当两个 Entry 对象的 key 的 hashCode() 返回值相同时，将由 key 通过 eqauls() 比较值决定是采用覆盖行为（返回 true），还是产生 Entry 链（返回 false）。)

## 3、“如果两个键的hashcode相同，你如何获取值对象？”

当我们调用get()方法，HashMap会使用键对象的hashcode找到bucket位置，然后获取值对象。如果有两个值对象储存在同一个bucket，将会遍历LinkedList直到找到值对象。找到bucket位置之后，会调用keys.equals()方法去找到LinkedList中正确的节点，最终找到要找的值对象。(当程序通过 key 取出对应 value 时，系统只要先计算出该 key 的 hashCode() 返回值，在根据该 hashCode 返回值找出该 key 在 table 数组中的索引，然后取出该索引处的 Entry，最后返回该 key 对应的 value 即可。)

## 4/HashMap和Hashtable的区别

就HashMap与HashTable主要从三方面来说。

一.历史原因:Hashtable是基于陈旧的Dictionary类的，HashMap是Java 1.2引进的Map接口的一个实现

二.同步性:Hashtable是线程安全的。而HashMap是线程序不安全的，在只有一个线程访问的情况下，效率要高于Hashtable

三.HashMap允许将null作为一个entry的key或者value，而Hashtable不允许。

## Map的特征

（1）Map不是collection的子接口或者实现类。Map是一个接口。

Map的每个Entry都持有两个对象，也就是一个   
 键一个值，Map可能会持有相同的值对象但键对   
 象必须是唯一的。

TreeMap也通过Comparator或者Comparable  
 维护一个排序顺序。

Map里你可以拥有随意个null值但最多只能有  
 一个null值。

Map接口最流行的几个实现类是HashMap、  
 LinkedHashMap、Hashtable和TreeMap。   
 （HashMap、TreeMap最常用）

## HashMap和Hashtable的区别

#### （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

Hashtable的实现方法里面都添加了synchronized关键字来确保线程同步，因此相对而言HashMap性能会高一些，我们平时使用时若无特殊需求建议使用HashMap，在多线程环境下若使用HashMap需要使用Collections.synchronizedMap()方法来获取一个线程安全的集合（Collections.synchronizedMap()实现原理是Collections定义了一个SynchronizedMap的内部类，这个类实现了Map接口，在调用方法时使用synchronized来保证线程同步,当然了实际上操作的还是我们传入的HashMap实例，简单的说就是Collections.synchronizedMap()方法帮我们在操作HashMap时自动添加了synchronized来实现线程同步，类似的其它Collections.synchronizedXX方法也是类似原理）

#### （2）HashMap可以使用null作为key，而Hashtable则不允许null作为key

虽说HashMap支持null值作为key，不过建议还是尽量避免这样使用，因为一旦不小心使用了，若因此引发一些问题，排查起来很是费事   
HashMap以null作为key时，总是存储在table数组的第一个节点上

#### （3）HashMap是对Map接口的实现，HashTable实现了Map接口和Dictionary抽象类

#### （4）HashMap的初始容量为16，Hashtable初始容量为11，两者的填充因子默认都是0.75

HashMap扩容时是当前容量翻倍即:capacity\*2，Hashtable扩容时是容量翻倍+1即:capacity\*2+1

###### （5）两者计算hash的方法不同

Hashtable计算hash是直接使用key的hashcode对table数组的长度直接进行取模

（6）HashMap和Hashtable的底层实现都是数组+链表结构实现

#### （7）HashSet和HashMap、Hashtable的区别

除开HashMap和Hashtable外，还有一个hash集合HashSet，有所区别的是HashSet不是key value结构，仅仅是存储不重复的元素，相当于简化版的HashMap，只是包含HashMap中的key而已

通过查看源码也证实了这一点，HashSet内部就是使用HashMap实现，只不过HashSet里面的HashMap所有的value都是同一个Object而已，因此HashSet也是非线程安全的，至于HashSet和Hashtable的区别，HashSet就是个简化的HashMap的

# List、Map、Set三个接口，存取元素时，各有什么特点？

List与Set都是单列元素的集合，它们有一个功共同的父接口Collection。

## Set里面不允许有重复的元素

存元素：add方法有一个boolean的返回值，当集合中没有某个元素，此时add方法可成功加入该元素时，则返回true；当集合含有与某个元素equals相等的元素时，此时add方法无法加入该元素，返回结果为false。

取元素：没法说取第几个，只能以Iterator接口取得所有的元素，再逐一遍历各个元素。

## List表示有先后顺序的集合

存元素：多次调用add(Object)方法时，每次加入的对象按先来后到的顺序排序，也可以插队，即调用add(int index,Object)方法，就可以指定当前对象在集合中的存放位置。

取元素：方法1：Iterator接口取得所有，逐一遍历各个元素

方法2：调用get(index i)来明确说明取第几个。

Map是双列的集合，存放用put方法:put(obj key,obj value)，每次存储时，要存储一对key/value，不能存储重复的key，这个重复的规则也是按equals比较相等。

取元素：用get(Object key)方法根据key获得相应的value。

也可以获得所有的key的集合，还可以获得所有的value的集合，

还可以获得key和value组合成的Map.Entry对象的集合。

List以特定次序来持有元素，可有重复元素。Set 无法拥有重复元素,内部排序。Map 保存key-value值，value可多值。

## ArrayList和LinkList的区别

ArrayList与LinkedList都是List接口的实现类,因此都实现了List的所有未实现的方法,只是实现的方式有所不同；而List接口继承了Collection接口,

Collection接口又继承了Iterable接口,因此List同时拥有了Collection与Iterable接口的特性.

ArrayList实现了List接口,它是以数组的方式来实现的,数组的特性是可以使用索引的方式来快速定位对象的位置,因此对于**快速的随机取得对象的需求,**

**使用ArrayList实现执行效率上会比较好.**

LinkedList是采用链表的方式来实现List接口的,它本身有自己特定的方法，如: addFirst(),addLast(),getFirst(),removeFirst()等. 由于是采用链

表实现的,因此在进行insert和remove动作时在效率上要比ArrayList要好得多!适合用来实现Stack(堆栈)与Queue(队列),前者先进后出，后者是先进先出.

在删除可插入对象的动作时，为什么ArrayList的效率会比较低呢?

因为ArrayList是使用数组实现的,若要从数组中删除或插入某一个对象，需要移动后段的数组元素，从而会重新调整索引顺序,调整索引顺序会消耗一定的时间，

所以速度上就会比LinkedList要慢许多. 相反,LinkedList是使用链表实现的,若要从链表中删除或插入某一个对象,只需要改变前后对象的引用即可!

LinkedList 是一个继承于AbstractSequentialList的双向链表。它也可以被当作堆栈、队列或双端队列进行操作。

LinkedList 实现 List 接口，能对它进行队列操作。

LinkedList 实现 Deque 接口，即能将LinkedList当作双端队列使用。

LinkedList 实现了Cloneable接口，即覆盖了函数clone()，能克隆。

LinkedList 实现java.io.Serializable接口，这意味着LinkedList支持序列化，能通过序列化去传输。

LinkedList 是非同步的。

## ArrayList和Vectoy的区别

Vector是在Collections API之前就已经产生了的, 而ArrayList是在JDK1.2的时候才作为Collection framework的一部分引入的.

它们都是在内部用一个Obejct[]来存储元素的.

1. 线程安全

Vector是同步的, 而ArrayList不是.

因为Vector是同步的, 所以它是线程安全的.

同样, 因为Vecotr是同步的, 所以他需要额外的开销来维持同步锁, 所以它要比ArrayList要慢.(理论上来说)

当然, 如果你对ArrayList有偏好, 你也可以用Collection.synchronizedList(List)来得到一个线程安全的List.

2. 容量增长

Vector允许用户设置capacityIncrement这样在每次需要扩充数组的size的时候, Vector会尝试按照预先设置的capacityIncrement作为

增量来设置, 而ArrayList则会把数组的大小扩大一倍.

比如现在同样一个长度为10的Vector和ArrayList, 我们把Vector的capacityIncrement设为1

那么我们在插入第11个对象的时候, Vector会将长度变成11, 然后分配空间, 然后将对象添加进去, 而ArrayList则会分配20个对

象的空间, 然后将对象添加进去.

如果capacityIncrement设为0或者负值, Vector就会做和ArrayList一样, 每次都将数组大小扩大一倍.

3. 性能比较

刚刚在前面已经说过了, 由于Vector是同步的, 而ArrayList不是, 所以Vector的性能要比ArrayList要稍第一点, 用性能换安全嘛.

# Maven

对项目jar包的一个统一管理，可以非常方便的帮助我们看jar中的源代码。Maven编译src目录的内容会编译到target里边的的classes目录中，现在部署项目中的时候maven会从target/classes中把文件和class拷贝到tomcat webINFO中的classes里边

## 使用Maven的好处:

Maven中使用约定，约定java源代码代码必须放在哪个目录下，编译好的java代码又必须放到哪个目录下，这些目录都有明确的约定。

　　Maven的每一个动作都拥有一个生命周期，例如执行 mvn install 就可以自动执行编译，测试，打包等构建过程

　　只需要定义一个pom.xml,然后把源码放到默认的目录，Maven帮我们处理其他事情

使用Maven可以进行项目高度自动化构建，依赖管理(这是使用Maven最大的好处)，仓库管理。

# Freemarker

FreeMarker是java的免费模板引擎,主要用于MVC中的view层,生成html展示数据给客户端,可以完全替代jsp。 FreeMarker是一个模板引擎，一个基于模板生成

文本输出的通用工具，使用纯Java编写,模板中没有业务逻辑,外部java程序通过数据库操作等生成数据传入template中, 然后输出页面。它能够生成各

种文本：HTML、XML、JSP、RTF、Java源代码等等，而且不需要Servlet环境，并且可以从任何源载入模板，如本地文件、数据库等等。

**优点：** 1、不能编写java代码，可以实现严格的mvc分离

2、性能非常不错

3、对jsp标签支持良好

4、内置大量常用功能，使用非常方便

5、宏定义（类似jsp标签）非常方便

6、使用表达式语言

**缺点：** 1、不是官方标准

2、用户群体和第三方标签库没有jsp多

**选择freemarker的原因**：

1、性能。velocity应该是最好的，其次是jsp，普通的页面freemarker性能最差（虽然只是几毫秒到十几毫秒的差距）。但是在复杂页面上（包含大量判断、日期金额格式化）的页面上，freemarker的性能比使用tag和el的jsp好。

2、宏定义比jsp tag方便

3、内置大量常用功能。比如html过滤，日期金额格式化等等，使用非常方便

4、支持jsp标签

5、可以实现严格的mvc分离

# HTTP协议

HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写，是用于从万维网服务器传输超文本到本地浏览器的传送协议；它基于TCP/IP通信协议来传递数据，是一个属于应用层的面向对象的协议；HTTP协议工作于客户端-服务端架构为上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。Web服务器根据接收到的请求后，向客户端发送响应信息。

HTTP使用统一资源标识符（Uniform Resource Identifiers, URI）来传输数据和建立连接。URL是一种特殊类型的URI，包含了用于查找某个资源的足够的信息URL,全称是UniformResourceLocator, 中文叫统一资源定位符,是互联网上用来标识某一处资源的地址；比如我们的项目前后台的交互，前台ajax方法通过向后台controller层的接口发送get或者post请求，请求的路径就是url，通过url来对controller层的接口发送请求得到数据。

工作原理：HTTP协议定义Web客户端如何从Web服务器请求Web页面，以及服务器如何把Web页面传送给客户端。HTTP协议采用了请求/响应模型。

客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，

响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。

get请求和post请求的区别：

1、GET提交的数据会放在URL之后，以?分割URL和传输数据，参数之间以&相连，如EditPosts.aspx?name=test1&id=123456. POST方法是把提交的数据放在HTTP包的Body中.

2、GET提交的数据大小有限制（因为浏览器对URL的长度有限制），而POST方法提交的数据没有限制.

3、GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值。

4、GET方式提交数据，会带来安全问题，比如一个登录页面，通过GET方式提交数据时，用户名和密码将出现在URL上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码.

# IO流

## IO流概述

IO流简单来说就是Input和Output流，IO流主要是用来处理设备之间的数据传输，Java对于数据的操作都是通过流实现，而java用于操作流的对象都在IO包中。

分类：按操作数据分为：字节流和字符流。 如：Reader和InpurStream

按流向分：输入流和输出流。如：InputStream和OutputStream

IO流常用的基类：

InputStream ， OutputStream

字符流的抽象基类：

\* Reader ， Writer

由上面四个类派生的子类名称都是以其父类名作为子类的后缀：

如：FileReader和FileInputStream

## 字符流简介：

\*字符流中的对象融合了编码表，也就是系统默认的编码表。我们的系统一般都是GBK编码。

\*字符流只用来处理文本数据，字节流用来处理媒体数据。

\*数据最常见的表现方式是文件，字符流用于操作文件的子类一般是FileReader和FileWriter。1. 字符流的缓冲区：BufferedReader和BufferedWreiter

\* 缓冲区的出现时为了提高流的操作效率而出现的.

\* 需要被提高效率的流作为参数传递给缓冲区的构造函数

\* 在缓冲区中封装了一个数组，存入数据后一次取出

## 字节流概述：

1、字节流和字符流的基本操作是相同的，但是要想操作媒体流就需要用到字节流。

2、字节流因为操作的是字节，所以可以用来操作媒体文件。（媒体文件也是以字节存储的）

3、读写字节流：InputStream 输入流（读）和OutputStream 输出流（写）

4、字节流操作可以不用刷新流操作。

5、InputStream特有方法：

int available();//返回文件中的字节个数

注：可以利用此方法来指定读取方式中传入数组的长度，从而省去循环判断。但是如果文件较大，而虚拟机启动分配的默认内存一般为64M。当文件过大时，此数组长度所占内存空间就会溢出。所以，此方法慎用，当文件不大时，可以使用。 字节流缓冲区

\* 字节流缓冲区跟字符流缓冲区一样，也是为了提高效率。

# 谈谈final、finally和finalize的区别

1.final：如果一个类被final修饰，意味着该类不能派生出新的子类，不能作为父类被继承。

因此一个类不能被声明为abstract，又被声明为final。

将变量或方法声明为final。

可以保证他们在使用的时候不被改变。其初始化可以在两个地方：一是其定义的地方，

也就是在final变量在定义的时候就对其赋值；二是在构造函数中。这两个地方只能选其中的一个，

要么在定义的时候给值，要么在构造函数中给值。被声明为final的方法也只能使用，不能重写。

2.finally：在异常处理的时候，提供finally块来执行任何的清除操作。如果抛出一个异常，

那么相匹配的catch字句就会执行，然后控制就会进入finally块，前提是有finally块。

3.finalize：finalize是方法名，java技术允许使用finalize()方法在垃圾收集器将对象从内存中

清除出去之前做必要的清理工作。这个方法是在垃圾收集器确认一个对象没有被引用时对这个对象调用的。

它是在Object类中定义的，因此，所有的类都继承了它。子类覆盖finalize()方法已整理系统资源或者执

行其他清理工作。finalize()方法是在垃圾收集器删除对象之前对这个对象调用的

# 运行时异常与一般异常有何异同

异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

# error和exception有什么区别

error 表示恢复不是不可能但很困难的情况下的一种严重问题。比如说内存溢出。不可能指望程序能处理这样的情况。 exception 表示一种设计或实现问题。也就是说，它表示如果程序运行正常，从不会发生的情况

# Java中的异常处理机制的简单原理和应用

异常是指java程序运行时（非编译）所发生的非正常情况或错误，与现实生活中的事件很相似，现实生活中的事件可以包含事件发生的时间、地点、人物、情节等信息，可以用一个对象来表示，Java使用面向对象的方式来处理异常，它把程序中发生的每个异常也都分别封装到一个对象来表示的，该对象中包含有异常的信息。

Java对异常进行了分类，不同类型的异常分别用不同的Java类表示，所有异常的根类为java.lang.Throwable，Throwable下面又派生了两个子类：Error和Exception，Error 表示应用程序本身无法克服和恢复的一种严重问题，程序只有死的份了，例如，说内存溢出和线程死锁等系统问题。Exception表示程序还能够克服和恢复的问题，其中又分为系统异常和普通异常，系统异常是软件本身缺陷所导致的问题，也就是软件开发人员考虑不周所导致的问题，软件使用者无法克服和恢复这种问题，但在这种问题下还可以让软件系统继续运行或者让软件死掉，例如，数组脚本越界（ArrayIndexOutOfBoundsException），空指针异常（NullPointerException）、类转换异常（ClassCastException）；普通异常是运行环境的变化或异常所导致的问题，是用户能够克服的问题，例如，网络断线，硬盘空间不够，发生这样的异常后，程序不应该死掉。

java为系统异常和普通异常提供了不同的解决方案，编译器强制普通异常必须try..catch处理或用throws声明继续抛给上层调用方法处理，所以普通异常也称为checked异常，而系统异常可以处理也可以不处理，所以，编译器不强制用try..catch处理或用throws声明，所以系统异常也称为unchecked异常。

提示答题者：就按照三个级别去思考：虚拟机必须宕机的错误，程序可以死掉也可以不死掉的错误，程序不应该死掉的错误；

# 请写出你最常见到的5个runtime exception

这道题主要考你的代码量到底多大，如果你长期写代码的，应该经常都看到过一些系统方面的异常，你不一定真要回答出5个具体的系统异常，但你要能够说出什么是系统异常，以及几个系统异常就可以了，当然，这些异常完全用其英文名称来写是最好的，如果实在写不出，那就用中文吧，有总比没有强！

所谓系统异常，就是…..，它们都是RuntimeException的子类，在jdk doc中查RuntimeException类，就可以看到其所有的子类列表，也就是看到了所有的系统异常。我比较有印象的系统异常有：NullPointerException、ArrayIndexOutOfBoundsException、ClassCastException。

Java基础部分

描述一下JVM加载class文件的原理机制

1. 查找当前ClassLoader中是否有此class的类对象，有则返回

2. 若没有的话，向上递归所有的父ClassLoader中有无此class类对象，有则返回

3. 若还没有，查找BootstrapClassLoader中有无此class类对象，有则返回

4. 若还没有的话，使用findClass或resolveClass加载类对象

a. 读取class二进制文件

b. 根据字节数组生成Class对象

c. 缓存到当前ClassLoader中

JVM加载class对象是懒加载，按需加载

# 堆和栈

在Java中，内存分为两种，一种是栈内存，另一种就是堆内存。

一、什么是堆内存

堆内存是Java内存中的一种，它的作用是用于存储Java中的对象和数组，当我们new一个对象或者创建一个数组的时候，就会在堆内存中开辟一段空间给它，用于存放。

特点：第一点：堆其实可以类似的看做是管道，或者说是平时去排队买票的的情况差不多，所以堆内存的特点就是：先进先出，后进后出，也就是你先排队，好，你先买票。

第二点：堆可以动态地分配内存大小，生存期也不必事先告诉编译器，因为它是在运行时动态分配内存的，但缺点是，由于要在运行时动态分配内存，存取速度较慢。

对象在堆中的分配是由Java虚拟机的自动垃圾回收器来管理；

二、什么是栈内存

栈内存是Java的另一种内存，主要是用来执行程序用的，比如：基本类型的变量和对象的引用变量。

特点：第一点：栈内存就好像一个矿泉水瓶，像里面放入东西，那么先放入的沉入底部，所以它的特点是：先进后出，后进先出

第二点：存取速度比堆要快，仅次于寄存器，栈数据可以共享，但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性

栈内存的分配机制：栈内存可以称为一级缓存，由垃圾回收器自动回收

数据共享：举例说明；例子：

int a = 3;

int b = 3;

第一步处理：1.编译器先处理int a = 3;

2.创建变量a的引用

3.在栈中查找是否有3这个值

4.没有找到，将3存放，a指向3

第二步处理：1.处理b=3

2.创建变量b的引用

3.找到，直接赋值

第三步改变：接下来

a = 4；

同上方法

a的值改变，a指向4，b的值是不会发生改变的

三、堆和栈的区别

不同处：1.堆内存用来存放由new创建的对象和数组。

2.栈内存用来存放方法或者局部变量等

3.堆是先进先出，后进后出

4.栈是后进先出，先进后出

相同点：1.都是属于Java内存的一种

2.系统都会自动去回收它，但是对于堆内存一般开发人员会自动回收它

GC是什么?为什么要有GC

GC是垃圾回收的意思(Gabage Collection)，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。线程对象在没有终止前，即使没有任何引用，也不会被垃圾回收。

只能建议JVM回收内存，不能强制，可以使用System.gc()建议执行。

GC有三种方式，串行回收、并行回收、混合回收。

# 垃圾回收的优点和原理，并考虑2种回收机制

Java语言中一个显著的特点就是引入了垃圾回收机制，使C++程序员最头疼的内存管理的问题迎刃而解，它使得Java程序员在编写程序的时候不再需要考虑内存管理。由于有垃圾回收机制，Java中的对象不再有"作用域"的概念，只有对象的引用才有"作用域"。垃圾回收可以有效的防止内存泄露，更有效的使用内存。垃圾回收器通常是作为一个单独的低级别的线程运行，对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

1. 按时间轮询，把没有引用的对象进行回收

2. 按内存的使用量超不超过报警值进行回收

3. 按CPU空闲时间进行回收

4. 程序建议JVM进行垃圾回收

GC有三种方式，串行回收、并行回收、混合回收。

# 垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？

对于GC来说，当程序员创建对象时，GC就开始监控这个对象的地址、大小以及使用情况。通常，GC采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当GC确定一些对象为"不可达"时，GC就有责任回收这些内存空间。可以。程序员可以手动执行System.gc()，通知GC运行，但是Java语言规范并不保证GC一定会执行。

# StringBuffer和StringBuilder的区别

相同之处，这两类都是可变长的字符串存储类，都实现了CharSequence接口

1.类型不同，因为不是一个类，也没有继承关系，做参数时不能共用

2.StringBuffer为线程安全类，StringBuilder为线程非安全类

3.StringBuffer性能低，StringBuilder性能高，如果在局部优先使用StringBuilder

4.JDK在1.5之前，字符串相加使用StringBuffer对象，在1.5之后使用StringBuilder对象

# String和StringBuffer的区别

JAVA平台提供了两个类：String和StringBuffer，它们可以储存和操作字符串，即包含多个字符的字符数据。这个String类提供了数值不可改变的字符串。而这个StringBuffer类提供的字符串进行修改。当你知道字符数据要改变的时候你就可以使用StringBuffer。典型地，你可以使用StringBuffers来动态构造字符数据。另外，String实现了equals方法，new String(“abc”).equals(new String(“abc”)的结果为true,而StringBuffer没有实现equals方法，所以，new StringBuffer(“abc”).equals(new StringBuffer(“abc”)的结果为false。

接着要举一个具体的例子来说明，我们要把1到100的所有数字拼起来，组成一个串。

StringBuffer sbf = new StringBuffer();

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

{

sbf.append(i);

}

上面的代码效率很高，因为只创建了一个StringBuffer对象，而下面的代码效率很低，因为创建了101个对象。

String str = new String();

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

{

str = str + i;

}

在讲两者区别时，应把循环的次数搞成10000，然后用endTime-beginTime来比较两者执行的时间差异，最后还要讲讲StringBuilder与StringBuffer的区别。String覆盖了equals方法和hashCode方法，而StringBuffer没有覆盖equals方法和hashCode方法，所以，将StringBuffer对象存储进Java集合类中时会出现问题

# 线程的理解

线程：我们可以在计算机上运行各种计算机软件程序。每一个运行的程序可能包括多个独立运行的线程（Thread）。 是一份独立运行的程序，有自己专用的运行栈。有时被称为轻量级进程(Lightweight Process，LWP），是程序执行流的最小单元。

## 要注意的一些重要术语：

1) sychronized意味着在一次仅有一个线程能够更改Hashtable。就是说任何线程要更新Hashtable时要首先获得同步锁，其它线程要等到同步锁被释放之后才能再次获得同步锁更新Hashtable。

2) Fail-safe和iterator迭代器相关。如果某个集合对象创建了Iterator或者ListIterator，然后其它的线程试图“结构上”更改集合对象，将会抛出ConcurrentModificationException异常。但其它线程可以通过set()方法更改集合对象是允许的，因为这并没有从“结构上”更改集合。但是假如已经从结构上进行了更改，再调用set()方法，将会抛出IllegalArgumentException异常。

3) 结构上的更改指的是删除或者插入一个元素，这样会影响到map的结构

## 多线程有几种实现方法?同步有几种实现方法

多线程有两种实现方法，分别是继承Thread类与实现Runnable接口

同步的实现方面有五种，分别是synchronized、wait与notify、sleep、suspend、join

synchronized: 一直持有锁，直至执行结束

wait():使一个线程处于等待状态，并且释放所持有的对象的lock，需捕获异常。

sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，需捕获异常，不释放锁。

notify():唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且不是按优先级。

notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。

## 启动一个线程是用run()还是start()

启动一个线程是调用start()方法，使线程就绪状态，以后可以被调度为运行状态，一个线程必须关联一些具体的执行代码，run()方法是该线程所关联的执行代码。

## 当一个线程进入一个对象的一个synchronized方法后，其它线程是否可进入此对象的其它方法

如果其它方法中使用当前对象作为锁对象，则不能；

如果其它方法中没有使用当前对象作为锁对象，则能。

## 线程的基本概念、线程的基本状态以及状态之间的关系

在多任务操作系统中，为了提高CPU的利用率，可以使用多进程编程。但对进程通信比较困难，进程间数据不能共享，因此可以使用多线程编程。一个进程至少包含一个主入口线程。

单个CPU，在同一时间只能处理一个线程的数据，但是操作系统的任务调度非常快，人眼无法识别，感觉上是多个线程同时执行。有的线程可以已经用完CPU，正在作磁盘操作，此时并不使用CPU，可以让出CPU资源给其它线程使用，提高效率。

1、新建状态(New)：新创建了一个线程对象。

2、就绪状态(Runnable)：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态(Running)：就绪状态的线程获取了CPU，执行程序代码。

4、阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

　　(一)、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。

　　(二)、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

　　(三)、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

5、死亡状态(Dead)：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

实现线程的两种方式：是继承Thread类或实现Runnable接口，但不管怎样，当new了这个对象后，线程就已经进入了初始状态

wait和sleep的区别：

线程访问：

锁池状态，之后等待锁释放，然后访问代码

wait

等待队列(释放资源)--->调用notify或者notifyall之后锁池状态--->( 等待锁释放)--->可运行状态--->运行状态---->访问代码

sleep,join

不释放资源-->结束后直接进入可运行状态--->运行状态---->访问代码

一个java控制台程序，默认运行两个线程，一个主线程，一个垃圾回收线程。

## 线程与进程的区别：

1.线程(Thread)与进程(Process)

进程定义的是应用程序与应用程序之间的边界，通常来说一个进程就代表一个与之对应的应用程序。不同的进程之间不能共享代码和数据空间，而同一进程的不同线程可以共享代码和数据空间。

2.一个进程可以包括若干个线程，同时创建多个线程来完成某项任务，便是多线程。