# 那些你很冒险的梦:

## 1、List 和 Set 的区别

set --其中的值不允许重复，无序的数据结构，list --其中的值允许重复，因为其为有序的数据结构。

## 2、HashSet 是如何保证不重复的

hashset add的方法，内部其实调用的是hashmap的put方法。hashset的值作为hashmap的key存进去，value是一个伪值。因为hashmap的key不能重复，所以。。。

## 3、HashMap 是线程安全的吗，为什么不是线程安全的（最好画图说明多线程环境下不安全）?

不安全，使用 java.util.Hashtable 类，此类是线程安全的。

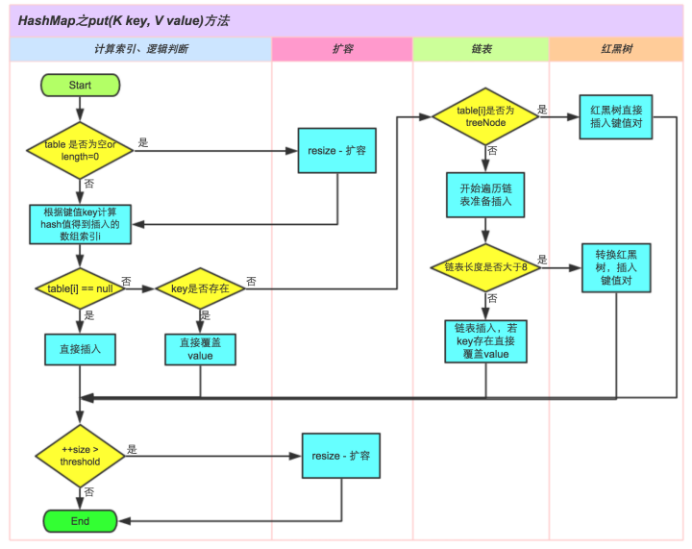
使用 java.util.concurrent.ConcurrentHashMap，此类是线程安全的。

使用 java.util.Collections.synchronizedMap() 方法包装 HashMap object，得到线程安全的Map，并在此Map上进行操作。

HashMap是一种无序的存储结构。HashMap的实现是假定元素是放在一个圆形的环上，

每次put进来的元素根据其hashCode计算该元素在圆环上索引，把该元素放到合适的位置。

红黑树是jdk1.8开始才有的。



## 4、HashMap 的扩容过程

## 5、HashMap 1.7 与 1.8 的 区别，说明 1.8 做了哪些优化，如何优化的？

## 6、final finally finalize

Final 用于定义变量：基本数据类型变量值不能变，引用类型引用不能变。

方法：方法不能被覆盖。

类：不可被继承。

finally是异常处理语句结构的一部分，表示总是执行。

return , break, continue try中有用到时，先执行finally。

finalize 是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，供垃圾收集时的其他资源回收，例如关闭文件等。

## 7、强引用 、软引用、 弱引用、虚引用

了解这个主要是为了解决OutOfMemory 内存溢出。

常规写的代码都属于强引用，其他的引用的在代码中可以通过关键字实现。softReference软引用，weakReference弱引用，phantomReference虚引用。

用了关键字之后，相当于给了虚拟机一个是否可以回收的依据，就可以通过编译的代码判断是否回收。（又涉及到垃圾回收机制）

例子：

public class Main {

public static void main(String[] args) {

SoftReference<String> sr = new SoftReference<String>(new String("hello"));

System.out.println(sr.get());

}

}

## 8、Java反射

//第一种方式获取Class对象

Student stu1 = new Student();//这一new 产生一个Student对象，一个Class对象。

Class stuClass = stu1.getClass();//获取Class对象

System.out.println(stuClass.getName());

//第二种方式获取Class对象

Class stuClass2 = Student.class;

System.out.println(stuClass == stuClass2);//判断第一种方式获取的Class对象和第二种方式获取的是否是同一个

//第三种方式获取Class对象

try {

Class stuClass3 = Class.forName("fanshe.Student");//注意此字符串必须是真实路径，就是带包名的类路径，包名.类名

System.out.println(stuClass3 == stuClass2);//判断三种方式是否获取的是同一个Class对象

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

//第一种方式获取Class对象

Student stu1 = new Student();//这一new 产生一个Student对象，一个Class对象。

Class stuClass = stu1.getClass();//获取Class对象

System.out.println(stuClass.getName());

//第二种方式获取Class对象

Class stuClass2 = Student.class;

System.out.println(stuClass == stuClass2);//判断第一种方式获取的Class对象和第二种方式获取的是否是同一个

//第三种方式获取Class对象

try {

Class stuClass3 = Class.forName("fanshe.Student");//注意此字符串必须是真实路径，就是带包名的类路径，包名.类名

System.out.println(stuClass3 == stuClass2);//判断三种方式是否获取的是同一个Class对象

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

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

## 注意：在运行期间，一个类，只有一个Class对象产生。

三种方式常用第三种，第一种对象都有了还要反射干什么。第二种需要导入类的包，依赖太强，不导包就抛编译错误。一般都第三种，一个字符串可以传入也可写在配置文件中等多种方法。

s

## 8.1、 object下的方法。



## 9、Arrays.sort 实现原理和 Collection 实现原理

Collection.sort()和arrays.sort()其实底层的实现都是调用的array.sort()。

## 10、LinkedHashMap的应用

HashMap是无序的，LinkedHashMap是有序的。

虽然LinkedHashMap增加了时间和空间上的开销，但是它通过维护一个额外的双向链表保证了迭代顺序。特别地，该迭代顺序可以是插入顺序，也可以是访问顺序。

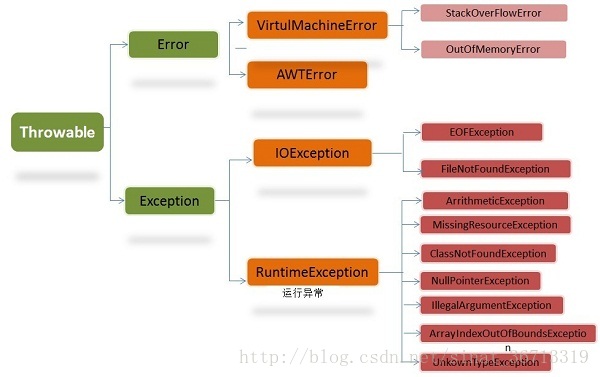
LinkedHashMap的默认实现是按插入顺序排序的。

## 11、cloneable接口实现原理

cloneable其实就是一个标记接口，只有实现这个接口后，然后在类中重写Object中的clone方法，然后通过类调用clone方法才能克隆成功，如果不实现这个接口，则会抛出CloneNotSupportedException(克隆不被支持)异常。

涉及浅克隆，和深度克隆，浅克隆是将基本数据类型的值复制给新的实例对象，但引用数据类型则指向的还是同一个引用，所以复制后两个对象的引用数据类型的值有一个有变动，则另一个也会变动，所以需要深度克隆，深度克隆则需要实现cloneable接口，基本数据类型因为在jvm内存中变量存储的值确定的值，非对象的引用。所以拷贝后的对象引用类型需要深度拷贝。

## 12、异常分类以及处理机制



Throwable是 Java 语言中所有错误或异常的超类。

Error时java程序运行时的系统异常应用程序不能抛出该类的对象。

Exception异常非为io异常: IOException、SQLException。和运行异常 : 如：NullPointerException、ClassCastException；一个是检查异常CheckedException。

处理异常的方式为1.抛出异常：一是throw,一个throws，还有一种系统自动抛异常。

Throw和throws的区别：

1. 位置不同：throws用在函数上，后面跟的是异常类，可以跟多个；而throw用在函数内，后面跟的是异常对象。
2. 功能不同：throws用来声明异常算是预处理异常，即知道可能会发生这个异常但并不一定会发生。

Throw则直接抛出问题对象给调用者。语句运行就结束了，下边的也不会执行。

1. 两者都是消极的处理方式，真正的处理异常应该交给函数处理。

2．捕获异常。用try catch{}

## 13、wait和sleep的区别

对于sleep()方法，我是属于Thread类中的。而wait()方法，则是属于Object类中的。

sleep()方法使方法暂停执行指定时间。会让出cpu。但是不会让出对象锁，监控状态一直保持着。

Wait()方法会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify()方法后本线程才进入对象锁定池准备。

即比如在synchronized代码中，sleep（）会一直占用着别的线程只能等待，wait方法则放弃了使用权，别的线程可以使用。

## 14、jdk1.7和1.8的区别

## 14、数组在内存中如何分配

New 一个数组或者为数组赋值时。便确定了数组的长度。在内存中初始化，声明的变量存贮在栈中，new 出的对象保存在堆中。

Synchronized（this）{

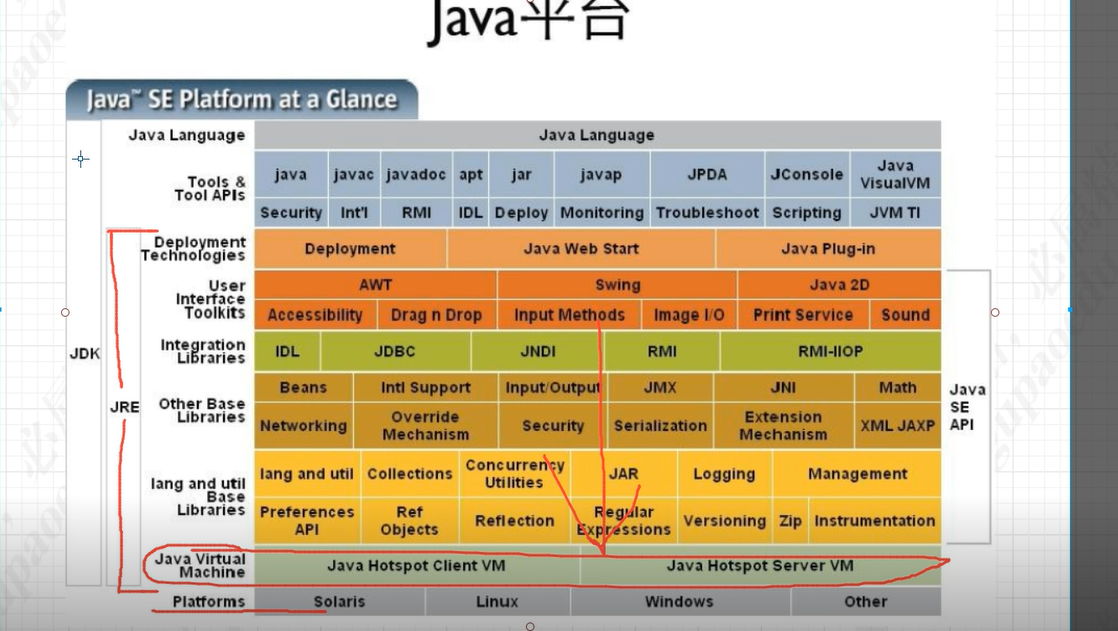
//wait();如果用wait方法则别的线程则可以进入此代码块。

//sleep();如果使sleep方法则一直占用别的线程不能进入。

}

## 15.jdk、jre、jvm

Jre包含jvm，jdk包含jre。



Jvm：运行转义层。

Jre：java运行时环境。

Jdk：类库，jiava文件编译器，监控器包含jre。

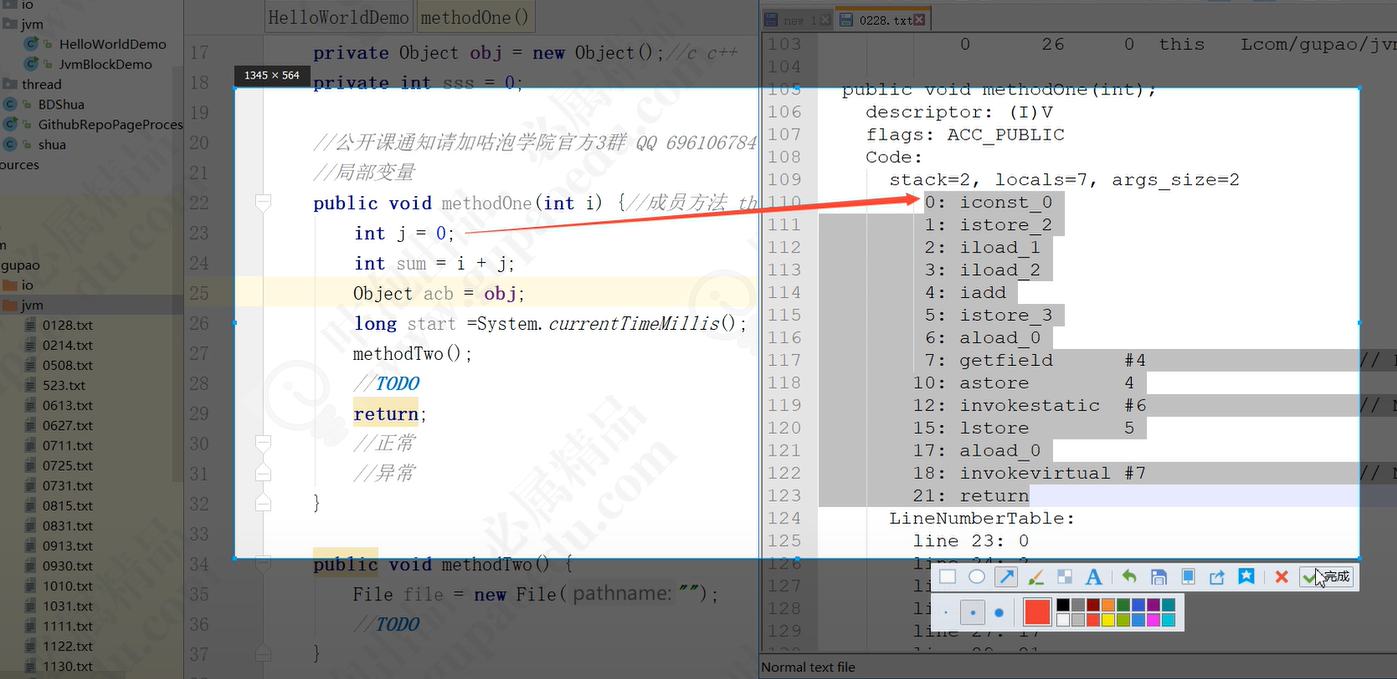
Jvm主要是内存管理（内存溢出），

类运行时，在jvm的运行时数据区（方法区、程序计数器

虚拟机栈、本地方法栈、heap）。

程序计数器：指向当前程序正在执行的字节码指令的地址（行号）。

虚拟机栈：存储当前线程运行方法是所需的数据、指令、放回地址—>最小的单位为栈帧，一个方法区一个栈帧（）。



Java里边最小的执行单位是线程。

# 那些你很冒险的梦:Java 并发

## 1、synchronized 的实现原理以及锁优化？

## 2、volatile 的实现原理？

## 3、Java 的信号灯？

## 4、synchronized 在静态方法和普通方法的区别？

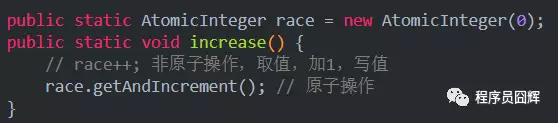
## 5、怎么实现所有线程在等待某个事件的发生才会去执行？

## 6、CAS？CAS 有什么缺陷，如何解决？

CAS（Compare-and-Swap），即比较并替换。CAS需要有3个操作数：内存地址V，旧的预期值A，即将要更新的目标值B。

CAS指令执行时，当且仅当内存地址V的值与预期值A相等时，将内存地址V的值修改为B，否则就什么都不做。整个比较并替换的操作是一个原子操作。

Java并发包原子操作类



缺陷1循环时间长开销很大。

缺陷2只能保证一个共享变量的原子操作。

缺陷3 ABA问题。为了解决这个问题，提供了一个带有标记的原子引用类“AtomicStampedReference”

## 7、synchronized 和 lock 有什么区别？

1）Lock不是Java语言内置的，synchronized是Java语言的关键字，因此是内置特性。Lock是一个类，通过这个类可以实现同步访问；

2）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。

区别： 1.用法不一样。synchronized既可以加在方法上，也可以加载特定的代码块上，括号中表示需要锁的对象。而Lock需要显示地指定起始位置和终止位置。synchronzied是托管给jvm执行的，Lock锁定是通过代码实现的。 2.在性能上来说，如果竞争资源不激烈，两者的性能是差不多的，而当竞争资源非常激烈时（即有大量线程同时竞争），此时Lock的性能要远远优于synchronized。所以说，在具体使用时要根据适当情况选择。 3.锁的机制不一样。synchronized获得锁和释放的方式都是在块结构中，而且是自动释放锁。而Lock则需要开发人员手动去释放，并且必须在finally块中释放，否则会引起死锁问题的发生。 4.Lock是一个接口，而synchronized是Java中的关键字,synchronized是内置的语言实现； 5.synchronized在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而Lock在发生异常时，如果没有主动通过unLock()去释放锁，则很可能造成死锁现象，因此使用Lock时需要在finally块中释放锁； 6.Lock可以让等待锁的线程响应中断，而synchronized却不行，使用synchronized时，等待的线程会一直等待下去，不能够响应中断；通过Lock可以知道有没有成功获取锁，而synchronized却无法办到。Lock可以提高多个线程进行读操作的效率。

## 8、Hashtable 是怎么加锁的 ？

## 9、HashMap 的并发问题？

## 10、ConcurrenHashMap 介绍？1.8 中为什么要用红黑树？

## 11、AQS

## 12、如何检测死锁？怎么预防死锁？

## 13、Java 内存模型？

## 14、如何保证多线程下 i++ 结果正确？

## 15、线程池的种类，区别和使用场景？

newCachedThreadPool：

newFixedThreadPool：

newSingleThreadExecutor:

NewScheduledThreadPool:

## 16、分析线程池的实现原理和线程的调度过程？

## 17、线程池如何调优，最大数目如何确认？

## 18、ThreadLocal原理，用的时候需要注意什么？

## 19、CountDownLatch 和 CyclicBarrier 的用法，以及相互之间的差别?

## 20、LockSupport工具

## 21、Condition接口及其实现原理

## 22、Fork/Join框架的理解

## 23、分段锁的原理,锁力度减小的思考

## 24、八种阻塞队列以及各个阻塞队列的特性

## 25、java队列有那些

# 那些你很冒险的梦:Spring

## 1、BeanFactory 和 FactoryBean？

## 2、Spring IOC 的理解，其初始化过程？

## 3、BeanFactory 和 ApplicationContext？

## 4、Spring Bean 的生命周期，如何被管理的？

## 5、Spring Bean 的加载过程是怎样的？

## 6、如果要你实现Spring AOP，请问怎么实现？

Aop的原理就是实现了jdk的动态代理（通过反射机制获取目标对象的方法）和cglib的动态代理。

## 7、如果要你实现Spring IOC，你会注意哪些问题？

## 8、Spring 是如何管理事务的，事务管理机制？

## 9、Spring 的不同事务传播行为有哪些，干什么用的？

事务的传播行为的7种类型：

|  |  |
| --- | --- |
| **事务传播行为类型** | **说明** |
| PROPAGATION\_REQUIRED | 支持当前事务，如果不存在 就新建一个 |
| PROPAGATION\_SUPPORTS | 支持当前事务，如果不存在，就不使用事务 |
| PROPAGATION\_MANDATORY | 支持当前事务，如果不存在，抛出异常 |
| PROPAGATION\_REQUIRES\_NEW | 如果有事务存在，挂起当前事务，创建一个新的事务 |
| PROPAGATION\_NOT\_SUPPORTED | 以非事务方式运行，如果有事务存在，挂起当前事务 |
| PROPAGATION\_NEVER | 以非事务方式运行，如果有事务存在，抛出异常 |
| PROPAGATION\_NESTED | 如果当前事务存在，则嵌套事务执行只对DataSourceTransactionManager 起效 |

## 10、Spring 中用到了那些设计模式？

## 11、Spring MVC 的工作原理？

## 12、Spring 循环注入的原理？

## 13、Spring AOP的理解，各个术语，他们是怎么相互工作的？

## 14、Spring 如何保证 Controller 并发的安全？

# 那些你很冒险的梦:Netty

## 1、BIO、NIO和AIO

## 2、Netty 的各大组件

## 3、Netty的线程模型

## 4、TCP 粘包/拆包的原因及解决方法

## 5、了解哪几种序列化协议？包括使用场景和如何去选择

## 6、Netty的零拷贝实现

## 7、Netty的高性能表现在哪些方面

# 那些你很冒险的梦:分布式相关

## 1、Dubbo的底层实现原理和机制

## 2、描述一个服务从发布到被消费的详细过程

## 3、分布式系统怎么做服务治理

Dubbo

## 4、接口的幂等性的概念

同步个接口相同的参数多次请求和一次请求对系统造成的影响应该是一样的。比如支付接口，重复点击多次，但实际只应该扣一次钱。

## 5、消息中间件如何解决消息丢失问题

## 6、Dubbo的服务请求失败怎么处理

## 7、重连机制会不会造成错误

## 8、对分布式事务的理解

## 9、如何实现负载均衡，有哪些算法可以实现？

## 10、Zookeeper的用途，选举的原理是什么？

## 11、数据的垂直拆分水平拆分。

## 12、zookeeper原理和适用场景

## 13、zookeeper watch机制

## 14、redis/zk节点宕机如何处理

## 15、分布式集群下如何做到唯一序列号

## 16、如何做一个分布式锁

## 17、用过哪些MQ，怎么用的，和其他mq比较有什么优缺点，MQ的连接是线程安全的吗

## 18、MQ系统的数据如何保证不丢失

## 19、列举出你能想到的数据库分库分表策略；分库分表后，如何解决全表查询的问题

## 20、zookeeper的选举策略

## 21、全局ID

# 那些你很冒险的梦:数据库

## mysql分页有什么优化

用主键或唯一索引，

用子查询SELECT \* FROM your\_table WHERE id <=

(SELECT id FROM your\_table ORDER BY id desc LIMIT ($page-1)\*$pagesize ORDER BY id desc LIMIT $pagesize

利用连接

## 2、悲观锁、乐观锁

加锁可以手动加，但一边mysql自动枷锁，根据create，insert，update自动加锁。跟事务本身的隔离级别有关系。

## 3、组合索引，最左原则

组合索引三个字段。

这两个查询语句都会用到索引(col1,col2)，mysql创建联合索引的规则是首先会对联合合索引的最左边的，也就是第一个字段col1的数据进行排序，在第一个字段的排序基础上，然后再对后面第二个字段col2进行排序。其实就相当于实现了类似 order by col1 col2这样一种排序规则。

减少开销。 建一个联合索引(col1,col2,col3)，实际相当于建了(col1),(col1,col2),(col1,col2,col3)三个索引。每多一个索引，

## 4、mysql 的表锁、行锁

## 5、mysql 性能优化

1. 为查询缓存优化你的查询。

2. EXPLAIN 你的 SELECT 查询。

3. 当只要一行数据时使用 LIMIT 1。

4. 为搜索字段建索引。

5. 在Join表的时候使用相当类型的例，并将其索引。

6. 千万不要 ORDER BY RAND()。

7. 避免 SELECT \*。

11. 尽可能的使用 NOT NULL。

16. 垂直分割

## 6、mysql的索引分类：B+，hash；什么情况用什么索引

## 7、事务的特性和隔离级别

# 那些你很冒险的梦:缓存

## 1、Redis用过哪些数据数据，以及Redis底层怎么实现

## 2、Redis缓存穿透，缓存雪崩

## 3、如何使用Redis来实现分布式锁

## 4、Redis的并发竞争问题如何解决

## 5、Redis持久化的几种方式，优缺点是什么，怎么实现的

## 6、Redis的缓存失效策略

## 7、Redis集群，高可用，原理

## 8、Redis缓存分片

## 9、Redis的数据淘汰策略

# 那些你很冒险的梦:JVM

## 1、详细jvm内存模型

## 2、讲讲什么情况下回出现内存溢出，内存泄漏？

## 3、说说Java线程栈

## 4、JVM 年轻代到年老代的晋升过程的判断条件是什么呢？

## 5、JVM 出现 fullGC 很频繁，怎么去线上排查问题？

## 6、类加载为什么要使用双亲委派模式，有没有什么场景是打破了这个模式？

## 7、类的实例化顺序

## 8、JVM垃圾回收机制，何时触发MinorGC等操作

## 9、JVM 中一次完整的 GC 流程（从 ygc 到 fgc）是怎样的

## 10、各种回收器，各自优缺点，重点CMS、G1

## 11、各种回收算法

## 12、OOM错误，stackoverflow错误，permgen space错误

# 那些你很冒险的梦:SpringCloud

## 1、什么是微服务

## 2、微服务之间是如何独立通讯的

## 3、springCloud和Dubbo有什么区别

## 4、springBoot 和springCloud 请你谈谈对他们的理解

## 5、什么事服务熔断，服务降级

## 6、微服务优缺点人别是什么，说下你在项目中遇到的坑

## 7、你所知道的微服务技术有哪些。

## 8、eureka和zookeeper都可以提供服务注册发现，说说两者区别。

# 那些你很冒险的梦:Web

## 一级缓存，二级缓存，三级缓存

## Web.xml

配置加载xml文件（基础文件，bean，监听等）。

1. 启动一个WEB项目的时候,容器(如:Tomcat)会去读它的配置文件web.xml.读两个节点: <listener></listener> 和 <context-param></context-param>

2.紧接着,容器创建一个ServletContext(上下文),这个WEB项目所有部分都将共享这个上下文.

3.容器将<context-param></context-param>转化为键值对,并交给ServletContext.

4.容器创建<listener></listener>中的类实例,即创建监听.

5.在监听中会有contextInitialized(ServletContextEvent args)初始化方法,在这个方法中获得ServletContext = ServletContextEvent.getServletContext();  
context-param的值 = ServletContext.getInitParameter("context-param的键");

6.得到这个context-param的值之后,你就可以做一些操作了.注意,这个时候你的WEB项目还没有完全启动完成.这个动作会比所有的Servlet都要早.  
换句话说,这个时候,你对<context-param>中的键值做的操作,将在你的WEB项目完全启动之前被执行.

7.举例.你可能想在项目启动之前就打开数据库.  
那么这里就可以在<context-param>中设置数据库的连接方式,在监听类中初始化数据库的连接.

8.这个监听是自己写的一个类,除了初始化方法,它还有销毁方法.用于关闭应用前释放资源.比如说数据库连接的关闭.

<https://www.cnblogs.com/jiaguozhilian/p/5819032.html>

## jdbc 配置

### 解析propertity文件（数据源配置）。

### DruidDataSource 数据源公共属性

### 加载Mybatis配置 mapper文件所在位置

### 扫描mapper接口

* 1. MapperScannerConfigurer 自动扫描 将Mapper接口生成代理注入到Spring

### 源文件

## spring-mvc 配置

### pring，mvc注解启用

* + 1. <!--启用spring的一些annotation -->
  1. <context:annotation-config/>
  2. <!-- 配置注解驱动 可以将request参数与绑定到controller参数上 -->
  3. <mvc:annotation-driven/>

### 扫描包

自动扫描该包，使SpringMVC认为包下用了@controller注解的类是控制器

<context:component-scan base-package="com.lhj.\*" />

### 静态资源映射

<mvc:resources mapping="/css/\*\*" location="/css/"/>

<mvc:resources mapping="/js/\*\*" location="/js/"/>

<mvc:resources mapping="/image/\*\*" location="/image/"/>

### 返回对象转换json



# 那些你很冒险的梦:AOP

# 那些你很冒险的梦:Log4j配置

### 1.需要依赖包

<dependency>  
<groupId>org.slf4j</groupId>  
<artifactId>slf4j-log4j12</artifactId>  
<version>1.6.4</version>  
</dependency>  
<dependency>  
<groupId>log4j</groupId>  
<artifactId>log4j</artifactId>  
<version>1.2.14</version>  
</dependency>  
<dependency>  
<groupId>org.slf4j</groupId>  
<artifactId>slf4j-api</artifactId>  
<version>1.7.7</version>  
</dependency>

### 2.resouse下增加



## 事务管理

## Filter过滤器

## Mapper 增强：自带插入，更新，删除，查询等方法

### Dao interface 继承Mymapper

### Maven 依赖

<dependency>

<groupId>tk.mybatis</groupId>

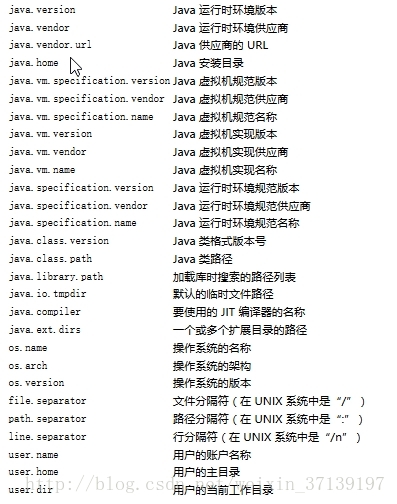
<artifactId>mapper</artifactId>

<version>${mapper.version}</version>

</dependency>

# 那些你很冒险的梦:Base

## System.getProperty()方法获取系统变量



<https://blog.csdn.net/gupao123456/article/details/81750056>

swagger

lombock