# 面试

# 常见问题领域：

**集群架构、大数据、高并发、分布式、高可用、性能调优**。

## 常见基础面试题

### java类集

* 接口：**List,Set,Map,Iterator,Enumeration**
* **List接口：**
  + **ArrayList包装了数组。支持随机访问。**
  + **LinkedList（链表的实现，搜索的时间复杂度n）**
* **Set接口：排序子类、HashSet与hashCode()和equals的关系。**
  + **HashSet：重复的判断依靠的是hashCode()和equals()，无序的。**
  + **TreeSet：有序的，依靠Comparable排序。**
  + **LinkedHashSet：继承了HashSet，但是有序的，增加顺序为保存顺序。**
* **Map接口：Map.Entry、Iterator输出、HashSet、WeakHashMap（弱引用）**

### 字符串哈希相等，equals相等吗，反过来呢？

如果哈希相同，那么equals相等。反过来也应该是相同的。

### Spring的工作原理，控制反转怎么实现的，过虑器过滤编码怎么实现

* Spring的核心组成：IOC&DI（工厂设计）、AOP（动态代理设计）；
  + Spring之中对于XML解析处理采用的是dom4j的实现；
  + Annotation的时候必须要有一个容器。
* 过滤器编码考虑两种情况
  + Struts1.x、Spring MVC、JSP + Servlet可以通过过滤器完成。
  + Struts2.x必须通过拦截器完成。
  + 实现：考虑可扩展性的配置，在配置文件里设置编码，程序运行的时候动态获得配置的编码：请求编码和回应编码

### 框架的源码看过没有

* 千万不要回答“没有”
* 框架的核心思想：反射+XML（Annotation）
* Struts2.x的设计：交由过滤器执行，而后过滤器交给控制器（Action）完成
* Spring MVC：基于方法的请求处理，所有的参数都提交到方法上，本质上还是一个DispatcherServlet。
* Hibernate：反射和dom4j的解析处理流程。

### 动态代理是怎么实现的

* 直接使用InvocationHandler接口进行实现，同时利用Proxy类设置动态请求对象。
* 使用CGLIB来避免对于“基于接口实现”的限制。

### action是单例还是多例

* Struts2.x 和 SpringMVC都是多例。
* Struts1.x是单例。
* 但是Struts2.x 和 SpringMVC是可以控制的，通过注解@scope=”prototype”来控制。

### 怎么配置Bean

由xml和annotation的扫描负责。

* XML使用<bean>标签，容器启动时可以创建
* annotation必须设置context命名空间，然后进行包的扫描。

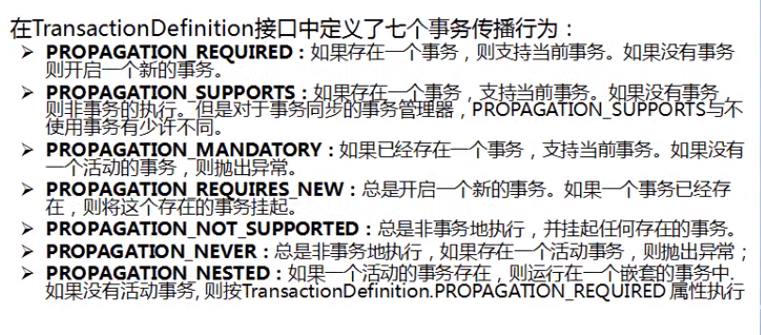
### Java的设计模式

工厂设计模式，代理设计模式，单例设计模式，合成设计模式，门面设计模式（JDBC）、装饰设计模式(PrintStream)、模板设计模式(Servlet)。

### 事务的控制

* Mysql数据库如果想要使用事务，则必须使用“type=innodb”这个数据引擎；
* 事务的核心控制commit、rollback。
* 在spring里面，利用AspectJ可以设置AOP的切面，而后进行声明式事务控制。

### 事务的传播属性



### 购物车是怎么实现的

购物车的实现可以基于：

* Session：浏览器关闭后消失
* Cookie：数据保存在本地，如果切换到手机无效
* 数据库：可以在不同的终端上持续操作

实现：Ajax处理操作，考虑到用户量大的情况，并且访问量频繁，则需要单独实现购物车的子系统模块。

### 统计一天的订单量

分时统计，单独配置一个文件进行计数（需要考虑同步，一同步就慢），不应该出现在抢购环节中。

### IN、Having、exist

* IN是判断具体的几项数据
* having是基于分组后的筛选，依然需要统计函数处理
* exist只是判断子查询里面是否有数据

### 定时任务

Java本身提供的定时任务：TimeTask、Timer,此类对于定时很难完成，它只能做频率，但是不准。所以定时开发会使用quartz组件，spring也提供自己的定时实现，可以在准确的时间执行。

### JVM的内存管理

内存分为：Java虚拟机栈，本地方法栈，堆，程序计数器、方法区。

所谓的垃圾处理操作只是堆内存：年轻代、老年代、永生代（JDK1.8移除）。

### 说一下缓存

* 缓存的主要目的是为了提高查询的效率，常见的两种缓存组件：EsCache（数据库）、OSCache（页面），除此之外还有缓存的数据库redis、memcached，其中redis可以将数据保存到磁盘上，并且支持的数据类型要多于memcached。Redis数据库每秒的并发访问量很高。

### 统计所有重名用户

考虑数据量问题，如果数据量很大，考虑使用**位图索引**，数据量小则分组统计即可。

### 使用Ajax处理函数有哪些

$.post()、$.get()、$.ajax()、$.jsonp()

### 简单描述一下webservice

Web服务调用，结合WSDL与SOAP行成远程方法调用；实现有两种：CXF、Jersey

### 描述一下RPC

RPC是一个远程过程调用，webservice是rpc一种实现。

## 语言基础

### default关键字

在java1.8之后，该关键字可以使用在接口定义中的方法修饰中。则该方法是一个普通方法。

### Java内存模型

Java的内存空间可以划分为以下几点：

* 新生代区：新生的对象都保存在此处，但不一定都一直存活；如果内存满了，则执行GC
* 旧生代区：如果有些对象发现要一直使用，则进入到旧生代区。如果要执行GC，先清理新生代区，如果发现空间仍然不足，则继续清理旧生代区。
* 永久区：永久区的数据不会清除，即使出现了“outofmemoryError”也不会清除。
* 调整虚拟机内存：-Xms（初始空间大小） -Xmx（最大） -Xmn（最小）
  + -Xms初始默认为物理内存的1/64,小于1G。
  + -Xmx最大分配内存，默认大小为物理内存的1/4，但是小于1G。
  + -Xmn设置新生代的堆内存大小。

### BinaryTree（二叉树）

对于树的操作考虑两个问题：

* 数据的存储问题：左，右
* 树的遍历问题：前序，中序、后序。

二叉树的操作常常使用中序，因为它可以排序。

基础实现：

|  |
| --- |
| **package** com.pcq.util;  **import** java.util.Arrays;  **public** **class** BinaryTree {  **private** Node root;//根节点  **private** **int** count; //统计个数  **private** Object[] retObj;  **private** **int** foot;  **public** BinaryTree() {  // **TODO** 自动生成的构造函数存根  }  **public** **static** **void** main(String[] args) {  BinaryTree bt = **new** BinaryTree();  bt.add("a");  bt.add("d");  bt.add("x");  bt.add("c");  System.***out***.println(Arrays.*toString*(bt.toArray()));  }  **private** **class** Node {  **private** Comparable data;//考虑到比较大小排序问题  **private** Node left;  **private** Node right;    **public** Node(Comparable data) {  **this**.data = data;  }  **public** **void** addNode(Node newNode) {  **if**(**this**.data.compareTo(newNode.data) < 0) {  **if**(**this**.right == **null**){  **this**.right = newNode;  } **else** {  **this**.right.addNode(newNode);  }  } **else** {  **if**(**this**.left == **null**) {  **this**.left = newNode;  } **else** {  **this**.left.addNode(newNode);  }  }  }    **public** **void** toArrayNode() {  **if**(**this**.left != **null**) {  **this**.left.toArrayNode();  }  BinaryTree.**this**.retObj[BinaryTree.**this**.foot++] = **this**.data;  **if**(**this**.right != **null**) {  **this**.right.toArrayNode();  }  }      }    **public** Object[] toArray() {  **if**(**this**.root == **null**) {  **return** **null**;  }  **this**.foot = 0;  **this**.retObj = **new** Object[**this**.count];  **this**.root.toArrayNode();  **return** **this**.retObj;  }  **public** **void** add(Object data) {  Comparable com = (Comparable)data;  Node newNode = **new** Node(com);//设置节点是为了排序  **if**(**this**.root == **null**) {  root = newNode;  } **else** {  **this**.root.addNode(newNode);  }  **this**.count++;  }  } |

### ClassLoader加载

自己编写的类加载器继承ClassLoader，最后一次执行。

Java程序类加载器分问两类：双亲加载

* 系统中的类由内部的最高类加载器加载，如果启动的时候加载的系统类太多了，那么一定会造成启动变慢，因此1.9提出了一个模块化的设计。
* 自己定义的类，会使用其它加载器。可以避免系统安全问题。
* 另外自定义的类加载器自定义类文件的来源，通过CLASSPATH加载的，可以通过远程或者文件进行加载。

### HashMap源码

* Map.Entry与HashMap.Node类
* 在hashMap的存储数据增加到阈值的时候，就会由链表变成红黑树，类似于二分查找。