# 面试题

## 1. Struts2与SpringMVC的区别

1.Struts2是类级别上的拦截，一个类对应一个request上下文，SpringMVC是方法级别的拦截，一个方法对应一个request上下文。而且Struts过滤后是去Struts配置文件中找Action，而SpringMVC过滤后是去controller中找对应于@RequestMapping注解的url绑定的方法，

从这里看Struts用起来更麻烦，因为你要每个类的请求你都要配置对应的拦截器 如：

[复制代码](javascript:void(0);)

1 <struts>

2 <package name="loginpackage" namespace="/sys" extends="struts-default" >

3 <action name="login\_\*" class="com.tax.action.LoginAction" method="{1}" >

4 <result name="mainUI" >/mainUI.jsp</result>

5 <result name="loginUI" >/loginUI.jsp</result>

6 </action>

7 </package>

8 </struts>

[复制代码](javascript:void(0);)

而SpringMVC的话：

1 @RequestMapping("api/fleet/delete")

2 @ResponseBody

3 public ReturnResult delete(FleetParam param, String userId) {

4 return fleetService.delete(param, userId);

5 }

2.也因为拦截器原因，导致Struts2的action比较乱，因为它要定义属性来获取请求中参数的数据，而属性在一个类的方法间是共享的（方法间不能独享request、response数据），所以会有点乱。而SpringMVC中请求参数与controller中方法的形参自动配对（在名字相同，或请求参数与形参的属性名相同，或通过@RequestParam注解指定条件下会自动将请求参数的值赋给形参）方法间可以独享request、response数据。

从这点来看，你会发现Struts2的Action看上去有点乱，而SpringMVC的Controller会更简单明了。

3.SpringMVC集成了Ajax，使用非常方便，只需一个注解@ResponseBody就可以实现，然后直接返回响应文本即可，而Struts2拦截器集成了Ajax，在Action中处理时一般必须安装插件或者自己写代码集成进去，使用起来也相对不方便。

## 2. MyBatis mapper.xml中使用静态常量或者静态方法

ongl中：

比如我有一个工具类com.wts.test.DateUtil，其中有一个方法isLeapYear(int year)，用于判断某年是否闰年。而在mapper的某个select中要根据是否闰年执行不同的查询。可以类似这样：

<if test="@com.wts.test.DateUtil@isLeapYear(year)==true">

select \* from tableA

</if>

<if test="@com.wts.test.DateUtil@isLeapYear(year)==false">

select \* from tableB

</if>

如果要使用常量的话，假设有常量类和常量Constant.CURRENT\_YEAR：

<if test=year==@com.wts.test.Consant@CURRENT\_YEAR>

select \* from tableC

</if>

 sql中：

使用静态方法：

<select id='testSelectA' .....>

select \* from tableA where year=${@com.wts.test.DateUtil@getYear()}

</select>

使用静态常量：

<select id='testSelectB' .....>

select \* from tableA where year=${@com.wts.test.Constant@CURRENT\_YEAR}

</select>

以上。转载请指明出处：<http://my.oschina.net/u/1758971/blog/682704>

## 3. JDK环境变量的配置及其作用分析

### 配置环境变量-1：新建JAVA\_HOME

变量名为：JAVA\_HOME

变量值为：jdk的安装的根目录，即第二步提到的C:\Program Files\Java\jdkx.x.x\_xx。

作用：

1. 为了方便引用，比如，你JDK安装在C:\Program Files\Java\jdk1.6.0目录里，则设置JAVA\_HOME为该目录路径, 那么以后你要使用这个路径的时候, 只需输入%JAVA\_HOME%即可, 避免每次引用都输入很长的路径串；2）归一原则, 当你JDK路径被迫改变的时候, 你仅需更改JAVA\_HOME的变量值即可, 否则,你就要更改任何用绝对路径引用JDK目录的文档, 要是万一你没有改全, 某个程序找不到JDK, 后果是可想而知的----系统崩溃!3）第三方软件会引用约定好的JAVA\_HOME变量, 不然, 你将不能正常使用该软件, 以后用JAVA久了就会知道, 要是某个软件不能正常使用, 不妨想想是不是这个问题。

### 配置环境变量-2：新建CLASSPATH

变量名为：CLASSPATH

变量值为：.;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar

 (要加.表示当前路径)

作用：这是一个很有趣,当然也比较折磨初学者的问题, 这个变量设置的目的是为了程序能找到相应的".class"文件, 不妨举个例子: 你编译一个JAVA程序---A.java, 会得到一个A.class的类文件,你在当前目录下执行java A, 将会得到相应的结果(前提是你已经设置CLASSPATH为"."). 现在, 你把A.class移到别的目录下(例如:"e:\"), 执行java A, 将会有NoClassDefFindError的异常,原因就是找不到.class文件, 现在你把CLASSPATH增加为:".;e:\"再运行java A, 看看会有什么结果)，一切正常, java命令通过CLASSPATH找到了.class文件!

### 配置环境变量-3：补充Path

变量名为：Path

变量值为：在后面追加.;%JAVA\_HOME%\bin;

(注意是追加，而不是替换！)

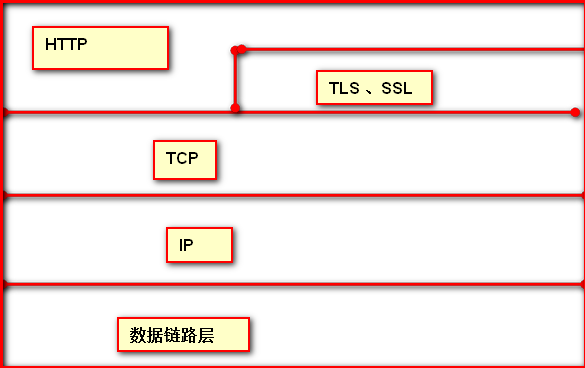
作用：道理很简单, 你想在任何时候都使用%JAVA\_HOME%\bin\java 等来执行java命令吗, 当然不会, 于是, 你可以选择把 %JAVA\_HOME%\bin添加到PATH路径下, 这样, 我们在任何路径下就可以仅用java来执行命令了.(当你在命令提示符窗口输入你个代码时,操作系统会在当前目录和PATH变量目录里查找相应的应用程序, 并且执行.)

## 4.[HTTP协议详解](https://www.cnblogs.com/it-cen/p/4234852.html)

### 1、HTTP是一个应用层协议，由请求和相应构成，是一个标准的客户端服务器模型

HTTP通常承载与TCP协议之上，有时也承载于TLS或SSL协议层之上，这个时候，就成了常说的HTTPS。默认HTTP端口为80，HTTPS的端口号是443

HTTP在OSI模型中的位置如下图：



### 2、HTTP协议的模型客户端发起请求，服务器回送相应

HTTP协议是一个无状态的协议，同一个客户端的这次请求和上次请求没有对应的关系。

### 3、HTTP协议是如何工作的

首先，客户端发送一个请求给服务器，服务器接收到这个请求后将生成一个相应返回给客户端。一次HTTP操作成为一个事务。

其工作过程分为四步：

（1）、客户机与服务器建立连接。点击某个链接，HTTP协议的工作开始；

（2）、建立连接后，客户端发送一个请求给服务器。格式为：前边是统一资源标识符（URL），中间是协议版本号，后边的MIME信息（包括请求修饰符、客户机信息和可能的内容）；

（3）、服务器街道请求后，给予相应的信息相应。格式为：首先是一个状态行（包括信息的协议版本号、一个成功或错误的代码），然后是MIME信息（包括服务器信息，实体信息和可能内容）；

（4）、客户端接收到服务器返回的信息并显示在用户的显示屏上，然后客户机与服务器断开连接。

### 4、下面简单介绍下HTTP协议中的一些主要的概念

#### （1）、请求

请求由三部分组成：请求行、消息报头、请求正文；



<1>请求行包括请求方法、请求URI、协议版本

请求方法有get、post、head等

<2>报头格式：名字+：+空格+值

Host：头域指定请求资源的Internet主机和端口号

User-Agent：包含浏览者的信息，主要是浏览器的名称版本和所用的操作系统

Accept：告诉服务器可以接受的文件格式

Cookie：用来标志一些信息

Cache-Control：指定请求和响应遵循的缓存机制

Referer：指定请求URI的源资源地址

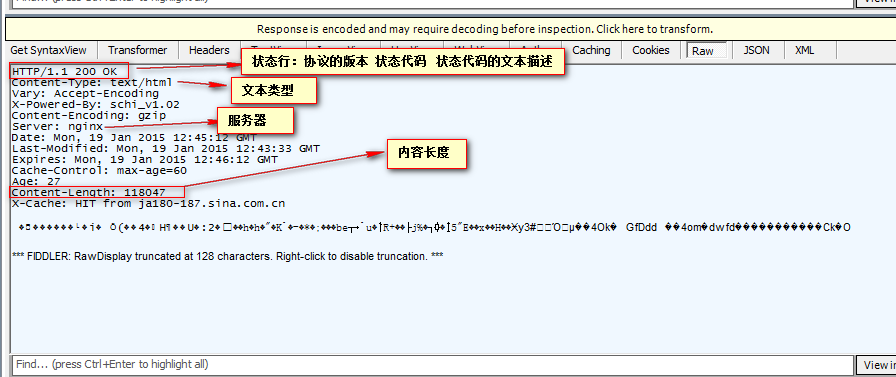
Content-Length：内容长度

Content-Range：响应资源的范围。可以实现断点续传，迅雷就是基于这个原理，使用多线程分段读取网络上的资源，最后在合并

Accept-Enconding：指定所能接受的编码方式

<3>、请求正文，可以没有

#### （2）、响应



状态吗由三位数组成，第一个数字定义了响应的类别，有五种可能取值：

1XX：指示信息——请求已接收，继续处理

2XX：成功——请求已被成功接收、理解、接受

3XX：重定向——要完成请求必须进行更进一步的操作

4XX：客户端错误——请求由语法错误或请求无法实现

5XX：服务器端错误——服务器未能实现合法的请求

常见状态码、状态描述和说明：

200 OK：客户端请求成功

400 Bad Request：客户端请求由语法错误、不能被服务器所理解

401 Unauthorize：请求未经授权

403 Forbidden：服务器收到请求，但是拒绝提供服务

404 Not Found：请求资源不存在，例如输入了错误的URL

500 Internal Server Error：服务器发生不可预期的错误

503 Server Unavailable：服务器当前不能处理客户端的请求，一段时间后可能恢复正常

### 5、抓包工具

分两类：

（1）、常规抓包工具：已IRIS、Wireshark为代表，这类软件可以抓取到整个局域网内所有的数据包，主要工作在数据传输层

（2）、专用抓包工具：只抓取某一协议，通常工作在应用层，最常见的就是对HTTP协议的抓取，如Fiddler、HttpWatch等

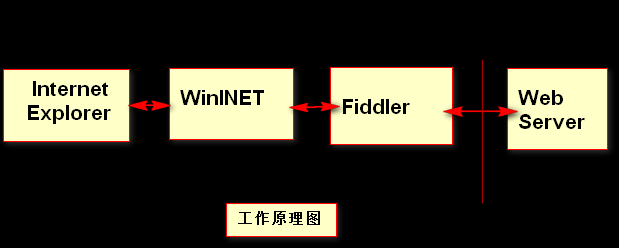
下面介绍一下Fiddler这款抓包软件的工作原理：

Fiddler是以代理服务器的方式监听系统的网络数据流动。运行软件后，就会在本地打开8888端口，网络数据流通过Fiddler进行中转时，可以监视HTTP/HTTPS数据流的记录并加以分析，甚至可以修改发送和接收的数据。

其工作原理是在浏览器（或者其他使用HTTP协议的进程）和服务器之间扮演代理的角色，这样所有的通信都要经过它。

其最大的一个特点就是可以中途修改HTTP通信内容。

工作原理如图：



Fiddler功能很强大，大家可以自行研究下。

## 5. mybatis中批量插入的两种方式(高效插入)

### MyBatis简介

MyBatis是一个支持普通SQL查询，存储过程和高级映射的优秀持久层框架。MyBatis消除了几乎所有的JDBC代码和参数的手工设置以及对结果集的检索封装。MyBatis可以使用简单的XML或注解用于配置和原始映射，将接口和Java的POJO（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录。

### 一、mybiats foreach标签

foreach的主要用在构建in条件中，它可以在SQL语句中进行迭代一个集合。foreach元素的属性主要有 item，index，collection，open，separator，close。item表示集合中每一个元素进行迭代时的别名，index指 定一个名字，用于表示在迭代过程中，每次迭代到的位置，open表示该语句以什么开始，separator表示在每次进行迭代之间以什么符号作为分隔 符，close表示以什么结束，在使用foreach的时候最关键的也是最容易出错的就是collection属性，该属性是必须指定的，但是在不同情况 下，该属性的值是不一样的，主要有一下3种情况：

如果传入的是单参数且参数类型是一个List的时候，collection属性值为list

如果传入的是单参数且参数类型是一个array数组的时候，collection的属性值为array

如果传入的参数是多个的时候，我们就需要把它们封装成一个Map了

具体用法如下:

[?](http://www.jb51.net/article/91951.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <insert id="insertBatch" parameterType="List">  INSERT INTO TStudent(name,age)  <foreach collection="list" item="item" index="index" open="("close=")"separator="union all">  SELECT #{item.name} as a, #{item.age} as b FROM DUAL  </foreach>  </insert> |

### 二、mybatis ExecutorType.BATCH

Mybatis内置的ExecutorType有3种，默认的是simple，该模式下它为每个语句的执行创建一个新的预处理语句，单条提交sql；而batch模式重复使用已经预处理的语句，并且批量执行所有更新语句，显然batch性能将更优； 但batch模式也有自己的问题，比如在Insert操作时，在事务没有提交之前，是没有办法获取到自增的id，这在某型情形下是不符合业务要求的

具体用法如下:

#### 方式一 spring+mybatis 的

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | //获取sqlsession  //从spring注入原有的sqlSessionTemplate  @Autowired  private SqlSessionTemplate sqlSessionTemplate;  // 新获取一个模式为BATCH，自动提交为false的session  // 如果自动提交设置为true,将无法控制提交的条数，改为最后统一提交，可能导致内存溢出  SqlSession session = sqlSessionTemplate.getSqlSessionFactory().openSession(ExecutorType.BATCH,false);  //通过新的session获取mapper  fooMapper = session.getMapper(FooMapper.class);  int size = 10000;  try{  for(int i = 0; i < size; i++) {  Foo foo = new Foo();  foo.setName(String.valueOf(System.currentTimeMillis()));  fooMapper.insert(foo);  if(i % 1000 == 0 || i == size - 1) {  //手动每1000个一提交，提交后无法回滚  session.commit();  //清理缓存，防止溢出  session.clearCache();  }  }  } catch (Exception e) {  //没有提交的数据可以回滚  session.rollback();  } finally{  session.close();  }  spring+mybatis |

#### 方法二:结合通用mapper sql别名最好是包名＋类名

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | public void  insertBatch(Map<String,Object> paramMap, List<User> list) throws Exception {  // 新获取一个模式为BATCH，自动提交为false的session  // 如果自动提交设置为true,将无法控制提交的条数，改为最后统一提交，可能导致内存溢出  SqlSession session = sqlSessionTemplate.getSqlSessionFactory().openSession(ExecutorType.BATCH, false);  try {  if(null != list || list.size()>0){  int lsize=list.size();  for (int i = 0, n=list.size(); i < n; i++) {  User user= list.get(i);  user.setIndate((String)paramMap.get("indate"));  user.setDatadate((String)paramMap.get("dataDate"));//数据归属时间  //session.insert("com.xx.mapper.UserMapper.insert",user);  //session.update("com.xx.mapper.UserMapper.updateByPrimaryKeySelective",\_entity);  session.insert(“包名+类名", user);  if ((i>0 && i % 1000 == 0) || i == lsize - 1) {  // 手动每1000个一提交，提交后无法回滚  session.commit();  // 清理缓存，防止溢出  session.clearCache();  }  }  }  } catch (Exception e) {  // 没有提交的数据可以回滚  session.rollback();  e.printStackTrace();  } finally {  session.close();  }  } |

以上所述是小编给大家介绍的mybatis中批量插入的两种方式(高效插入)，希望对大家有所帮助，如果大家有任何疑问请给我留言，小编会及时回复大家的。在此也非常感谢大家对脚本之家网站的支持！

## 6. Redis应用

### 1、redis数据库的优势

### (1) 速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)

### (2) 支持丰富数据类型，支持string，list，set，sorted set，hash

### (3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行

### (4) 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

### 2. redis的应用

1） 大数据时代淘宝、微信、以及微博等都广泛的使用了redis数据库，将一些固定不变的数据例如学校，区域等固定的信息保存在关系型数据库中。然后对于经常变化的数据例如淘宝每个节日都会有比较热门的搜索显示在搜索框，当节日过去关键字自动删除，为了便于管理，可以将这些数据保存在redis数据库中，并设置过期时间，到达时间就自动删除。

2）为了缓解数据库压力，微博首先将发送的微博保存到redis数据库，自己可以立即查看到，然后将内存中的数据同步到关系型数据库。

### 3.ERP中Redis的应用

系统录入一个新员工的同时会将新员工的基础信息写入到所有指定的Redis数据库中，然后等待考勤机那一方的主动读取成功后，当前redis数据库中的数据失效。借此可分担数据库的压力。

### 7. 乐观锁和悲观锁的区别

#### 悲观锁(Pessimistic Lock)

顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。

#### 乐观锁(Optimistic Lock)

顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库如果提供类似于write\_condition机制的其实都是提供的乐观锁。

两种锁各有优缺点，不可认为一种好于另一种，像乐观锁适用于写比较少的情况下，即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。但如果经常产生冲突，上层应用会不断的进行retry，这样反倒是降低了性能，所以这种情况下用悲观锁就比较合适。

## 8. [RabbitMQ和kafka从几个角度简单的对比](https://www.cnblogs.com/davidwang456/p/4076097.html)

业界对于消息的传递有多种方案和产品，

本文就比较有代表性的两个MQ(rabbitMQ,kafka)进行阐述和做简单的对比

在应用场景方面，RabbitMQ,遵循AMQP协议，由内在高并发的erlanng语言开发，用在实时的对可靠性要求比较高的消息传递上。

kafka是Linkedin于2010年12月份开源的消息发布订阅系统,它主要用于处理活跃的流式数据,大数据量的数据处理上。

#### 1)在架构模型方面

RabbitMQ遵循AMQP协议，RabbitMQ的broker由Exchange,Binding,queue组成，其中exchange和binding组成了消息的路由键；客户端Producer通过连接channel和server进行通信，Consumer从queue获取消息进行消费（长连接，queue有消息会推送到consumer端，consumer循环从输入流读取数据）。rabbitMQ以broker为中心；有消息的确认机制。

kafka遵从一般的MQ结构，producer，broker，consumer，以consumer为中心，消息的消费信息保存的客户端consumer上，consumer根据消费的点，从broker上批量pull数据；无消息确认机制。

#### 2)在吞吐量

kafka具有高的吞吐量，内部采用消息的批量处理，zero-copy机制，数据的存储和获取是本地磁盘顺序批量操作，具有O(1)的复杂度，消息处理的效率很高。

rabbitMQ在吞吐量方面稍逊于kafka，他们的出发点不一样，rabbitMQ支持对消息的可靠的传递，支持事务，不支持批量的操作；基于存储的可靠性的要求存储可以采用内存或者硬盘。

#### 3)在可用性方面

rabbitMQ支持miror的queue，主queue失效，miror queue接管。

kafka的broker支持主备模式。

#### 4)在集群负载均衡方面

kafka采用zookeeper对集群中的broker、consumer进行管理，可以注册topic到zookeeper上；通过zookeeper的协调机制，producer保存对应topic的broker信息，可以随机或者轮询发送到broker上；并且producer可以基于语义指定分片，消息发送到broker的某分片上。

rabbitMQ的负载均衡需要单独的loadbalancer进行支持。

原文：<http://wbj0110.iteye.com/blog/1974988>

收集的rabbitmq资料如下：

<http://jzhihui.iteye.com/category/195005>

<http://lynnkong.iteye.com/blog/1699684>

<http://blog.csdn.net/anzhsoft/article/details/19607841>

<http://ybbct.iteye.com/blog/1562326>

## 9. RabbitMQ要点简介

### 1、****什么是RabbitMQ****

RabbitMQ是由Erlang语言编写的实现了**高级消息队列协议**（AMQP）的开源消息代理软件（也可称为 面向消息的中间件）。支持Windows、Linux/Unix、MAC OS X操作系统和包括JAVA在内的多种编程语言。

AMQP，即Advanced Message Queuing Protocol，一个提供统一消息服务的应用层标准高级消息队列协议，是应用层协议的一个开放标准，为面向消息的中间件设计。基于此协议的客户端与消息中间件可传递消息，并不受 客户端/中间件 不同产品，不同的开发语言等条件的限制。

### 2、****RabbitMQ中的重要概念****

（1）**Broker**：经纪人。提供一种传输服务，维护一条从生产者到消费者的传输线路，保证消息数据能按照指定的方式传输。粗略的可以将图中的RabbitMQ Server当作Broker。

（2）**Exchange**：消息交换机。指定消息按照什么规则路由到哪个队列Queue。

（3）**Queue**：消息队列。消息的载体，每条消息都会被投送到一个或多个队列中。

（4）**Binding**：绑定。作用就是将Exchange和Queue按照某种路由规则绑定起来。

（5）**RoutingKey**：路由关键字。Exchange根据RoutingKey进行消息投递。

（6）**Vhost**：虚拟主机。一个Broker可以有多个虚拟主机，用作不同用户的权限分离。一个虚拟主机持有一组Exchange、Queue和Binding。

（7）**Producer**：消息生产者。主要将消息投递到对应的Exchange上面。一般是独立的程序。

（8）**Consumer**：消息消费者。消息的接收者，一般是独立的程序。

（9）**Channel**：消息通道，也称信道。在客户端的每个连接里可以建立多个Channel，每个Channel代表一个会话任务。

### 3、****RabbitMQ的使用流程****

AMQP模型中，消息在producer中产生，发送到MQ的exchange上，exchange根据配置的路由方式投递到相应的Queue上，Queue又将消息发送给已经在此Queue上注册的consumer，消息从queue到consumer有push和pull两种方式。

消息队列的使用过程大概如下：

（1）客户端连接到消息队列服务器，打开一个channel。

（2）客户端声明一个exchange，并设置相关属性。

（3）客户端声明一个queue，并设置相关属性。

（4）客户端使用routing key，在exchange和queue之间建立好Binding关系。

（5）生产者客户端投递消息到exchange。

（6）exchange接收到消息后，就根据消息的RoutingKey和已经设置的binding，进行消息路由（投递），将消息投递到一个或多个队列里。

（7）消费者客户端从对应的队列中获取并处理消息。

### ****4、RabbitMQ的优缺点****

#### **优点**

（1）由Erlang语言开发，支持大量协议：AMQP、XMPP、SMTP、STOMP。

（2）支持**消息的持久化**、**负载均衡**和**集群**，且集群易扩展。

（3）具有一个Web监控界面，易于管理。

（4）安装部署简单，上手容易，功能丰富，强大的社区支持。

（5）支持消息确认机制、灵活的消息分发机制。

#### **缺点**

（1）由于牺牲了部分性能来换取稳定性，比如消息的持久化功能，使得RabbitMQ在大吞吐量性能方面不及Kafka和ZeroMQ。

（2）由于支持多种协议，使RabbitMQ非常重量级，比较适合企业级开发。

因此当需要一个稳定的、高可靠性的、功能强大且易于管理的消息队列可以选择RabbitMQ。如果对消息吞吐量需求较大，且不在乎消息偶尔丢失的情况可以使用Kafka。

### ****5、Exchange类型****

#### ****5.1、Direct Exchange****

（1）名称：直接交换器类型

（2）默认的预先定义exchange名字：**空字符串**或者**amq.direct**

（3）作用描述：根据Binding指定的Routing Key，将符合Key的消息发送到Binding的Queue。可以构建点对点消息传输模型。

如图中RoutingKey分别是error、info、warning，其中error被Binding（绑定）到queue1和queue2上，info和warning被Binding到queue2上。当消息的RoutingKey是error，这条消息将被投递到queue1和queue2中（相当于消息被复制成两个分别投放到两个queue中），然后分别被Consumer1和Consumer2处理。如果消息的RoutingKey是info或者warning，这条消息只会被投递到queue2中，然后被Consumer2处理。如果消息的RoutingKey是其他的字符串，这条消息则会被丢弃。

#### ****5.2、Fanout Exchange****

（1）名称：广播式交换器类型

（2）默认的预先定义exchange名字：**amq.fanout**

（3）作用描述：将同一个message发送到所有同该Exchange 绑定的queue。**不论RoutingKey是什么**，这条消息都会被投递到所有与此Exchange绑定的queue中。

广播式交换器类型的工作方式：不使用任何参数将queue和Exchange进行Binding，发布者publisher向Exchange发送一条消息（注意：直接交换器类型中的producer变成了publisher，其中隐含了两种交换器的区别），然后这条消息被**无条件的**投递给所有和这个Exchange绑定的queue中。

如图中，没有RoutingKey的限制，只要消息到达Exchange，都会被投递到queue1和queue2中，然后被对应的Consumer处理。

#### ****5.3、Topic Exchange****

（1）名称：主题交换器类型

（2）默认的预先定义exchange名字：**amq.topic**

（3）作用描述：根据Binding指定的RoutingKey，Exchange对key进行**模式匹配**后投递到相应的Queue，**模式匹配时符号“#”匹配一个或多个词，符号“\*”匹配正好一个词，而且单词与单词之间必须要用“.”符号进行分隔**。此模式可以用来支持经典的**发布/订阅**消息传输模型-使用主题名字空间作为消息寻址模式，将消息传递给那些部分或者全部匹配主题模式的queue。

  如图中，假如消息的RoutingKey是American.action.13，这条消息将被投递到Q1和Q2中。假如RoutingKey是American.action.13.test（注意：此处是四个词），这条消息将会被丢弃，因为没有routingkey与之匹配。假如RoutingKey是Chinese.action.13，这条消息将被投递到Q2和Q3中。假如RoutingKey是Chinese.action.13.test，这条消息只会被投递到Q3中，#可以匹配一个或者多个单词，而\*只能匹配一个词。

#### ****5.4、Headers Exchange****

（1）名称：标题交换器类型

（2）默认的预先定义exchange名字：**amq.match**和**amq.headers**

（3）作用描述：同direct exchange类似，不同之处是不再使用Routing Key路由，而是使用headers（Message attributes）进行匹配路由到指定Queue。

Headers类型的exchange使用的比较少，它也是忽略routingKey的一种路由方式。是使用Headers来匹配的。Headers是一个键值对，可以定义成HashTable。发送者在发送的时候定义一些键值对，接收者也可以再绑定时候传入一些键值对，两者匹配的话，则对应的队列就可以收到消息。匹配有两种方式all和any。这两种方式是在接收端必须要用键值"x-mactch"来定义。all代表定义的多个键值对都要满足，而any则代码只要满足一个就可以了。fanout，direct，topic exchange的routingKey都需要要字符串形式的，而headers exchange则没有这个要求，因为键值对的值可以是任何类型。

代码示例请参考文章[《RabbitMQ学习之Headers交换类型（java）》](http://blog.csdn.net/leixiaotao_java/article/details/78905205)。

## 10. Kafka简介、基本原理、执行流程与使用场景

### 一、简介

Apache Kafka是分布式发布-订阅消息系统，在 [kafka官网](http://kafka.apache.org/)上对 kafka 的定义：一个分布式发布-订阅消息传递系统。 它最初由LinkedIn公司开发，Linkedin于2010年贡献给了Apache基金会并成为顶级开源项目。Kafka是一种快速、可扩展的、设计内在就是分布式的，分区的和可复制的提交日志服务。

几种分布式系统消息系统的对比：

推荐相关文章：[各消息队列对比，Kafka深度解析，众人推荐，精彩好文！](http://blog.csdn.net/allthesametome/article/details/47362451)   
<http://blog.csdn.net/allthesametome/article/details/47362451>

### 二、Kafka基本架构

它的架构包括以下组件：

1、话题（Topic）：是特定类型的消息流。消息是字节的有效负载（Payload），话题是消息的分类名或种子（Feed）名；

2、生产者（Producer）：是能够发布消息到话题的任何对象；

3、服务代理（Broker）：已发布的消息保存在一组服务器中，它们被称为代理（Broker）或Kafka集群；

4、消费者（Consumer）：可以订阅一个或多个话题，并从Broker拉数据，从而消费这些已发布的消息；

上图中可以看出，生产者将数据发送到Broker代理，Broker代理有多个话题topic，消费者从Broker获取数据。

### 三、基本原理

我们将消息的发布（publish）称作 producer，将消息的订阅（subscribe）表述为 consumer，将中间的存储阵列称作 broker(代理)，这样就可以大致描绘出这样一个场面：

生产者将数据生产出来，交给 broker 进行存储，消费者需要消费数据了，就从broker中去拿出数据来，然后完成一系列对数据的处理操作。

乍一看返也太简单了，不是说了它是分布式吗，难道把 producer、 broker 和 consumer 放在三台不同的机器上就算是分布式了吗。看 kafka 官方给出的图：

多个 broker 协同合作，producer 和 consumer 部署在各个业务逻辑中被频繁的调用，三者通过 zookeeper管理协调请求和转发。这样一个高性能的分布式消息发布订阅系统就完成了。

图上有个细节需要注意，producer 到 broker 的过程是 push，也就是有数据就**推送**到 broker，而 consumer 到 broker 的过程是 pull，是通过 consumer 主动去拉数据的，而不是 broker 把数据主懂发送到 consumer 端的。

### 四、Zookeeper在kafka的作用

上述，提到了Zookeeper，那么Zookeeper在kafka的作用是什么？

（1）无论是kafka集群，还是producer和consumer都依赖于zookeeper来保证系统可用性集群保存一些meta信息。

（2）Kafka使用zookeeper作为其分布式协调框架，很好的将消息生产、消息存储、消息消费的过程结合在一起。

（3）同时借助zookeeper，kafka能够生产者、消费者和broker在内的所以组件在无状态的情况下，建立起生产者和消费者的订阅关系，并实现生产者与消费者的负载均衡。

### 五、执行流程

首先看一下如下的过程：

我们看上面的图，我们把 broker 的数量减少，叧有一台。现在假设我们按照上图进行部署：

（1）Server-1 broker 其实就是 kafka 的 server，因为 producer 和 consumer 都要去还它。 Broker 主要还是做存储用。

（2）Server-2 是 zookeeper 的 server 端，它维持了一张表，记录了各个节点的 IP、端口等信息。

（3）Server-3、 4、 5 他们的共同之处就是都配置了 zkClient，更明确的说，就是运行前必须配置 zookeeper的地址，道理也很简单，这之间的连接都是需要 zookeeper 来进行分发的。

（4）Server-1 和 Server-2 的关系，他们可以放在一台机器上，也可以分开放，zookeeper 也可以配集群。目的是防止某一台挂了。

简单说下整个系统运行的顺序：

（1）启动zookeeper 的 server

（2）启动kafka 的 server

（3）Producer 如果生产了数据，会先通过 zookeeper 找到 broker，然后将数据存放到 broker

（4）Consumer 如果要消费数据，会先通过 zookeeper 找对应的 broker，然后消费。

### 六、Kafka的特性

（1）高吞吐量、低延迟：kafka每秒可以处理几十万条消息，它的延迟最低只有几毫秒，每个topic可以分多个partition, consumer group 对partition进行consume操作；

（2）可扩展性：kafka集群支持热扩展；

（3）持久性、可靠性：消息被持久化到本地磁盘，并且支持数据备份防止数据丢失；

（4）容错性：允许集群中节点失败（若副本数量为n,则允许n-1个节点失败）；

（5）高并发：支持数千个客户端同时读写；

（6）支持实时在线处理和离线处理：可以使用Storm这种实时流处理系统对消息进行实时进行处理，同时还可以使用Hadoop这种批处理系统进行离线处理；

### 七、Kafka的使用场景

（1）日志收集：一个公司可以用Kafka可以收集各种服务的log，通过kafka以统一接口服务的方式开放给各种consumer，例如Hadoop、Hbase、Solr等；

（2）消息系统：解耦和生产者和消费者、缓存消息等；

（3）用户活动跟踪：Kafka经常被用来记录web用户或者app用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到kafka的topic中，然后订阅者通过订阅这些topic来做实时的监控分析，或者装载到Hadoop、数据仓库中做离线分析和挖掘；

（4）运营指标：Kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告；

（5）流式处理：比如spark streaming和storm；

（6）事件源；

相关阅读：

1、[Apache Kafka：下一代分布式消息系统](http://blog.csdn.net/xlgen157387/article/details/77211729)

参考文章：

1、<http://www.cnblogs.com/likehua/p/3999538.html>   
2、<http://www.jianshu.com/p/a82b8619e56c>   
3、<http://blog.csdn.net/ychenfeng/article/details/74980531>

## 11.spring和springboot的区别

想了解区别，其实就是Spring Boot提供了哪些特征：

1. Spring Boot可以建立独立的Spring应用程序；
2. 内嵌了如Tomcat，Jetty和Undertow这样的容器，也就是说可以直接跑起来，用不着再做部署工作了。
3. 无需再像Spring那样搞一堆繁琐的xml文件的配置；
4. 可以自动配置Spring；
5. 提供了一些现有的功能，如量度工具，表单数据验证以及一些外部配置这样的一些第三方功能；
6. 提供的POM可以简化Maven的配置

## 12. Mybatis中#和$的区别

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 #user\_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 $user\_id$，如果传入的值是111,那么解析成sql时的值为order by user\_id, 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](https://www.2cto.com/database/)对象，例如传入表名.

6.一般能用#的就别用$.

MyBatis排序时使用order by 动态参数时需要注意，用$而不是#

## 13. redis和mangodb的区别

转载：<http://blog.csdn.net/dodolzg/article/details/43865241>

### 简介

MongoDB更类似Mysql，支持字段索引、游标操作，其优势在于查询功能比较强大，擅长查询JSON数据，能存储海量数据，但是不支持事务。

Mysql在大数据量时效率显著下降，MongoDB更多时候作为关系数据库的一种替代。

### 内存管理机制

Redis数据全部存在内存，定期写入磁盘，当内存不够时，可以选择指定的LRU算法删除数据。

MongoDB数据存在内存，由linux系统mmap实现，当内存不够时，只将热点数据放入内存，其他数据存在磁盘。

### 支持的数据结构

Redis支持的数据结构丰富，包括hash、set、list等。

MongoDB数据结构比较单一，但是支持丰富的数据表达，索引，最类似关系型数据库，支持的查询语言非常丰富。

### 性能

二者性能都比较高，应该说都不会是瓶颈。

### 可靠性

二者均支持持久化。

### 集群

         MongoDB集群技术比较成熟，Redis从3.0开始支持集群。

### 不适用场景

Ø  需要使用复杂sql的操作

Ø  事务性系统

## 14. 几种解决高并发的处理方式

1.将数据保存到redis上,redis是一个美而小的nosql的数据库,处理 速度非常快,每秒10W+的处理速度

2.使用高性能,高服务的关系型数据,优化sql

3.使用F5/nginx进行负载均衡(dubbo)

     (1)一般都是使用软负载nginx,F5的属于硬件的负载,来减少开发的部分成本

     (2)当服务的调用量越来越大的时候,会对每天的响应时间,进行统计,动态的调整权重

4.统计一些高请求的的数据,进行一些搜索引擎操作,例如Lucene全文检索,数据直接在这里访问,来减少服务的压力

## 15. 处理高并发、大数据下的基本技能

1、扎实的编程基础知识，比如jvm的堆栈、hashmap、queue、线程基础知识  
2、负载均衡知识，nginx,apache各种负载均衡策略及每个策略使用场景  
3、数据库开设读写分离  
4、上千万数据分库分表策略，每个策略的使用场景，时间分表、地区分表、hash分表  
5、海量数据采用NOSQL型数据库如阿里云的表格存储  
6、性能调优，诊断性能的思路及解决方法， jps、jstack、jmap、jhat、jstat等命令,linux的top等命令。  
7、缓存使用,redis、memcache的区别及可以支持的吞吐量。  
8、mysql优化，慢sql的诊断思路及解决方法。  
9、通信协议http、tcp，http协议格式，tcp的原理。  
10、页面的静态化处理  
11、项目的并发压力测试

## 16. 数据库高并发解决方法总结

一个项目刚开始的时候是为了实现基本功能，随着版本和功能的迭代，大数据和高并发成了软件设计必须考虑的问题!

本质很简单，一个是慢，一个是等。

两者是相互关联的，因为慢，所以要等，因为等，所以慢，解决了慢，也就解决了等，解决了等，也就解决了慢。

关键是如何解决慢和等，核心一个是**短**，一个是**少**，一个是**分流,**最后一个是**集群/横向扩张/读写分离/建立主从**。

### 短是指路径要短

典型的mvc结构是请求->controller->model->dao->view，然后把页面返回给用户。要想短的话，

1，页面静态化- 用户可以直接获取页面，不用走那么多流程，比较适用于页面不频繁更新。

2，使用缓存- 第一次获取数据从数据库准提取，然后保存在缓存中，以后就可以直接从缓存提取数据。不过需要有机制维持缓存和数据库的一致性。

3，使用储存过程-那些处理一次请求需要多次访问数据库的操作，可以把操作整合到储存过程，这样只要一次数据库访问就可以了。

4，批量读取 - 高并发情况下，可以把多个请求的查询合并到一次进行，以减少数据库的访问次数

5，延迟修改 - 高并发情况下，可以把多次修改请求，先保存在缓存中，然后定时将缓存中的数据保存到数据库中，风险是可能会断电丢失缓存中的数据，

6,  使用索引 - 索引可以看作是特殊的缓存，尽量使用索引就要求where字句中精确的给出索引列的值。

### 少是指查询的数据要少

1，分表 - 把本来同一张表的内容，可以按照地区，类别等分成多张表，很简单的一个思路，但是要尽量避免分出来的多表关联查询。

2，分离活跃数据 - 例如登录用户业务，注册用户很多，但是活跃的登录用户很少，可以把活跃用户专门保存一张表，查询是先查询活跃表，没有的话再查总表，这也类似与缓存啦。

3， 分块 - 数据库层面的优化，对程序是透明的，查询大数据只用找到相应块就行。

### **分流**三种

1，集群 - 将并发请求分配到不同的服务器上，可以是业务服务器，也可以是数据库服务器。

2，分布式 - 分布式是把单次请求的多项业务逻辑分配到多个服务器上，这样可以同步处理很多逻辑，一般使用与特别复杂的业务请求。

3，CDN - 在域名解析层面的分流，例如将华南地区的用户请求分配到华南的服务器，华中地区的用户请求分配到华中的服务器。

### **横向扩展**

大型网站为了软解大量的并发访问，除了在网站实现分布式负载均衡，远远不够。到了数据业务层、数据访问层，如果还是传统的数据结构，或者只是单单靠一台服务器扛，如此多的数据库连接操作，数据库必然会崩溃，数据丢失的话，后果更是 不堪设想。这时候，我们会考虑如何减少数据库的联接，一方面采用优秀的代码框架，进行代码的优化，采用优秀的数据缓存技术如：memcached,如果资金丰厚的话，必然会想到假设服务器群，来分担主数据库的压力。Ok切入今天微博主题，利用MySQL主从配置，实现读写分离，减轻数据库压力。这种方式，在如今很多网站里都有使用，也不是什么新鲜事情，今天总结一下，方便大家学习参考一下。

概述：搭设一台Master服务器（win8.1系统，Ip：192.168.0.104），搭设两台Slave服务器（虚拟机——一台Ubuntu，一台 Windows Server 2003）

原理：主服务器（Master）负责网站NonQuery操作，从服务器负责Query操作，用户可以根据网站功能模特性块固定访问Slave服务器，或者自己写个池或队列，自由为请求分配从服务器连接。主从服务器利用MySQL的二进制日志文件，实现数据同步。二进制日志由主服务器产生，从服务器响应获取同步数据库。

具体实现：

#### 1、在主从服务器上都装上MySQL数据库，windows系统鄙人安装的是mysql\_5.5.25.msi版本，Ubuntu安装的是mysql-5.6.22-linux-glibc2.5-i686.tar

windows安装mysql就不谈了，一般地球人都应该会。鄙人稍微说一下Ubuntu的MySQL安装，我建议不要在线下载安装，还是离线安装的好。大家可以参考 <http://www.linuxidc.com/Linux/2013-01/78716.htm> 这位不知道大哥还是姐妹，写的挺好按照这个就能装上。在安装的时候可能会出现几种现象，大家可以参考解决一下：

（1）如果您不是使用root用户登录，建议 su - root 切换到Root用户安装，那就不用老是 sudo 了。

（2）存放解压的mysql 文件夹，文件夹名字最好改成mysql

（3）在./support-files/mysql.server start 启动MySQL的时候，可能会出现一个警告，中文意思是启动服务运行读文件时，忽略了my.cnf文件，那是因为my.cnf的文件权限有问题，mysql会认为该文件有危险不会执行。但是mysql还会启动成功，但如果下面配置从服务器参数修改my.cnf文件的时候，你会发现文件改过了，但是重启服务时，修改过后的配置没有执行，而且您 list一下mysql的文件夹下会发现很多.my.cnf.swp等中间文件。这都是因为MySQL启动时没有读取my.cnf的原因。这时只要将my.cnf的文件权限改成my\_new.cnf的权限一样就Ok，命令：chmod 644 my.cnf就Ok

http://images.cnitblog.com/blog/157872/201412/141051539312957.png

（4）Ubuntu中修改文档内容没有Vim，最好把Vim 装上，apt-get install vim,不然估计会抓狂。

这时候我相信MySQL应该安装上去了。

#### 2、配置Master主服务器

（1）在Master MySQL上创建一个用户‘repl’，并允许其他Slave服务器可以通过远程访问Master，通过该用户读取二进制日志，实现数据同步。

1 mysql>create user repl; //创建新用户

2 //repl用户必须具有REPLICATION SLAVE权限，除此之外没有必要添加不必要的权限，密码为mysql。说明一下192.168.0.%，这个配置是指明repl用户所在服务器，这里%是通配符，表示192.168.0.0-192.168.0.255的Server都可以以repl用户登陆主服务器。当然你也可以指定固定Ip。

3 mysql> GRANT REPLICATION SLAVE ON \*.\* TO 'repl'@'192.168.0.%' IDENTIFIED BY 'mysql';

（2）找到MySQL安装文件夹修改my.Ini文件。mysql中有好几种日志方式，这不是今天的重点。我们只要启动二进制日志log-bin就ok。

 在[mysqld]下面增加下面几行代码

1 server-id=1 //给数据库服务的唯一标识，一般为大家设置服务器Ip的末尾号

2 log-bin=master-bin

3 log-bin-index=master-bin.index

（3）查看日志

mysql> SHOW MASTER STATUS;  
+-------------------+----------+--------------+------------------+  
| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |  
+-------------------+----------+--------------+------------------+  
| master-bin.000001 | 1285 | | |  
+-------------------+----------+--------------+------------------+  
1 row in set (0.00 sec)

重启MySQL服务

#### 3、配置Slave从服务器（windows）

（1）找到MySQL安装文件夹修改my.ini文件，在[mysqld]下面增加下面几行代码

1 [mysqld]

2 server-id=2

3 relay-log-index=slave-relay-bin.index

4 relay-log=slave-relay-bin

重启MySQL服务

（2）连接Master

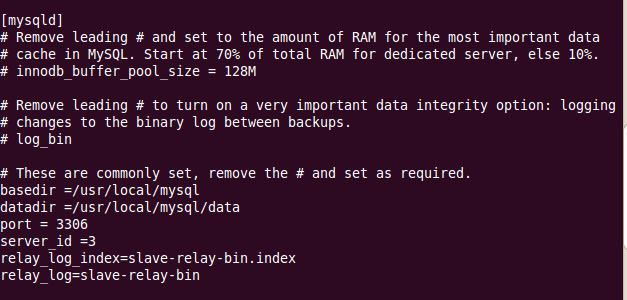
change master to master\_host='192.168.0.104', //Master 服务器Ip  
master\_port=3306,  
master\_user='repl',  
master\_password='mysql',   
master\_log\_file='master-bin.000001',//Master服务器产生的日志  
master\_log\_pos=0;

（3）启动Slave

start slave;

#### 4、Slave从服务器（Ubuntu）

（1）找到MySQL安装文件夹修改my.cnf文件,vim my.cnf



（2） ./support-files/myql.server restart 重启MySQL服务  ,  ./bin/mysql 进入MySQL命令窗口

（3）连接Master

change master to master\_host='192.168.0.104', //Master 服务器Ip  
master\_port=3306,  
master\_user='repl',  
master\_password='mysql',   
master\_log\_file='master-bin.000001',//Master服务器产生的日志  
master\_log\_pos=0;

（4）启动Slave

start slave;

OK所有配置都完成了，这时候大家可以在Master Mysql 中进行测试了，因为我们监视的时Master mysql  所有操作日志，所以，你的任何改变主服务器数据库的操作，都会同步到从服务器上。创建个数据库，表试试吧。。。

## 17. SQL中存储过程和函数的区别

本文部分内容转自<http://www.cnblogs.com/lengbingshy/archive/2010/02/25/1673476.html>

本质上没区别。只是函数有如：只能返回一个变量的限制。而存储过程可以返回多个。而函数是可以嵌入在sql中使用的,可以在select中调用，而存储过程不行。执行的本质都一样。   
     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   
      1.    一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。   
      2.    对于存储过程来说可以返回参数，而函数只能返回值或者表对象。   
      3.    存储过程一般是作为一个独立的部分来执行（EXEC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。   
      4.    当存储过程和函数被执行的时候，SQL Manager会到procedure cache中去取相应的查询语句，如果在procedure cache里没有相应的查询语句，SQL Manager就会对存储过程和函数进行编译。   
      Procedure cache中保存的是执行计划 (execution plan) ，当编译好之后就执行procedure cache中的execution plan，之后SQL SERVER会根据每个execution plan的实际情况来考虑是否要在cache中保存这个plan，评判的标准一个是这个execution plan可能被使用的频率；其次是生成这个plan的代价，也就是编译的耗时。保存在cache中的plan在下次执行时就不用再编译了。

## 18. IOC与DI的区别

### **一、IOC介绍**

1.IOC是控制反转。

2.传统资源查找方式：要求组件向容器发起请求，查找资源作为回应，容器适时返回资源。

3.IOC查找方式：容器会主动将资源提供给它所管理的组件，组件只需要选择一个合适的方式来接收资源，也被称为查找的被动式。

4.创建对象实例的控制权从代码控制剥离到IOC容器控制(之前的写法，由程序代码直接操控使用new关键字)，实际就是你在xml文件控制，控制权的转移是所谓反转，侧重于原理。

### **二、DI介绍**

1.DI是依赖注入

2.IOC的另一种表达方式：组件以一些预先定义好的方式(如：setter方法)接收来自容器的资源注入。

3.相对于IOC而言，这种表述更加直接。

4.创建对象实例时，为这个对象注入属性值或其它对象实例，侧重于实现。

#### **属性注入(最常用)**

1.通过set方法注入Bean的属性值或依赖的对象。

2.name属性：制定Bean属性set方法后的属性名。

3.value属性值：可以使(value)子节点指定属性值。

#### **构造器注入**

1.通过构造器注入Bean属性值或依赖对象，它保证Bean实例在实例化后可以使用。

2.构造器注入在<constractor-arg>一样。

### **三、区别**

1.它们是spring核心思想的不同方面的描述。

2.依赖注入和控制反转是对同一件事情的不同描述，从某个方面讲，就是它们描述的角度不同。

依赖注入是从应用程序的角度在描述，可以把依赖注入描述完整点：应用程序依赖容器创建并注入它所需要的外部资源；

而控制反转是从容器的角度在描述，描述完整点：容器控制应用程序，由容器反向的向应用程序注入应用程序所需要的外部资源。

## 19. 说说Spring中AOP和IOC的概念以及在spring中是如何应用的

**AOP:面向切面编程**

**IOC:依赖倒置或者叫做控制反转**

**关于spring中的AOP和IOC的具体含义，楼主上网查阅了相关资料，现在分享给大家:**

### ****1. IOC（Inverse of Control）：控制反转，也可以称为依赖倒置****

**所谓依赖，从程序的角度看，就是比如A要调用B的方法，那么A就依赖于B，反正A要用到B，则A依赖于B。所谓倒置，你必须理解如果不倒置，会怎么着，因为A必须要有B，才可以调用B，如果不倒置，意思就是A主动获取B的实例：B b = new B()，这就是最简单的获取B实例的方法（当然还有各种设计模式可以帮助你去获得B的实例，比如工厂、Locator等等），然后你就可以调用b对象了。所以，不倒置，意味着A要主动获取B，才能使用B；到了这里，就应该明白了倒置的意思了。倒置就是A要调用B的话，A并不需要主动获取B，而是由其它人自动将B送上门来。  
        形象的举例就是：  
        通常情况下，假如你有一天在家里口渴了，要喝水，那么你可以到你小区的小卖部去，告诉他们，你需要一瓶水，然后小卖部给你一瓶水！这本来没有太大问题，关键是如果小卖部很远，那么你必须知道：从你家如何到小卖部；小卖部里是否有你需要的水；你还要考虑是否开着车去；等等等等，也许有太多的问题要考虑了。也就是说，为了一瓶水，你还可能需要依赖于车等等这些交通工具或别的工具，问题是不是变得复杂了？那么如何解决这个问题呢？  
        解决这个问题的方法很简单：小卖部提供送货上门服务，凡是小卖部的会员，你只要告知小卖部你需要什么，小卖部将主动把货物给你送上门来！这样一来，你只需要做两件事情，你就可以活得更加轻松自在：  
        第一：向小卖部注册为会员。  
        第二：告诉小卖部你需要什么。  
  
        这和Spring的做法很类似！Spring就是小卖部，你就是A对象，水就是B对象  
        第一：在Spring中声明一个类：A  
        第二：告诉Spring，A需要B  
  
        假设A是UserAction类，而B是UserService类**

1. **<bean id="userService" class="org.leadfar.service.UserService"/>**
2. **<bean id="documentService" class="org.leadfar.service.DocumentService"/>**
3. **<bean id="orgService" class="org.leadfar.service.OrgService"/>**
5. **<bean id="userAction" class="org.leadfar.web.UserAction">**
6. **<property name="userService" ref="userService"/>**
7. **</bean>**

**在Spring这个商店（工厂）中，有很多对象/服务：userService,documentService,orgService，也有很多会员：userAction等等，声明userAction需要userService即可，Spring将通过你给它提供的通道主动把userService送上门来，因此UserAction的代码示例类似如下所示：**

1. **package org.leadfar.web;**
2. **public class UserAction{**
3. **private UserService userService;**
4. **public String login(){**
5. **userService.valifyUser(xxx);**
6. **}**
7. **public void setUserService(UserService userService){**
8. **this.userService = userService;**
9. **}**
10. **}**

**在这段代码里面，你无需自己创建UserService对象（Spring作为背后无形的手，把UserService对象通过你定义的setUserService()方法把它主动送给了你，这就叫依赖注入！），当然咯，我们也可以使用注解来注入。Spring依赖注入的实现技术是：动态代理**

### **2.AOP:即面向切面编程**

**面向切面编程的目标就是分离关注点。什么是关注点呢？就是你要做的事，就是关注点。假如你是个公子哥，没啥人生目标，天天就是衣来伸手，饭来张口，整天只知道玩一件事！那么，每天你一睁眼，就光想着吃完饭就去玩（你必须要做的事），但是在玩之前，你还需要穿衣服、穿鞋子、叠好被子、做饭等等等等事情，这些事情就是你的关注点，但是你只想吃饭然后玩，那么怎么办呢？这些事情通通交给别人去干。在你走到饭桌之前，有一个专门的仆人A帮你穿衣服，仆人B帮你穿鞋子，仆人C帮你叠好被子，仆人C帮你做饭，然后你就开始吃饭、去玩（这就是你一天的正事），你干完你的正事之后，回来，然后一系列仆人又开始帮你干这个干那个，然后一天就结束了！**

**AOP的好处就是你只需要干你的正事，其它事情别人帮你干。也许有一天，你想裸奔，不想穿衣服，那么你把仆人A解雇就是了！也许有一天，出门之前你还想带点钱，那么你再雇一个仆人D专门帮你干取钱的活！这就是AOP。每个人各司其职，灵活组合，达到一种可配置的、可插拔的程序结构。**

**从Spring的角度看，AOP最大的用途就在于提供了事务管理的能力。事务管理就是一个关注点，你的正事就是去访问数据库，而你不想管事务（太烦），所以，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/回滚事务！**

**-----乐于分享，共同进步！**

**资料转载于博客:**[**http://blog.csdn.net/eson\_15/article/details/51090040**](http://blog.csdn.net/eson_15/article/details/51090040)

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/qq_32575047/article/details/79056141>

## 20. 写一个线程死锁

在实际编程中，要尽量避免出现死锁的情况，但是让你故意写一个死锁的程序时似乎也不太简单（有公司会出这样的面试题），以下是一个简单的死锁例子。

线程的同步化可能会造成死锁，死锁发生在两个线程相互持有对方正在等待的东西（实际是两个线程共享的东西）。只要有两个线程和两个对象就可能产生死锁。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | /\*\*  \* 一个简单的死锁类  \* 当DeadLock类的对象flag==1时（td1），先锁定o1,睡眠500毫秒  \* 而td1在睡眠的时候另一个flag==0的对象（td2）线程启动，先锁定o2,睡眠500毫秒  \* td1睡眠结束后需要锁定o2才能继续执行，而此时o2已被td2锁定；  \* td2睡眠结束后需要锁定o1才能继续执行，而此时o1已被td1锁定；  \* td1、td2相互等待，都需要得到对方锁定的资源才能继续执行，从而死锁。  \*/  public class DeadLock implements Runnable {      public int flag = 1;      //静态对象是类的所有对象共享的      private static Object o1 = new Object(), o2 = new Object();      @Override      public void run() {          System.out.println("flag=" + flag);          if (flag == 1) {              synchronized (o1) {                  try {                      Thread.sleep(500);                  } catch (Exception e) {                      e.printStackTrace();                  }                  synchronized (o2) {                      System.out.println("1");                  }              }          }          if (flag == 0) {              synchronized (o2) {                  try {                      Thread.sleep(500);                  } catch (Exception e) {                      e.printStackTrace();                  }                  synchronized (o1) {                      System.out.println("0");                  }              }          }      }        public static void main(String[] args) {            DeadLock td1 = new DeadLock();          DeadLock td2 = new DeadLock();          td1.flag = 1;          td2.flag = 0;          //td1,td2都处于可执行状态，但JVM线程调度先执行哪个线程是不确定的。          //td2的run()可能在td1的run()之前运行          new Thread(td1).start();          new Thread(td2).start();        }  } |

## 21. MD5和base64的结合

### 1. MD5 是一种加密机制

通过update函数来加密。通过digest来得到加密后的字节数组。  
如：MessageDigest md5=MessageDigest.getInstance("MD5");  
md5.update(byte\_s);//这里就是需要加密的字符串参数  
md5.update(password.getBytes());//这里就是需要加密的字符串参数  
  
byte[] byte\_password=md5.digest();//这里是得到加密后的字节数组  
  
2. BASE64是一种加解密的

可以加密也可以解密。通过BASE64Encoder来加密。通过BASE64Decoder来解密。  
  
  
3.BASE64Encoder是一种加密机制

是将字节数组通过加密机制得到加密后的字符串。 byte[]---->string  
如：BASE64Encoder base64Encoder=new BASE64Encoder();  
String string\_s=base64Encoder.encode(byte\_s);  
  
  
4. BASE64Decoder是一种解密机制

是将加密后的字符串通过解密机制得到字节数组。byte[]<-----string  
如： BASE64Decoder base64Decoder=new BASE64Decoder();  
byte\_s=base64Decoder.decodeBuffer(s\_dbPassword);  
  
5.还有一个类是SecureRandom这是一种得到随机的字节数组的一个方法  
如：SecureRandom secureRandom=new SecureRandom();  
byte[] byte\_s=new byte[12];  
secureRandom.nextBytes(byte\_s);  
  
  
6.接下来就是讲MD5和BASE64结合起来来进行加密  
如：public static String create\_MD5\_BASE64\_password(String password) throws Exception{  
SecureRandom secureRandom=new SecureRandom();  
byte[] byte\_s=new byte[12];  
secureRandom.nextBytes(byte\_s);  
  
MessageDigest md5=MessageDigest.getInstance("MD5");  
md5.update(byte\_s);  
md5.update(password.getBytes());  
  
byte[] byte\_password=md5.digest();  
BASE64Encoder base64Encoder=new BASE64Encoder();  
String string\_s=base64Encoder.encode(byte\_s);  
String string\_password=base64Encoder.encode(byte\_password);  
return string\_s+string\_password;  
}  
  
  
7.这是MD5和BASE64结合起来，判断字符串和字符串加密后的是否一致

也可以理解为解密  
如：public static boolean panduan\_new\_MD5\_BASE64\_password(String password,String dbPassword) throws Exception{  
String s\_dbPassword=dbPassword.substring(0,16);  
String dbPassword\_=dbPassword.substring(16,dbPassword.length());  
byte[] byte\_s=null;  
BASE64Decoder base64Decoder=new BASE64Decoder();  
byte\_s=base64Decoder.decodeBuffer(s\_dbPassword);  
  
MessageDigest md5=MessageDigest.getInstance("MD5");  
md5.update(byte\_s);  
md5.update(password.getBytes());  
  
byte[] byte\_array=md5.digest();  
byte[] byte\_array\_=base64Decoder.decodeBuffer(dbPassword\_);  
  
if(Arrays.equals(byte\_array,byte\_array\_)){//这里数组函数。用来判断两个字节数组是否相同  
return true;  
}else{  
return false;  
}  
}

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/yumolan4325/article/details/78953238>

## 22. JSON书写格式

### JSON是什么

JSON ( JavaScript Object Notation) ，是一种数据交互格式。

### 为什么有这个技术

Json之前，大家都用 XML 传递数据。XML 是一种纯文本格式，所以适合在网络上交换数据，但是 XML 格式比较复杂，终于道格拉斯·克罗克福特（Douglas Crockford）发明了JSON 这种超轻量级的数据交换格式。

### JSON 如何使用

#### - 数据格式

JSON 实际上是 JavaScript 的一个子集，所以JSON 的数据格式和 JavaScript 是对应的:

- number => JS number

- boolean => JS boolean

- string => JS string //

- null => JS null

- array => JS Array 的表达方式 []

- object => JS {} 表达式

#### - 注意事项

* JSON 规定字符集是UTF-8，字符串必须使用""，Object 的键也必须使用""
* 数组或对象的最后一个成员，不能加 **逗号**

#### - JS 内置两个Json方法

JSON.stringify( {} , [ ] , "")

//参数一 ：要序列化的数据（object）

//参数二 ：控制对象的键值，只想输出指定的属性，传入一个数组

//参数三 ：序列化后，打印输出的格式（一个Tab ，可以更直观查看json）

任何把 JavaScript 变成 Json ，就是把这个对象序列化为Json字符串，然后才可以通过网络传递；   
收到一个Json格式的字符串，

JSON.parse( json.DATA ) //传入json字符串

如果我们收到一个JSON格式的字符串，只需要把它反序列化成一个JavaScript对象，就可以在JavaScript中直接使用这个对象了。

#### 实例

1.数组方式 [ ]

[{

"id" : 1 ,

"name" : "xiaoming"

},{

"id" : 2 ,

"name" : "xiaohong"

}]

2.对象方式 { }

//前后端分离，推荐后端返回给前端数据格式

{

"status" : 0 , //执行状态码

"msg" : "SUCCESS", //说明文字信息，没有为NULL

"data" :[{ //对象中嵌套数组，数组是返回的数据，

"id" : 1 ,

"name" : "xiaohong"

},{

"id" : 2,

"name" : "xiaoming"

}]

}

3.反例

{"id" : ox16 } //不合法，数值需要是十进制

{"name" : underfined } //不合法，没有该值

[{

"name" : NUll,

"school" : function() {

console.log("该写法是错误的")

}//不合法

}]//json中不能使用自定义函数，或系统内置函数

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/yuzhangsir/article/details/76599940>

## 23. 深入了解面向切面编程 Spring底层两种动态代理实现原理及区别

Spring提供的实现动态代理有两种方式,

一个是被代理对象需要实现JDK提供的动态代理接口。通过cglib的jar包实现动态代理,该方法只需要对目标对象继承即可

Spring支持两种方法,那么我们在使用spring进行动态代理时究竟使用的哪一种方法呢？spring优先支持实现接口的方式,如果没有接口则使用cglib方式。下面我们看一看这两种方法有什么区别。 下面我们通过JDK实现动态代理,需求是如下：有一个服务层对象ServiceImpl 如下

他们有三个方法,增删改。我们知道执行增删改操作时需要开启事务，提交事务。接下来我们通过动态代理的方式为这三个方法织如事务控制。以伪代码的形式。因为JDK动态代理需要使用接口,那我们为他创建一个接口。

接下来我们使用ServiceImpl的代理工厂生成代理对象，创建代理工厂类ServiceProxyFactory如下

第一眼看上去可能有点混乱,一步一步来,既然代理工厂生成代理对象,我们首先创建一个getInstance方法返回代理对象。JDK提供了一个Proxy下的一个静态方法newProxyInstance()，帮助我们生成代理对象。该方法需要传入三个参数,

第一个参数为的类加载器, Java有三种类加载器，这里不做重点描述,提供任意一个类的class对象有一个方法getClassLoader()返回一个classLoader对象,源码中该对象是被final修饰的,防止黑客修改类加载以达到不可告人的目的,sun公司在jvm中有着大量的代码维护反射和类加载器,防止被坏人利用。第二个参数为Class<?>[] 类型数组,什么意思呢,这里可以传入任意的Class类,那么如何获取Class类数组呢,通过任意的class对象getInterfaces()方法可以获得Class[],但是这里我们必须传入需要被代理对象的class对象。为什么呢,前面说过需要的参数为Class<?>[]，由于Class类实现了Type接口,在生成代理对象的过程中Class数组的泛型类型？会被替换成传入的class类型的类型,就是说我传入一个A.class.getInterfaces(),进入到底层时会变成Class<A>[]。因为我们这里必须传入被代理对象的class的Class[]。第三个参数需要传入一个执行器对象,该执行器必须符合InvocationHandler接口规范,就是实现这个接口的invoke方法。该方法同样要求传入三个参数,

参数一:局部变量:被代理对象实例

参数二：Method类型对象,该参数有JDK的Proxy负责传入,method为我们被代理对象的执行方法对象。

参数三：Object[],它是被执行方法参数，在方法执行中参数由Proxy负责传入至Object[]中。

那么第3个参数我们实质只需要传入被代理对象实例即可。那么传入被代理方法对象有两种方法,一个是构造器传入,另一个是set方法传入,也可以使用spring注入。我们使用method对象调用invoke()方法,该方法需要两个参数,第一个参数：全局变量的被代理对象实例,这里与全局变量被代理对象实例做一个区分帮助大家更清晰的理解。看图中,被代理对象实例由set方法传入全局变量si,然后si作为InvocationHandler的实现类下的invoke方法的参数method对象下的invoke方法参数传入。 这个参数也就是invoke的调用对象。第二个参数就是被代理对象实例调用的方法参数,也就是上方的参数三Object[]。该invoke方法返回一个Object对象,因为我使用的自动生成变量所以这个Object对象名也是invoke,不好意思！最后我们将该对象invoke返回即可。这样就可以创建一个代理对象了,我们使用动态代理的目的是在原编码不变的基础上对方法进行增强,请看下图

该方法相当于对图中三行代码进行一个封装,封装成了一个方法。执行顺序自上而下。下面看一下如果使用代理对象调用被代理对象的方法

1先创建一个被代理对象实例,用于传入代理对象工厂。

2然后创建代理对象工厂实例,为内部的被代理对象赋值。

3代理对象工厂实例调用getInstance方法生成代理对象

4代理对象调用增强后的方法

下面请看执行结果

执行的是我们增强后的方法，这就说明代理对象的作用已经达到了。这就是JDK提供的动态代理。下面我们看看由第三方人员提供的拓展类库cglib是如何实现动态代理的。

这就是cglib创建动态代理的代码,具体就不详细介绍了,只需了解即可。下面看执行代码

结果还需要看吗？

可以看到cglib的代理方式不需要传入被代理对象实例即可完成动态代理。cglib和jdk代理有什么区别呢？jdk的代理对象与被代理对象实现了同一接口,底层通过clone()方法,将实现类的方法复制过来在按照invoke中的配置对方法进行增强。而cglib代理则是代理对象继承了被代理对象的方法,然后自身进行增强。这两种代理思想在外来的自动生成代码方向会经常被使用,实现完全体代码的复用性。以上就是springAOP的实现原理了。

如果您认为本文对您产生了积极的影响请分享给其他人,我们虽然不具备发明计算机的能力但是我们可以成为更好的计算机使用使用者,cn互联网前进的每一步,都有你的一步！关注作者,每天都有最新的互联网技术,纯原创文章精心手写,能扩展的地方我都尽量扩展,大家有遇到的问题也可以找我帮助解决在评论区留言即可,让我们一起进步，感谢！

# 非面试题

## 离职原因

## 个人规划