## 说明Jsp中errorPage的作用，如何应用？

要想进行错误页的设置，需要满足以下两个条件：  
1、在错误能够处理的页上的page指令中，**增加：isErrorPage=”ture”，表示可以处理错误**。  
2、在每一个应用制定errorPage包，进行isErrorPage为true的页面。  
如果现在希望配置一种全局的错误处理的话，则可以在web.xml中进行配置。

## Statement和PreparedStatement有什么区别？哪个效率高？

PreparedStatement代表的是一个预编译的SQL。这两种对象的区别主要体现在以下的三个方面：

第一是使用方面的区别，statement执行的SQL语句必须是一个完整的SQL，而对于PreparedStatement来说，可以使用“？”作为SQL语句当中的占位符，然后使用PreparedStatement的setXXX方法来给占位符赋值，最后在执行；

第二个区别是使用Statement时，如果SQL当中出现了“‘”或者“-”等符号时，需要使用转义字符来进行转义，而在PreparedStatement当中，如果占位符的值当中有这些符号，PreparedStatement会自动的进行转义；

最后一个区别是PreparedStatement会讲SQL语句进行预编译，每次执行的时候只需要将参数设置给相应的占位符就可以运行。而使用Statement时，SQL语句时每次都要进行编译，所以PreparedStatement的效率相对较高。

防止依赖注入。

JDBC API使用Java的反射机制来实现Java程序和JDBC驱动的松耦合。所有操作都是通过**JDBC接口**完成的，而驱动只有在通过Class.forName反射机制来加载的时候才会出现。

Connection类中提供了4个事务处理方法:

* setAutoCommit(Boolean autoCommit):设置是否自动提交事务,默认为自动提交,即为true,通过设置false禁止自动提交事务;
* commit():提交事务;
* rollback():回滚事务.
* savepoint:保存点
  + 注意：savepoint不会结束当前事务，普通提交和回滚都会结束当前事务的

**JDBC要提升读取数据的性能**，可以指定通过结果集（ResultSet）对象的**setFetchSize()方法**指定每次抓取的记录数（典型的空间换时间策略）；要提升更新数据的性能可以使用PreparedStatement语句构建批处理，将若干SQL语句置于一个批处理中执行。

## [web.xml文件中可以配置哪些内容？](https://www.cnblogs.com/tiancai/p/9306147.html)

web.xml用于配置Web应用的相关信息，如：监听器（listener）、过滤器（filter）、Servlet、相关参数、会话超时时间、安全验证方式、错误页面等

## 静态和动态包含的区别与联系

　　注：下文将包含有其它文件的jsp文件称为主体文件，比如上文中的index.jsp文件。将被包含的文件称为包含文件，比如上文中的header.jsp文件。

1. <%@ include file=” ”%>是指令元素。<jsp:include page=” ”/>是行为元素
2. 最终编译成java文件的数目不同。（从上面的例子可以看出）
   1. 静态包含在转换成为java文件的时候将包含文件的内容“复制”到主体文件，然后作为一个整体编译。最终编译为一个java文件。
   2. 动态包含**是各个jsp文件分别转换，分别编译。**最终编程成多个java文件。
3. 执行时间不同

静态包含发生在：JSP---->java文件阶段。

动态包含发生在：执行class文件阶段。动态加入。

1. 静态包含在两个文件中**不能有相同的变量**，动态包含允许。

由于静态包含相当于将包含文件内容直接复制到主体文件中，如果出现相同的变量，就会出现覆盖等问题，导致文件出错。而动态包含相当于调用不同的jsp，变量所在的空间不同，自然不会出现覆盖等现象。

1. 无论是动态包含还是静态包含，其request对象都是相同的。也就是同一个request对象。

静态包含最终编译成一个java文件，有一个request对象很好理解。而动态包含最终编译成多个jsp文件，为何会使用一个request对象呢？其实这些jsp组合的过程是一个请求转发的过程，自然也使用同一个request对象了。

## redirect的状态码是多少？

forward的地址栏不会变化，响应的内容是服务器通过其他URL读取到的；

redirect的地址栏会变化，是服务器返回**301/302**状态码告诉客户端去请求新的URL地址。

## Servlet中如何获取用户配置的初始化参数以及服务器上下文参数？

可以通过重写Servlet接口的init(ServletConfig)方法并通过ServletConfig对象的getInitParameter()方法来获取Servlet的初始化参数。

**可以通过ServletConfig对象的getServletContext()方法获取ServletContext对象，**并通过该对象的getInitParameter()方法来获取服务器上下文参数。当然，ServletContext对象也可以在处理用户请求的方法（如doGet()方法）中通过请求对象的getServletContext()方法来获得。

## 使用标签库有什么好处？如何自定义JSP标签？

使用标签库的好处包括以下几个方面：  
- 分离JSP页面的**内容和逻辑**，简化了Web开发；  
- 开发者可以创建自定义标签来封装业务逻辑和显示逻辑；  
- 标签具有**很好的可移植性、可维护性和可重用性；**  
- 避免了对Scriptlet（小脚本）的使用（很多公司的项目开发都不允许在JSP中书写小脚本）

自定义JSP标签包括以下几个步骤：  
- 编写一个Java类实现实现Tag/BodyTag/IterationTag接口（开发中通常不直接实现这些接口而是继承TagSupport/BodyTagSupport/SimpleTagSupport类，这是对缺省适配模式的应用），重写doStartTag()、doEndTag()等方法，定义标签要完成的功能  
- 编写扩展名为tld的标签描述文件对自定义标签进行部署，tld文件通常放在WEB-INF文件夹下或其子目录中  
- 在JSP页面中使用taglib指令引用该标签库

## [什么是DAO模式，这样的设计模式的优点？](https://www.cnblogs.com/cst11021/p/4774954.html)

DAO模式实际上是两个模式的组合，Data Accessor模式和Active Domain Object模式。

Data Accessor模式：封装了数据访问实现的机制，通过提供黒盒式数据存取接口，实现数据访问和业务逻辑的分离。

Active Domain Object模式：**实现业务数据的对象化封装。**

**数据存储逻辑的分离；数据访问底层实现的分离；资源管理和调度的分离；数据抽象**：通过对底层数据的封装，开发人员可以使用面向对象思想对数据进行操作。

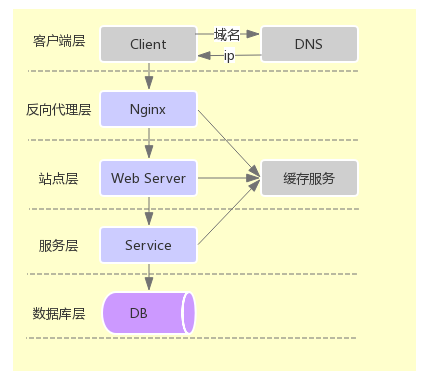
**dao是数据持久化层，负责数据库操作，model负责数据模型，和数据表对应。**

## Spring MVC注解的优点

* 注解最大的好处就是简化了XML配置；其实大部分注解一定确定后很少会改变
* 它可以充分利用 Java 的反射机制获取类结构信息，这些信息可以有效减少配置的工作。
* 编译期校验，错误的注解在编译期间就会报错。注解在java代码中，从而避免了额外的文件维护工作。

## 如何设计一个秒杀系统

秒杀系统不单单适用于电商的抢购场景，其实一切涉及到**大并发的场景**都适合于使用该套秒杀系统方案。



（1）客户端层：手机或PC端操作的客户端页面，域名通过DNS解析路由到NG

（2）反向代理层：一般通过NG作为反向代理，**将客户端请求**均衡路由到后端站点服务，NG也可以**水平扩展为多实例**，且每个实例可单独部署为主从的高可用方案。

（3）站点层：站点层可以水平扩展为多个实例部署，以此来均衡来自客户端请求产生的**高并发负载，**多个web server之间的session信息可以集中存储于**分布式缓存服务（Redis**，MemCache）中。

（4）服务层：服务层也可水平扩展为多**个实例部署**，即时下最火的**微服务**方式

（5）数据库层：数据库层的常见部署方式，如**读写分离，分库分表**等

**秒杀系统的难点**

**1）瞬时大并发**：如果系统没有经过限流或者熔断处理，那么系统瞬间就会崩掉，就好像被DDos攻击一样；**尽量将请求拦截在上游。**

**2）超卖**：

**3）性能**：当遇到大并发和超卖问题后，必然会引出另一个问题，那就是性能问题，如何保证在大并发请求下，系统能够有好的性能，让用户能够有更好的体验，不然每个用户都等几十秒才能知道结果，那体验必然是很糟糕的；**读多写少的业务场景充分利用好缓存**

**秒杀系统方案**

从整个秒杀系统的架构其实和一般的互联网系统架构本身没有太多的不同，核心理念还是通**过缓存、异步、限流来保证系统的高并发和高可用。**下面从一笔秒杀交易的流程来描述下秒杀系统架构设计的要点：

**浏览器端：**

1）对于大促时候的秒杀活动， 一般运营会配置静态的活动页面(**按钮置灰：禁止用户重复提交请求; 通过JS控制：在一定时间内只能提交一次请求**)，配置静态活动页面主要有两个目的一方面是为了便于在各种社交媒体分发，另一方面是因为秒杀活动页的流量是大促期间最大的，**通过配置成静态页面可以将页面发布在公有云上动态的横向扩展；**

2）将秒杀活动的静态页面提前刷新到**CDN节点**，通过**CDN节点的页面缓存**来缓解访问压力和公司网络带宽，CDN上缓存js、css和图片；**Nginx反向代理实现web server端的水平扩展**

3）将**活动H5页面部署在公有云的web server上**，使用公有云最大的好处就是能够根据活动的火爆程度动态扩容而且成本较低，同时将访问压力隔离在公司系统外部；

**服务端控制器层**

**在服务端控制层需要针对同一个访问uid，限制访问频率。**

4）在提供真正商品秒杀业务功能的app server上，需要进行**交易限流、熔断控制**，防止因为秒杀交易影响到其他正常服务的提供，我们在限流和熔断方面使用了hystrix，在核心交易的**controller层通过hystrix进行交易并发限流控制，**当交易流量超出我们设定的限流最大值时，**会对新交易进行熔断处理固定返回静态失败报文,** **实现削峰的常用的方法有利用缓存和消息中间件等技术。**

5）服务降级处理，除了上面讲到的限流和熔断控制，我们还设定了降级开关，对于首页、购物车、订单查询、大数据等功能都会进行一定程度的服务降级，例如我们会对首页原先动态生成的大数据页面布局降级为所有人看到的是一样的页面、购物车也会降级为不在一级页面的tabbar上的购物车图标上显示商品数量、历史订单的查询也会提供时间周期较短的查询、大数据商品推荐也会提供一样的商品推荐，通过这样的降级处理能够很好的保证各个系统在大促期**间能够正常的提供最基本的服务，**保证用户能够正常下单完成付款。

6）**防止超卖**一般是通过在数据库上**实施乐观锁**来完成，使用乐观锁虽然比for update这种悲观锁方式性能要好很多，但是还是无法满足秒杀的上万并发需求，我们的方案其实也很简单实时**库存的扣减在缓存中进行，异步扣减数据库中的库存，保证缓存中和数据库中库存的最终一致性。**

分布式缓存是redis，使用了**redis集群**方案稳定性和高可用方面还是比较有保证的，因为**redis是单线程写，所以也不用担心线程安全的问题，redis自身就能够保证数据的强一致性，**在下单的事务中包含了**实时扣减缓存中的库存和异步发送队列**(采用消息队列缓存请求)，由队列处理器再异步从队列中取出订单根据订单信息扣减库存系统数据库中的商品数量。

**使用队列处理**：将请求放入队列排队处理，以可控的速度来访问底层DB

利用消息中间件和缓存实现简单的秒杀系统

Redis是一个分布式缓存系统，支持多种数据结构，我们可以利用Redis轻松实现一个强大的秒杀系统。

我们可以采用Redis 最简单的key-value数据结构，用一个原子类型的变量(AtomicInteger)作为key，把用户id作为value，库存数量便是原子变量的最大值。对于每个用户的秒杀，我们使用 RPUSH key value插入秒杀请求， 当插入的秒杀请求数达到上限时，停止所有后续插入。

然后我们可以在台启动多个工作线程，使用 LPOP key 读取秒杀成功者的用户id，然后再操作数据库做最终的下订单减库存操作。

当然，上面Redis也可以替换成消息中间件如ActiveMQ、RabbitMQ等，也可以将缓存和消息中间件 组合起来，缓存系统负责接收记录用户请求，消息中间件负责将缓存中的请求同步到数据库。

## IoC容器的设计与实现

<https://blog.csdn.net/u010994169/article/details/71169986>

Spring可以理解为一个工厂，**负责对象的创建和对象间关系的维护。**

IoC是对对象进行控制反转,也成为依赖注入.依赖控制反转(把对象的控制权从业务的业务对象手中转交给平台或者框架)的模式有很多种,在spring中IoC容器是实现这个模式的载体.

    依赖注入的方式:1.接口注入;2.setter注入;3.构造器注入.IoC设计中,setter和构造器注入是主要的注入方式。

容器中起作用的主要数据类型是**BeanDefinition**,它是对依赖反转模式中**管理的对象依赖关系的数据抽象**,依赖反转功能都是围绕着对这个BeanDefinition的处理完成的。

1.首先创建一个Student类和StudentService类如下：

Student类，name，add，selfIntroduce();

public class StudentService {

private Student student;

public Student getStudent() {

return student;

}

public void setStudent(Student student) {

this.student = student;}

2.接着编写Spring的配置文件：application.xml.

<?xml version="1.0" encoding="UTF-8"?>

<beans>

<bean id="Student" class="cn.shu.pojo.Student">

<property name="name" value="Hover"/>

<property name="add" value="China"/>

</bean>

<bean id="StudentService" class="cn.shu.service.StudentService">

<property name=“Student” ref="Student"/>

</bean>

</beans>

3. Ioc最为核心的部分**ClassPathXMLApplicationContext**。这个类采用dom4j对application.xml文档进行解析。大致思路是：**通过读取配置文件中定义的全路径类名，利用java反射技术创建该类的实例。**首先定义HashMap用于存储创建的实例。Key值就是配置文件的<bean>标签“id”,value值就是对应的实例。然后该类需要提供  public Object getBean(String name)方法返回实例对象。

public class ClassPathXMLApplicationContext implements ApplicationContext{

public ClassPathXMLApplicationContext(String config\_file) {

URL url = this.getClass().getClassLoader().getResource(config\_file);

file = new File(url.toURI());

XMLParsing();}

String cls = bean.getAttributeValue("class");**//全路径类名**

Object **obj** = Class.forName(cls).newInstance();

Method[] method = obj.getClass().getDeclaredMethods();

List<Element> list = **bean.getChildren**("property");

if (name.startsWith("set")) {、、、、、

method[n].invoke(obj, el.getAttribute("value").getValue());

method[n].invoke(obj,map.get(el.getAttribute("ref").getValue()));

map.put(id, obj);}

public Object getBean(String name) {

return map.get(name);

}