## 基础篇

### jdk jre jvm三者之间的关系

java开发者工具包 java运行环境 java虚拟机

jdk>>jre>>jvm

### 面向对象(oop)

面向对象是对面向过程的封装,封装了类的属性和方法

### 对象创建的过程

对象创建之前,首先类被加载(先加载父类再加载子类),执行静态代码块,执行构造器((先构造父类再构造子类))。创建对象实例静态方法和类有关,直接使用类名.方法来进行调用

### java的反射机制

当类加载器将类加载进jvm之后,jvm会创建每一个类的元数据对象(Class),java语言允许通过元数据对象动态的创建对象实例,这种机制就称为java的反射机制,基本上所有框架的底层都用到了反射机制,spring、mybatis、servlet都用到了.

### 抽象类和接口有什么区别?

抽象类可以有抽象方法,也可以没有抽象方法。但是包含抽象方法的类必是抽象类。可以有成员变量。接口当中只能有抽象方法,不能有普通方法,可以有静态常量,接口可以看做是一种特殊的抽象类，它们都不能被实例化。

### 方法的重写与重载有什么区别?

重写是指子类继承父类,重写父类的方法

重载是指在一个类中,可以有多个重名的方法,方法名相同,但是形参列表不同

### java当中的多态是什么?

多态是指可以创建父类对象的引用指向子类的对象,方法的调用只能调用父类的方法

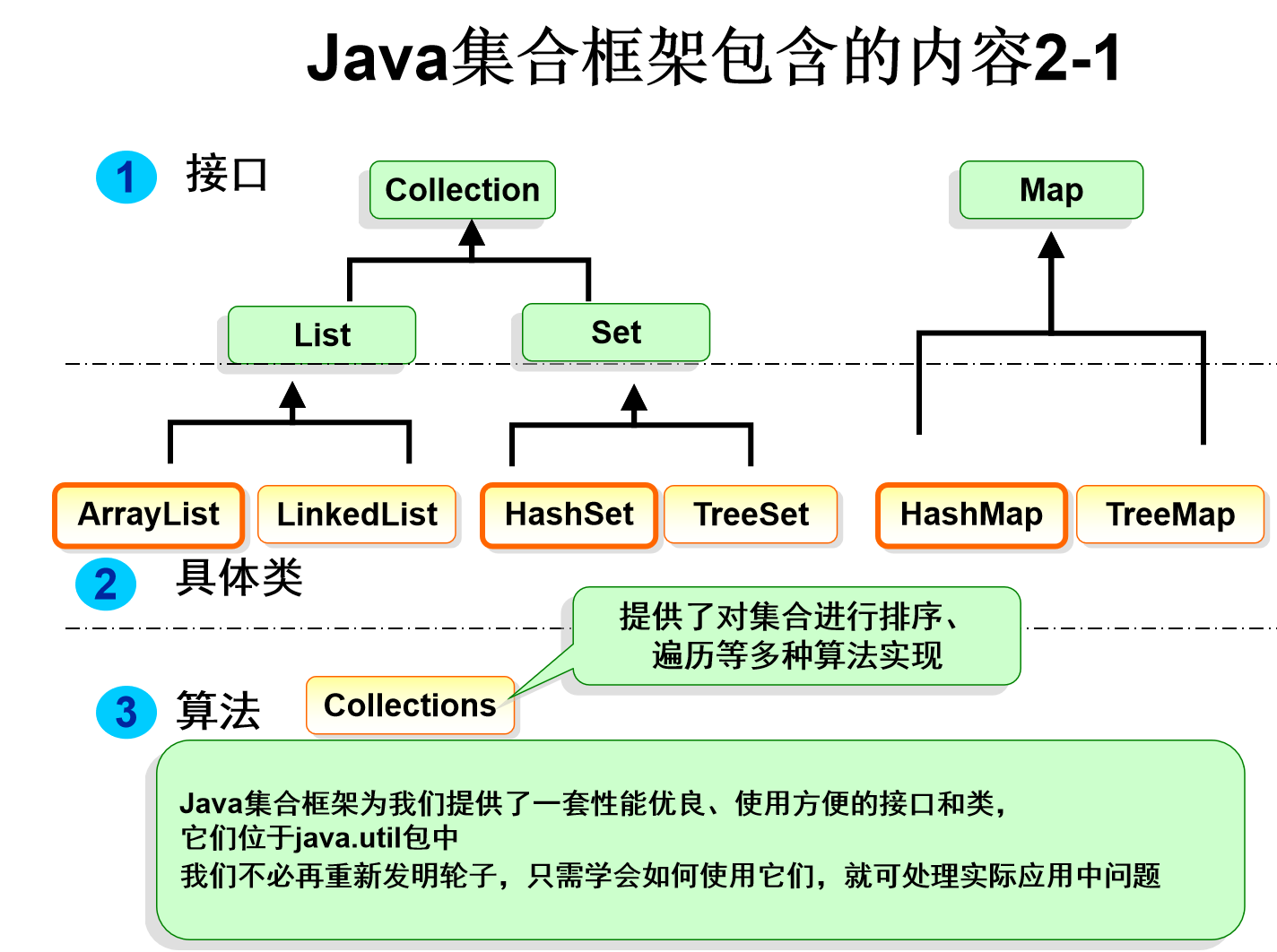
### 框架是如何反射创建对象的呢?

框架的配置有xml形式、注解形式

xml可以被解析,解析的技术有dom4j、sax解析

三大框架使用的解析方式是dom4j。

### 集合类



#### 9.1常用的数据结构 数组和链表

#### 9.2数组和链表有什么区别?

数组会在内存当中开辟一块连续的存储空间,需要指定长度

数组当中的每一个元素都有索引,查询的话通过索引可以直接定位到

某一个元素,效率很高,但是进行删除的话,数组会进行移动,

所以效率很低.

链表不需要连续的存储单元,链表中的上一个元素通过指针指向

下一个元素,所以链表结构进行查询的话,头部尾部比较快,

中间元素查找速度比较慢,但是删除的话,只需要删除前后指针,重新

建立指针就可以了,所以删除的性能很高.

#### 9.3 java当中的集合类或集合框架（重点）

collection接口是集合类的顶层接口,collections是工具类

collection有两个子接口,一个list接口,一个是set接口

list接口有序可重复,set接口无序不可重复

list接口常用的实现类有

arrayList 基于数组实现的

linkedList 基于链表实现的 区别参考数组和链表区别

vector 基于数组实现的,但是是线程安全的(synchronized)

copyOnwriteArrayList 复制数组副本来实现的

set接口常用实现类

HashSet 基于hashMap来实现的,实现不可重复是通过 hashCode方法和equals方法进行两次比较,先比较hashCode,再通过equals

TreeSet 基于二叉树来实现的,可以对元素进行排序(排序规则默认是自然顺序,可以自定义比较器,实现自定义排序)

Map接口(key value结构的)

map接口的实现类

HashMap key不可重复,无序,可以为null

实现原理 基于数组和链表来实现的

当存入一组键值对的时候,先对key进行hash,然后映射到一个初始化长度为16的数组上,当不同的key产生hash碰撞的时候,value会通过链表结构来进行存储,jdk1.8之后对hashMap进行了改进,当链表长度达到临界值8,会通过红黑树来存储value,hashMap有两个参数,一个是初始化数组长度16,负载因子0.75,当满足扩容阈值的时候(当数组的12个元素被存满,并且有hash碰撞了),动态扩容,以2倍的增长方式进行扩容.

HashTable

HashTable是线程安全的hashMap（synchronized机制）key不准许为null

TreeMap基于二叉树来实现的,可对key进行自然排序（自定义比较器,写比较规则）

ConcurrentHashMap是线程安全的,对整个hash桶采用分段策略,拆分成若干个段segment,对每一个段上锁（synchronized）,极大的提高了并发修改的效率.

### 线程

#### 10.1进程和线程有什么区别?

进程是指应用程序在运行时的状态,进程会占用系统的cpu以及内存资源

线程是指进程当中的执行流程,多个线程协同工作,会共同使用进程的资源,

才确保进程的功能得以实现.

#### 10.2 java当中如何实现线程呢?

1:继承Thread类

2:实现runnable接口,只是任务类,还需手动创建线程对象

3:线程池机制来创建和管理线程 ThreadPool

singleThreadPool 创建一个线程的线程池

fixedThreadPool 创建固定数量的线程池

cachedThreadPool 数量可缓存的线程池

ScheduledThreadPool 有定时功能的线程池

#### 10.3线程的生命周期

新建状态(new一个线程的时候)

调用线程.start方法的时候,等待cpu分配时间片

可运行状态(就绪状态）

当cpu分配到了时间片之后,线程开始运行,运行run方法当中的内容

运行状态

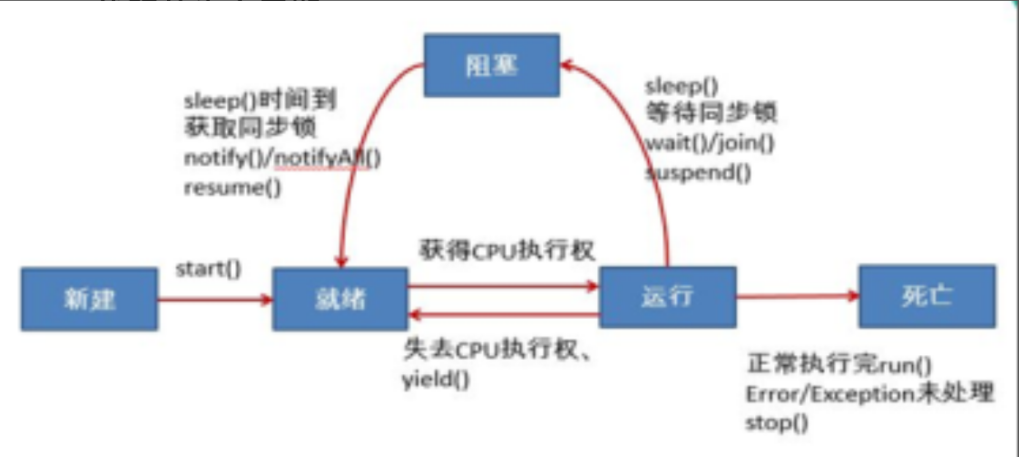
当线程遇到sleep、wait等方法的时候,线程会进入阻塞状态

阻塞状态

当休眠时间到期,或者被notify了,线程又回到了可运行状态

死亡状态

线程运行结束之后,就会销毁,被jvm的gc回收



#### 10.4多线程并发或线程安全问题如何解决?（重点）

1:通过volatile 关键字修饰变量,可以实现线程之间的可见性,

避免变量脏读的出现,底层是通过限制jvm指令的重排序来实现的

适用于一个线程修改,多个线程读的场景

2:通过synchronized锁(任意对象)来实现线程同步,自动锁的思想,

底层实现原理:当有线程进入同步代码块之后,利用jvm的计数器将

锁的标记置为1,当别的线程再想进入的时候,发现锁的标记为1,

该线程就去锁池等待,当第一个线程出来之后,锁的标记会置为0,

之后cpu会随机分配一个线程再次进入同步代码块.

3:通过lock锁的机制,进行手动lock,和unlock,但是这种很容易出现死锁。

注意加锁以及解锁的顺序,就可以避免死锁

4:通过线程安全的集合类,可以解决并发问题

ConcurrentHashMap

CopyonWriteArrayList

5:使用并发包下面的原子类,底层使用的是cas机制（乐观锁）,可以解决并发问题 atomicInteger 线程安全的原子整型类

6:使用线程池来创建和管理线程,也可以一定程度上解决并发问题

7:使用ThreadLocal来修饰变量,可以解决并发问题

ThreadLocal底层是怎么实现的?

多个线程会复制一份threadLocao变量的副本进行操作,互不影响,来保证线程安全的

#### 10.5线程之间如何进行通信?

1:使用synchronized锁的wait和notify进行等待和唤醒

2:使用lock锁的condition的await和signal进行等待唤醒

#### 10.6悲观锁和乐观锁

悲观锁的思想对线程并发持悲观态度,使用互斥锁的机制来解决并发问题

乐观锁的思想是使用cas(compareAndSwapper)机制,不上锁,但也能解决并发问题

int count=100;

最后一次修改的时间戳或者版本号;

1:先查询一下变量的值以及它的最后一次修改的时间戳;

2:在本地修改变量的值

3：对变量进行修改的时候,先比对一下最后一次修改的时间戳是否发生变化了,如果没发生变化,修改变量的值, 如果发生变化了,重试以上流程

线程的使用场景:

1:项目当中使用线程池(注册完毕发邮件和短信)

2:tomcat本身就是通过线程的机制来处理用户的请求

IO(input输入 output输出)

数据存储的形式

文件形式 类似于湖泊

IO流的形式 类似于河流

File file=new File("d:\\1.txt");

FileInputStream in=new FileInputStream("d:\\1.txt")；

java当中IO流的分类

字节流 InputStream(抽象类) OutputStream

FileInputStream FileOutputStream (实现类)

装饰者模式对传统的字节流进行了封装,增加了缓冲区功能

BufferedInputStream BufferedOutputStream(带有缓冲区的字节流)

字符流 Reader writer

FileReader FileWriter

BufferedReader BufferedWriter

### 网络协议

#### 11.1iso7层模型

物理层 数据链路层 网络层 传输层 会话层 表示层 应用层

tcp/udp属于传输层的协议

#### 11.2 tcp和udp有什么区别?

tcp是面向连接的,建立连接前需要进行三次握手,断开连接前需要进行

四次挥手,可靠的不会造成数据丢失的,可以简单理解成打电话,效率低

udp协议是面向无连接,不可靠有可能造成数据丢失,可以理解成发电报,

效率高

#### 11.3 http协议和https协议区别

http协议属于应用层的协议(封装的tcp),现在大多的互联网都是通过Http协议进行连接,是免费协议,https协议是安全的http协议,此协议是采用对称加密算法来保证数据传送的, 安全性,是收费协议

#### 11.4请详细描述下一次http请求到最后响应这之间的过程?

客户端发起http请求,因为http协议是基于tcp协议,会首先经过dns解析,找到目标服务器,发起tcp连接进行三次握手,之后连接进入tomcat,tomcat会对连接进行处理,处理完毕之后响应一个html文档给客户端,客户端通过浏览器进行可视化,断开连接的时候进行四次挥手.

#### 11.5简述一下tcp协议三次握手或四次挥手的过程?

1:客户端发送syn包

2:服务端接收到syn包之后,回复syn+ack

3:客户端接收到服务端的syn+ack之后,会回复一个ack

### Javaweb

#### 12.1Javaweb的核心

Socket是web的核心,线程是程序运行的根本

客户端发送一个请求-->tomcat（socketserver监听8080端口号）-->创建一个线程

运行业务逻辑代码(servlet引擎)

servlet是javaweb开发的核心,充当控制器用来接收和处理用户的请求

#### 12.2 servlet的生命周期:

新建

初始化

提供服务

销毁

#### 12.3既然是单例线程安全如何保障?

1:尽量不要使用全局变量

2:请看java并发编程

#### 12.4协议有哪几种请求方式？

get post put delete

get和post请求是最常用的请求方式

#### 12.5 get和post请求有什么区别?

get请求是通过请求行url来传输参数的,安全性比较低,传输参数的大小不超过2KB效率快

post请求是通过请求正文来提交数据的,安全性比较高,传输数据的大小不受限制,效率低



#### 12.6请求的转发和重定向有什么区别?

请求的转发使用的是同一个request对象,地址栏不发生改变

请求的重定向本质是两次请求,地址栏发生改变

#### **12.7 servlet当中有哪些域对象?**

HttpServletrequest Httpsession servletContext

request的作用域是一次请求响应内

session的作用域是一次会话

application的作用域是从tomcat启动到tomcat关闭

HttpSession是tomcat提供以及创建的用于存储客户端会话信息的对象

HttpSession的生命周期由tomcat进行管理,存储在tomcat内存当中.

tomcat默认session的有效时间是30分钟,可以通过web.xml或代码

去手动设置session的失效时间.

request.getSession(true); 去取到当前用户的session对象,如果不存在则创建一个返回

request.getSession(false);去取到当前用户的session,如果不存在则返回null。

#### **12.8 HttpSession的实现原理？为什么tomcat可以根据不同的用户取到不同的session？**

session是通过cookie来进行对应的存取的,当tomcat创建用户的session的时候,会向浏览器响应一个记录着该用户sessionId的cookie,("JSESSIONID","5435dsadfdsfsfdsf")

浏览器接收到cookie之后会将cookie保存在浏览器内存中,以后该客户端每次访问tomcat的时候,都会携带着该cookie。

#### 12.9 Cookie是什么？

HttpCookie是存储在浏览器端的会话对象,cookie可以存储在浏览器的内存或硬盘中.

cookie只能存储字符串类型的键值对("JSESSIONID","5435dsadfdsfsfdsf")。

cookie常用于记住密码、购物车等功能的实现.

#### 12.10 Filter

filter是tomcat提供用于资源保护的过滤器,可以保护项目当中的资源(静态资源、动态资源).

filter的使用根据业务场景,可以进行多层拦截,从外到内.

#### 12.11 Listener

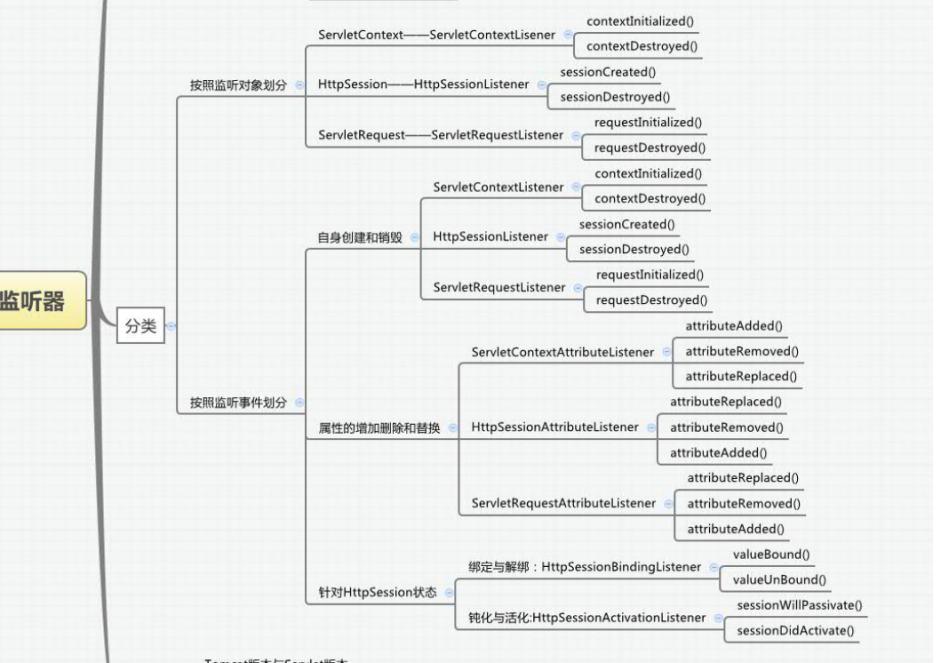
监听器机制是tomcat提供的用来监视域对象的,request、session、servletcontext

监听request的话,统计每天项目的访问量 PV ,每次请求的处理时间

监听session的话,可以统计出用户的上下线时间,以及在线用户等功能

监听servletContext 可以做一些初始化工作 比如spring的监听器 7

文本的传输格式

2

## HTML、XML、JSON

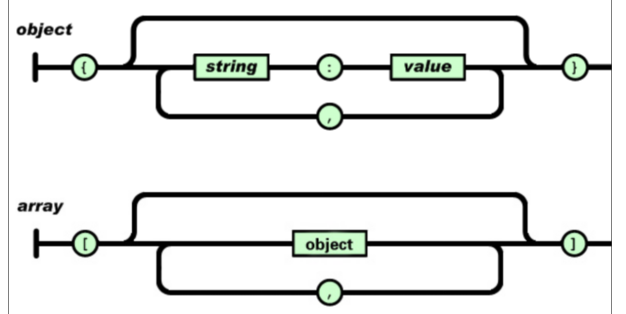
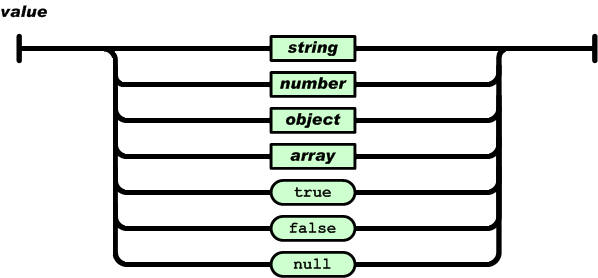
HTML 标签固定,不区分大小写,除此之外和XML一样

XML 标签随意,严格区分大小写,有且只有一个根标签

xsd、dtd文件都是xml的约束文件,约束了xml中可以使用哪些标签,标签的使用顺序

JSON 语法结构简洁、轻量级 {"username":"张三","password":"123456","teacher":{"teacher\_name":"涂旭","age":"18"}}

{属性名1:属性值1,属性名2:属性值2} 属性和属性值必须得"" ,属性值支持嵌套



#### 12.12 同步与异步的区别

同步请求 同步请求会使客户端处于阻塞状态,直至服务端响应完毕,

基本上所有的请求都是同步请求,类似于打电话

异步请求 异步请求不会使客户端处于阻塞,把请求发出去,然后服务端响应之后会进行异步回调把数据传送给客户端,类似于寄信

ajax请求就是异步请求,用来局部刷新,隐形请求

## 框架篇

### 框架是什么东西?

框架是一组抽象封装的设计,是一种软件半成品,程序员必须得遵守框架的api,

配置配置文件和遵循代码规范,极高的提高了开发效率.

### Spring

#### 1.1 spring的特性?谈谈你对spring的理解?spring用来干嘛?

**spring有两大核心,IOC（Inversion of Control控制反转）和AOP(Aspect Oriented Programming面向切面)**

IOC即控制反转,通常我们在项目当中需要手动new去创建对象,这种方式不利于

对对象的管理,现在我们将对象的创建权利反转给spring容器,这就是控制反转。

spring容器在实例化对象的时候,会根据对象之间的依赖关系,自动完成属性

注入工作,这就是**依赖注入(DI:Dependency Injection )**

AOP即面向切面编程,底层是通过动态代理的机制来实现的,**支持jdk和cglib两种**。

**默认通过jdk动态代理**。通常我们在项目当中,一些公共功能的实现可以通过aop来

进行解耦和统一实现,比如事务管理、日志、权限等等。

我们在项目当中的事务管理是这样配置的

1:通过配置文件进行配置，此时项目自动应用配置的事务特性。

2：通过注解进行配置，此时每个类或类中的方法都要添加注解

#### **1.2 jdk的动态代理和cglib动态代理有什么区别?**

jdk的动态代理要求业务类必须得实现业务接口,底层是通过生成业务接口的动态代理实现类来完成功能增强

cglib不需要业务类实现接口,底层是通过衍生出当前业务类的子类对象来完成功能增强

#### 1.3 你看过spring的源码么？说说aop的源码?

平常看过一些博客,了解一部分spring的源码,是spring的ioc有两大核心

接口**beanFactory和applicationContext,**beanFactory是通过工厂模式来生产bean,**默认生产的是单例bean**。

applicationContext是beanFactory的子接口,增强了BeanFactory的功能。通常我们

都是使用applicationContext,其中ClassPathXmlApplicationContext

来初始化IOC容器

#### 1.4 Aop的源码

aop通过设置切面,当切面设置的目标类的方法被调用时,aop框架会拦截

此次调用,我看过pointCut类的源码,里面有两个核心属性,ClassFilter 类过滤器

MethodMatcher 方法匹配器,所以aop来可以进行拦截.拦截之后,aop机制会通过

jdk或cglib帮咱们生成动态代理对象,调用增强类的增强方法,进行功能织入。

#### 1.5 aop的实现原理

首先通过aop配置切面以及切面的增强逻辑，当spring的容器中目标类的对象的方法被调用时，aop框架就会拦截此次调用，aop框架会帮你生成一个动态代理对象，先运行增强类中的前置增强方法，再运行原业务类的业务方法，最后再运行增强类的后置增强方法

注意：aop默认通过jdk提供的proxy来生成动态代理对象，但是也可以手动指定使用cglib来生成动态代理对象

#### 1.6 **spring当中bean的作用域有哪些?**

singleton 单实例

protoType 多实例

request 请求结束就销毁

session 会话结束就销毁

globalSession 全局

#### 1.7 **spring当中bean的生命周期**

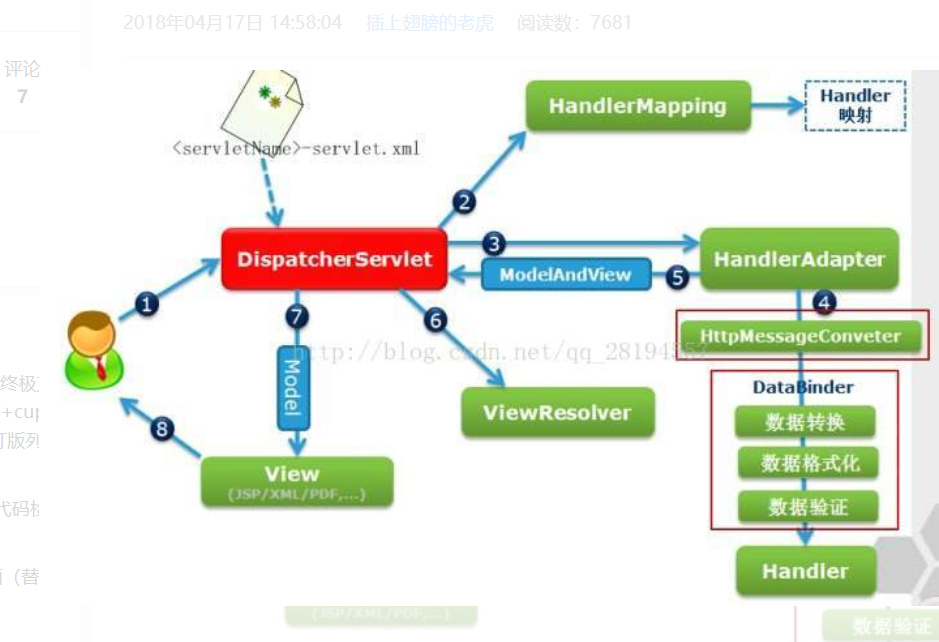
bean的创建 依赖注入 功能增强 根据作用域进行销毁

#### 1.8 spring当中的单例,是如何保障它线程安全的？

spring是通过ThreadLocal来保障单例线程安全的

### 2. Springmvc

springmvc是spring旗下的开发mvc的一个模块,底层封装的是servlet,完成了对servlet的优化处理。



#### 2.1 springmvc的原理？springmvc的工作流程?springmvc的源码你看过吗？（重点）

首先请求到达核心控制器dispatcherServlet,dispatcherServlet会判断请求

后缀是否满足格式要求,如果满足格式要求,会查询handlerMapping,查找

对应的handler,调用handlerAdapter进行参数绑定等适配工作,之后调用具体的

handler进行业务逻辑处理,返回一个modelAndView对象给dispatcherServlet。

dispatcherServlet调用viewResolver进行视图解析渲染,然后返回给客户端

#### 2.2 springmvc的常用注解有哪些?（重点）

Controller

RequestMapping

RequestBody

ResponseBody

PathVariable

RequestParam

#### 2.3 springmvc的拦截器怎么写?

定义一个类,**实现了handlerInterceptor接口**,重写 preHandle postHandle

afterCompletion 三个方法,之后在springmvc的配置文件当中 使用

**<mvc:interceptors>**里面可以配置多个interceptor

#### 2.4 springmvc和struts2有什么区别?

**springmvc的核心控制器是servlet**,**struts2**的核心过滤器是 strutsPreparedAndExcute**Filter**

**springmvc的handler是单例的存在,面向方法编程的设计**,struts2的处理器

是多实例,是面向类的开发,每次请求都会创建一个处理器,oop的设计思想

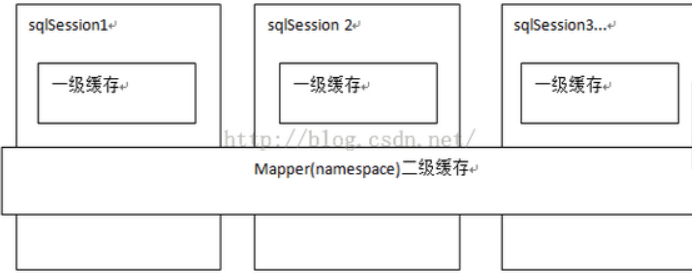
### 3.Mybatis

mybatis是一个半自动化轻量级的**ORM(Object Relational Mapping)**框架,程序员可以手写sql语句,使用起来比较灵活

**3.0 mybatis的架构**



#### 3.1mybatis的二级缓存



**mybatis是自动开启一级缓存的,sqlsession级别**

**mybatis的二级缓存需要手动开启的,mapper级别(application应用),**

通常项目当中,对于字典表(大量查询,很少修改)的数据,可以使用mybatis的二级缓存机制,

提高查询效率。比如省市县、汽车品牌、配件类别、企业信息等等

### 3.2 使用Mybatis的二级缓存

1）在核心配置文件SqlMapConfig.xml中加入<setting name="cacheEnabled" value="true"/>

2）在UserMapper.xml中开启二缓存<cache></cache>

3）结果映射的pojo实现 java.io.serializable接口

#### 3.2 mybatis如何主键回填?

在insert标签里面,加入**<selectKey>**select last\_insert\_id()</selectKey>标签

#### 3.3 **mybatis的mapper代理底层怎么实现?**

mybatis是先通过xml解析,反射,最后通过动态代理机制来生成mapper接口的实现类对象namespace的值等于mapper接口的类路径、sql标签的id值等于mapper接口的方法名

#### **3.4 mybatis使用了哪些设计模式？你了解过哪些设计模式?**

mybatis当中 **sqlSessionFactoryBuilder 建造者模式**

**sqlSessionFactory 单例 singleTon**

**sqlSession 工厂模式 beanFactory**

**mapper 动态代理 springaop**

springmvc handlerAdapter 适配器模式

BufferedReader 装饰者设计模式

思考题:装饰者模式和动态代理有什么区别?

#### 3.5 **#{}和${}有什么区别(重点)**

**#{}代表占位符传参**,底层是通过preparedStatement来实现的,对sql语句预编译

**可以防止sql注入,更安全**

**${}代表字符串拼接,不安全**

#### 3.6 **mybatis的常用sql标签有哪些**

<select><insert><update><delete><sql>

动态sql

<if><foreach><where><set>

### **4.hibernate**

hibernate是全自动的orm框架,程序员不需要手写sql,通过hql语句来进行

查询以及结果集自动映射。

#### 4.1 **hibernate的对象状态**

临时状态

持久化状态

托管状态

#### **4.2 hibernate的缓存机制**

一级缓存 session级别

二级缓存 sessionFactory级别

#### 4.3 **mybatis和hibernate有什么区别?**

互联网项目使用Mybatis居多

mybatis 手写sql 更加灵活, 对于复杂查询的支持较好

企业的办公软件用的比较多

hibernate 通过对象关系操作数据库 单表操作,比较高效,多表查询支持不好

## 数据库篇

### 你是怎么设计数据库的？

设计数据库首先要遵循三大范式要求：原子性、依赖性、关联性

1.原子性是指数据库表的每一列都是不可分割的原子数据项

2.依赖性是指实体的属性完全依赖于主关键字。

3.关联性是指任何非主属性不依赖于其它非主属性

对于数据库设计来说，不仅仅要考虑范式要求，为了节省查询效率，允许适当的有一些冗余字段。

### 关系型数据库

关系型数据是面向对象的设计,每一张表都是用来描述一类事物的,每一列都是用

来描述事物的属性的,每一行都是一条记录、一个对象。以下是常用的:

mysql mysql属于oracle旗下的免费的轻量级的 (使用面广,互联网公司)

oracle 收费的重量级的数据库(常用于银行、保险、石油这些项目)

sqlServer (windows平台的开发 、net的语言)

DB2 类似于oracle（银行项目）

#### 1.1数据库索引是什么?

索引是数据库提供的利于快速查询的机制,索引类似于书签目录,

当查询条件那一列做了索引之后,那么数据库会去硬盘索引文件

当中找到满足条件记录的物理位置,直接就可以定位以及获取记录.

#### 1.2索引的种类有哪些?

1:主键索引

2:唯一索引

3:单列索引

4:外键索引

前面四种命中规则,查询条件当中包含该列 列名=列值

5:组合索引 命中规则是最左原则 比如对 a b c 三列创建组合索引,

一旦使用到a这一列就会命中组合索引

select \* from items where a=? and b=? 命中索引

select \* from items where a=? 命中索引

select \* from items where b=? and c=? 没有命中

#### 1.3 sql怎么调优,索引命中怎么知晓？（重点）

使用explain关键字查看sql的执行计划,可以看到该sql的

索引命中情况、索引命中类型、命中率等相关的信息,通过这种方式

可以对sql语句进行调优,对关键查询的条件创建索引以及注意一下sql语句的

用法

尽量不使用like,尽量避免在某一列上进行运算,

不使用 in not in 使用exist not exist等等代替

#### 1.4索引的常用数据结构?

hash索引 hash 均匀分布

B树索引 B树 分布有序

比如我对A列分别建立Hash索引和B树索引,当我频繁对A列的数据进行修改的时候,

性能高,但是如果按照范围进行查找的话,B树索引的性能高(where A>100 and A<1000)

#### 1.5索引有什么优缺点?

索引可以提高查询的效率,但是对记录进行增删改的时候,由于数据库需要去维护

索引文件,所以说如果频繁修改记录的话,影响数据库性能。我们在项目当中创建

索引的时候,只是针对于那些频繁查询的字段创建索引,通常情况下创建索引的列

不会超过6个

#### 1.6 sql语句(笔试题甚至面试时)

group by 分组 子查询 内连接 常用函数 count

#### 1.7你用过存储过程么?

存储过程是预编译的,类似于函数,存储过程可以写sql语句加流程语句

create Producer A(输入参数,输出参数)

begin

//可以写sql while if语句

end

call A();

当时我们数据库当中做了一个批量更新,使用到了存储过程

存储过程的优点,一次编写,多次调用；缺点在分布式环境下,移植性有问题

单库使用存储过程比较多.

#### 1.8 mysql的存储引擎有哪些,有什么区别

常用的就是InnoDB 数据库默认引擎是InnoDB ,它对大规模写入的支持较好,支持数据库事务,支持外键,主要用于主从机构当中的主服务器

MYISAM 它对大规模读的支持比较好,主要用于从服务器

#### 1.9数据库的事务隔离级别

有4种:

读 未提交

读 已提交

可重复性读

串行化

mysql默认隔离级别 可重复性读

#### 1.10 Oracle的主键自增以及分页

通过序列 sequence 来实现自增

oracle不支持limit语句的,需要通过伪列rownum 进行两次子查询来实现分页

### 非关系型数据库

#### 什么是NoSQL

NoSQL,泛指非关系型数据库，NoSQL即Not-Only\_SQL,它可以作为关系型数据库的良好补充。NoSQL数据库的产生就是为了解决大规模数据库集合

多重数据库种类带来的挑战，尤其是大数据应用的难题。Nosql说白了就是不能使用sql了，不是关系型数据库，解决高并发的问题。

因为像mysql或oracle单机能够承受的并发数最多也就几千

#### 1 Redis

Redis是用C语言开发的一个开源的高性能键值对数据库。它通过提供多种键值数据类型来适用不同场景下的存储需求，目前为止redis支持的键值数据类型有五种：

1.字符串类型 string: redis中没有使用C语言的字符串表示，而是自定义一个数据 结构叫SDS（simple dynamic string）即简单动态字符串。

2.散列类型 hash：hash叫散列类型，它提供了字段和字段值的映射。字段值只能是字符串类型，不支持散列类型、集合类型等其它类型

3.列表类型 list

4.集合类型 set

5.有序集合类型

redis的字符串是二进制安全的，

##### 0.1什么是二进制安全？

简单理解就是存入什么数据取出的还是什么数据

##### 0.2 redis hash（散列类型）存储比关系数据库的好处？

两者都可以存放对象，对于关系型数据来说，存储数据比较严格、有事务控制，安全性更高，

但是关系型数据库在并发压力下性能很低，所以使用redis hash类型可以在一定程序上

替代关系型数据库，因为redis单机并发量支持10万+

##### 0.3 redis sorted set介绍

在集合类型的基础上有序集合类型为集合中的每个元素都关联一个分数，这使得我们不仅可以完成插入、删除和判断元素是否存在在集合中，

还能够获得分数最高或最低的前N个元素、获取指定分数范围内的元素等与分数有关的操作。

在某些方面有序集合和列表类型有些相似。

1、二者都是有序的。

2、二者都可以获得某一范围的元素。

但是，二者有着很大区别：

1、列表类型是通过链表实现的，获取靠近两端的数据速度极快，而当元素增多后，访问中间数据的速度会变慢。

2、有序集合类型使用散列表实现，所有即使读取位于中间部分的数据也很快。

3、列表中不能简单的调整某个元素的位置，但是有序集合可以（通过更改分数实现）

4、有序集合要比列表类型更耗内存。

##### 1.1 redis的哨兵机制

redis主从之间可以通过哨兵进行检查,如果主节点出现故障,会自动切换从节点为主

##### 1.2 redis cluster集群原理

redis cluster集群默认16384个hash槽,集群搭建成功之后,需要给每一个主节点

分配hash槽。当外部数据插入的时候,会对key进行crc16然后对16384取模,这样

就计算出哪个节点对该数据进行管理。我们在项目当中采用的3主3从的结构,主从之间通过哨兵,出现故障自动切换.

##### 1.3 你们在项目当中哪些地方用到redis？

1：我们项目当中一些字典表数据,通过redis进行缓存提高查询性能,比如品牌表

省市县表等等

2:我们利用redis的过期策略,短信验证码存入redis,两分钟之内有效

3:我们整个项目架构采用redis充当session容器,实现分布式环境下的session共享

4:我们利用redis的list队列,基于push pop的原子性,实现商品抢购秒杀的场景

##### 1.4 缓存的穿透和雪崩问题,你们是如何解决的?

穿透 顾名思义,就是大量的请求穿过缓存层,达到数据库,造成数据库压力过大

举例：你登录淘宝之后,订单搜索,你输入一个无效的订单,通过压测工具恶意攻击

解决方案：1.每次请求时,可以在redis层,做一个bitMap,先去bitmap中查询一下是否存在该条件,进行过滤

2.每次请求,查询数据库就算不存在,我也将查询条件和null在redi中进行缓存

雪崩是指在某一个时间点,大量的缓存同时失效,请求达到数据库,造成数据库压力过大

解决方案:对大量数据设置过期时间时,哪怕业务需要同时失效,我也不设置同一个时间,分别对每一组key value 设置失效时间,让每一组key value的失效时间间隔个几毫秒

##### 1.5 redis是单线程执行的,为什么性能这么高?

redis通过C语言实现了多路复用 轮询的机制,保证了高性能(NIO)

##### 1.6 Redis的应用场景

内容缓存（数据查询、短连接、新闻内容、商品内容等等）。（最多使用）

分布式集群架构中的session分离。

聊天室的在线好友列表。

任务队列。（秒杀、抢购、12306等等）100件iphonex免费赠送,

应用排行榜。

网站访问统计。

数据过期处理（可以精确到毫秒）

##### 1.7持久化

Redis的高性能是由于其将所有数据都存储在了内存中，为了使Redis在重启之后仍能保证数据不丢失，需要将数据从内存中同步到硬盘中，这一过程就是持久化。

Redis支持两种方式的持久化，一种是RDB方式，一种是AOF方式。可以单独使用其中一种或将二者结合使用。

###### 1.7.1 RDB持久化

RDB方式的持久化是通过快照（snapshotting）完成的，当符合一定条件时Redis会自动将内存中的数据进行快照并持久化到硬盘。

RDB是Redis默认采用的持久化方式，在redis.conf配置文件中默认有此下配置：

save 900 1

save 开头的一行就是持久化配置，可以配置多个条件（每行配置一个条件），每个条件之间是“或”的关系

###### 1.7.2 AOF持久化

默认情况下Redis没有开启AOF（append only file）方式的持久化，可以通过appendonly参数开启：appendonly yes

开启AOF持久化后每执行一条会更改Redis中的数据的命令，Redis就会将该命令写入硬

盘中的AOF文件。AOF文件的保存位置和RDB文件的位置相同，都是通过dir参数设置的，默认的文件名是appendonly.aof，

可以通过appendfilename参数修改：appendfilename appendonly.aof.

###### 1.7.3 问题总结：

通过RDB方式实现持久化，一旦Redis异常退出，就会丢失最后一次快照以后更改的所有数据。这就需要开发者根据具体的应用场合，

通过组合设置自动快照条件的方式来将可能发生的数据损失控制在能够接受的范围。如果数据很重要以至于无法承受任何损失，则可以考虑使用AOF方式进行持久化。

项目当中必须使用rdb,手动开启aof，来保证数据的完整性

##### 1.8 Redis的停止：

强行终止redis进程可能会导致redis持久化数据丢失

正确的方法应该是向redis发送shutdown命令

##### 1.9 Redis实例

一个redis进程就是一个redis实例，一台服务器可以同时有多个redis实例，不同的redis实例提供不同的服务端口对外提供服务，

每个redis实例之间互相影响。每个redis实例都包括自己的数据库，数据库中可以存储自己的数据。

##### 2.0多数据库测试

一个Redis实例可以包括多个数据库，客户端可以指定连接某个redis实例的哪个数据库，就好比一个mysql中创建多个数据库，客户端连接时指定连接哪个数据库。

一个redis实例最多可提供16个数据库，下标从0到15，客户端默认连接第0号数据库，也可以通过select选择连接哪个数据库

## 调优篇

### 1.jvm调优

#### **1.1堆分为新生代和老年代**

新生代主要存放的是新创建的对象,新生代主要是采用复制算法进行

minorGC，老年代存放的是在新生代阶段被多次gc回收还没有回收掉的对象,

老年代也会进行垃圾回收gc(majorGC)，采用的是标记-整理算法,老年代进行GC的时候

会造成一个应用程序线程中断的效果，会影响到应用程序的性能.

jvm提供了很多种垃圾回收策略,垃圾回收器 CMS、G1(采用多线程进行GC)

#### 1.2**什么时候对象可以进行垃圾收集?？**

1:引入计数法 (解决不了循环引用的问题)（早期虚拟机）

2:可达性分析法 引用链，当一个对象到GC Roots没有任何引用链（即GC Roots到对象不可达）时，

则证明此对象是不可用的(现代虚拟机使用的)

#### **1.3垃圾收集算法?**

1:复制算法

2:标记清除算法

3:标记整理算法

4:分代管理算法(新生代使用复制算法，年老代使用标记整理算法)

#### 1.4**对象的引用类型**

1：强引用（Object obj=new Object()）

2：弱引用

3：软引用

4: 虚引用

#### **1.5 jvm的调优机制:**

通常我们在项目当中，需要对jvm进行调优，以防止内存溢出的现象。

主要通过调整jvm的启动参数,包括最小堆内存xms、最大堆内存xmx 栈内存xss,

永久代permSize。由于jdk1.8永久代被弃用了，使用操作系统本地内存，

所以不需要调节永久代大小了.

还包括使用并行的垃圾收集器，比如商用的cms、G1垃圾收集器

通常情况下，我们将最小堆内存和最大堆内存设置的一样，防止内存回落

造成GC。在生产环境，我们通过使用Jconsole JvisualVm 对jvm可以进行监控

#### 1.6通过监控发现，cpu和内存使用率居高不下,该怎么排查问题?(重点)

1.可以通过linux top命令 查看机器的cpu和内存使用情况然后通过jstack获取java进程的线程使用情况,然后查看日志，排查问题

2.使用jvm监控工具 Jconsole JvisualVM 远程连接到服务器，分析线程和堆栈内存，排查问题，最有可能的问题是:代码死循环，线程死锁、jvm内存溢出等

#### **1.7 jdk1.8之后的新特性?**

1：支持函数式编程 Lambda 表达式

2：方法引用

3：集合的stream处理

jdk的内存结构发生重大变化

永久代弃用，class信息使用本地内存存储

常量池移动到堆当中

堆内存、栈内存、永久代内存、方法区内存

垃圾收集器选择 CMS G1 串行、并行

tomcat的bin目录下面的 catalina.bat catalina.sh

eclipse installerJres

-server

-Xms3G

-Xmx3G

-Xss256k

-XX:PermSize=128m

-XX:MaxPermSize=128m

-XX:+UseParallelOldGC

-XX:+HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath=/usr/aaa/dump

-XX:+PrintGCDetails

-XX:+PrintGCTimeStamps

-Xloggc:/usr/aaa/dump/heap\_trace.txt

-XX:NewSize=1G -XX:MaxNewSize=1G

jvm的内存线程线程监控

jconson

jvisualVm

通过Linux 相关命令去查看该jvm的内存使用情况、线程情况等等

### 2数据库调优（重点）

1:数据库结构优化

设计数据库的时候,要预估单表的峰值,因为以Mysql为例,当单表的

数据超过千万条,查询速度就特别缓慢了,这里就需要对数据库进行

分库分表（数据分片）,为了提高查询和写入性能,mysql可以采取主从结构

主数据库采用innodb引擎,从数据库采用myIsam引擎,主从之间通过

同步binlog日志实现数据一致

2:SQL优化,本质是通过创建索引,使用explain关键字,查询索引命中情况,

对sql进行调优

3:使用cache层,预先对经常常用的数据库数据进行缓存,以减少数据库的压力

提高数据库性能

### 3. tomcat调优

#### 3.1 tomcat的工作原理

tomcat的核心组件包括server(服务器) service(服务) conector（连接器）

Executor(执行器) engine(servlet引擎).

server当中有一个catalina服务,调用conector组件接收用户的连接,conector底层

是通过socketServer监听8080端口号接收客户端的tcp连接的,接收到连接之后,

会调用 Executor(执行器)创建或捞取线程执行业务逻辑处理,最后再调用servlet

引擎完成请求以及响应

#### 3.2 tomcat调优

1.tomcat默认的线程创建方式,是一次请求,创建一个新的线程来处理

优化方案,可以配置线程池来创建和管理线程,可以设置线程数量

<Executor name="tomcatThreadPool" namePrefix="catalina-exec-"

maxThreads="150" minSpareThreads="4"/>

2.tomcat默认的连接处理方式是BIO,阻塞的IO,容易造成客户端堵塞

优化方案：使用http11Nio协议来进行连接的处理

<Connector executor="tomcatThreadPool"

port="8088" protocol="org.apache.coyote.http11.Http11NioProtocol"

connectionTimeout="20000"

maxConnections="10000"

redirectPort="8443" />

3:因为tomcat是基于jvm的,所以应该对jvm的启动参数进行调优(堆栈内存、垃圾收集器)

悲观锁和乐观锁

mysql如何实现悲观锁和乐观锁

1：mysql提供了表锁和行锁的机制,这就是悲观锁的体现

select \* from user where id=1 for update

2:mysql没有提供乐观锁的机制,需要自己手动实现

更新user表当真 id=1的username的值

每一条记录都有一个最后一次修改时间,

进行更新操作之前,先把上一次的最后一次修改时间查询出

执行sql语句 update user set username ="新值" where id=1 and

updateTime=上次查询的那个时间

公平锁和非公平锁

公平锁是指线程运行是有序的,FIFO RetrantLock(true)

非公平锁是指线程运行允许插队,具有随机性 RetrantLock(false)

## 分布式篇

### Dubbo

dubbo是阿里开源的一个分布式服务治理rpc框架,主要用于服务拆分

dubbo提供了注册中心的机制,注册中心支持zookeeper、redis、memcached

等。我们在项目当中使用zookeeper集群来充当dubbo的注册中心。

dubbo支持多种rpc协议,比如dubbo、hessian、thrift等

我们在项目当中使用dubbo协议进行rpc,dubbo协议是基于netty的nio的http

长连接,dubbo还提供了心跳检测机制。dubbo除了注册中心之外,还提供了

dubbo-admin dubbo-monitor进行服务监控和预警。

Dubbo采用全Spring配置方式，透明化接入应用，对应用没有任何API侵入，只需用Spring加载Dubbo的配置即可， Dubbo基于Spring的Schema扩展进行加载。

#### 1核心部分包含

##### 1.1远程通讯:

提供对多种基于长连接的NIO框架抽象封装，包括多种线程模型，序列化，以及“请求-响应”模式的信息交换方式。

##### 1.2集群容错:

提供基于接口方法的透明远程过程调用，包括多协议支持，以及软负载均衡，失败容错，地址路由，动态配置等集群支持

##### 1.3 服务发现机制

1:服务提供者启动之后,会向注册中心注册自己的服务名和服务地址

2:服务消费者启动之后,会向注册中心订阅指定的服务

3:注册中心会向服务消费者推送所订阅服务的地址列表(如服务提供者地址变更也会重新推送)

4:服务消费者通过负载均衡策略,选择一个地址,进行rpc调用

#### 2.Dubbo能做什么？

透明化的远程方法调用，就像调用本地方法一样调用远程方法，只需简单配置，没有任何API侵入。

软负载均衡及容错机制，可在内网替代F5等硬件负载均衡器，降低成本，减少单点。

服务自动注册与发现，不再需要写死服务提供方地址，注册中心基于接口名查询服务提供者的IP地址，并且能够平滑添加或删除服务提供者。

#### **3如果注册中心不可用,那么服务消费者还能调用服务么?**

dubbo提供了服务的本地缓存机制,短期之内,服务消费者还是可以调用服务的

#### 4现在我有两套环境,一套是开发环境,一套是测试环境,共用同一个注册中心,如果保障开发环境的服务消费者调用的一定是开发环境的服务提供者？

在发布和订阅服务的时候指定group或version

#### 5 **dubbo默认失败重试次数以及超时时间**

失败重试2次 1000毫秒，可以根据业务场景，手动设置

#### 扩展：BIO、NIO、AIO有什么区别?

BIO 同步阻塞的io,客户端发起连接之后,直到响应,这当中一直是阻塞状态(tomcat默认BIO)

NIO 同步非阻塞,服务器实现模式为一个请求一个线程,

即客户端发送的连接请求都会注册到多路复用器上,

多路复用器轮询到连接有I/O请求时才启动一个线程进行处理。（netty、tomcat调优）

AIO 异步的io （linux native AIO）

### 分布式锁

#### redis实现

(注意redis cluster集群不支持 setNx)

分布式场景下,多个相同应用的节点有可能会对同一资源进行操作,

这就涉及到锁的问题。

举例:我们的系统需要每天定时和支付宝、微信等第三方接口,

在夜里2点进行统一对账

可以使用redis的setNx来实现分布式锁

02:00:00 有10个进程想要运行定时任务

1号进程首先获得锁了,

setNx lock 02:10:00

2-10号发现setNx失败了,返回0

当2号进程发现 lock的锁的value值小于当前系统时间了,就可以断定

这个锁失效了,2号线程进程通过getset方法重新把锁的时间更新,并获得

之前的锁的失效时间,如果当前时间小于锁的失效时间,那么意味着有别的进程已经

重新获得锁了.

#### zookeeper实现

还可以使用zookeeper的watch机制,可以监听某一个数据节点,一旦该节点变更

会通知监听者

### zookeeper

Zookeeper是一个java语言开发的分布式框架,重要用于分布式环境协调服务,

Zookeeper 实现了观察者模式，监听各个节点

比如充当dubbo的注册中心.zookeeper集群启动时,会进行投票选举,

1/2以上原则,选举出leader,其余的为follower。leader主要负责写的操作,

follower主要负责读,主从之间提供了心跳检测的机制.当1/2以上的follower

ping不同leader的时候,就会认为leader宕机了,所有的follower会再次进行

投票选举,选举出新的leader。zookeeper集群实现了cap理论,cp 一致性和

分区容错性,存活原则1/2法则。

### Nginx

#### 1、请解释一下什么是Nginx?

Nginx是一个web服务器和方向代理服务器，用于HTTP、HTTPS、SMTP、POP3和IMAP协议。

#### 2、请列举Nginx的一些特性。

反向代理/L7负载均衡器

嵌入式Perl解释器

动态二进制升级

可用于重新编写URL，具有非常好的PCRE支持

#### 3、请解释Nginx如何处理HTTP请求。

Nginx使用反应器模式。主事件循环等待操作系统发出准备事件的信号，这样数据就可以从套接字读取，在该实例中读取到缓冲区并进行处理。单个线程可以提供数万个并发连接。

#### 4、 使用“反向代理服务器”的优点是什么?

反向代理服务器可以隐藏源服务器的存在和特征。它充当互联网云和web服务器之间的中间层。这对于安全方面来说是很好的，特别是当您使用web托管服务时。

#### 5、请列举Nginx服务器的最佳用途。

Nginx服务器的最佳用法是在网络上部署动态HTTP内容，使用SCGI、WSGI应用程序服务器、用于脚本的FastCGI处理程序。它还可以作为负载均衡器。

#### 6、请解释Nginx服务器上的Master和Worker进程分别是什么?

Master进程：读取及评估配置和维持

Worker进程：处理请求

## 权限管理

### Shiro

shiro是apache旗下的权限控制框架,shiro有四大核心

认证器(authenticator)、授权器(Authorizer)、会话管理器(sessionManager)、

密码管理器(Cryptography)

我们在项目当中使用shiro的时候,需要配置shiroFilter、指定securityManager、

登录url、非法访问的url、权限过滤规则等等,认证和授权通过自定义Realm

重写dogetAuthentication 登录认证方法以及dogetAuthorization方法 授权方法,

shiro不仅仅提供url级别的权限控制,我们还通过shiro标签 hasrole、haspermisstion

实现按钮级别的权限管理.

session管理这一块,我们使用shiro整合redis-cluster实现了分布式环境下的

session分离,手动开发sessionDao,重写session的增删改查方法

SOA（分布式）

SOA即面向服务架构

SAAS PAAS 服务化之后的系统

RPC

RPC即远程过程调用

RPC有三要素 socket 动态代理 序列化

RPC的常用技术实现 hessian thrift avro soapWebService

webService的两种技术实现

1:基于rest协议的webService接口 http+json (互联网项目)

2：基于soap协议的webService接口 （传统型项目)

简单对象访问协议

http

xml

wsdl

基于jax-ws实现 https://www.jianshu.com/p/94addb458778

使用wsimport -keep 地址 生成java客户端

## 消息中间件

### 0.市面上常用的消息中间件的优缺点以及使用场景

kafka 高吞吐(百万级别的数据写入)、高可用(数据备份、zookeeper) 大数据平台日志收集

RocketMQ 吞吐量(几十万级) 大型互联网公司

rabbitMQ 吞吐量(万级)、高并发 中小型互联网公司 提供了事务以及消息确认机制

activeMq 吞吐量(万级) 小型或者非互联网公司

### 1.rabbitMQ

MQ称为消息队列,用于服务之间同步调用解耦，实现数据的最终一致性

它的常用API Connection channel exchange topic routingKey等等

rabbitMQ是使用erlang语言开发的，天然支持高并发，提供了消息的确认机制，

保证了消息的可靠性。

#### 1.1 rabbitMQ的消息发布模式

basic 点对点

exchange 发布订阅

#### 1.2 rabbitMQ如何保证消息不会丢失?

消息丢失很有可能在生产者、mq服务、消费者任意一个环节出现

生产者可以使用channel事务机制、confirm机制，前者能保证消息传递的

可靠性，但是吞吐量会下降，这时候可以使用confirm机制，mq收到消息之后

会进行ack回调通知

mq采用持久化对消息进行硬盘存储，可以保证mq异常关机之后，数据不会丢失

消费者 关闭rabbitMQ提供的自动ACK消息确认机制，改为消费者处理完消息之后,

手动ACK(基于tcp的ack事务包)

#### 1.3如何保证mq当中消息的幂等性（不被重复消费）

生产者在生产消息的时候做幂等控制(使用全局、流程唯一的id)

举例:生产者使用订单id、用户id的时候，先判断该消息是否已经存储了（redis）,然后再

发送到mq。

消费者消费消息的时候，也是使用全局唯一的标识，先判断是否已经消费过了，

可以借助redis或数据库(唯一索引)，来保证不被重复消费

#### 1.4对于分布式环境的id生成？

使用统一的id发号器来生成主键(根据业务维度设计主键、末位有序)

#### 1.5 你们项目什么地方使用到了mq？

1.用户抢购秒杀成功之后，生成订单之后，我们将订单号存储到了rabbitMQ,

(我们通过死信队列、死信交换器、普通队列)来实现延迟队列,如果限定的

支付时间之内，用户没有付款，那么我们需要消费mq当中的消息，操作数据库

设置订单为已过期、库存回滚、积分回滚、优惠券回滚。

2.为了保证redis缓存和数据库的数据一致性，我们采用rabbitMQ进行解耦，

mysql数据变更通过binlog日志会记录下来,我们通过生产者监听mysql的binlog

日志,将要更新的数据写入到mq，另一端消费者消费数据，更新到缓存当中

## 7.SpringBoot快速开发框架

springboot是spring4.0之后提供的一个自动化启动框架,采用习惯优于配置

的理念,可以进行自动化配置(EnableAutoConfiguration),框架采用

注解+properties(或yaml)代替传统的xml配置,极大的提高了开发效率.

springboot通过main方法启动,而且内置web容器(tomcat),打包方式为jar

springboot的常用注解有

@springbootapplication 标识启动类

@configuration 标识配置类

@EnableAutoConfiguration 开启自动配置

@ComponentScan 注解扫包器

@MapperScan 扫描mybatis的mapper接口

@RestController rest风格返回json

### springboot自动配置的原理

spring-boot-autoconfigure jar包下meta-inf文件夹下,

spring.factories配置文件提供了各种技术框架的默认配置

我看过很多自动配置类的源码,比如springmvc,自动配置里

指定dispatcherServlet请求路径为/,指定了处理器映射器

requestMappingHandlerMapping等组件的配置

### 什么是SpringBoot

SpringBoot是快速开发的Spring框架，能够快速整合主流框架，简化xml配置，采用全注解化，内置Http服务器（如tomcat、jetty等），通过java部署运行。

### 为什么要用SpringBoot

快速开发，快速整合，配置简化、内嵌服务容器

### SpringBoot启动方式

主类@SpringBootApplication注解或添加@ComponentScan和@EnableAutoConfiguration注解，使用@SpringBootApplication时注意自动扫描当前包

### SpringBoot与SpringMVC 区别

SpringMVC是SpringBoot的Web开发框架

### SpringBoot与SpringCloud 区别

SpringBoot是快速开发的Spring框架，SpringCloud是完整的微服务框架， SpringCloud依赖于SpringBoot。

### SpringBoot中用那些注解

@EnableAutoConfiguration作用

自动扫描并添加jar包依赖

@SpringBootApplication原理

是一个组合注解，相当于@EnableAutoConfiguration和@ComponentScan

### SpringBoot热部署使用什么？

devtools

### 热部署原理是什么？

热部署的实现原理主要依赖java的类加载机制，在实现方式可以概括为在容器启动的时候起一条后台线程，定时的检测类文件的时间戳变化，如果类的时间戳变掉了，则重新加载整个应用的class文件，同时重启服务，重新部署。

### 热部署原理与热加载区别是什么

热加载是在运行时重新加载class文件，不会重启服务。

### 你们项目中异常是如何处理

在web项目中，使用全局捕获异常返回统一错误信息。

### SpringBoot如何实现异步执行

在启动类添加@EnableAsync表示开启对异步任务的支持，在异步服务上添加@Async

### SpringBoot多数据源拆分的思路

先在properties配置文件中配置两个数据源，创建分包mapper，使用@ConfigurationProperties读取properties中的配置，使用@MapperScan注册到对应的mapper包中

### SpringBoot多数据源事务如何管理

第一种方式是在service层的@TransactionManager中使用transactionManager指定DataSourceConfig中配置的事务

第二种是使用jta-atomikos实现分布式事务管理

### SpringBoot如何实现打包

进入项目目录在控制台输入mvn clean package，clean是清空已存在的项目包，package进行打包

### SpringBoot性能如何优化

如果项目比较大，类比较多，不使用@SpringBootApplication，采用@Compoment指定扫包范围

在项目启动时设置JVM初始内存和最大内存相同

将springboot内置服务器由tomcat设置为undertow

### SpringBoot执行流程

使用SpringApplication.run()启动，在该方法所在类添加@SpringBootApplication注解，该注解由@EnableAutoConfiguration和@ComponentScan等注解组成，@EnableAutoConfiguration自动加载SpringBoot配置和依赖包，默认使用@ComponentScan扫描当前包及子包中的所有类，将有spring注解的类交给spring容器管理

### SpringBoot底层实现原理

使用maven父子包依赖关系加载相关jar包，使用java操作Spring的初始化过程生成class文件，然后用java创建tomcat服务器加载这些class文件

### SpringBoot装配Bean的原理

通过@EnableAutoConfiguration自动获取配置类信息，使用反射实例化为spring类，然后加载到spring容器

### springboot的版本问题

springboot 1.5 搭配spring 4

springboot 2.0.2 搭配spring 5

## 微服务框架SpringCloud

### 请说一下你们项目使用到的springcloud中所用的组件？

Eureka 注册中心

Ribbon 本地负载均衡

Fegin 服务调用

RestTemplent 服务调用

Hystrix 熔断降级

Zuul 网关

链路追踪

配置中心

### 什么是SpringCloud

SpringCloud是微服务的一种解决方案，依赖SpringBoot实现。包含注册中心(eureka)、客户端负载均衡(Ribbon)、网关(zull)、分布式锁、分布式会话等。

### 为什么要使用SpringCloud

SpringCloud是一套非常完整的微服务解决方案，俗称“微服务全家桶”，几乎内置了微服务所使用的各种技术，可以不必集成第三方依赖。

### SpringCloud服务注册发现原理

每个SpringCloud服务器启动后向注册中心注册本服务器信息，如服务别名、服务器IP、端口号等，其他服务进行请求时先根据服务别名从注册中心获取到目标服务器IP和端口号，并将获取到的信息缓存到本地，然后通过本地使用HttpClient等技术进行远程调用。

### SpringCloud 支持那些注册中心

Eureka、Consul、Zookeeper

### 现在Eureka闭源了，可以通过什么注册中心替代Eureka呢？

Consul或Zookeeper

### 谈谈你对微服务服务治理的思想

### Eureka如何实现高可用

启动多台Eureka服务器，然后作为SpringCloud服务互相注册，客户端从Eureka集群获取信息时，按照注册的Eureka顺序对第一个Eureka进行访问。

### @LoadBalanced注解的作用

开启客户端负载均衡。

### Nginx与Ribbon的区别

Nginx是反向代理同时可以实现负载均衡，nginx拦截客户端请求采用负载均衡策略根据upstream配置进行转发，相当于请求通过nginx服务器进行转发。Ribbon是客户端负载均衡，从注册中心读取目标服务器信息，然后客户端采用轮询策略对服务直接访问，全程在客户端操作。

### Ribbon底层实现原理

Ribbon使用discoveryClient从注册中心读取目标服务信息，对同一接口请求进行计数，使用%取余算法获取目标服务集群索引，返回获取到的目标服务信息。

### SpringCloud有几种调用接口方式

使用Feign和RestTemplate

### DiscoveryClient的作用

可以从注册中心中根据服务别名获取注册的服务器信息。

### 谈谈服务雪崩效应

雪崩效应是在大型互联网项目中，当某个服务发生宕机时，调用这个服务的其他服务也会发生宕机，大型项目的微服务之间的调用是互通的，这样就会将服务的不可用逐步扩大到各个其他服务中，从而使整个项目的服务宕机崩溃.发生雪崩效应的原因有以下几点

1.单个服务的代码存在bug. 2请求访问量激增导致服务发生崩溃(如大型商城的枪红包，秒杀功能). 3.服务器的硬件故障也会导致部分服务不可用.

### 在微服务中，如何保护服务?

一般使用使用Hystrix框架，实现服务隔离来避免出现服务的雪崩效应，从而达到保护服务的效果。当微服务中，高并发的数据库访问量导致服务线程阻塞，使单个服务宕机，服务的不可用会蔓延到其他服务，引起整体服务灾难性后果，使用服务降级能有效为不同的服务分配资源,一旦服务不可用则返回友好提示，不占用其他服务资源，从而避免单个服务崩溃引发整体服务的不可用.

### 服务雪崩效应产生的原因

因为Tomcat默认情况下只有一个线程池来维护客户端发送的所有的请求，这时候某一接口在某一时刻被大量访问就会占据tomcat线程池中的所有线程，其他请求处于等待状态，无法连接到服务接口。

### 谈谈Hystrix服务保护的原理

通过服务降级、服务熔断、服务隔离为高并发服务提供保护。

### 谈谈服务降级、熔断、服务隔离

服务降级：当客户端请求服务器端的时候，防止客户端一直等待，不会处理业务逻辑代码，直接返回一个友好的提示给客户端。

服务熔断是在服务降级的基础上更直接的一种保护方式，当在一个统计时间范围内的请求失败数量达到设定值（requestVolumeThreshold）或当前的请求错误率达到设定的错误率阈值（errorThresholdPercentage）时开启断路，之后的请求直接走fallback方法，在设定时间（sleepWindowInMilliseconds）后尝试恢复。

服务隔离就是Hystrix为隔离的服务开启一个独立的线程池，这样在高并发的情况下不会影响其他服务。服务隔离有线程池和信号量两种实现方式，一般使用线程池方式。

### 服务降级底层是如何实现的？

Hystrix实现服务降级的功能是通过重写HystrixCommand中的getFallback()方法，当Hystrix的run方法或construct执行发生错误时转而执行getFallback()方法。

### 分布式配置中心有那些框架？

Apollo(阿波罗)、zookeeper、springcloud config。

### 分布式配置中心的作用？

动态变更项目配置信息而不必重新部署项目。

### SpringCloud Config 可以实现实时刷新吗？

springcloud config实时刷新采用SpringCloud Bus消息总线。

### 什么是网关?

网关相当于一个网络服务架构的入口，所有网络请求必须通过网关转发到具体的服务。

### 网关的作用是什么

统一管理微服务请求，权限控制、负载均衡、路由转发、监控、安全控制黑名单和白名单等

### 网关与过滤器有什么区别

网关是对所有服务的请求进行分析过滤，过滤器是对单个服务而言。

### 常用网关框架有那些？

Nginx、Zuul、Gateway

### Zuul与Nginx有什么区别？

Zuul是java语言实现的，主要为java服务提供网关服务，尤其在微服务架构中可以更加灵活的对网关进行操作。Nginx是使用C语言实现，性能高于Zuul，但是实现自定义操作需要熟悉lua语言，对程序员要求较高，可以使用Nginx做Zuul集群。

### 既然Nginx可以实现网关？为什么还需要使用Zuul框架

Zuul是SpringCloud集成的网关，使用Java语言编写，可以对SpringCloud架构提供更灵活的服务。

### 如何设计一套API接口

考虑到API接口的分类可以将API接口分为开发API接口和内网API接口，内网API接口用于局域网，为内部服务器提供服务。开放API接口用于对外部合作单位提供接口调用，需要遵循Oauth2.0权限认证协议。同时还需要考虑安全性、幂等性等问题。

### ZuulFilter常用有那些方法

Run()：过滤器的具体业务逻辑

shouldFilter()：判断过滤器是否有效

filterOrder()：过滤器执行顺序

filterType()：过滤器拦截位置

### 如何实现动态Zuul网关路由转发

通过path配置拦截请求，通过ServiceId到配置中心获取转发的服务列表，Zuul内部使用Ribbon实现本地负载均衡和转发。

### Zuul网关如何搭建集群

使用Nginx的upstream设置Zuul服务集群，通过location拦截请求并转发到upstream，默认使用轮询机制对Zuul集群发送请求。

## 话术篇

### 通过redis实现用户购物车

我们要做的是将购物车商品信息存储到redis cluster集群中

1.通过token从redis cluster集群中取到用户的id

细节：从cookie中取到token，然后拼串：”login:”+token，以此串为键，从redis cluster中取到字段为”id”的值userId

2.拼串：”shopping:”+userId，以此串为键，将商品id和商品数量信息存储到redis cluster集群

细节：可以先通过”shopping:”+userId从redis cluster集群中取一下购物车信息，如果没有商品，那就直接添加进去就行了；如果有商品，那就需要判断是不是存在和所选商品一样的商品，如果存在，就在原来的基础上让商品数量加1即可，如果不存在，则直接添加进去

### cookie存储游客购物车

如果cookie被禁用，使用js的sessionStorage 来存储购物车信息

sessionStorage 是HTML5新增的一个会话存储对象，用于临时保存同一窗口(或标签页)的数据，在关闭窗口或标签页之后将会删除这些数据。

游客购物车我们是通过cookie来存放商品信息的，当游客选择好商品，点击加入购物车之后，会将商品id传到后台controller，这个时候我们可以先获取到用户请求携带的cookie，然后找一下有没有以这个购物车命名的那个cookie，如果没有的话，我们就先通过传过来的商品id去查商品信息，把结果放在list集合里面，然后通过fastJson的jar包把list转换成json类型，这个时候用的是JSON.toJSONString(list)的方法，然后对这个json进行utf-8编码(URLEncoder.encode(json,"utf-8"))，存到cookie中去；

如果之前就已经存过cookie的话，我们是先将那个cookie取出，这个时候的值是json类型，我们需要先对这个json进行解码(URLDecoder.decode(json,"utf-8"))，然后通过fastJson的jar包把json转换成list类型，这个时候用的是JSON.parseArray(json,ItemsVo.class)方法，然后遍历这个list，判断一下传过来的商品id和list里面商品的id有没有一致的，如果有的话就把list里面的商品那个对象的数量属性+1，如果没有，就去数据库里面把该商品查出来，然后放到list集合里面就可以了，最后再把list转换成json类型，编码存到cookie里面就可以了

那么如果要向删除游客购物车里面的商品该怎么做呢？

当用户点击删除的时候，通过ajax异步请求将所要删除的商品的id传到后台删除的controller

中，首先要取到cookie信息，找出cookie里面所存放的购物车商品信息，这个商品信息是以json类型进行存储的，我们需要将这个json解码并转换成list集合，然后通过Iterator迭代器遍历这个list，找出要删除的那些商品，通过remove方法，将其移除掉，最后得到的那个list就是删除之后的，然后将这个list再次转换成json类型，编码并存到cookie中，最后返回一条信息给ajax的回调函数，让其刷新一下页面

### 游客购物车和用户购物车合并

当用户以游客的身份将商品加入购物车，然后登录之后，我们需要将游客的购物车中的商品合并到用户的购物车里面，这个时候我们需要获取到两种信息，一种是cookie里面的商品信息，另一种是用户购物车里面的商品信息，cookie里面的商品信息我们可以通过cookie来获取到，接下来是获取用户购物车信息，用户的购物车信息我们是通过以用户的id为键，以商品id和购买数量组成的map集合为值，存储到redis cluster集群里面的，存储的类型是hash类型，所以我们要先获取到用户的id，通过cookie携带的token令牌，从redis cluster集群中获取到用户id，然后在通过用户id获取到他的购物车信息，这个是map类型的，键是商品id，值是商品数量；

获取到这两种信息之后，我们就能将其进行合并，首先将cookie里面的购物车信息转换成list，用户的购物车map类型通过keySet()方法，把商品id存在set集合里面，然后通过双重循环，外层是list遍历，内层将set集合通过Iterator进行遍历，找到有没有相同的id，如果有的话，就通过jedisCluster将集群中对应商品id的数量+cookie中对应商品id的数量即可，如果没有就将cookie中的商品id和数量通过jedisCluster的hset方法添加到集群中

### 支付功能实现

当用户点击购买之后，会进入订单详情页面，在这个订单详情页面中，需要设置收货人的相关信息，比如收货人姓名、收获地址、联系方式等；还要显示订单的相关信息，包括订单号、订单总金额、商品详情以及对应的价格和数量；然后选择支付方式(微信或支付宝)，还有付款按钮；

比如说用户选了支付宝支付,点击了付款,请求进入后台之后，我们需要调用支付宝提供的接口，支付宝这个接口api要求传一些参数，比如: 付款金额，订单标题，订单介绍，外部订单号(一个唯一的订单号，用于调用支付宝其他接口进行查询交易，退款..) ，支付宝方会创建预交易记录，我们判断支付宝回应的状态码，成功的话呢，就可以获取到支付二维码图片信息(非必需) 根据我们经理的要求呢，需要将我们网站的logo的绘制到二维码中间，所以我就将收到的二维码用第三方jar工具处理过后，才将其返回给前台，前台收到二维码之后呢，显示到页面,会用js脚本每隔2秒请求后台查询订单是否已经完成支付，后台的查询是携带着订单号去请求支付宝的查询接口查询支付状态，这样可以减轻我们数据库服务器的压力。如果支付成功的话呢，会控制进行跳转。当然，时间久了二维码也会失效，会提示用户刷新

为了将支付宝那边的支付记录的支付状态同步到我们的数据库中，我们向支付宝注册了一个回调地址，这个地址(控制器的requestMapping地址)的作用是如果二维码被支付成功，支付宝会请求我们这个地址，并且会携带支付成功的订单号，以及其他参数，获取到这些参数后就可以同步支付状态到我们的数据库中

（支付安全） 还有一点就是微信支付和支付宝支付，首先要注册商家服务,然后再申请支付功能，为了保证安全，申请支付功能通过之后可以设置公钥和私钥，支付平台会利用私钥和公钥提供验签功能(采用RSA非对称加密算法)保证安全，防止恶意请求篡改参数

(对账功能) 但是还有一个场景，就是支付成功之后，在进行回调的时候，网络超时了，这就会使订单状态无法更新，这个时候有两种解决方案：一是用户自己联系人工客服，提供订单号等信息，客服会去核对订单信息以及支付情况，人工客服手动修改订单状态；还有一种就是对账，定时的去微信或支付宝进行对账，写一个定时任务，每隔一段时间就去微信或支付宝去核对支付信息

### redis实现抢购秒杀功能

要实现抢购秒杀功能，事先将需要加入抢购的商品id和开始抢购的时间以hash类型存放在redis cluster集群中，还需要本次抢购商品的库存，以及通过list类型存放的与库存数等量的商品序列号；

当用户进入抢购页面时，需要向用户展示出商品的图片，商品名，抢购价格，库存数和剩余库存，以及开始抢购的时间等信息，当用户点击抢购的时候，首先在前台判断一下是否达到了开始抢购的时间，如果时间满足，就通过ajax将请求发送到抢购的controller，携带的参数是商品id，然后再次确认是否到了开始抢购的时间；

我们是通过redis里面的list类型实现抢购的，如果有用户抢购成功，就将事先存好的商品序列号pop出来一个，一直到pop出空值，这就表示商品已经全部被抢完了；

当然，很多抢购商品基本上都是每人只能抢一件，在此之前我们还要先判断一下该用户是否已经抢购成功一次了，因此需要将抢购成功的用户的id存入redis cluster集群中

### 分布式

说一说你对分布式的理解：

分布式就是一种分而治之的思想，用于解决互联网大并发的应用场景。

对于传统的项目，比如ssm、ssh，所有的功能都耦合在一起，一起进行开发以及部署。优点：开发快捷，部署简单。缺点：所有的功能全部耦合在一起，不利于二次开发；在大并发场景下，单点功能会影响整个系统的稳定性。

分布式架构，比如dubbo、spring cloud等；dubbo是面向rpc，spring cloud是面向http；dubbo的优势在于rpc的调用是通过局域网，不受网络带宽的影响，速度稍快；spring cloud的优势在于比dubbo多了统一的网关以及熔断路由的功能，spring cloud算是严格意义上的微服务，dubbo只是rpc的治理；现在互联网公司用dubbo(HSF、JSF)、spring cloud平分秋色，但是趋势是向spring cloud靠近；分布式架构可以解决由大并发导致的所有功能全部失效的问题。

注意：每一个服务都使用自己独立的数据库

服务：是指由controller等组成的供客户端调用的接口

dubbo架构：

dubbo是一个开源的分布式服务治理的rpc框架，它提供了服务的注册和发现机制，当服务提供者启动的时候会向注册中心写入自己的服务名以及服务器地址，服务消费者启动的时候，会向服务中心订阅某一服务，注册中心会将该服务对应的服务器地址推送给消费者，如果服务提供者有地址变更，注册中心会推送新的地址给服务消费者，然后服务消费者通过负载均衡算法，选择一个服务提供的url，进行RPC调用；dubbo的rpc协议默认是dubbo协议，dubbo协议的底层是通过基于netty的tcp长连接的NIO，NIO有三个核心组件：channel、selector、buffer，channel是多路复用的管道，selector是轮询器，buffer是数据缓冲区；dubbo支持多协议注册中心，我们在项目当中通常使用zookeeper充当注册中心；dubbo还提供了监控中心，监控服务的使用情况。

dubbo的网关：

dubbo没有自己的统一网关，一般用nginx服务器来做dubbo的网关；nginx用来开发网关的配置文件，指定请求转发的策略以及转发给谁，配置文件当中得写死ip地址和端口号；但是对于dubbo架构，api层的机器如何平滑的添加tomcat？

可以通过zookeeper集群来实现，当api层的机器启动之后，会向zookeeper集群写入自己的地址，在nginx服务器中写一个lua脚本，监听zookeeper集群当中记录的所有的tomcat的地址，一但有新机器添加或移除，nginx就通过lua脚本重新生成负载均衡的配置文件，直接热加载

两个环境公用同一个注册中心，怎么能保证环境的服务消费者调用到的一定是环境服务提供者：给服务分组 group version版本号