一、个人技能部分

1、docker

docker是一个用于部署系统，解决环境问题的容器技术。Docker设计的目的就是要加强开发环境与应用程序要部署的生产环境的一致性。Docker的目标之一就是缩短代码从开发、测试到部署、上线运行的周期，让应用程序具备可移植性，易于构建，并易于协作。Docker还鼓励面向服务的体系结构和微服务架构。Docker推荐单个容器只运行一个应用程序或进程，这样就形成了一个分布式的应用程序模型，在这种模型下，应用程序或者服务都可以表示为一系列内部互联的容器，从而使分布式部署应用程序，扩展或调试应用程序都变得非常简单，同时也提高了程序的内省性。

1.不同的应用程序可能会有不同的应用环境，有些软件安装之后会有端口之间的冲突，这时候，可以使用虚拟机来实现隔离，但是使用虚拟机的成本太高，而且消耗硬件。

2.不同的软件的环境都不一样，比如：你用的是乌班图，里面有个数据库，现在要迁移到centos中，但是此时需要从新在centos安装数据库，如果版本不一致，或者不支持，就会出现问题。比较麻烦。有了docker之后就不用这么麻烦了，直接将开发环境搬运到不同的环境即可。

3.在服务器负载方面，如果你单独开一个虚拟机，那么虚拟机会占用空闲内存的，docker部署的话，这些内存就会利用起来。

docker就是用于部署项目,解决环境问题的软件技术(实现虚拟化,比传统的虚拟机技术要好).特别适合微服务。

镜像是Docker生命周期中的构建或者打包阶段，而容器则是启动或者执行阶段。容器基于镜像启动，一旦容器启动完成后，我们就可以登录到容器中安装自己需要的软件或者服务。

Docker五大优势：持续集成、版本控制、可移植性、隔离性、安全性

docker的组件：

registry:中央注册中心

images:就是下载镜像文件

client:就是操作docker的客户端（命令）

containter:就是docker容器，需要运行在doker服务中

Docker镜像是由文件系统叠加而成（是一种文件的存储形式）。最底端是一个文件引导系统，即bootfs，这很像典型的Linux/Unix的引导文件系统。Docker用户几乎永远不会和引导系统有什么交互。实际上，当一个容器启动后，它将会被移动到内存中，而引导文件系统则会被卸载，以留出更多的内存供磁盘镜像使用。Docker容器启动是需要的一些文件，而这些文件就可以称为Docker镜像。

Docker容器操作：

查看正在运行容器：docker ps

查看所有的容器（启动过的历史容器）：docker ps -a

查看最后一次运行的容器：docker ps -l

查看停止的容器：docker ps -f status=exited

创建一个交互式容器并取名为mycentos：docker run -it --name=mycentos centos:7 /bin/bash

退出当前容器：exit

创建一个守护式容器：docker run -di --name=mycentos2 centos:7

登录守护式容器：docker exec -it mycentos2 /bin/bash

停止正在运行的容器：docker stop mycentos2

启动已运行过的容器：docker start mycentos2

将文件拷贝到容器内：docker cp 需要拷贝的文件或目录 容器名称:容器目录

将文件从容器内拷贝出来：docker cp 容器名称:容器目录 需要拷贝的文件或目录

创建容器并挂载宿主机目录到容器中的目录下：

docker run -di -v /usr/local/myhtml:/usr/local/myhtml --name=mycentos3 centos:7

添加参数-privileged=true来解决挂载的目录没有权限的问题：

docker run -di --privileged=true -v /root/test:/usr/local/test --name=mycentos4 centos:7

查看容器运行的各种数据:docker inspect mycentos2
直接输出IP地址:docker inspect --format='{{.NetworkSettings.IPAddress}}' mycentos2
删除指定的容器： 这个命令只能删除已经关闭的容器，不能删除正在运行的容器
docker rm $CONTAINER_ID/NAME
删除所有的容器：
docker rm `docker ps -a -q`或者： docker rm $(docker ps -aq)
将容器保存为镜像：docker commit pinyougou_nginx mynginx
镜像备份：docker save -o mynginx.tar mynginx
镜像恢复：docker load -i mynginx.tar

springboot微服务部署：
1.在本地开发完成微服系统打包，将其copy到linux系统中
2.创建dockefile文件：

```
#my dockerfile ljh
FROM java:8
MAINTAINER ljh
ADD demo-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
EXPOSE 8080
```

3.构建镜像：docker build -t demoappimage.
4.创建容器：docker run -di --name=myapp1 -p 8080:8080
5.浏览器中访问系统：：http://192.168.25.132:8080/hello

3、Docker容器有几种状态？
答：有四种状态：运行、已暂停、重新启动、已退出。

4、Dockerfile中最常见的指令是什么？
答：FROM：指定基础镜像；LABEL：功能是为镜像指定标签；RUN：运行指定的命令；CMD：容器启动时要运行的命令。

5、Dockerfile中的命令COPY和ADD命令有什么区别？
答：一般而言，虽然ADD和COPY在功能上类似，但是首选COPY。那是因为它比ADD更易懂。COPY仅支持将本地文件复制到容器中，而ADD具有一些功能（如仅限本地的tar提取和远程URL支持），这些功能并不是很明显。因此，ADD的最佳用途是将本地tar文件自动提取到镜像中，如ADD rootfs.tar.xz /。

7、解释基本的Docker使用工作流程是怎样的？
答：（1）从Dockerfile开始，Dockerfile是镜像的源代码；（2）创建Dockerfile后，可以构建它以创建容器的镜像。镜像只是“源代码”的“编译版本”，即Dockerfile；（3）获得容器的镜像后，应使用注册中心新分发容器。注册中心就像一个git存储库，可以推送和拉取镜像；（4）接下来，可以使用该镜像来运行容器。

8、如何在生产中监控Docker？
答：Docker提供docker stats和docker事件等工具来监控生产中的Docker。我们可以使用这些命令获取重要统计数据的报告。
Docker统计数据：当我们使用容器ID调用docker stats时，我们获得容器的CPU，内存使用情况等。它类似于Linux中的top命令。
Docker事件：Docker事件是一个命令，用于查看Docker守护程序中正在进行的活动流。一些常见的Docker事件是：attach，commit，die，detach，rename，destroy等。

9、什么类型的应用程序无状态或有状态更适合Docker容器？
答：最好为Docker Container创建无状态应用程序。我们可以从应用程序中创建一个容器，并从应用程序中取出可配置的状态参数。现在我们可以在生产环境和具有不同参数的QA环境中运行相同的容器。这有助于在不同场景中重用相同的镜像。另外，无状态应用程序比有状态应用程序更容易使用Docker容器进行扩展。

2、maven

Maven是由Apache开发的一个工具。用来管理java项目(依赖管理，项目构建， 分模块开发)

maven的作用：

1.依赖管理： maven对项目的第三方构件（jar包）进行统一管理。向工程中加入jar包不要手工从其它地方拷贝，通过maven定义jar包的坐标，自动从maven仓库中去下载到工程中。

2.项目构建： maven提供一套对项目生命周期管理的标准，开发人员、和测试人员统一使用maven进行项目构建。项目生命周期管理：编译、测试、打包、部署、运行。

3.模块分发：maven对工程分模块构建，提高开发效率。

maven的好处：

1. 节省磁盘空间
2. 可以一键构建
3. 可以跨平台
4. 应用在大型项目时可以提高开发效率（jar包管理，maven的工程分解，可分模块开发）

Maven的常用命令：

clean命令：清除编译产生的target文件夹内容

compile命令：该命令可以对src/main/java目录的下的代码进行编译

test命令：测试命令,或执行src/test/java/下所有junit的测试用例

package命令：打包项目，如果是JavaSe的项目,打包成jar包，如果是JavaWeb的项目,打包成war包

install命令：包后将其安装在本地仓库，当我们执行了install 也会执行compile test package

依赖的作用范围：

compile 编译、测试、运行【默认】

provided 编译、测试

runtime 运行、测试

test 测试

解决maven的jar包冲突问题：

1：第一声明优先原则

2：路径近者优先

3：直接排除原则

maven的聚合和继承：

聚合：把项目的多个模块聚合在一起，使用一条命令进行构建，即一条命令实现构建多个项目；

继承：可以将各个模块相同的依赖和插件配置提取出来，在简化POM的同时还可以促进各个模块配置的一致性。

Git/SVN

https://www.cnblogs.com/qcloud1001/p/9884576.html

3、JDBC

Java Data Base Connectivity，java数据库连接技术。jdbc是一套接口规范，运行开发人员根据接口规
范连接访问操作不同的数据库。
JDBC的好处：
1、使用一套接口可以操作不同的数据库，使用非常简单，不用关系底层网络编程通信
2、切换数据库非常方便，只需要修改很少的内容就可以完成数据库切换，比如：mysql数据库切换为oracle
jdbc操作数据库步骤：
1、导入mysql数据库驱动包
2、注册mysql数据库驱动（DriverManager）
3、使用驱动获取数据库连接(Connection)
4、使用数据库连接创建运输器(Statement)
5、使用运输器发送sql并获取返回的结果（ResultSet）
6、结果集通过游标指针读取数据
7、释放资源

4、myBatis

什么是框架:框架是对常见底层功能进行封装成一套具有可重用设计的组件；
框架解决了哪些问题：
1、解决技术通用问题
2、解决开发效率问题
3、解决稳定性问题
分层开发下常见的框架:SSM
WEB层：springMVC框架、status
业务层：spring框架
Dao层：JDBC、mybatis框架、hibernate

mybatis的优点：
1. 轻量级框架：轻量级：对底层其他技术不依赖或轻依赖　重量级：对底层其他技术重依赖
   mybatis只是对JDBC封装，不依赖其他任何技术，导2个jar包和写几个配置文件即可
2. 解除SQL与程序代码的耦合：SQL语句和代码的分离，提高了可维护性。
   java代码写在java类中.sql代码写在配置文件中，可维护性高
3. Mybatis的核心优势：ORM框架（Object Relational Mapping）：对象关系映射框架
   含义：将关系型数据库的sql语句映射成技术语言的对象
   例子：
   ```
   select * from student;查询多条记录------->List
   select * from student where id=1;---------->Student对象
   ```
   上面的操作都是由mybatis框架自动映射完成，不需要开发人员去封装
4. Mybatis支持缓存，可以提高性能。
   执行一条sql语句结果就会缓存下来，下次还要执行同一条sql语句就不会执行，直接将缓存的结果返回

MyBatis如何获取数据库连接？
1、配置mybatis核心主配置文件sqlMapConfig.xml配置数据库信息、事务、连接池
2、读取配置文件的输入流
3、创建连接会话工厂构建类：SqlSessionFactoryBuilder [了解]
   SqlSessionFactoryBuilder.build(输入流)
4、根据构建类创建连接会话工厂类：SqlSessionFactory （连接池对象）
   SqlSessionFactory.openSession();
5、使用连接会话工厂类获取数据库连接： SqlSession

主配置文件中要编写的内容：
1. 引入约束文件(固定的，使用模板)
2. 根标签 <configuration>
3. <properties resource="jdbc.properties"/> 引入外部的properties属性文件,其中
   jdbc.properties就是要引入的文件前提是这个外部的文件和主配置文件同级

4. <typeAliases> <package name="com.itheima.pojo"/> </typeAliases>,进行别名配置，其中name属性的值就是要扫描的包名,通过包扫描之后，该包里面的所有类都会配置别名，别名就是类名(不区分大小写)

5. 在environments标签里面配置数据库的环境信息

    1. 数据源dataSource的信息
    2. 事务的信息(了解)

6. 加载映射配置文件，通过包扫描的方式

    <mappers><package name="com.itheima.dao"/></mappers>，其中name属性的值就是要加载的所有映射配置文件所在的包名

延迟加载：

查询多表，只使用其中一个表的数据，就执行一个表的查询，如果后面再使用其他表数据，再执行其他表的查询；不用不查询，使用的时候再查询。

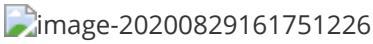
一对一查询：association　　一对多，多对多：collection

一级缓存：在一个sqlSession会话对象内执行查询操作会进行缓存
二级缓存：在一个接口内(一个名称空间内)不同SqlSession会话对象执行的查询操作都会进行缓存，跨sqlSession缓存。说明:接口映射配置文件中有namespace="接口类全名"，这代表一个名称空间内。

二级缓存如何开启：

1. 在核心配置文件中的settings标签中开启
2. 要在对应的映射配置文件中添加<catch/>标签
3. 要进行二级缓存的pojo类必须实现Serializable接口

ThreadLocal的底层原理是一个Map，key为当前线程对象，value就是存储的数据，这个数据在一个线程内是共享的
ThreadLocal的作用:可以实现一个线程内只操作一个数据,只共享一个数据；在多线程的时候,每个线程操作自己的数据



5、Spring

spring的优点：

1.方便解耦，简化开发(基础重要功能)

通过Spring提供的IOC容器，可以将对象之间的依赖关系交由Spring进行控制，避免硬编码所造成的过度程序耦合。有了Spring，用户不必再为单实例模式类、属性文件解析等这些很底层的需求编写代码，可以更专注于上层的应用。

2. AOP编程的支持(亮点)

通过Spring提供的AOP功能，方便进行面向切面的编程，许多不容易用传统OOP实现的功能可以通过AOP轻松应付。

3.声明式事务的支持 (简化事务)

在Spring中，通过声明式方式灵活地进行事务的管理，提高开发效率和质量

4.方便程序的测试 (集成Junit测试框架)

可以用非容器依赖的编程方式进行测试，例如：Spring对Junit4支持，可以通过注解方便的测试Spring程序。

5.方便集成各种优秀框架 (亮点、优势)

Spring不排斥各种优秀的开源框架，相反，Spring可以降低各种框架的使用难度，Spring提供了对各种优秀框架（如Struts、Hibernate、Hessian、Quartz）等的直接支持。

6.降低Java EE API的使用难度(了解)

Spring对很多难用的Java EE API（如JDBC，JavaMail，远程调用等）提供了一个薄薄的封装层，通过Spring的简易封装，这些Java EE API的使用难度大为降低

7.轻量级框架，全局异常处理，容器，MVC框架

IOC：

控制反转，对象的使用者不是创建者，把对象的创建和管理交给Spring框架。要使用的时候，直接从Spring工厂(容器)里面取，作用是解耦。

bean标签的scope属性：用来描述bean的作用范围

singleton：单例模式。spring创建bean对象时会以单例方式创建。(默认)

prototype：多例模式。spring创建bean对象时会以多例模式创建。

request：针对Web应用。spring创建对象时，会将此对象存储到request作用域。

session：针对Web应用。spring创建对象时，会将此对象存储到session作用域。

实例化Bean的三种方式：

1. 无参构造方法方式
2. 静态工厂方式（工厂方法是静态方法）
3. 实例工厂实例化的方法

spring 中的工厂：

BeanFactory是老版本使用的工厂,已经被废弃，ApplicationContext 是现在使用的工厂

ClassPathXmlApplicationContext：它是从类的根路径下加载配置文件

FileSystemXmlApplicationContext：它是从磁盘路径上加载配置文件，配置文件可以在磁盘任意位置

AnnotationConfigApplicationContext:当使用注解配置容器对象时，需要使用此类来创建spring容器，它用来读取注解

ApplicationContext加载方式是框架启动时就开始创建所有单例的bean,存到了容器里面，BeanFactory加载方式是用到bean时再加载。

DI：

依赖注入，我们交给Spring创建的Bean里面可能有一些属性(字段)，Spring帮我创建的同时也把Bean的一些属性(字段)给赋值。

DI的方式：

1. 构造方法方式注入
2. set方法方式的注入
3. P名称空间注入，其实底层也是调用set方法，只是写法有点不同
4. 静态工厂
5. 实例工厂

Spring5 的新特性：

1.该版本是基于jdk8编写的， 所以jdk8以下版本将无法使用

2.@Nullable和@NotNull注解来显示表明可为空的参数和以及返回值。这样就够在编译的时候处理空值而不是在运行时抛出 NullPointerExceptions。

3.Spring Framework 5.0带来了Commons Logging桥接模块的封装，它被叫做spring-jcl而不是标准的 Commons Logging。当然，无需任何额外的桥接，新版本也会对 Log4j 2.x，SLF4J，JUL(java.util.logging) 进行自动检测。

4.核心容器的更新：支持候选组件索引作为类路径扫描的替代方案，使用了组件索引，应用程序启动时间大大缩减

5.JetBrains Kotlin 语言支持；

6.响应式编程风格：响应式堆栈WEB框架。这个堆栈完全的响应式且非阻塞，可以进行少量线程的扩展；

7. Junit5 支持

AOP：

全称是Aspect Oriented Programming，即面向切面编程。在不修改源码的基础上，对我们的已有方法进行增强。说白了就是把我们程序重复的代码抽取出来，在需要执行的时候，使用动态代理的技术，进行增强。

作用：在程序运行期间，不修改源码对已有方法进行增强。

优势： 减少重复代码 提高开发效率 维护方便

AOP 这种思想还没有出现的时候，我们解决切面的问题思路无非有以下两种：

1. 方式一:通过静态方法实现(缺点:需要修改源码,后期不好维护)

   把需要添加的代码抽取到一个地方，然后在需要添加那些方法中引用

2. 方式二:通过继承方案来解决(缺点:需要修改源码,继承关系复杂,后期不好维护)

   抽取共性代码到父类，子类在需要的位置，调用父类方法。

AOP 的底层动态代理实现有两种方案：

一种是使用JDK的动态代理。基于接口实现，它的底层是创建接口的实现代理类，实现扩展功能。也就是我们要增强的这个类，实现了某个接口，那么我就可以使用这种方式。而另一种方式是使用了cglib的动态代理，基于继承实现，这种主要是针对没有接口的方式，那么它的底层是创建被增强目标类的子类，实现扩展功能。

AOP中的术语：

JoinPoint：连接点(所有可以被增强的方法)，类里面哪些方法可以被增强，这些方法称为连接点。在spring的AOP中，指的是业务层的类的所有现有的方法。

Pointcut：切入点(具体项目中真正已经被增强的方法)，在类里面可以有很多方法被增强，但是实际开发中，我们只对具体的某几个方法而已，那么这些实际增强的方法就称之为切入点

Advice：通知/增强（具体用于增强方法的代码)。增强的逻辑、称为增强，比如给某个切入点(方法)扩展了校验权限的功能，那么这个校验权限即可称之为增强或者是通知

- 通知分为：
- 前置通知：在原来方法之前执行.
- 后置通知：在原来方法之后执行. 特点：可以得到被增强方法的返回值
- 环绕通知：在方法之前和方法之后执行. 特点:可以阻止目标方法执行
- 异常通知：目标方法出现异常执行. 如果方法没有异常,不会执行. 特点:可以获得异常的信息
- 最终通知：指的是无论是否有异常，总是被执行的。

Aspect：切面(所有的通知都是在切面中的)

为什么HikariCP为什么越来越火?

效率高　一直在维护　SpringBoot2.0现在已经把HikariCP作为默认的连接池了

为什么HikariCP被号称为性能最好的Java数据库连接池?

1.优化并精简字节码：HikariCP利用了一个第三方的Java字节码修改类库Javassist来生成委托实现动态代理

2.优化代理和拦截器：减少代码，代码量越少，一般意味着运行效率越高、发生bug的可能性越低.

3.自定义集合类型ConcurrentBag：ConcurrentBag的实现借鉴于C#中的同名类，是一个专门为连接池设计的lock-less集合，实现了比LinkedBlockingQueue、LinkedTransferQueue更好的并发性能。ConcurrentBag内部同时使用了ThreadLocal和CopyOnWriteArrayList来存储元素，其中CopyOnWriteArrayList是线程共享的。ConcurrentBag采用了queue-stealing的机制获取元素：首先尝试从ThreadLocal中获取属于当前线程的元素来避免锁竞争，如果没有可用元素则再次从共享的CopyOnWriteArrayList中获取。此外，ThreadLocal和CopyOnWrite ArrayList在ConcurrentBag中都是成员变量，线程间不共享，避免了伪共享(false sharing)的发生。

4.使用FastList替代ArrayList：FastList是一个List接口的精简实现，只实现了接口中必要的几个方法。JDK ArrayList每次调用get()方法时都会进行rangeCheck检查索引是否越界，FastList的实现中去除了这一检查，只要保证索引合法那么rangeCheck就成为了不必要的计算开销(当然开销极小)。

编程式事务：

PlatformTransactionManager 平台事务管理器是一个接口，实现类就是Spring真正管理事务的对象。

常用的实现类：

DataSourceTransactionManager　:JDBC开发的时候(JDBCTemplate,MyBatis)，使用事务管理。

HibernateTransactionManager　:Hibernate开发的时候，使用事务管理。

声明式事务：

Spring的声明式事务的作用是：无论spring集成什么dao框架，事务代码都不需要我们再编写了，声明式的事务管理的思想就是AOP的思想。面向切面的方式完成事务的管理。声明式事务有两种,xml配置方式和注解方式。

事务的传播行为的作用：

如果Service层的这个方法中，除了调用了Dao层的方法之外，还调用了本类的其他的Service方法，那么在调用其他的Service方法的时候，必须保证两个service处在同一个事务中，确保事物的一致性。事务的传播特性就是解决这个问题的。

保证在同一个事务里面：

- PROPAGATION_REQUIRED:默认值，也是最常用的场景.
  如果当前没有事务，就新建一个事务，
  如果已经存在一个事务中，加入到这个事务中。
- PROPAGATION_SUPPORTS:
  如果当前没有事务，就以非事务方式执行。
  如果已经存在一个事务中，加入到这个事务中。

- PROPAGATION_MANDATORY
  如果当前没有有事务，就抛出异常；
  如果已经存在一个事务中，加入到这个事务中。

保证不在同一个事物里：

- PROPAGATION_REQUIRES_NEW
  如果当前有事务，把当前事务挂起,创建新的事务但独自执行
- PROPAGATION_NOT_SUPPORTED
  如果当前存在事务，就把当前事务挂起。不创建事务
- PROPAGATION_NEVER
  如果当前存在事务，抛出异常

spring循环依赖：

- 当两个bean内部互相注入的时候,会产生依赖问题,spring容器在初始化时,无法判断优先加载哪一个,导致爆出bean初始化异常：

依赖问题解决方案：

只要手动指定加载先后顺序,就可以解决;通过set注入,对其中任意一个bean添加@lazy注解;constuor注入,在构造参数前添加@lazy注解;

spring容器的启动过程：

第一步，资源定位，首先我们找到Spring配置文件的位置，创建类ClassPathResource，传入参数配置文件路径path；

第二步，创建IOC容器类DefaultListableBeanFactory，这个类实现了BeanDefinitionRegistry接口，这个接口有一个回调函数，后面调用能向IOC注册bean的定义信息；

第三步，读取配置文件，并将bean注册到IOC容器中去，具体步骤是先创建配置文件读取器XmlBeanDefinitionReader；再调用里面的读取文件的方法，并传入资源类loadBeanDefinitions（resource），从中读出spring的配置信息，并调用Bean注册回调函数向容器中注册Bean；

6、Springmvc

三层架构：

- 表现层：WEB层，用来和客户端进行数据交互的。表现层一般会采用MVC的设计模型（springmvc、struts）
- 业务层：处理具体的业务逻辑的（spring）
- 持久层：用来操作数据库的（myBatis、Hibernate、JDBC）

MVC：Model View Controller 模型视图控制器，每个部分各司其职。

- Model：数据模型，JavaBean的类，用来进行数据封装
- View：指JSP、HTML用来展示数据给用户
- Controller：用来接收用户的请求，整个流程的控制器。用来进行数据校验等

SpringMVC是一种基于Java实现的MVC设计模型的请求驱动类型的轻量级WEB层框架

作用：用于替代我们之前Web里面的Servlet的相关代码

1. 参数绑定(获得请求参数)　2.调用业务　3.响应　4.分发转向等等

SpringMVC的三大组件：处理器映射器、处理器适配器、视图解析器

什么东西配置在web.xml里面：

之前web里面学习到的Servlet、Filter、Listener，其它的东西都配置在springmvc的配置文件里面

客户端传入给服务器端的请求参数一般分为两种：

1.formdata类型的请求参数，例如:"username=aobama&pwd=123&nickname=圣枪游侠"，

2.json类型的请求参数，例如使用axios发送异步的post请求携带的请求参数，而RequestBody注解的作用就是将json类型的请求参数封装到POJO对象或者Map中

转发和重定向区别：

1. 转发是一次请求，重定向是两次请求
2. 转发路径不会变化，重定向的路径会改变

3. 转发只能转发到内部的资源,重定向可以重定向到内部的(当前项目里面的)也可以是外部的(项目以外的)
4. 转发可以转发到web-inf里面的资源， 重定向不可以重定向到web-inf里面的资

解决静态资源会被DispatcherServlet拦截的问题，只会发生在DispatcherServlet的映射路径配置的是"/"的时候
方案1: <mvc:resource mapping="/js/**" location="/js/"/> ,这种方案不建议使用
方案2: 将所有静态资源交由DefaultServlet进行处理， <mvc:default-servlet-handler />
方案3: 将DispatcherServlet的映射路径配置成 "*.do",那么他就只会拦截以.do结尾的请求

文件上传要求:
浏览器端：1.提交方式 post 2.文件上传框input type="file" 3.表单的entype属性 multipart/form-data
服务器端：借助第三方组件(jar,框架)实现文件上传，serlvet3.0(原生的文件上传的API) commons-fileupload
SpringMVC(底层封装了:commons-fileupload)

Spring MVC的处理器拦截器类似于Servlet开发中的过滤器Filter，用于对处理器(自己编写的Controller)进行预处理和后处理。用户可以自己定义一些拦截器来实现特定的功能。拦截器链（Interceptor Chain）。拦截器链就是将拦截器按一定的顺序联结成一条链。在访问被拦截的方法或字段时，拦截器链中的拦截器就会按其之前定义的顺序被调用。
我们要想自定义拦截器，要求必须实现：HandlerInterceptor接口。
拦截器与过滤器的区别：
（1）拦截器是基于java的反射机制的，而过滤器是基于函数回调。
（2）拦截器不依赖于servlet容器，而过滤器依赖于servlet容器。
（3）拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。
（4）拦截器可以访问action上下文、值栈里的对象，而过滤器不能。
（5）在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次。
（6）拦截器可以获取IOC容器中的各个bean，而过滤器就不行。

多个过滤器的执行顺序:
1. 如果采用配置文件方式配置过滤器，那么就按照过滤器的配置先后顺序执行
2. 如果采用注解方式配置过滤器，那么就按照类名的排序执行

能够让专门的配置文件做专门的事情:
1. springmvc的配置文件:
    1. 包扫描
    2. 加载mvc注解驱动
    3. 配置springmvc的各种组件:视图解析器、文件解析器、异常处理器、类型转换器、拦截器等等
    4. 导入其它的配置文件

2. spring的配置文件:
    1. ioc和依赖注入的配置
    2. 声明式事务的配置
    3. aop的配置
    4. 导入其它的配置文件

3. spring-mybatis.xml,做整合的事情
4. log4j.properties 日志框架的配置文件
5. db.properties 数据库环境的配置文件
6. web.xml,里面是配置Servlet和Filter、Listener
7. mybatis的主配置文件、mybatis的映射配置文件

7、SpringBoot

基本介绍:

SpringBoot是Spring开源组织下的子项目，主要是简化了使用Spring的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手。

优点：

1.独立运行：Spring Boot而且内嵌了各种servlet容器，Tomcat、Jetty等，现在不再需要打成war包部署到容器中，Spring Boot只要打成一个可执行的jar包就能独立运行，所有的依赖包都在一个jar包内；

2.简化配置：spring-boot-starter-web启动器自动依赖其他组件，简少了maven的配置；

3.自动配置：Spring Boot能根据当前类路径下的类、jar包来自动配置bean，如添加一个spring-boot-starter-web启动器就能拥有web的功能，无需其他配置。自动配置也可以理解为，约定优于配置。

4.无代码生成和XML配置：Spring Boot配置过程中无代码生成，也无需XML配置文件就能完成所有配置工作，这一切都是借助于条件注解完成的，这也是Spring4.x的核心功能之一。

获取配置文件中的值我们一般有几种方式：

1.@value注解的方式 只能获取简单值

2.Environment的方式

3.@ConfigurationProperties

激活profile的方式（了解）

1.配置文件的方式 spring.profiles.active=test

2.运行是指定参数 java -jar xxx.jar --spring.profiles.active=test

3.jvm虚拟机参数配置 -Dspring.profiles.active=dev

springBoot版控制的原理；

自己的springboot项目继承于spring-boot-starter-parent,而他又继承与spring-boot-dependencies ，dependencies中定义了各个版本。通过maven的依赖传递特性从而实现了版本的统一管理。

SpringBoot的自动配置原理：

condition接口是spring4之后提供给了的接口，增加条件判断功能，用于选择性的创建Bean对象到spring容器中。@Conditional(value = OnClassCondition.class) 当符合指定类的条件返回true的时候则执行被修饰的方法，放入spring容器中。

@SpringBootApplication注解里面有@EnableAutoConfiguration，这种@Enable开头就是springboot中定义的一些动态启用某些功能的注解，他的底层实现原理实际上用的就是@import注解导入一些配置，自动进行配置，加载Bean。

在启动类的注解@springbootapplication注解里面又修饰了@compnetScan注解，该注解的作用用于组件扫描包类似于xml中的context-componet-scan，如果不指定扫描路径，就扫描该注解修饰的启动类所在的包以及子包。

import注解用于导入其他的配置，让spring容器进行加载和初始化。import的注解有以下的几种方式使用：

1.直接导入Bean

2.导入配置类

3.导入ImportSelector的实现类，通常用于加载配置文件中的Bean

4.导入ImportBeanDefinitionRegistrar实现类

自动配置的流程：

1.@import注解 导入配置

2.selectImports导入类中的方法中加载配置返回Bean定义的字符数组

3.加载META-INF/spring.factories 中获取Bean定义的全路径名返回

4.最终返回回去即可

SpringBoot的监控：

SpringBoot自带监控功能Actuator，可以帮助实现对程序内部运行情况监控，比如监控状况、Bean加载情况、配置属性、日志信息等。

Spring Boot Admin是一个开源社区项目，用于管理和监控SpringBoot应用程序

SpringBoot部署：

war包部署时，要修改启动类配置，需要继承SpringBootServletInitializer，目的是指定要使用外部的tomcat容器运行程序。

springboot事件监听机制：

1. CommandLineRunner 在容器准备好了之后可以回调 @componet修饰即可
2. ApplicationRunner 在容器准备好了之后可以回调 @componet修饰即可
3. ApplicationContextInitializer 在spring在刷新之前调用该接口的方法用于：做些初始化工作通常用于web环境，用于激化配置，web上下问的属性注册。注意：需要配置META/spring.factories 配置之后才能加载调用
4. SpringApplicationRunListener 也需要配置META/spring.factories 配置之后才能加载调用他是SpringApplication run方法的监听器，当我们使用SpringApplication调用Run方法的时候触发该监听器回调方法。注意：他需要有一个公共的构造函数，并且每一次RUN的时候都需要重新创建实例

Spring的整个启动过程：准备Environment--发布事件--创建上下文、bean--刷新上下文--结束

启动流程主要分为三个部分，第一部分进行SpringApplication的初始化模块，配置一些基本的环境变量、资源、构造器、监听器，第二部分实现了应用具体的启动方案，包括启动流程的监听模块、加载配置环境模块、及核心的创建上下文环境模块，第三部分是自动化配置模块，该模块作为springboot自动配置核心。

https://blog.csdn.net/hfmbook/article/details/100507083

8、SpringCloud

9、SpringData

Spring Data Jpa
Spring Data ES

10、RestFul

在目前主流的三种Web服务交互方案中，REST相比于SOAP（Simple Object Access protocol，简单对象访问协议）以及XML-RPC更加简单明了，无论是对URL的处理还是对Payload的编码，REST都倾向于用更加简单轻量的方法设计和实现。REST并没有一个明确的标准，而更像是一种设计的风格。
优点：它结构清晰、符合标准、易于理解、扩展方便
特性：

资源（Resources）：网络上的一个实体，或者说是网络上的一个具体信息。它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的存在。可以用一个URI（统一资源定位符）指向它，每种资源对应一个特定的URI。要获取这个资源，访问它的URI就可以，因此URI即为每一个资源的独一无二的识别符。

表现层（Representation）：把资源具体呈现出来的形式，叫做它的表现层（Representation）。比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式。

状态转化（State Transfer）：每发出一个请求，就代表了客户端和服务器的一次交互过程。

HTTP协议，是一个无状态协议，即所有的状态都保存在服务器端。因此，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生“状态转化”（State Transfer）。而这种转化是建立在表现层之上的，所以就是 “表现层状态转化”。具体说，就是HTTP协议里面，四个表示操作方式的动词： GET、POST、PUT、DELETE。它们分别对应四种基本操作：GET用来获取资源， POST用来新建资源， PUT用来更新资源， DELETE用来删除资源。

11、Sql语句调优经验

查询慢，索引失效

12、Redis

主从复制，以及哨兵和集群之间区别：

主从复制：是redis实现高可用的一个策略。将会有主节点和从节点，从节点的数据完整的从主节点中复制一份。

哨兵：当系统节点异常宕机的时候，开发者可以手动进行故障转移恢复，但是手动比较麻烦，所以通过哨兵机制自动进行监控和恢复。为了解决哨兵也会单点故障的问题，可以建立哨兵集群。

集群：即使使用哨兵，redis每个实例也是全量存储，每个redis存储的内容都是完整的数据，浪费内存且有木桶效应。为了最大化利用内存，可以采用集群，就是分布式存储。这个时候可以使用redis集群。将不同的数据分配到不同的节点中，这样就可以横向扩展，扩容。

如何实现分布式锁，哪些地方用到了分布式锁：

秒杀模块中：下单完成后，修改库存的时候用到了，从redis中取出商品判断库存数量和更改库存的操作放到一个事务中，防止出现超卖；另外，支付失败时，需要删除排队信息、抢单信息、订单信息及恢复库存，需要放到一个事务中。

缓存雪崩的解决方案：

1.限流（令牌桶或者漏桶）

2.数据预热（loadding 到服务器redis中）

3.热点数据要均匀的分步到不同的redis中（可以使用算法：一致性hash算法）

4.采用大数据分析实现冷热数据隔离，将热点Key均匀分布。

5.多级缓存的方式。（分布式缓存+本地缓存（MyBatis）的方式来实现）+ hystrix

缓存穿透：Redis缓存中没有，数据库也没有的情况　解决方案：将null值存入redis，并设置一个过期时间。

缓存空对象会有两个问题：

第一，空值做了缓存，意味着缓存层中存了更多的键，需要更多的内存空间 （ 如果是攻击，问题更严重），比较有效的方法是针对这类数据设置一个较短的过期时间，让其自动剔除。

第二，缓存层和存储层的数据会有一段时间窗口的不一致，可能会对业务有一定影响。例如过期时间设置为5分钟，如果此时存储层添加了这个数据，那此段时间就会出现缓存层和存储层数据的不一致，此时可以利用消息系统或者其他方式清除掉缓存层中的空对象。

思考一下就可以想到,当我们设置了SETNX后,如果马上挂掉了,那我们再次EXPIRE 指令是不是就会无法生效,出现了资源锁死的情况,独占资源所以针对这样问题redis，在2.6.12版本过后,有了一个新的决绝方案

```
SET key value [EX seconds] [PX millisecounds] [NX|XX]
EX seconds:设置键的过期时间为second秒
PX millisecounds:设置键的过期时间为millisecounds 毫秒
NX:只在键不存在的时候,才对键进行设置操作
XX:只在键已经存在的时候,才对键进行设置操作
SET操作成功后,返回的是OK,失败返回NIL
例如： set name zhangsan  ex 30 nx
```

(1)在多线程环境下,使用redis中的setnx可避免发生线程不安全问题;当一个线程执行该命令成功后,会返回一个值为1,那么当下一个线程要进行操作时,看到该状态就会阻塞,等到上一个线程操作完任务,下一个就会重新进行setnx;

13、分布式事务

分布式事务指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于不同的分布式系统的不同节点之上,且属于不同的应用，分布式事务需要保证这些操作要么全部成功，要么全部失败。本质上来说，分布式事务就是为了保证不同数据库的数据一致性。

1. 2PC：基于XA协议的两阶段提交

优点： 尽量保证了数据的强一致，适合对数据强一致要求很高的关键领域。

缺点： 牺牲了可用性，对性能影响较大，不适合高并发高性能场景，要求数据库支持XA协议

2. TCC：补偿事务

1. tcc 性能要优于xa　适用于高并发的场景
2. 实现分布式事务 不依赖于具体支持RM的的数据库，可以任意的数据库，逻辑在业务层实现
3. 数据一致性 弱于XA
4. 业务逻辑都需要程序员自己来实现 很麻烦。应用层的一种补偿方式

3. 本地消息表：（异步确保）

核心思想是将分布式事务拆分成本地事务进行处理，消息生产方，需要额外建一个消息表，并记录消息发送状态。然后消息会经过MQ发送到消息的消费方。如果消息发送失败，会进行重试发送。消息消费方，需要处理这个消息，并完成自己的业务逻辑。此时如果本地事务处理成功，表明已经处理成功了，如果处理失败，那么就会重试执行。如果是业务上面的失败，可以给生产方发送一个业务补偿消息，通知生产方进行回滚等操作。生产方和消费方定时扫描本地消息表，把还没处理完成的消息或者失败的消息再发送一遍。如果有靠谱的自动对账补账逻辑，这种方案还是非常实用的。

1.避免了使用分布式事务的XA tcc ,通过MQ实现数据的最终一致性。

2.实现简单

3.性能有保证　特别符合高并发的场景。

劣势：消息表会耦合到业务系统中

4. MQ事务消息，仅rocketMQ支持

在业务方法内要向消息队列提交两次请求，一次发送消息和一次确认消息。如果确认消息发送失败了RocketMQ会定期扫描消息集群中的事务消息，这时候发现了Prepared消息，它会向消息发送者确认，所以生产方需要实现一个check接口，RocketMQ会根据发送端设置的策略来决定是回滚还是继续发送确认消息。这样就保证了消息发送与本地事务同时成功或同时失败。

几种实现方式的对比；

+ XA　强一致性　性能差　（支持XA协议的关系型数据库）
+ TCC　弱一致性　性能好　需要写很多补偿代码
+ MQ　数据最终一致性 性能最好 消息丢失 重复消费 数据不一致。

5.Seate

seata中有两种分布式事务实现方案，AT及TCC

AT模式主要关注多DB访问的数据一致性，当然也包括多服务下的多DB数据访问一致性问题

TCC 模式主要关注业务拆分，在按照业务横向扩展资源时，解决微服务间调用的一致性问题

AT模式：

Transaction Coordinator （TC）： 事务协调器，维护全局事务的运行状态，负责协调并驱动全局事务的提交或回滚。

Transaction Manager（TM）： 控制全局事务的边界，负责开启一个全局事务，并最终发起全局提交或全局回滚的决议。

Resource Manager （RM）： 控制分支事务，负责分支注册、状态汇报，并接收事务协调器的指令，驱动分支（本地）事务的提交和回滚。

第一阶段：Seata的JDBC数据源代理通过对业务SQL的解析，把业务数据在更新前后的数据镜像组织成回滚日志，利用本地事务的ACID特性，将业务数据的更新和回滚日志的写入在同一个本地事务中提交。这样，可以保证任何提交的业务数据的更新一定有相应的回滚日志存在,基于这样的机制，分支的本地事务便可以在全局事务的第一阶段提交，并马上释放本地事务锁定的资源。这也是Seata和XA事务的不同之处，两阶段提交往往对资源的锁定需要持续到第二阶段实际的提交或者回滚操作，而有了回滚日志之后，可以在第一阶段释放对资源的锁定，降低了锁范围，提高效率，即使第二阶段发生异常需要回滚，只需找对undolog中对应数据并反解析成sql来达到回滚目的，同时Seata通过代理数据源将业务sql的执行解析成undolog来与业务数据的更新同时入库，达到了对业务无侵入的效果。

第二阶段：如果决议是全局提交，此时分支事务此时已经完成提交，不需要同步协调处理（只需要异步清理回滚日志）.如果决议是全局回滚，RM收到协调器发来的回滚请求，通过XID和Branch ID找到相应的回滚日志记录，通过回滚记录生成反向的更新SQL并执行，以完成分支的回滚。

TCC模式：

基本思路就是使用侵入业务上的补偿及事务管理器的协调来达到全局事务的一起提交及回滚。

14、Zookeeper

ZooKeeper 本质上是一个分布式的小文件存储系统。提供基于类似于文件系统的目录树方式的数据存储，并且可以对树中的节点进行有效管理。从而用来维护和监控存储的数据的状态变化。通过监控这些数据状态的变化，从而可以达到基于数据的集群管理。ZooKeeper适用于存储和协同相关的关键数据，不适合用于大数据量存储。

应用场景：

1.注册中心 2.配置中心 3.分布式锁 4.分布式队列 5.负载均衡　　分布式锁：没了解

核心功能：

1.管理用户程序提交的数据
2.为用户程序提供数据节点监听服务

角色:leader、follower、observer

1.Leader作为整个ZooKeeper集群的主节点，负责响应所有对ZooKeeper状态变更的请求。它会将每个状态更新请求进行排序和编号，以便保证整个集群内部消息处理的FIFO。

2.Follower的逻辑就比较简单了。除了响应本服务器上的读请求外，follower还要处理leader的提议，并在leader提交该提议时在本地也进行提交。

如果ZooKeeper集群的读取负载很高，或者客户端多到跨机房，可以设置一些observer服务器，以提高读取的吞吐量。 3.Observer和Follower比较相似，只有一些小区别：首先observer不属于法定人数，即不参加选举也不响应提议；其次是observer不需要将事务持久化到磁盘，一旦observer被重启，需要从leader重新同步整个名字空间。

特性：

1.Zookeeper：一个leader，多个follower组成的集群

2.全局数据一致：每个server保存一份相同的数据副本，client无论连接到哪个server，数据都是一致的

3.分布式读写，更新请求转发，由leader实施

4.更新请求顺序进行，来自同一个client的更新请求按其发送顺序依次执行

5.数据更新原子性，一次数据更新要么成功，要么失败

6.实时性，在一定时间范围内，client能读到最新数据

ZooKeeper的数据模型：

层次模型。层次模型常见于文件系统。层次模型和key-value模型是两种主流的数据模型。

ZooKeeper使用文件系统模型主要基于以下两点考虑：

1. 文件系统的树形结构便于表达数据之间的层次关系。
2. 文件系统的树形结构便于为不同的应用分配独立的命名空间（namespace）。

节点的分类：持久性znode、临时性znode、持久顺序性znode、临时顺序性Znode

半数存活机制：只要有半数以上的节点存活,集群就能提供服务(==适合装在奇数台机器上==)

常用命令：

-e:表示普通临时节点 -s:表示带序号节点

1.查询所有命令 help

2.查询根路径下的节点 ls /zookeeper

3.创建普通永久节点 create /app1 "helloworld"

4.创建带序号永久节点 create -s /hello "helloworld"

5.创建普通临时节点 create -e /app3 'app3'

6.创建带序号临时节点 create -e -s /app4 'app4'

7.查询节点数据 get /app1

8.修改节点数据 set /app1 'hello'

9.删除节点 delete /hello0000000006

10.递归删除节点 rmr /hello

11.查看节点状态 stat /zookeeper

ZooKeepe常用Java API：

1.原生Java API（不推荐使用）

2.Apache Curator（推荐使用）

3.ZkClient（不推荐使用）

zookeeper的watch机制：

zookeeper的订阅发布也就是watch机制，是一个轻量级的设计。因为它采用了一种推拉结合的模式。一旦服务端感知主题变了，那么只会发送一个事件类型和节点信息给关注的客户端，而不会包括具体的变更内容，所以事件本身是轻量级的，这就是所谓的“推”部分。然后，收到变更通知的客户端需要自己去拉变更的数据，这就是“拉”部分。watche机制分为添加数据和监听节点。

Curator在这方面做了优化，Curator引入了Cache的概念用来实现对ZooKeeper服务器端进行事件监听。Cache是Curator对事件监听的包装，其对事件的监听可以近似看做是一个本地缓存视图和远程ZooKeeper视图的对比过程。而且Curator会自动的再次监听，我们就不需要自己手动的重复监听了。

Curator中的cache共有三种：

- NodeCache（监听和缓存根节点变化）
- PathChildrenCache（监听和缓存子节点变化）
- TreeCache（监听和缓存根节点变化和子节点变化）

投票选举机制：

1.全新集群（paxos：算法）

共五人参加选举，第一轮只能选自己，获得选票大于5/2(即2)时，成为leader.

回合一:A启动，开始第一轮选举，A选择自己.投票结果:[(A-->A)]A只获得一票，小于2所以不能成为leader，进入LOOKING(竞选状态)。

回合二:B启动起来，投票给自己，告诉其他人，并接收到A的(第一回合)投票信息;此时，A接收到B的(第二回合)投票信息，A知道进入第二回合了，重新投票，由于B的leaderID大，所以A投给了B.投票结果:[(A-->B)，(B-->B)]B获得两票，但是仍没有超过2，所以A，B进入LOOKING(竞选状态)。

回合三:C启动起来，投票给自己，告诉其他人，并接收到了A和B的投票信息此时，AB也接收到了C的投票信息，一看第三回合了，重新投票，由于C的leaderID最大，就将自己的一票都给了C投票结果:[(A-->C)，(B-->C)，(C-->C)]，C获得三票，大于2，所以C进入leaderinng状态

剩下回合，由于C获得的票数已经超过半数，所以D和E的投票是不能改变现实的有人可能会疑问，D的leader比C大啊，他们应该都选D啊?不要误解了，当集群中没有leader的时候，大家在新的回合会将自己的一票投给leaderID大的同学，但是如果已经有了leader，那在新的回合将不会改变他们的投票信息

2.非全新集群的选举机制(数据恢复)

那么，初始化的时候，是按照上述的说明进行选举的，但是当zookeeper运行了一段时间之后，有机器down掉，重新选举时，选举过程就相对复杂了。需要加入数据id、leader id和逻辑时钟。

数据id：数据新的id就大，数据每次更新都会更新id。

Leader id：就是我们配置的myid中的值，每个机器一个。

逻辑时钟：这个值从0开始递增，每次选举对应一个值，也就是说： 如果在同一次选举中，那么这个值应该是一致的 ； 逻辑时钟值越大，说明这一次选举leader的进程更新.

选举的标准就变成：

1. 逻辑时钟小的选举结果被忽略，重新投票
2. 统一逻辑时钟后，数据id大的胜出
3. 数据id相同的情况下，leader id大的胜出，根据这个规则选出leader。

2、如何查看一个节点是否注册成功?

2.1 在服务器上看：

1）查找zookeeper的目录： find / -name zookeeper

2）进入zookeeper的bin目录： /data/opt/src/zookeeper-3.4.9/bin

3）执行zkcli.sh命令， ./zkCli.sh

4）查看有哪些zookeeper节点： ls /

5）查看注册了哪些服务， ls /daily_orderServer_group（节点名称）

2.2 在dubboadmin的网页看更方便

1）可以搜索服务名，比如： CancelOrderService

2）或者搜索应用名，比如： orderServiceServerApplication

3）或者搜索机器IP， 比如：192.168.1.222:20886

备注：如果不知道应用名或者机器IP的配置：

1）在linux执行命令查找dubbo配置目录：

find / -name orderservice_server_apphome.properties

2）执行命令查看dubbo配置内容：

cat /data/appHome/orderServiceServerConfig/orderservice_server_apphome.properties

https://blog.csdn.net/weixin_34126557/article/details/91832026?biz_id=102&utm_term=zookeeper%E5%A6%82%E4%BD%95%E6%9F%A5%E7%9C%8B%E8%8A%82%E7%82%B9%E6%B3%A8%E5%86%8C%E6%88%90%E5%8A%9F&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-91832026&spm=1018.2118.3001.4187](https://blog.csdn.net/weixin_34126557/article/details/91832026?biz_id=102&utm_term=zookeeper如何查看节点注册成功&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-91832026&spm=1018.2118.3001.4187)

3、如何删除一个节点

ls：查看所有节点　找到要删除的节点
删除指令：delete　/node_1/node_1_100000001
delete只能删除不包含子节点的节点　如果含有子节点 使用rmr命令 rmr　/node_1

15、Dubbo

架构演进：
1：单体架构全部功能集中在一个项目内（耦合度高）。
2：垂直架构按照业务进行切割，形成小的单体项目（代码冗余）。
3：SOA架构（项目一）
面向服务的架构（SOA）是一个组件模型,全称为Service-Oriented Architecture，它将应用程序的不同功能单元（称为服务）进行拆分，并通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种各样的系统中的服务可以以一种统一和通用的方式进行交互。可以使用dubbo作为调度的工具（RPC协议）
4：微服务架构（项目二）
将系统服务层完全独立出来，抽取为一个一个的微服务
特点一：抽取的粒度更细，遵循单一原则，数据可以在服务之间完成数据传输（一般使用restful请求调用资源）
特点二：采用轻量级框架协议传输。（可以使用springcloudy）（http协议）
特点三：每个服务都使用不同的数据库，完全独立和解耦

RPC：Remote Procedure Call 远程过程调用，是一种进程间的通信方式，分为同步调用和异步调用
Java中的RPC框架：RMI、Hessian、Dubbo、spring Cloud等，RPC的框架，要求传递的参数和实体类要实现序列化

Dubbo：
一款高性能的Java RPC框架。Dubbo提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。
注意：
1.消费者与提供者应用名称不能相同
2.如果有多个服务提供者，名称不能相同，通信端口也不能相同
3.只有服务提供者才会配置服务发布的协议，默认是dubbo协议，端口号是20880
Dubbo支持的协议有：dubbo、rmi、hessian、http、webservice、rest、redis等
dubbo协议采用单一长连接和NIO异步通讯，适合于小数据量、大并发的服务调用，以及服务消费者机器数远大于服务提供者机器数的情况。不适合传送大数据量的服务，比如传文件，传视频等，除非请求量很低。
dubbo协议：
- 连接个数：单连接
- 连接方式：长连接
- 传输协议：TCP
- 传输方式：NIO异步传输
- 序列化：Hessian二进制序列化
- 适用范围：传入传出参数数据包较小（建议小于100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要用dubbo协议传输大文件或超大字符串。
- 适用场景：常规远程服务方法调用

rmi协议：
- 连接个数：多连接
- 连接方式：短连接
- 传输协议：TCP
- 传输方式：同步传输
- 序列化：Java标准二进制序列化
- 适用范围：传入传出参数数据包大小混合，消费者与提供者个数差不多，可传文件
- 适用场景：常规远程服务方法调用，与原生RMI服务互操作

默认的情况下，dubbo调用的时间为一秒钟，如果超过一秒钟就会报错

Dubbo 提供了多种均衡策略（包括随机random、轮询roundrobin、最少活跃调用数leastactive），默认为random随机调用。配置负载均衡策略，既可以在服务提供者一方配置（@Service(loadbalance = "roundrobin")），也可以在服务消费者一方配置（@Reference(loadbalance = "roundrobin")）

实现配置中心：
1：在zookeeper中添加数据源所需配置
2：在dubbo-common中导入jar包
3：修改数据源，读取zookeeper中数据源所需配置数据
（1）在dubbo-common中创建工具类：SettingCenterUtil,继承PropertyPlaceholderConfigurer
（2）编写载入zookeeper中配置文件，传递到Properties属性中
（3）重写processProperties方法
（4）修改spring-dao.xml
4：watch机制
（1）添加监听
（2）获取容器对象，刷新spring容器：SettingCenterUtil,实现ApplicationContextAware

16、RabbitMQ

应用场景：
做微信支付的时候用到了，支付微服务发送消息，订单微服务处理消息
具体使用逻辑：
1.用户调用支付微服务（changgou-service-pay）生成微信支付二维码的时候，传递exchange、routingkey、queue信息给微信服务器
2.用户支付完成后，微信服务器通过回调地址通知支付微服务（changgou-service-pay），拿到微信服务器透传过来exchange、routingkey、queue信息进行消息的发送
3.订单微服务（changgou-service-order）收到消息，更新订单表里面对应字段的状态
这里就实现了支付微服务和订单微服务的解耦，这里也实现了支付功能和订单状态修改的异步处理

mq保证消息的顺序 具体实现 参考面试宝典

17、ES

1、如何对查询出来的商品进行排序？sort
2、如何实现IK分词器的动态变化？
3、大量数据如何导入？分批次导入，多次线程
4、实际工作中遇到了什么问题？有什么优化？
5、到底什么时候使用filter？ 什么时候使用Query？

介绍：
1.开源的Elasticsearch（简称ES）是开源搜索引擎，基于Lucene。
2.可以通过RESTful API进行全文搜索。
3.近实时，es从数据写入到数据被搜索到有一个延时（大概1秒），基于es执行的搜索和分析可以达到秒级。
4.作为一个大型分布式集群（数百台服务器）技术，处理海量数据（PB级）数据，服务大公司；也可以运行在单机上，服务小公司
5.基本概念索引、类型、文档、字段。
使用步骤：
搭建集群
安装ik分词器，配置自定义分词词库
定义Bean
定义索引名称、类型名称、文档名称
定义映射关系
哪些需要分词-》name
哪些不需要分词-》categoryName，brandName

规格使用动态类型Object-》private Map<String, Object> specMap
索引初始化
动态索引更新
关键字搜索
聚合查询（普通域、动态域），显示品牌和规格
多条件过滤查询
范围过滤查询
分页搜索
排序搜索
高亮搜索

18、Linux

常用命令：
查看磁盘使用空间：df -hl复制代码
查看网络是否连通：netstat
查看各类环境变量:env
查找命令的可执行文件：whereis [-bfmsu][-B <目录>...][-M <目录>...][-S <目录>...][文件...]
which 只能查可执行文件　whereis 只能查二进制文件、说明文档，源文件等
当前系统支持的所有命令的列表： compgen -c
显示当前所在目录：pwd

19、Vue

Vue.js 的目标是通过尽可能简单的API实现响应的数据绑定和组合的视图组件。
常用指令：
v-if：根据表达式的值的真假条件渲染元素
v-show：根据表达式之真假值，切换元素的display CSS 属性
v-for：循环指令
v-bind：属性绑定
v-on：时间绑定
v-model：实现表单输入和应用状态之间的双向绑定

1. 理解MVVM双向绑定

2. vue的入门,一些基本格式步骤(重点)
    1. 在要使用vue的页面引入vue.js文件
    2. 准备一个vue的容器
    3. 在容器下面添加一个script标签，创建vue对象
    4. 在vue对象中添加参数
        1. el ： "#容器的id"
        2. data ： {数据模型}
        3. methods ： {定义的方法}

3. 通过插值表达式实现数据和视图的绑定(不重要)

4. 使用vue绑定各种事件:(重点)
        1. 点击事件 @click="函数"
        2. 键盘按键按下事件 @keydown="函数($event)",$event就是当前的事件,它的keyCode属性就是获取按键的建码值，preventDefault()阻止事件发生
        3. 鼠标移入事件 @mouseover="函数"

5. v-text和v-html 绑定标签体的内容(重点)

6. v-bind可以绑定属性:(了解)
    v-bind:href="数据模型"　　可以简写 :href="数据模型"

7. v-model可以绑定表单项的值(重点)
   v-model="数据模型"

8. v-for 进行遍历(重点)
   v-for="(遍历出来的元素,元素的下标) in 要遍历的数据模型" ， v-for是使用在需要重复创建的标签上

9. v-if和v-show控制标签的显示和隐藏(了解)
      他俩绑定的是boolean类型的数据
      v-if 是通过删除标签来进行隐藏的
      v-show 是通过设置display属性来进行隐藏的

10. 生命周期和钩子函数
      1. created()钩子函数，在页面加载的时候，发送异步请求获取服务器端的数据(重点)
      2. mounted()钩子函数，在页面加载的时候，获取视图里的数据
      3. mounted函数在created函数之后执行

11. axios发送异步请求(最重点)
    1. get请求
      1. 请求路径url
      2. 请求参数,get方式无论采用哪种方式携带请求参数，其实都是在url路径后面通过"?"携带，所以Servlet里面都是使用request.getParameter()获取
      3. 处理响应数据
         .then(resopnse=>{}) response就是服务器端的响应对象(包含响应的行头体)，所以我们如果需要获取响应体的内容则需要response.data获取
    2. post请求
      只有请求参数的携带和get不同
        1. 可以通过url后面的"?"携带
        2. 在请求体中携带json类型的参数

20、数据结构

十大排序算法：
https://www.cnblogs.com/onepixel/articles/7674659.html
二叉搜索树：
1.所有非叶子结点至多拥有两个儿子（Left和Right）；
2.所有结点存储一个关键字；
3.非叶子结点的左指针指向小于其关键字的子树，右指针指向大于其关键字的子树；
二叉搜索树的问题：不平衡时，例如每个节点都只有一个子节点，查找效率就是线性查找，失去二分查找的意义，因此平衡二叉查找树应运而生。使用平衡算法使二叉树变成平衡二叉树，平衡算法就是一种在二叉搜索树中插入和删除结点的策略。

B树（平衡的多路搜索树，并不是二叉的）：
定义：
1.定义任意非叶子结点最多只有M个儿子；且M>2；
2.根结点的儿子数为[2，M]；
3.除根结点以外的非叶子结点的儿子数为[M/2，M]；
4.每个结点存放至少M/2-1（取上整）和至多M-1个关键字；（至少2个关键字）
5.非叶子结点的关键字个数=指向儿子的指针个数-1；
6.非叶子结点的关键字：K[1]，K[2]，…，K[M-1]；且K[i] < K[i+1]；
7.非叶子结点的指针：P[1]，P[2]，…，P[M]；其中P[1]指向关键字小于K[1]的子树，P[M]指向关键字大于K[M-1]的子树，其它P[i]指向关键字属于(K[i-1]，K[i])的子树；
8.所有叶子结点位于同一层；
搜索：
从根结点开始，对结点内的关键字（有序）序列进行二分查找，如果命中则结束，否则进入查询关键字所属范围的儿子结点；重复，直到所对应的儿子指针为空，或已经是叶子结点。

特性：

1.关键字集合分布在整颗树中；

2.任何一个关键字出现且只出现在一个结点中；

3.搜索有可能在非叶子结点结束；

4.其搜索性能等价于在关键字全集内做一次二分查找；

5.自动层次控制；

由于限制了除根结点以外的非叶子结点，至少含有M/2个儿子，确保了结点的至少利用率。B树就是B-树，

B+树：

定义：

1.其定义基本与B树相同，除了：

2.非叶子结点的子树指针与关键字个数相同；

3.非叶子结点的子树指针P[i]，指向关键字值属于[K[i]，K[i+1])的子树（B-树是开区间）；

5.为所有叶子结点增加一个链指针；

6.所有关键字都在叶子结点出现；

搜索：

B+的搜索与B-树也基本相同，区别是B+树只有达到叶子结点才命中（B-树可以在非叶子结点命中），其性能也等价于在关键字全集做一次二分查找。

特性：

1.所有关键字都出现在叶子结点的链表中（稠密索引），且链表中的关键字恰好是有序的；

2.不可能在非叶子结点命中；

3.非叶子结点相当于是叶子结点的索引（稀疏索引），叶子结点相当于是存储（关键字）数据的数据层；

4.更适合文件索引系统；

B*树：

是B+树的变体，在B+树的非根和非叶子结点再增加指向兄弟的指针；B*树定义了非叶子结点关键字个数至少为(2/3)*M，即块的最低使用率为2/3（代替B+树的1/2）。所以，B*树分配新结点的概率比B+树要低，空间使用率更高；

小结：

1.二叉搜索树：二叉树，每个结点只存储一个关键字，等于则命中，小于走左结点，大于走右结点；

2.B（B-）树：多路搜索树，每个结点存储M/2到M个关键字，非叶子结点存储指向关键字范围的子结点；所有关键字在整颗树中出现，且只出现一次，非叶子结点可以命中；

3.B+树：在B-树基础上，为叶子结点增加链表指针，所有关键字都在叶子结点中出现，非叶子结点作为叶子结点的索引，B总是到叶子结点才命中；

4.B*树：在B+树基础上，为非叶子结点也增加链表指针，将结点的最低利用率从1/2提高到2/3；

为什么说B+树比B树更适合数据库索引？

1、 B+树的磁盘读写代价更低：B+树的内部节点并没有指向关键字具体信息的指针，因此其内部节点相对B树更小，如果把所有同一内部节点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多，一次性读入内存的需要查找的关键字也就越多，相对IO读写次数就降低了。

2、B+树的查询效率更加稳定：由于非终结点并不是最终指向文件内容的结点，而只是叶子结点中关键字的索引。所以任何关键字的查找必须走一条从根结点到叶子结点的路。所有关键字查询的路径长度相同，导致每一个数据的查询效率相当。

3、由于B+树的数据都存储在叶子结点中，分支结点均为索引，方便扫库，只需要扫一遍叶子结点即可，但是B树因为其分支结点同样存储着数据，我们要找到具体的数据，需要进行一次中序遍历按序来扫，所以B+树更加适合在区间查询的情况，所以通常B+树用于数据库索引。

红黑树：

HashMap原理：

1.1 rehash哈希冲突

哈希冲突指的是 在多个线程同时rehash同一个hashmap的过程中，可能会出现两个线程同时操作同一条链表产生链表节点之间相互应用的情况，此时在查询该链表时就会发生死循环。

解决方案：我们可以通过ConcurrentHashMap来解决。在多线程操作中，ConcurrentHashMap集合会对每一条链表都进行单独加锁，这样就只会有一个线程操作一条链表，避免了rehash哈希冲突，保证了线程安全。

1.2 HashMap怎么存储null键

当我们用put方法向HashMap中存入null键时，永远会放在第一个节点【主要是由于底层addEntry(hash，key，value，i)方法，当key为null，hash参数设置为0，默认添加到第一个节点】；

如果集合中已经存在null键，那么就会覆盖到已经存在的null键中的value值，返回值为被替代的value；如果不存在null键，则直接插入到数组第一个节点位置，返回null；

1.3 Hash算法

我们使用HashMap存储数据或者获取数据是调用put（）方法和get方法（）都需要获取key键的hash值，而hash值的计算主要分为两步：

1、第一步：根据key键值 调用hashcode方法 获hashcode值

hashcode值得计算 是通过对象的内部物理地址转换成一个整数，然后该整数通过hash函数的算法（比如hash表中有八个位置，地址值转换为17，17%8=1 ，那么该对象的hashcode值为1）得到一个hashcode值。

2、第二步 使用底层的hash方法 计算hash值【put和get方法】，是通过hashcode值 与自身无符号右移16位后做异 或运算得到的结果，（做异或运算的目的是 能更好的保留各部分的特征，减少哈希碰撞）

扩展：获取hash值后 对槽位数-1进行取余 得到存储槽的位置；

1.4 树化的条件

当长度大于8时，会由链表（函数n/2）转为红黑树（log(n)），红黑树的查询效率更高；当小于6时，就不需要转化为红黑树；

21、IO

JAVA NIO

NIO主要有三大核心部分： Channel(通道)， Buffer(缓冲区)， Selector(选择器)。传统IO基于字节流和字符流进行操作，而NIO基于Channel和Buffer进行操作，数据总是从通道读取到缓冲区中，或者从缓冲区写入到通道中。 Selector(选择区)用于监听多个通道的事件（比如：连接打开，数据到达）。因此，单个线程可以监听多个数据通道。 NIO和传统IO之间第一个最大的区别是，IO是面向流的，NIO是面向缓冲区的。

NIO 的缓冲区

Java IO 面向流意味着每次从流中读一个或多个字节，直至读取所有字节，它们没有被缓存在任何地方。此外，它不能前后移动流中的数据。如果需要前后移动从流中读取的数据，需要先将它缓存到一个缓冲区。

NIO 的缓冲导向方法不同。数据读取到一个它稍后处理的缓冲区，需要时可在缓冲区中前后移动。这就增加了处理过程中的灵活性。但是，还需要检查是否该缓冲区中包含所有您需要处理的数据。而且，需确保当更多的数据读入缓冲区时，不要覆盖缓冲区里尚未处理的数据。Buffer，故名思意 缓冲区，实际上是一个容器，是一个连续数组。 Channel 提供从文件、网络读取数据的渠道，但是读取或写入的数据都必须经由Buffer。

Channel

首先说一下Channel，国内大多翻译成“通道”。 Channel和IO中的Stream(流)是差不多一个等级的。 只不过 Stream是单向的，譬如：InputStream， OutputStream，而Channel是双向的，既可以用来进行读操作，又可以用来进行写操作。NIO中 Channel的主要实现有：

1. FileChannel
2. DatagramChannel
3. SocketChannel
4. ServerSocketChannel

这里看名字就可以猜出个所以然来：分别可以对应文件IO、UDP和TCP（Server 和 Client）

Selector

Selector类是NIO的核心类， Selector能够检测多个注册的通道上是否有事件发生，如果有事件发生，便获取事件然后针对每个事件进行相应的响应处理。这样一来，只是用一个单线程就可以管理多个通道，也就是管理多个连接。这样使得只有在连接真正有读写事件发生时，才会调用函数来进行读写，就大大地减少了系统开销，并且不必为每个连接都创建一个线程，不用去维护多个线程，并且避免了多线程之间的上下文切换导致的开销。

22、JS、JQuery

js部分的内容

1.1 BOM:浏览器对象模型，说白了就是使用js代码操作浏览器

1. 三种弹框:
   1.警告框，alert()
   2.确认框，confirm(),返回值类型是boolean，点击确定就返回true，点击取消就返回false
   3.输入框，prompt()，返回值就是用户输入的内容，如果点击取消则返回null
2. 两种定时器:
   1. setTimeout(要执行的任务,定时时间毫秒)只执行一次的定时器
   2. setInterval(要执行的任务,定时时间毫秒),循环执行的定时器，方法的返回值就是这个定时器的id
   3. clearInterval(定时器的id)，清除某个定时器
3. location对象的href属性，该属性可以获取地址栏上的地址，或者是设置地址栏上的地址并且访问

1.2 DOM:文档对象模型，说白了就是使用js代码操作页面上的标签，以及标签的属性、文本、内容、样式

1. 查找标签(获取到标签):
   1. 根据id查找，document.getElementById(id),这里找到的是一个标签
   2. 根据类名查找，document.getElementsByClassName(类名)，这里找到的是多个标签(标签的数组)
   3. 根据name属性的值进行查找，document.getElementsByName(name)，这里找到的也是多个标签(标签的数组)
   4. 根据标签名进行查找，document.getElementsByTagName(标签名)，这里找到的也是多个标签(标签的数组)

2. 操作标签的属性:
   1. 标签.属性名，操作标签内置的属性(这种使用方式一般用在常用的属性那)
   2. setAttribute("属性名","属性值"),设置标签的属性
   3. getAttribute("属性名")，获取标签的属性
   4. removeAttribute("属性名")，删除标签的属性

3. 操作标签里面的文本: innerText 可以获取和设置标签体内的文本
   hello world

4. 操作标签体内的html内容: innerHTML 可以获取和设置标签体内的html内容
   hello world

5. 操作标签的css样式:
   1. 标签.style.样式名 可以获取和设置css样式，此时实际上操作的是标签的行内样式
   2. 标签.className ,这是给标签设置类名，设置了类名就能够使用对应的类选择器，其实这种方式操作的是内部样式

6. 创建标签，document.createElement(标签名)

7. 添加子标签，父标签.appendChild(子标签)

8. 删除标签， 标签.remove()

JQuery:

1. 什么jQuery，它的作用是什么
   1.1 jQuery是js的一个轻量级框架
   1.2 它的作用是简化js的DOM(操作页面的标签)、animate(动画)、event(事件)的代码
   但是jQuery没有对应的简化BOM的代码

2. 怎么在项目中使用jQuery，怎么将jQuery引入到对应的html中?
   2.1 在项目中准备一个文件夹(js)，将jquery.js文件拷贝进来

2.2 在要使用jQuery的html页面中，使用script标签的src属性引入jQuery文件

3. 对比jQuery对象和js对象
3.1 什么是jQuery对象： 使用jQuery的选择器获取的对象$("#d1")
3.2 什么是js对象： 在JavaScript中，如果它不是jQuery对象那它就是js对象
3.3 js对象只能操作js的属性和方法，jQuery对象只能操作jQuery的方法
比如一个input标签，如果你是使用document.getElementById("ipt")获取的对象，那么你可以使用.value获取value属性，如果你是使用$("#ipt")获取的对象，那么你可以使用.val()获取value属性

4. jQuery对象和js对象之间的相互转换
4.1 将js对象转换成jQuery对象： $(js对象)
4.2 将jQuery对象转换成js对象：
jQuery对象的本质:其实是一个标签的数组，该数组中的每一个元素都是js对象，
所以要将jQuery对象转换成js对象，只需要遍历这个jQuery对象中的每一个元素就可以了

5. this关键字，表示当前触发事件的这个标签，它一定是一个js对象

6.jQuery的选择器
6.1 jquery的选择器有什么作用： 帮助我们获取到想要的标签
6.2 常用的jQuery的选择器：
1. 基本选择器(掌握):
1. id选择器： 根据id获取标签，$("#id")
2. 类选择器： 根据class属性获取标签， $(".类名")
3. 标签选择器/元素选择器： 根据标签名获取标签， $("标签名")

2. 其它选择器(了解):
1. 层级选择器:
$("A B")
$("A>B")
$("A+B")
$("A~B")

2. 属性选择器:
$("[属性名]")
$("[属性名=属性值]")
$("[属性名!=属性值]")
$("[属性名^=属性值]") 属性值以什么开头的标签
$("[属性名$=属性值]") 属性值以什么结尾的标签
$("[属性名*=属性值]") 属性值包含某个字符串的标签

3. 基本过滤选择器:
$(":odd")
$(":even")
$(":gt(下标)") great than 大于
$(":lt(下标)") less than 小于
$(":eq(小标)")
$(":first")
$(":last")

4. 表单属性选择器
$(":text")
$(":password")
$(":checkbox")
$(":button") ....

5. 表单属性过滤选择器
$(":checked") 找到所有选中的单选、多选框

```
            $(":enabled")   找到所有可用的表单项
            $(":disabled")   找到所有不可用的表单项
            $(":selected")   找到所有选中的下拉框

7. 使用jQuery操作DOM
   1. 查找标签: 使用jQuery的选择器进行查找
   2. 操作标签的文本和值:
      2.1 操作表单项的value属性:    val(),方法中不传参数代表获取，方法中传参数代表设置
      2.2 操作标签体的文本: text(),方法中不传参数代表获取,方法中传参数代表设置
      2.3 操作标签体的html内容: html(),方法中不传参数代表获取,方法中传参数代表设置

   3. 操作标签的属性:
      3.1 attr()方法，如果只传入属性名代表获取属性，如果同时传入属性名和属性值代表设置属性
      3.2 prop()方法，如果只传入属性名代表获取属性，如果同时传入属性名和属性值代表设置属性

      怎么去选择到底使用attr()方法操作属性还是prop()方法操作属性呢?
      如果属性是boolean类型的，比如checked、disabled、selected 我们就是使用prop()方法操作
      其它属性都是用attr()方法操作

   4. 操作标签的样式:
      4.1 操作行内样式: css()方法，如果只传入样式名代表获取样式，如果传入样式名和样式值代表设置样式
      4.2 操作内部样式，其实就是操作类名: addClass()添加类名，removeClass()移除类名

   5. 创建、添加、删除标签:
      1. 创建标签: $("标签名")
      2. 添加标签:
         1. 父标签.append(子标签)   往父标签的最后添加子标签
         2. 父标签.prepend(子标签)   往父标签的最前面添加子标签
         3. A标签.before(标签)   在A的前面添加标签(兄弟关系)
         4. A标签.after(标签)   在A的后面添加标签(兄弟关系)
         5. 标签.remove()  删除标签
         6. 标签.empty()  清除标签内的所有子标签

8. 使用jQuery操作event(事件):
   1. jQuery对象.click(匿名函数)、jQuery对象.mouseover(匿名函数)
   2. jQuery的事件切换:
      1. 可以采用链式写法: jQuery对象.click(匿名函数).mouseover(匿名函数)
      2. hover(匿名函数1,匿名函数2)  同时绑定鼠标移入和移出事件
```

23、MySQL

索引的几种类型分别是：普通索引、唯一索引、主键索引、组合索引、全文索引：

1.普通索引：仅加速查询

2.唯一索引：加速查询 + 列值唯一（可以有null）

3.主键索引：加速查询 + 列值唯一（不可以有null）+ 表中只有一个

4.组合索引：多列值组成一个索引，专门用于组合搜索，其效率大于索引合并

5.全文索引：对文本的内容进行分词，进行搜索

全文索引是将存储在数据库中的整本书或整篇文章中的任意内容信息查找出来的技术。它可以根据需要获取全文中有关章，节，段，句，词等信息，也可以进行各种统计和分析。之前的MySQL数据库中，INNODB存储引擎并不支持全文索引技术，大多数的用户转向MyISAM存储引擎，不过这可能进行表的拆分，并需要将进行全文索引的数据存储为MyIsam表。这样的确能够解决逻辑业务的需求，但是却丧失了INNODB存储引擎的事务性，而这在生产环境应用中同样是非常关键的。

从INNODB1.2.x版本开始，INNODB存储引擎开始支持全文索引，其支持myisam的全部功能，并且还支持其他的一些特性。

倒排索引：

全文索引通常使用倒排索引来实现。倒排索引同B+树索引一样，也是一种索引结构。它在辅助表中存储了单词与单词自身在一个或多个文档中所在的位置之间的映射。这通常利用关联数组实现，其拥有两种表现形式。

inverted file index，其表现形式为{单词， 单词所在文档的ID}

full inverted index，其表现形式为{单词，（单词所在文档的ID， 再具体文档中的位置）}

【具体来说就是inverted file index只能根据单词找到对应的文档，而full inverted index不仅能找到对应的文档，还能找到单词在文档中的具体位置】

主键索引和非主键索引查询的区别：

主键：唯一、非空、自增

24、编码

(1) 类名首字母大写，字段、方法以及对象的首字母应该小写；
(2) 为了常规用途而创建一个类时，请采取"经典形式"，并包含对下述元素的定义：
equals() hashCode() toString() clone()（implement Cloneable） implement Serializable
(3) 使类尽可能短小精悍，而且只解决一个特定的问题。
(4) 让一切东西都尽可能地"私有"--private
(5) 避免使用"魔术数字"我们应创建一个常数，并为其使用具有说服力的描述性名称，并在整个程序中都采用常数标识符
(6)多加注释

25、JDK1.8新特性

1、lambda表达式

Lambda是一个匿名函数,我们可以把lambda表达式理解为是一段可以传递的代码(将代码像数据一样进行传递),写出更简洁,更灵活的代码。

语法： lambda的参数列表 -> lambda表达式需要执行的功能 即lambda体

（）-> sout("hello world")

2、函数式接口

Consumer<T> 消费性接口

Supplier<T> 攻击型接口

Function(T,R) 函数向接口

Predicate<T> 断言型接口

3、方法引用和构造器引用

4、Stream API

什么是流？

流是数据渠道，用于操作数据源（集合，数组等）所生成的元素序列，集合讲的数据，流讲的是计算

注意：Stream 自己不会存储元素

Stream 不会改变元对象,相反,它们会返回一个新的Stream

Stream 操作时延迟执行的,这意味着他们会等到需要结果时才执行(单例模式:懒汉)

如何使用

创建stream （从一个集合,数组中获取一个Stream）

中间操作(对数据元中的数据进行操作处理)

终止操作 （执行中间操作链,返回结果）

5、接口中默认方法与静态方法

6、新时间日期API

26、JVM

1、JVM内存空间

PC寄存器： 创建程序计数器，就是一个指针，指向方法区中的方法字节码

虚拟机栈：JVM Stack ：八种基本数据类型、对象引用（引用指针）【只存放局部变量】

本地方法栈：Native Method Stack

Java调用非java代码的接口，此方法用非java语言实现
方法区 Method Area
用于存储虚拟机加载的类信息 class信息
堆 Java Heap
储存对象的实例
方法区和堆都是线程共享的，在JVM启动时创建，在JVM停止时销毁，而Java虚拟机栈、本地方法栈、程序计数器是线程私有的，随线程的创建而创建，随线程的结束而死亡。

GC机制：
五个内存区域中，有3个是不需要进行垃圾回收的：本地方法栈、程序计数器、虚拟机栈。因为他们的生命周期是和线程同步的，随着线程的销毁，他们占用的内存会自动释放。所以，只有方法区和堆区需要进行垃圾回收，回收的对象就是那些不存在任何引用的对象。

判断是否是垃圾数据：
根搜索算法：从一个叫GC Roots的根节点出发，向下搜索，如果一个对象不能达到GC Roots的时候，说明该对象不再被引用，可以被回收。
可作为GC Roots的对象包含以下几种：
1.虚拟机栈(栈帧中的本地变量表)中引用的对象。
2.方法区中静态属性引用的对象
3.方法区中常量引用的对象
4.本地方法栈中(Native方法)引用的对象
Minor GC机制 和 Full GC机制
我们说的堆内存里面主要分三块： 分别是新生代 和老生代和持久代；
新生代里面大致分为Eden区和Survivor区，Survivor区又分为大小相同的两部分：FromSpace和ToSpace。新建的对象都是从新生代分配内存，Eden区不足的时候，会把存活的对象转移到Survivor区。当新生代里面Eden区满时，就会触发Minor GC（也称作Youn GC）。
老生代或者持久代里面垃圾回收机制称为 Full GC机制（Major GC），速度比Minor GC机制慢10倍以上。当老生代满时会触发Full GC，会同时回收新生代、老生代；当持久代【方法区】满时也会触发Full GC，会导致Class、Method元信息的卸载
GC算法：
新生代主要使用复制算法： 将不需要清除的对象复制到一块区域，清除其他的无用对象；缺点是需要额外的空间和移动
老生代主要使用标记压缩算法： 将不需要清除的对象进行标记，清除未标记的对象，然后把活的对象向空闲空间移动，再更新引用对象的指针。效率高，也解决了内存碎片的问题，缺点是需要移动对象。
2、线上CPU过高的问题解决、
1.查询占用cpu最高的线程id
```
ps -eLo pid,lwp,pcpu | grep 15285|sort -nk 3
```
2.导出JAVA线程栈信息
命令：kill -3 [PID] 或者 jstack [PID]
3.从栈信息中找到线程数多的几个
命令：sort 文件名 | uniq -c | sort -nk 1
4、分析线程数最多的前几个线程
如果是功能点占用cpu高，那么可以从业务和技术两个方面优化：
4/1 弹性时间：对高使用率的请求，分散到不同时间 比如队列，异步，减少统一时间的请求
4/2 批处理或定时任务
如果是GC进程占用cpu更多，则使用JVM自带的性能监控和故障分析工具VisualVM，查看垃圾回收活动是否过于频繁，如果是GC次数过多，就查询堆存储中对象大小最大的对象，根据占用内存最大的对象去查询代码中接口，分析代码问题 ，查看sql语句是否可以优化，是否有死循环

二、天虹在线超市

1、项目有哪些优化点?

1、广告缓存这块，之前只采用redis进行广告的缓存，并发量大的时候，并没有减轻Tomcat压力，后期考虑采用OpenResty+Redis进行缓存，使用lua脚本使nginx直接处理请求没减轻了tomcat服务器的压力。
2、之前基于Redis实现单点登录，把session数据存放在redis中统一管理。后期决定采用Spring Security OAuth2.0+JWT实现单点登录，Spring security Oauth2让使用Oauth2协议变的规范，而且容易和Spring Security整合，另外jwt前面使用非对称加密，进行jwt的校验的时候，使用公钥校验，更为安全和方便。

2、什么是单点登录，为什么采用Spring Security+OAuth2.0+JWT?

1、单点登录SSO也就是Single Sign-On，用户只需要登录一次就可以访问所有相互信任的应用系统，避免重复登录。
2、Spring security Oauth2让使用Oauth2协议变的规范，而且容易和Spring Security整合，另外jwt使用非对称加密，进行jwt的校验的时候，使用公钥校验，更为安全和方便。

OAuth 2.0定义了四种授权方式：
    授权码模式（authorization code）
    简化模式（implicit）
    密码模式（resource owner password credentials）
    客户端模式（client credentials）

采用OAuth2.0的原因？
    OAuth2.0协议中规定了单点登录的一些流程，包括令牌的颁发、校验和刷新等，不用OAuth2.0我们也要自己实现这些功能，使用OAuth2.0开发比较方便。

3、介绍一下广告模块

OpenResty：
    基于Ngnix和Lua的高性能web平台，内部集成精良的Lua库、第三方模块、依赖项。用于方便搭建能够处理高并发、扩展性极高的动态web应用、web服务、动态网关；
    使用Lua脚本调用Ngnix支持的C以及Lua模块，快速构建10K～1000K单机并发连接的高性能web应用系统；
    OpenResty的目标是让web服务直接运行在Nginx服务内部，利用Ngnix的非阻塞IO模型，对HTTP客户端请求和后端DB进行一致的高性能响应；
    OpenResty实际上是Nginx+LuaJIT的完美组合；
【广告缓存的载入与读取】使用到了OpenRestry
    先查询openresty本地缓存如果没有，其中openresty本地缓存使用LRU算法。
    再查询redis中的数据，如果没有
    再查询mysql中的数据，但凡有数据 则返回即可
    也就是，nginx很强了，OpenRestry让nginx更强。
参考面试宝典

4、介绍下lua

    项目第四天：Lua是一种轻量小巧的脚本语言，用标准C语言编写并以源代码形式开放，其设计目的是为了嵌入应用程序中，从而为应用程序提供灵活的扩展和定制功能。
特性：
    1.支持面向过程(procedure-oriented)编程和函数式编程(functional programming)；
    2.自动内存管理：只提供了一种通用类型的表（table），用它可以实现数组，哈希表，集合，对象；
    3.语言内置模式匹配；闭包(closure)；函数也可以看做一个值；提供多线程（协同进程，并非操作系统所支持的线程）支持；
    4.通过闭包和table可以很方便地支持面向对象编程所需要的一些关键机制，比如数据抽象，虚函数，继承和重载等。
应用场景：
    游戏开发
    独立应用脚本

Web 应用脚本
扩展和数据库插件如：MySQL Proxy 和 MySQL WorkBench
安全系统，如入侵检测系统
redis中嵌套调用实现类似事务的功能
web容器中应用处理一些过滤 缓存等等的逻辑，例如nginx。
lua有交互式编程和脚本式编程：
Lua 交互式编程模式可以通过命令 lua -i 或 lua 来启用，将 Lua 程序代码保持到一个以 lua 结尾的文件，并执行，该模式称为脚本式编程。
两个减号是单行注释 --[[
多行注释
多行注释
--]]
默认的情况下，定义一个变量都是全局变量，如果要用局部变量 需要声明为local。
Lua中的数据类型：nil、boolean、number、string、userdata、function、thread 和 table

模块：
模块类似于一个封装库，从Lua 5.1 开始，Lua加入了标准的模块管理机制，可以把一些公用的代码放在一个文件里，以API接口的形式在其他地方调用，有利于代码的重用和降低代码耦合度。
require函数，用于引入其他的模块，类似于java中的类要引用别的类的效果。

5、Nginx的缓存

广告是怎么存入的？什么形式？
json字符串（状态是可用的，图片的连接url和fastDFS上面的绝对路径），cjson实现，key是：category_id

6、Canal

Canal：的常见使用场景之一就是数据同步，比如把mysql的数据同步到elasticsearch
cannal的原理：
1.canal伪装自己为mysql slave，向mysql master发送dump协议
2.mysql master收到dump请求，开始推送binary log给canal
3.canal解析binary log对象
4.生产环境下，cannal也需要高可用
除了Canal还能用什么技术实现监听数据库变化？
数据同步除了用cannal还可以用logstash,logstash是周期性查询,cannal是基于binlog日志，理论是不需要额外增加mysql负担的

7、微服务架构相对于soa架构有什么优势?

1.SOA（Service Oriented Architecture）“面向服务的架构”:他是一种设计方法，其中包含多个服务，服务之间通过相互依赖最终提供一系列的功能。一个服务通常以独立的形式存在于操作系统进程中。各个服务之间通过网络调用。

2.微服务架构:其实和SOA架构类似，微服务是在SOA上做的升华，微服务架构强调的一个重点是“业务需要彻底的组件化和服务化”，原有的单个业务系统会拆分为多个可以独立开发、设计、运行的小应用。这些小应用之间通过服务完成交互和集成。

微服务架构 = 80%的SOA服务架构思想 + 100%的组件化架构思想 + 80%的领域建模思想

SOA架构特点：

系统集成：站在系统的角度，解决企业系统间的通信问题，把原先散乱、无规划的系统间的网状结构，梳理成规整、可治理的系统间星形结构，这一步往往需要引入 一些产品，比如ESB、以及技术规范、服务管理规范；这一步解决的核心问题是【有序】

系统的服务化：站在功能的角度，把业务逻辑抽象成可复用、可组装的服务，通过服务的编排实现业务的快速再生，目的：把原先固有的业务功能转变为通用的业务服务，实现业务逻辑的快速复用；这一步解决的核心问题是【复用】

业务的服务化：站在企业的角度，把企业职能抽象成可复用、可组装的服务；把原先职能化的企业架构转变为服务化的企业架构，进一步提升企业的对外服务能力；“前面两步都是从技术层面来解决系统调用、系统功能复用的问题”。第三步，则是以业务驱动把一个业务单元封装成一项服务。这一步解决的核心问题是【高效】

微服务架构特点：

1.通过服务实现组件化:开发者不再需要协调其它服务部署对本服务的影响。

2.按业务能力来划分服务和开发团队:开发者可以自由选择开发技术，提供 API 服务

3.去中心化:每个微服务有自己私有的数据库持久化业务数据每个微服务只能访问自己的数据库，而不能访问其它服务的数据库某些业务场景下，需要在一个事务中更新多个数据库。这种情况也不能直接访问其它微服务的数据库，而是通过对于微服务进行操作。数据的去中心化，进一步降低了微服务之间的耦合度，不同服务可以采用不同的数据库技术（SQL、NoSQL等）。在复杂的业务场景下，如果包含多个微服务，通常在客户端或者中间层（网关）处理。

6、为什么将soa架构更换为微服务架构?

1.soa架构的服务之间还是存在相互依赖，服务的划分粒度还不够细，耦合度较高，不利于系统扩展;
2.soa架构系统之间的通信和治理（网关，消息等）还需要依赖第三方组件，配置文件管理较困难;
3.微服务架构的服务划分粒度更细，原有的单个业务系统会拆分为多个可以独立开发、设计、运行的小应用。各服务之间的耦合度更低，有利于系统扩展，提高开发效率;
4.微服务采用springCloud框架开发，内部集成了各种组件，使用方便;
5.微服务可以是跨平台的，可以用任何一种语言开发

7、单点登录

```
1.用到了哪些技术?
    1.sso-->效果-->方案
    2.spring security 框架（认证、授权）
    3.oauth2.0 协议-->规范
        authorization-->授权-->accessToken
            1.颁发accessToken-->提到了可以有4种模式
                （密码-->oauth/token 授权码模式-->oauth/authorize）
            2.校验accessToken-->oauth/check_token
            3.刷新accessToken-->oauth/token
                    refresh_token-->xxxxxx
                    grant_type-->refresh_token
    4.jwt-->规范
        jwt由base64(头部).base64(载荷).sign(base64(头部).base64(载荷))
        签证算法有两种，分别是HS256和RS256。一个是对称加密，一个是非对称加密
    5.加密方式：对称加密->1个密码 aes des HS256  非对称加密->1个秘钥对 rsa HS256
        非对称加密可以做3个事情
```

1.公钥加密-->私钥解密
2.私钥加密-->公钥解密
3.私钥签名-->中间人修改-->公钥校验
6.spring security oauth
方便基于oauth2.0协议完成授权和认证
开发授权服务器
开发资源服务器
7.第三方登录：qq,微博,微信,github-->登录我们的系统

2.提问的方式梳理?
2.1解释什么是sso?
password.jd.com card.jd.com user.jd.com
2.2怎么理解spring security?
WebSecurityConfigurerAdapter
@EnableWebSecurity
AuthenticationManager
UserDetailsService
2.3怎么理解spring security oauth?
@EnableResourceServer
@EnableAuthorizationServer
AuthorizationServerConfigurerAdapter
TokenStore
AccessTokenConverter
2.4谈一下你对非对称加密了解?
1.公钥加密-->私钥解密
2.私钥加密-->公钥解密
3.私钥签名-->中间人修改-->公钥校验===>主要是这里
2.5如果网页/h5/app要实现三方登录,后台开发需要做什么事情?
参考开放平台

https://wiki.open.qq.com/wiki/%E3%80%90QQ%E7%99%BB%E5%BD%95%E3%80%91%E7%BD%91%E7%AB%99%E6%8E%A5%E5%85%A5

https://developers.weixin.qq.com/doc/oplatform/Website_App/WeChat_Login/Wechat_Login.html
1.用户在我们的网页-->点击qq登录按钮(前端)-->要么前端跳/后台跳(redirect_url)-->跳转qq登录页面-->登录完成-->拿到accessToken(qq颁发的)
2.回调自己服务器
1.走到我们的一个controller方法里面
2.有了accessToken
1.头像,昵称,邮箱-->对用户可见
2.三方平台的唯一标识-->openUserId-->静默注册了一个新用户-->找到与之对应的自己平台的userId
2.6我们自己的系统要基于oauth2.0协议去完成授权(颁发token)用什么授权模式?
密码模式 简单，自己的系统,自己的user,自己user对应的password
2.7spring security oauth可以对accessToken持久化,持久化方式有哪些,为什么我们用JwtTokenStore?
redis-->持久化到服务器了
jdbc-->持久化到服务器了
jwt-->持久化到我们的浏览器客户端
2.8spring security oauth为什么对token的签名使用非对称加密方式?
1.校验方便
怎么校验?
框架校验spring security oauth
怎么方便?
1.放入public.key
2.给spring security oauth传入public.key
@Bean

```
                    public JwtAccessTokenConverter jwtAccessTokenConverter() {
                        JwtAccessTokenConverter converter = new
                        JwtAccessTokenConverter();
                        converter.setVerifierKey(getPubKey());
                        return converter;
                    }
```

2.安全性：非对称加密,安全性高于对称加密

2.9为什么生成token,我们要基础oauth2.0协议?

方便，关于token相关的操作规范都提供了如何颁发、如何校验、如何刷新

2.10为什么我们做sso,我们要基础oauth2.0协议?

因为oauth2.0有那么一套关于accessToken的颁发,校验,刷新

2.11如果要完成qq登录,是否需要了解oauth2.0协议?

不怎么了解都行

2.12什么时候让用户登录?

网关控制请求需要token的一些url的时候有可能有跳：（订单、支付、加入购物车、地址列表）

2.13什么情况会让你跳?

cookie/header/params没有

2.14什么时候颁发token?

登录成功之后-->基于oauth2.0协议的密码模式请求成功后,spring security oauth就会把accessToken给我们.

2.15什么时候校验token,如何校验,谁去校验?

1.spring-security-oauth框架校验

2.受保护的资源服务器/资源服务器校验/加了@EnableResourceServer注解微服务校验/使用了public.key的微服务校验

2.16spring-security-oauth框架从哪里去取出token?

header

key:Authorization

value:[bearer token]

2.17用户客户端-->网关-->微服务-->我们需要怎么处理token?

假设用户客户端携带了正确的token-->网关过滤器(AuthorizeFilter)-->加到请求头

2.18用户客户端-->网关-->微服务-->feign-->微服务-->我们需要怎么处理token?

假设用户客户端携带了正确的token-->拦截器(MyFeignInterceptor implements RequestInterceptor)-->加到请求头

3.怎么考虑技术选型?

sessionId+redis-->下去自己看一下

https://blog.csdn.net/qq_22172133/article/details/82291112

cas

jwt+拦截器

jwt+(spring security)+(spring security oauth)+(spring cloud gateway)

4.涉及几张表?

1.t_user

2.oauth_client_details

5.涉及几个接口?

1.登录

2.颁发-->登录,刷新,校验

6.开发流程、开发步骤是怎样的?

自己想下

7.哪些地方有难度或者印象深刻?

概念多,容易混

8.开发过程中有没有遇到什么问题?

概念多,容易混

9.开发了多久?

```
    2个星期
        1.网关-->修改
        2.授权中心-->独立
        3.资源服务器还需要修改
            a1->修改
            a2->修改
            a3->修改
            a4->修改
```

8、搜索模块

```
1.用到了哪些技术?
    es

2.怎么考虑技术选型?
    es

3.涉及几张表?
    一张(tb_sku)

4.涉及几个接口?
    2个:导入，查询

5.开发流程、开发步骤是怎样的?
    参考面试宝典

6.哪些地方有难度或者印象深刻?
    1.es建立索引的时候,映射建立需要思考
        1.哪些分词
        2.哪些不分词
        3.规格怎么存到es

7.开发过程中有没有遇到什么问题?
    skuMapper.saveAll(skuInfos);
    skuInfos东西多
        会不会内存溢出?
        会不会太慢?  Elasticsearch的_bulk来批量插入  Bulk的api+多线程
        spring-data-es.jar如何导入大量数据
        查询不够快?
    官方:query和filter,query比较慢,filter比较快-->不是很理解
    我们能感受到的
        1.query-->filter+filter+filter->满
        2.query-->query+query+query-->快
            最基本结论--->dsl结构可能引入效率不一样

8.开发了多久?
    一周
```

三、SCC-SUBMES

1、权限模块

RBAC 基于角色的权限访问控制，以角色为中心，权限与角色相关联，用户通过成为适当的角色而拥相应的权限。

1.用到了哪些技术?
    spring security
    shiro

2.怎么考虑技术选型?
    spring security 对比 shiro
     优点:
     1: Spring Security基于Spring开发，项目中如果使用Spring作为基础，配合Spring Security做权限更加方便，而Shiro需要和Spring进行整合开发
     2: Spring Security功能比Shiro更加丰富些，例如安全防护
     3: Spring Security社区资源比Shiro丰富
     缺点:
      1: Shiro的配置和使用比较简单，Spring Security上手复杂
      2: Shiro依赖性低，不需要任何框架和容器，可以独立运行，而Spring Security依赖于Spring容器

3.涉及几张表?
    5张表: user role permission menu
    7张表-->最主流
    user user_role role permission role_permission menu role_menu

4.涉及几个接口?
    1. 登录接口->在登录的时候会查询
        用户-->role->权限
    2. 用户所拥有的菜单
        用户-->role->菜单

5.需求是怎样的?
        1.没有登录,不能访问任何资源(url)
        2.不同用户,看到的菜单不同-->不同用户是不同角色
        3.不同用户,可以操作的controller/controller里面方法/controller里面方法的具体代码块
        4.需要自定义登录界面-->spring security默认的登录界面太丑了

6.开发流程、开发步骤是怎样的?
    1.登录接口完成接口
    2.用户所拥有的菜单接口
    3.没有登录,不能访问任何资源-->spring security进行配置
    4.自定义登录界面-->
        1.完成界面(div+css)
        2.spring security配置
            1.loginPage
            2.loginProcessingUrl
    5.不用用户,可以操作的controller/controller里面方法/controller里面方法的具体代码块
        1.登录的时候-->赋值权限-->需要自定义userDetail-->loadUserByUsername
        2.应用权限
            1.开启注解支持-->
            @EnableGlobalMethodSecurity
            2.使用相关注解-->
            @PreAuthorize("hasAnyAuthority('CHECKITEM_DELETE')")
    6.不用用户,看到的菜单不同-->只有管理系统才是
        进入页面的时候,去加载 vue-->mounted()-->加载

7.哪些地方有难度或者印象深刻?
    spring-security方式也多
    spring-security-->xml
    spring-security-->bean

spring-security配置项多
1.放行配置
2.拦截配置
3.loginPage/loginProcessingUrl
4.自定义认证器-->找我们自己的userDetail
5.告诉db里面的密码使用的什么加密方式
6.csrf
7.httpBasic

8.开发过程中有没有遇到什么问题?
管理系统使用了iframe,只能<security:frame-options policy="SAMEORIGIN"/>
csrf
httpBasic

9.开发了多久?
两周

Spring Security
是Spring提供的安全认证服务的框架。使用Spring Security可以帮助我们来简化认证和授权的过程。常用的权限框架除了Spring Security，还有Apache的shiro框架。
认证：系统提供的用于识别用户身份的功能，认证的目的是让系统知道你是谁。只需要用户表就可以了，在用户登录时可以查询用户表t_user进行校验，判断用户输入的用户名和密码是否正确。
授权：用户认证成功后，需要为用户授权，其实就是指定当前用户可以操作哪些功能。用户必须完成认证之后才可以进行授权，首先可以根据用户查询其角色，再根据角色查询对应的菜单，这样就确定了用户能够看到哪些菜单。然后再根据用户的角色查询对应的权限，这样就确定了用户拥有哪些权限。所以授权过程会用到上面7张表。

基本配置：
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                  http://www.springframework.org/schema/beans/spring-beans.xsd
                  http://www.springframework.org/schema/mvc
                  http://www.springframework.org/schema/mvc/spring-mvc.xsd
                  http://code.alibabatech.com/schema/dubbo
                  http://code.alibabatech.com/schema/dubbo/dubbo.xsd
                  http://www.springframework.org/schema/context
                  http://www.springframework.org/schema/context/spring-
context.xsd
                          http://www.springframework.org/schema/security
                          http://www.springframework.org/schema/security/spring-
security.xsd">
    <!--一：定义哪些链接可以放行-->
    <!--
      http：用于定义相关权限控制
      指定哪些资源不需要进行权限校验，可以使用通配符
    -->
    <security:http security="none" pattern="/js/**" />
    <security:http security="none" pattern="/css/**" />
    <security:http security="none" pattern="/img/**" />
    <security:http security="none" pattern="/plugins/**" />

    <!--开启注解方式权限控制-->
```

```
    <security:global-method-security pre-post-annotations="enabled" />

    <!--
        二：定义哪些链接不可以放行，即需要有角色、权限才可以放行
        http：用于定义相关权限控制
        auto-config：是否自动配置，设置为true时框架会提供默认的一些配置，例如提供默认的登录
页面、登出处理等，设置为false时需要显示提供登录表单配置，否则会报错
        use-expressions：用于指定intercept-url中的access属性是否使用表达式
    -->
    <security:http auto-config="true" use-expressions="true">
        <security:headers>
            <!--设置在页面可以通过iframe访问受保护的页面，默认为不允许访问-->
            <security:frame-options policy="SAMEORIGIN"></security:frame-
options>
        </security:headers>
    </security:http>

        <!--只要认证通过就可以访问-->
        <!--
            intercept-url：定义一个拦截规则
            pattern：对哪些url进行权限控制
            access：在请求对应的URL时需要什么权限，默认配置时它应该是一个以逗号分隔的角色列
表，
              请求的用户只需拥有其中的一个角色就能成功访问对应的URL
            isAuthenticated()：需要经过认证后才能访问（不是匿名用户）
        -->
        <security:intercept-url pattern="/pages/**"  access="isAuthenticated()"
/>

        <!--
          form-login：定义表单登录信息
        -->
        <security:form-login login-page="/login.html"
                             username-parameter="username"
                             password-parameter="password"
                             login-processing-url="/login.do"
                             default-target-url="/pages/main.html"
                             authentication-failure-url="/login.html"
                             always-use-default-target="true"/>

          <!--csrf：对应CsrfFilter过滤器 disabled：是否启用CsrfFilter过滤器，如果使用自
定义登录页面需要关闭此项，否则登录操作会被禁用（403）-->
        <security:csrf disabled="true"></security:csrf>

        <!--
          logout：退出登录
          logout-url：退出登录操作对应的请求路径
          logout-success-url：退出登录后的跳转页面
        -->
        <security:logout logout-url="/logout.do"
                         logout-success-url="/login.html" invalidate-
session="true"/>
    </security:http>

    <!--配置密码加密对象-->
    <bean id="passwordEncoder"

 class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
```

```
    <!--
        三：认证管理，定义登录账号名和密码，并授予访问的角色、权限
        authentication-manager：认证管理器，用于处理认证操作
    -->
    <security:authentication-manager>
        <!--
            authentication-provider：认证提供者，执行具体的认证逻辑
        -->
        <security:authentication-provider user-service-
ref="springSecurityUserService">
            <!--指定密码加密策略-->
            <security:password-encoder ref="passwordEncoder">
</security:password-encoder>
        </security:authentication-provider>
    </security:authentication-manager>
</beans>
```

在spring-security.xml文件中配置，指定哪些资源可以匿名访问：

```
<!--
  http：用于定义相关权限控制
  指定哪些资源不需要进行权限校验，可以使用通配符
-->
<security:http security="none" pattern="/js/**" />
<security:http security="none" pattern="/css/**" />
<security:http security="none" pattern="/login.html"/>
```

修改spring-security.xml文件，加入表单登录信息的配置：

```
<!--
  form-login：定义表单登录信息
            login-page="/login.html"：表示指定登录页面
            username-parameter="username"：使用登录名的名称，默认值是username
            password-parameter="password"：使用登录名的密码，默认值是password
            login-processing-url="/login.do"：表示登录的url地址
            default-target-url="/index.html"：登录成功后的url地址
            authentication-failure-url="/login.html"：认证失败后跳转的url地址，失败后
指定/login.html
            always-use-default-target="true"：登录成功后，始终跳转到default-target-
url指定的地址，即登录成功的默认地址
-->
<security:form-login login-page="/login.html"
                     username-parameter="username"
                     password-parameter="password"
                     login-processing-url="/login.do"
                     default-target-url="/index.html"
                     authentication-failure-url="/login.html"/>
```

修改spring-security.xml文件，关闭CsrfFilter过滤器：

```
<!--
  csrf：对应CsrfFilter过滤器
  disabled：是否启用CsrfFilter过滤器，如果使用自定义登录页面需要关闭此项，否则登录操作会被禁
用（403）
Spring-security采用盗链机制，其中_csrf使用token标识和随机字符，每次访问页面都会随机生成，然
后和服务器进行比较，成功可以访问，不成功不能访问（403：权限不足）。
-->
<security:csrf disabled="true"></security:csrf>
```

从数据库查询用户信息：

1：定义UserService类，实现UserDetailsService接口。
2：修改spring-security.xml配置（注入UserService）

bcrypt：将salt随机并混入最终加密后的密码，验证时也无需单独提供之前的salt，从而无需单独处理salt问题
spring security中的BCryptPasswordEncoder方法采用SHA-256 +随机盐+密钥对密码进行加密。SHA系列是Hash算法，不是加密算法，使用加密算法意味着可以解密（这个与编码/解码一样），但是采用Hash处理，其过程是不可逆的。
（1）加密(encode)：注册用户时，使用SHA-256+随机盐+密钥把用户输入的密码进行hash处理，得到密码的hash值，然后将其存入数据库中。
（2）密码匹配(matches)：用户登录时，密码匹配阶段并没有进行密码解密（因为密码经过Hash处理，是不可逆的），而是使用相同的算法把用户输入的密码进行hash处理，得到密码的hash值，然后将其与从数据库中查询到的密码hash值进行比较。如果两者相同，说明用户输入的密码正确。这正是为什么处理密码时要用hash算法，而不用加密算法。因为这样处理即使数据库泄漏，黑客也很难破解密码。

配置多种校验规则（对访问的页面做权限控制）

```
<security:intercept-url pattern="/index.html" access="isAuthenticated()">
</security:intercept-url>
<security:intercept-url pattern="/a.html" access="isAuthenticated()">
</security:intercept-url>
<security:intercept-url pattern="/b.html" access="hasAuthority('add')">
</security:intercept-url>
<security:intercept-url pattern="/c.html" access="hasRole('ROLE_ADMIN')">
</security:intercept-url>
<security:intercept-url pattern="/d.html" access="hasRole('ABC')">
</security:intercept-url>
```

注解方式权限控制（对访问的Controller类做权限控制）

```
<security:global-method-security pre-post-annotations="enabled">
</security:global-method-security>
```

同时使用注解，在Controller类中的方法上添加：@PreAuthorize(value = "hasRole('ROLE_ADMIN')")

退出登录

```
<security:logout logout-url="/logout.do" logout-success-url="/login.html"
invalidate-session="true"></security:logout>
```

Spring Security在认证成功后会将用户信息保存到框架提供的上下文对象中，所以此处我们就可以调用Spring Security框架提供的API获取当前用户的username并展示到页面上

```
@RequestMapping(value = "/getUsername")
public Result getUsername(){
    try {
        // 从SpringSecurity中获取认证用户的信息
        User user =
(User)SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        return new Result(true,
MessageConstant.GET_USERNAME_SUCCESS,user.getUsername());
    } catch (Exception e) {
        e.printStackTrace();
        return new Result(false, MessageConstant.GET_USERNAME_FAIL);
    }
}
```

2、工艺模块

工艺参数新增逻辑
canal：

```
        <!--起步依赖-->
```

```
        <dependency>
            <groupId>com.xpand</groupId>
            <artifactId>starter-canal</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
#canal配置
canal:
  client:
    instances:
      example:
        host: 192.168.211.132
        port: 11111
```

启动类：

```
@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
@EnableCanalClient
@EnableEurekaClient
@EnableFeignClients(basePackages =
{"com.changgou.content.feign","com.changgou.item.feign"})
public class CanalApplication {
    public static void main(String[] args) {
        SpringApplication.run(CanalApplication.class, args);
    }
}
```

监听类：

```
@CanalEventListener
public class CanalDataEventListener {

    @Autowired
    private ContentFeign contentFeign;
    @Autowired
    private PageFeign pageFeign;
    @Autowired
    private StringRedisTemplate stringRedisTemplate;

    /**
    * @Description: 自定义监听数据库操作
    * @Param: [entryType, rowData]
    * @Return: void
    * @Author: Wangqibo
    * @Date: 2020/8/13/0013
    */
    @ListenPoint(destination = "example",
                 schema = "changgou_content",
                 table = {"tb_content", "tb_content_category"},
                 eventType = {
                    CanalEntry.EventType.UPDATE,
                    CanalEntry.EventType.DELETE,
                    CanalEntry.EventType.INSERT})
    public void onEventCustomUpdate(CanalEntry.EventType
eventType,CanalEntry.RowData rowData){
        //获取categoryId的值
        String categoryId = getCategoryId(eventType, rowData);
        //调用feign获取该分类下所有广告集合
        Result<List<Content>> result =
contentFeign.findByCategoryId(Long.valueOf(categoryId));
        //更新redis
```

```
        stringRedisTemplate.boundValueOps("content_" +
categoryId).set(JSON.toJSONString(result.getData()));
    }

    /**
    * @Description: 根据不同操作获取categoryId
    * @Param: [eventType, rowData]
    * @Return: java.lang.String
    * @Author: Wangqibo
    * @Date: 2020/8/13/0013
    */
    private String getCategoryId(CanalEntry.EventType
eventType,CanalEntry.RowData rowData){
        String categoryId = "";
        if (eventType == CanalEntry.EventType.DELETE) {
            //如果是删除，获取beforeList
            List<CanalEntry.Column> beforeColumnsList =
rowData.getBeforeColumnsList();
            for (CanalEntry.Column column : beforeColumnsList) {
                if (column.getName().equalsIgnoreCase("category_id")) {
                    categoryId = column.getValue();
                    return categoryId;
                }
            }
        }else {
            //如果是添加和修改，获取afterList
            List<CanalEntry.Column> afterColumnsList =
rowData.getAfterColumnsList();
            for (CanalEntry.Column column : afterColumnsList) {
                if (column.getName().equalsIgnoreCase("category_id")) {
                    categoryId = column.getValue();
                    return categoryId;
                }
            }
        }
        return categoryId;
    }
}
```

3、质量模块

QDN单的实现流程

4、参数管理

```
1.用到了哪些技术?
    1.5个请求
        post-->json-->Bean
        post-->file-->Multipart-->fastDFS/7牛云（单张、多张）
        get-->分类-->3级联动
        单表->自关联   3个表->关联查询
        get->品牌选择-->关联查询
        get->模板选择-->关联查询
    2.存db 存tb_sku,tb_spu-->多表-->事务
        编程事务、声明事务（注解、xml）
    3.id的生成：雪花算法
    4.时间问题-->统一
```

```
        db:dateTime
        Bean:Date
        前端:var
        4.1 我们jdbc连接数据的时候：url:
jdbc:mysql://localhost:3306/changgou_oauth?
useUnicode=true&characterEncoding=utf-
8&useSSL=false&allowMultiQueries=true&serverTimezone=GMT%2B8
        4.2 返回前端：Bean(Date)-->springboot(jackjson)-->json-->前端
                统一配置
                    spring:
                        jackson:
                            date-format: java.text.SimpleDateFormat
                            time-zone: GMT+8
                单个配置
                    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", locale =
"zh",timezone="GMT+8")
                    private Date createTime;//添加日期

2.怎么考虑技术选型？
    不涉及

3.涉及几张表?
    数据来源-->5张表
    数据存储-->2张表

4.涉及几个接口?
    6个get
    2个post

5.开发流程、开发步骤是怎样的？
    1.图片上传单独完成
    2.完成数据来源相关的接口-->6个
    3.完成add接口-->6个

6.哪些地方有难度或者印象深刻？开发过程中有没有遇到什么问题？
    fastDfs图片上传过大,可能有问题
    声明事务到底用注解/xml 有去思考
    时区问题

7.开发了多久？
    一周
```

5、数据导入导出

```
```

四、状元教育综合管理系统

1、第三方登录

```
流程：
    1.完成开放平台的应用创建，获取相关信息（app_ID、app_KEY）
    2.下载官方demo运行查看
    3.前端点击qq发起请求
```

4.后台收到请求，调用sdk里面的方法，弹出qq授权页面
5.用户授权完成，qq授权服务器，回调结果
6.后台处理授权结果，拿到AccessToken
7.通过AccessToken可以换取想要的信息
8.头像，昵称
9.三方平台的唯一标识（openId）
10.在认证中心用户里按照openid进行查询，匹配到的登录，未匹配到的进行新用户注册或与存在用户绑定。
11.我们也可以不使用他们提供的sdk，可以选择使用Spring Social实现QQ社交登录

处理点击qq图标的请求：

```
 /**
     * 第三方登录 授权
     *
     * @return
     */
    @RequestMapping(value = "/to_qq")
    public void toLogin(HttpServletRequest request, HttpServletResponse
response) {
    response.setContentType("text/html;charset=utf-8");
    try {
        // 【看这里】
        response.sendRedirect(new Oauth().getAuthorizeURL(request));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

处理授权结果

```
 @RequestMapping(value = "/to_third_login/callback")
    public void loginBack(HttpServletRequest request, HttpServletResponse
response) {
    response.setContentType("text/html; charset=utf-8");
    try {
        PrintWriter out = response.getWriter();
        AccessToken accessTokenObj = (new
Oauth()).getAccessTokenByRequest(request);

        String accessToken = null,
        openID = null;
        long tokenExpireIn = 0L;
        if (accessTokenObj.getAccessToken().equals("")) {
            //               我们的网站被CSRF攻击了或者用户取消了授权
            //               做一些数据统计工作
            System.out.print("没有获取到响应参数");
        } else {
            accessToken = accessTokenObj.getAccessToken();
            tokenExpireIn = accessTokenObj.getExpireIn();

            request.getSession().setAttribute("demo_access_token", accessToken);
            request.getSession().setAttribute("demo_token_expirein",
String.valueOf(tokenExpireIn));

            // 利用获取到的accessToken 去获取当前用的openid -------- start
            //【看这里】三方平台唯一标识
            OpenID openIDObj = new OpenID(accessToken);
            openID = openIDObj.getUserOpenID();
```

```
            out.println("欢迎你，代号为 " + openID + " 的用户!");
            request.getSession().setAttribute("demo_openid", openID);
            out.println("<a href=" + "/shuoshuoDemo.html" + " target=\"_blank\">
去看看发表说说的demo吧</a>");
            // 利用获取到的accessToken 去获取当前用户的openid --------- end

            out.println("<p> start ----------------------------------利用获取到的
accessToken,openid 去获取用户在Qzone的昵称等信息 ---------------------------- start
</p>");
            UserInfo qzoneUserInfo = new UserInfo(accessToken, openID);
            UserInfoBean userInfoBean = qzoneUserInfo.getUserInfo();
            out.println("<br/>");
            if (userInfoBean.getRet() == 0) {
                out.println(userInfoBean.getNickname() + "<br/>");
                out.println(userInfoBean.getGender() + "<br/>");
                out.println("<image src=" +
userInfoBean.getAvatar().getAvatarURL30() + "/><br/>");
                out.println("<image src=" +
userInfoBean.getAvatar().getAvatarURL50() + "/><br/>");
                out.println("<image src=" +
userInfoBean.getAvatar().getAvatarURL100() + "/><br/>");
            } else {
                out.println("很抱歉，我们没能正确获取到您的信息，原因是： " +
userInfoBean.getMsg());
            }
            out.println("<p> end ----------------------------------利用获取到的
accessToken,openid 去获取用户在Qzone的昵称等信息 ---------------------------- end
</p>");
        }
    } catch (Exception e) {
    }
```

2、Quartz

Quartz是Job scheduling（作业调度）领域的一个开源项目，Quartz既可以单独使用也可以跟spring框架整合使用，在实际开发中一般会使用后者。使用Quartz可以开发一个或者多个定时任务，每个定时任务可以单独指定执行的时间，例如每隔1小时执行一次、每个月第一天上午10点执行一次、每个月最后一天下午5点执行一次等。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                  http://www.springframework.org/schema/beans/spring-beans.xsd
                  http://www.springframework.org/schema/context
                  http://www.springframework.org/schema/context/spring-
context.xsd">
    <!-- 注册自定义Job -->
    <bean id="jobDemo" class="com.itheima.job.JobDemo"></bean>
    <!-- 1：创建JobDetail对象,作用是负责通过反射调用指定的Job，注入目标对象，注入目标方法 -
->
    <bean id="jobDetail"

 class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean
">
        <!-- 注入目标对象 -->
        <property name="targetObject" ref="jobDemo"/>
```

```
        <!-- 注入目标方法 -->
        <property name="targetMethod" value="run"/>
    </bean>
    <!-- 2：注册一个触发器，指定任务触发的时间 -->
    <bean id="myTrigger"
class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
        <!-- 注入JobDetail -->
        <property name="jobDetail" ref="jobDetail"/>
        <!-- 指定触发的时间，基于Cron表达式（0/10表示从0秒开始，每10秒执行一次） -->
        <property name="cronExpression">
            <value>0/10 * * * * ?</value>
        </property>
    </bean>
    <!-- 3：注册一个统一的调度工厂，通过这个调度工厂调度任务 -->
    <bean id="scheduler"
class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
        <!-- 注入多个触发器 -->
        <property name="triggers">
            <list>
                <ref bean="myTrigger"/>
            </list>
        </property>
    </bean>
</beans>

为什么选用Quartz而不使用spring Task?
```

3、JasperSoft Studio

4、JasperReport

```
<dependency>
  <groupId>net.sf.jasperreports</groupId>
  <artifactId>jasperreports</artifactId>
  <version>6.8.0</version>
</dependency>

原理：
    - JRXML：报表填充模板，本质是一个xml文件
    - Jasper：由JRXML模板编译成的二进制文件，用于代码填充数据
    - Jrprint：当用数据填充完Jasper后生成的对象，用于输出报表
    - Exporter：报表输出的管理类，可以指定要输出的报表为何种格式
    - PDF/HTML/XML：报表形式

开发流程：
    1. 制作报表模板
    2. 模板编译
    3. 构造数据
    4. 填充数据
    5. 输出文件
API：
@Test
public void testJasperReports()throws Exception{
    String jrxmlPath =

 "D:\\ideaProjects\\projects111\\jasperdemo\\src\\main\\resources\\demo.jrxml";
```

```
    String jasperPath =

 "D:\\ideaProjects\\projects111\\jasperdemo\\src\\main\\resources\\demo.jasper";

    //编译模板
    JasperCompileManager.compileReportToFile(jrxmlPath,jasperPath);

    //构造数据
    Map paramters = new HashMap();
    paramters.put("reportDate","2019-10-10");
    paramters.put("company","itcast");
    List<Map> list = new ArrayList();
    Map map1 = new HashMap();
    map1.put("name","xiaoming");
    map1.put("address","beijing");
    map1.put("email","xiaoming@itcast.cn");
    Map map2 = new HashMap();
    map2.put("name","xiaoli");
    map2.put("address","nanjing");
    map2.put("email","xiaoli@itcast.cn");
    list.add(map1);
    list.add(map2);

    //填充数据
    JasperPrint jasperPrint =
        JasperFillManager.fillReport(jasperPath,
                                     paramters,
                                     new JRBeanCollectionDataSource(list));

    //输出文件
    String pdfPath = "D:\\test.pdf";
    JasperExportManager.exportReportToPdfFile(jasperPrint,pdfPath);
}
```

5。阿里云短信服务API

短信服务（Short Message Service）是阿里云为用户提供的一种通信服务的能力，支持快速发送短信验证码、短信通知等。 三网合一专属通道，与工信部携号转网平台实时互联。电信级运维保障，实时监控自动切换，到达率高达99%。短信服务API提供短信发送、发送状态查询、短信批量发送等能力，在短信服务控制台上添加签名、模板并通过审核之后，可以调用短信服务API完成短信发送等操作。

模板内容是：传智健康 验证码${number}，您正进行传智健康系统的身份验证，打死不告诉别人！

其中${number}为动态参数，需要我们后续在代码中控制。

手机快速登录：

（1）发送验证码

1. 获得用户输入的手机号码
2. 生成动态验证码(4或者6位)
3. 使用阿里云发送短信验证码
4. 把验证码存到redis(存5分钟)

（2）登录

1. 获得用户输入的信息(Map)
2. 取出redis里面的验证码和用户输入的验证码进行校验
3. 如果校验通过
    - 判断是否是会员，不是会员，自动注册为会员
    - 保存用户的登录状态(CAS或者自己手动签发token ，我们这里可以使用Cookie存储用户信息)

6、ECharts

7、POI

Apache POI是用Java编写的免费开源的跨平台的Java API，Apache POI提供API给Java程序对Microsoft Office格式档案读和写的功能，其中使用最多的就是使用POI操作Excel文件。jxl：专门操作Excel

POI结构：

- HSSF － 提供读写Microsoft Excel XLS格式档案的功能
- XSSF － 提供读写Microsoft Excel OOXML XLSX格式档案的功能（我们使用）
- HWPF － 提供读写Microsoft Word DOC格式档案的功能
- HSLF － 提供读写Microsoft PowerPoint格式档案的功能
- HDGF － 提供读Microsoft Visio格式档案的功能
- HPBF － 提供读Microsoft Publisher格式档案的功能
- HSMF － 提供读Microsoft Outlook格式档案的功能

我们使用：XSSF － 提供读写Microsoft Excel OOXML XLSX格式档案的功能

```
 // 读取excel
@Test
public void readExcel1() throws IOException {
    //创建工作簿
    XSSFWorkbook workbook = new XSSFWorkbook("D:\\read.xlsx");
    //获取工作表，既可以根据工作表的顺序获取，也可以根据工作表的名称获取
    XSSFSheet sheet = workbook.getSheetAt(0);
    //遍历工作表获得行对象
    for (Row row : sheet) {
        //遍历行对象获取单元格对象
        for (Cell cell : row) {
            //获得单元格中的值
            String value = cell.getStringCellValue();
            System.out.println(value);
        }
    }
    workbook.close();
}

// 创建excel
@Test
public void createExcel() throws IOException {
    //在内存中创建一个Excel文件
    XSSFWorkbook workbook = new XSSFWorkbook();
    //创建工作表，指定工作表名称
    XSSFSheet sheet = workbook.createSheet("传智播客");

    //创建行，0表示第一行
    XSSFRow row = sheet.createRow(0);
    //创建单元格，0表示第一个单元格
    row.createCell(0).setCellValue("编号");
    row.createCell(1).setCellValue("姓名");
    row.createCell(2).setCellValue("年龄");

    XSSFRow row1 = sheet.createRow(1);
    row1.createCell(0).setCellValue("1");
    row1.createCell(1).setCellValue("小明");
    row1.createCell(2).setCellValue("10");
```

```
    XSSFRow row2 = sheet.createRow(2);
    row2.createCell(0).setCellValue("2");
    row2.createCell(1).setCellValue("小王");
    row2.createCell(2).setCellValue("20");

    //通过输出流将workbook对象下载到磁盘
    FileOutputStream out = new FileOutputStream("D:\\itcast.xlsx");
    workbook.write(out);
    out.flush();//刷新
    out.close();//关闭
    workbook.close();
}
```

```
         <!--POI报表-->
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
        </dependency>
```

导出excel文件

```
@RequestMapping(value = "/exportBusinessReport",method = RequestMethod.GET)
    @PreAuthorize("hasAnyAuthority('REPORT_VIEW')")//查看统计报表的权限控制
    public Result exportBusinessReport(HttpServletRequest request,
HttpServletResponse response){
        try {
            //1.获取模板需要的数据
            Map<String,Object> result = reportService.getBusinessReportData();
            //2.获取模板
            String temlateRealPath =
request.getSession().getServletContext().getRealPath("template")+
File.separator+"report_template.xlsx";
            //3.通过poi技术往单元格填充数据
            XSSFWorkbook workbook = new XSSFWorkbook(new FileInputStream(new
File(temlateRealPath)));
            XLSTransformer transformer = new XLSTransformer();
            transformer.transformWorkbook(workbook, result);
            //4.通过输出流返回浏览器下载本地
            ServletOutputStream outputStream = response.getOutputStream();
            //设置文件类型
            response.setContentType("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");
            //设置文件名称 name一定不能写错 固定的区分大小写
            response.setHeader("content-
Disposition","attachment;filename=report.xlsx");
            workbook.write(outputStream);
            outputStream.flush();//刷新
            outputStream.close();//释放资源
            workbook.close();//关闭workbook
            //5.资源释放关闭
            return null; //成功null
        } catch (Exception e) {
            e.printStackTrace();
        }
        return new Result(false,MessageConstant.GET_BUSINESS_REPORT_FAIL);
```

```
    }
```

8、数据导入导出

```
1.用到了哪些技术?
    1. poi-->读写xls,pdf,word
        org.apache.poi->xls
    2.jasperReports->pdf
    3.自己查->docx doc
    4.easypoi

2.怎么考虑技术选型?
    org.apache.poi->xls

3.涉及几张表?
    看具体的报表（产品表、订单表、物流信息、客户信息）

4.涉及几个接口?
    1.导入-->批量添加
        1.下载一个模板:java实现文件下载有两者方式
            方式一:window.location.href
="../../template/ordersetting_template.xlsx";
            方式二:res.setHeader("Content-
Disposition","attachment;filename="+filename);
        两者区别是什么?自己下去看一下
        2.上传数据,就是文件上传
            1.前端content-type
            2.如果用springmvc 需要配置多媒体视图解析器,如果是springboot.默认就有了
            3.可以通过配置,限制文件的大小
            4.后台通过multipartFile接收
            5.poi读取收到的xls
    2.导出
        1.查数据
        2.根据模板,写入内容
        3.文件的下载

5.开发流程、开发步骤是怎样的?
    分导入导出自己再去分析下

6.哪些地方有难度或者印象深刻?
    1.文件的上传下载
    2.xls读取以及写入
        1.读取的数据量很大-->可能有问题
        2.写入的数据量很大-->100w记录-->xls-->
            https://www.cnblogs.com/swordfall/p/8298386.html
         系统要求导入40万条excel数据，采用poi方式，服务器出现内存溢出情况。

7.开发过程中有没有遇到什么问题?
    cell-->文本-->782300-->poi读出出来的值是-->7.823E5
    可以用DecimalFormat对这个double转换一下得到的就是我们要的值了
        DecimalFormat df = new DecimalFormat("0");
        String val = df.format(cell.getNumericCellValue());

8.开发了多久?
    一周
```

五、小程序

1、说一说yapi平台如何使用?

2、答题模块

3、七牛云API

常见的图片存储方案：
    方案一：使用nginx搭建图片服务器
    方案二：使用开源的分布式文件存储系统，例如Fastdfs、HDFS等
    方案三：使用云存储，例如阿里云、七牛云等

```
public class TestQiniu {
    // 上传本地文件
    @Test
    public void uploadFile(){
        //构造一个带指定Zone对象的配置类
        Configuration cfg = new Configuration(Zone.zone0());
        //...其他参数参考类注释
        UploadManager uploadManager = new UploadManager(cfg);
        //...生成上传凭证，然后准备上传
        String accessKey = "liyKTcQC5TP1LrhgZH6Xem8zqMXbEtVgfAINP53w";
        String secretKey = "f5zpuzKAPceEMG77-EK3XbwqgOBRDXDawG4UHRtb";
        String bucket = "itcast_health";
        //如果是Windows情况下，格式是 D:\\qiniu\\test.png，可支持中文
        String localFilePath = "D:/2.jpg";
        //默认不指定key的情况下，以文件内容的hash值作为文件名
        String key = null;
        Auth auth = Auth.create(accessKey, secretKey);
        String upToken = auth.uploadToken(bucket);
        try {
            Response response = uploadManager.put(localFilePath, key, upToken);
            //解析上传成功的结果
            DefaultPutRet putRet = new Gson().fromJson(response.bodyString(),
DefaultPutRet.class);
            System.out.println(putRet.key);
            System.out.println(putRet.hash);
        } catch (QiniuException ex) {
            Response r = ex.response;
            System.err.println(r.toString());
            try {
                System.err.println(r.bodyString());
            } catch (QiniuException ex2) {
                //ignore
            }
        }
    }
}
```