一、HashMap的put流程？（易点云、乌鸫等公司）

> key的hash值运算方式。
>
> - HashMap中如果传入的key是null，hash值固定就是0
> - 如果不为null， **会将key.hashCode的结果，进行高低位的亦或运算，得到一个结果，就是hash值。**
>   - 高低16位的^运算，是为了让hashCode的高16位，也能参与到计算存储位置的运算中，从而减少hash冲突的出现。
>
> 数组索引位置的计算。
>
> - 基于 **数组长度 - 1 再跟key的hash值做&运算** ，得到要存储的数组索引位置。
> - 为啥HashMap的长度必须是2的n次方？
>   - 如果长度不是2的n次方，会导致在计算索引位置时，hash冲突变多，导致HashMap查询变慢。
>
> 写入到数组，链表，红黑树。
>
> - 数组：当判断数据要插入的数组索引位置上，没数据，直接扔数组上。
> - 红黑树：如果发现数组上的数据类型是TreeNode，那就将数据插入到红黑树中。。
> - 链表：循环找到链表的最后一个位置，挂上。
> - 修改value：无论存储哪个位置，都会判断hash是否一致，如果一致，再判断 == 或者 eq是否为true，为true覆盖value值。
>
> 计数器
>
> - 直接对size属性做++即可，因为是单线程的集合。 如果是覆盖value，不会++。
>
> **如果你对ConcurrentHashMap掌握的好？** ，你就说一嘴，这玩应线程不安全。安全的话吗，要用啥等.....

二、项目中设计模式的应用（众安保险、国人通）

> low的方案：直接说你项目中用到了什么框架，框架里面涉及了什么设计模式~~
>
> **But，之前我听录音，有面试官问问题的时候，直接说，别聊框架，说你自己的实际落地的设计模式。**
>
> 所以咱们要 **提前准备** 项目中落地的设计模式，虽然我知道很多同学，实际项目没涉及到落地设计模式，但是你面试必须有！！！
>
> 如果贴合项目聊：
>
> - low方案：单例，工厂，代理。
> - 正常最好聊：策略，责任链，观察者，模板.....
>
> 比如我们做的短信平台项目，里面涉及到了多个校验的操作，我们希望这些校验操作之间可以做到解耦，并且根据客户的情况不同，做到一个定制化的校验。
>
> 比如客户A： 1，3，2，5，6，4顺序校验
>
> 比如客户B： 1，2，3 顺序校验
>
> 针对每个客户保存他要做的校验规则，存储客户的某个字段里。

可以将校验规则向上抽取一个接口，所有的校验规则都去实现这个接口，编写具体的逻辑。

然后利用Spring4版本中的泛型注入，直接从Spring的一级缓存里，将这个校验接口的所有实现类注入到我的业务代码。

根据客户存储的校验规则查询出来，然后在一级缓存（Map），获取对应的校验实现类去执行校验逻辑。

你想想Mybatis的Cache，再想想Servlet规范中的Filter，再想想SpringMVC中的Interceptor。。。

https://www.mashibing.com/course/1957

三、ThreadLocal你知道么（京东）

知道，回答完毕。

传参。 你可以和上面聊策略 + 责任链的设计模式结合在一起。

聊一下具体的项目中哪个业务涉及到了。

ThreadLocal存储数据的方式。

ThreadLocal本身不存储数据，他只是一个key。

真正存储数据的，是线程对象Thread当中的一个Map。

这个Map的底层是一个Entry数组，每一个Entry都可以存储key和value。

其中的key，就是ThreadLocal。

可以声明多个ThreadLocal对象，但是存储数据的，就是线程中的内个Entry数组。

内存泄漏问题。

ThreadLocal有两个内存泄漏问题

key的内存泄漏，这个问题已经被解决了，因为ThreadLocal内部对Key的引用是弱引用。

value的内存泄漏问题，在线程池操作ThreadLocal时，因为线程一致没有被回收，Entry数组他就一直在，前面如果ThreadLocal被回收掉了，但是value还在，导致value占用内存，但是你还查询不到。还有一个安全问题，上次逻辑存储的数据，在下次逻辑里又查询出来了。所以value的内存泄漏问题，需要咱们在使用完毕后，主动的remove，避免下次操作出现问题。

四、ThreadLocal如何实现主子线程之间的数据同步（京东）

用共享变量去实现父子线程之间的数据同步。

一般面试要聊InheritableThreadLocal，一定是父线程主动的去创建的子线程才可以。

如果是子线程给父线程传递数据，那就是采用共享编程，或者作为返回值。

具体的实现原理，看这个。

https://www.mashibing.com/course/2751

五、接口抽象类区别（上海启腾）

忒基础了，跟Java中的八大基本数据类型差不离。如果不会，别等面试官说话，你直接说，今天先到这吧。

从几个维度去聊

- 首先说属性，接口属性都是public static final修饰的常量，抽象类就没啥限制了。
- 另外就是方法的纬度了，在JDK1.8之后，interface增强了，其实方法的维度和抽象类区别不大，抽象类可以声明抽象方法，非抽象方法，静态方法都可以，但是interface增强后也可以，默认是抽象方法，default修饰可以写方法体，static修饰可以提供静态方法。但是interface的访问修饰符都是public。
- 接口不能写静态代码块，抽象类可以。接口的设计初衷和用途就决定了，他的目的是定义行为规范，目的不是实现具体逻辑。只是JDK1.8的新特性和这个初衷有点相违背。
- 其次，这哥俩都不实例化。
- 抽象类单继承，限制大。 接口多线程，更舒服。一个extends继承，一个implement实现。

六、请解释RESTful API的设计原则，并说明如何在Java中实现这些原则（滴滴司乘）

RESTful API他本质就是一个架构风格，他不是标准。可以不去遵循，也可以就遵循一部分，当然，也可以全部遵循。

然后再聊一下RESTful API的几个风格：

- 服务端不保存会话状态信息，会话管理内一套，不玩了。要求每个发送请求，在请求的报文中携带好必要的信息，比如现在的JWT，将会话信息的token扔到请求头里。说白了会话信息存储到客户端。
- 前端后端分离，后端不做什么请求转发，重定向这种操作，后端的每一个接口都是一个资源。
- 对请求的路径也有一定的要求，不会在请求路径上搞什么?key=value传参，而是基于路径本身传参或者是基于请求体中的JSON传参。
- 其次就是利用不同的请求方式代表不同的操作，比如/user，GET请求代表查询，POST请求代表添加，PUT请求代表修改，DELETE代表删除。

> - 响应数据层面，利用不同的状态码标识不同的事情，201 - created，代表添加成功等。。。。
>
> Java中实现非常简单，Spring完美的支持了架构风格。
>
> - 不存储会话，你就上JWT，搞Token，还能做到去中心化。
> - 前后端分离没啥说的啊。
> - SpringMVC提供了@PathVariable接收路径参数，@RequestBody接收请求体参数。
> - SpringMVC提供了@GetMapping，@PostMapping，@PutMapping.....
> - 状态码也支持，有HttpStatus。。。。

七、Java后端架构设计的原则是什么？请详细说明（马士兵教育）

> 官方一些的回答方式，架构设计的原则，固定内几个：（我个人不推荐）
>
> - 单一职责：一个类，就专门一干一个事，别搞乱七八糟的。
> - 里氏替换：子类可以拓展父类功能，但是别影响其他功能的正确性。
> - 开闭原则：功能可以加，但是不能改。
> - 依赖倒置：面向接口编程。
> - 接口隔离：.....
>
> 这个聊法本质没啥问题，中规中矩，但是没法体现你的履历给你带来的提升。。
>
> 我可能更推荐这种方式：
>
> - 可维护性：后期功能迭代，不可避免，不要因为一个小的改动，导致大面积的修改代码，做到高内聚低耦合！！
>   - 比如我们短信平台，在涉及做缓存和搜索功能时，我会单独提供一套缓存服务和搜索服务，甚至支付也一样，单独提供一个支付的服务。这下额服务只对外暴露基本的功能接口。如果后期涉及到了一些政策原因，其他原因，导致你需要去替换中间件，比如将Redis缓存中间件替换为其他的国产化的缓存组件。 因为前面的设计，我不需要去改动我的业务服务里的任何内容，只需要在缓存服务中，将之前Redis的API，替换为国产化的缓存的API即可，其他的不需要动。
> - 可扩展性：需要主动追加一些额外的功能或者方向时，可以更方面的动态增减功能，其他功能不会受到影响。
>   - 比如我们短信平台，在策略以及接口模块中的校验时，采用了策略设计模式 + 责任链模式做到动态可插拔的校验规则，如果后期需要拓展校验规则，只需要去继承校验接口去做具体的实现，然后修改用户的校验规则字段，就可以很方便的、扩展功能。
> - 安全性：安全是每个产品都必须考虑的点，所有很多加密算法要考虑好，比如你们的数据传输时，可以上非对称加密，对称加密，各种国密，（AES，RSA，SM1，2，3，4会就聊），还有一些敏感数据的过滤，在一些XSS攻击，注入攻击，CSRF攻击，这些都需要考虑，而且后期HTTPS也是必上的。
> - 性能：比如基本的CND，DNS优化，缓存的设计，分库分表，合理的中间件，多线程。。。。
> - 容错兜底：比如熔断，降级，做好兜底。 限流方便的考虑，还有MQ做削峰，部署的时候要规避单点故障问题，每个服务至少两台节点。 还可以考虑一些异地多活，其次还有数据的冗余备份。
> - 监控.....普罗米修斯。
> - 弹性伸缩.....GraalVM

八、描述Java中的异常处理机制，包括checked和unchecked异常的区别，以及如何自定义异常类（马士兵教育）

> Java中处理异常的一些机制：
>
> - try-catch：捕获异常的
> - throw：抛出异常的
> - throws：声明方法抛出异常
> - SpringMVC有异常处理器：全局的异常处理
>
> ---
>
> 聊第二个问题，可以先说一下异常的结构，顶级父类，Throwale，下面俩子类Error和Exception其中Execption里分为运行时异常（unchecked），检查时异常（checked）
>
> - checked：就是在编译时期，就存在的，当咱们做一些操作时，比如IO操作，文件可能不存在，提前向上抛出IOException。
> - unchecked：运行时异常，就是程序运行后，在执行代码时，可能会出现的异常，比如NPE，索引越界等等。。
>
> ---
>
> 自定义异常这，记住，一定是继承RuntimeException，这样才能更好的适配Spring的声明式事务，如果抛出的不是RuntimeException， Spring声明式事务会失效。
>
> 自定义异常就是自己在可能发生异常的地方，经过一定的逻辑判断，就可以手动抛出异常，自定义异常中最好存储code和message信息，以便在抛出异常后，可以快速定个位到哪个逻辑出现的什么问题。code和message可以在枚举中维护。

九、描述Java中基本数据类型的种类及其特点，如何选择合适的数据类型来优化内存使用（马士兵教育）

> 八个基本数据类型，得会。
>
> byte，short，int，long，float，double，boolean，char。
>
> 一般干活的时候，就用int居多，没问题，但是面试的话，就展开聊聊。
>
> 如果单纯的从减少内存占用来聊，肯定是根据具体的业务选择占用字节数小的类型更好。
>
> - 比如年龄，一般byte够用，是在不成，咱来个short。
> - 而且特别是数据库的字段，肯定是在业务合理的情况下，使用较小的类型，因为这样一个页可以存储更多的行，更多的数据就可以存放在MySQL的Buffer Pool中提升查询性能。
> - 其次，还是要贴近业务，比如咱们在做主键索引类型时，还是要选择long类型，一般咱们会选择分布式ID的雪花算法等方式生成ID，64位的界限更长。
> - 比如存储金额，一般也不会推荐采用Float，或者Double，因为Java中这种浮点型计算有误差，一般针对这种类型，咱们会选择BigDecimal，或者是采用Long类型，比如金额以厘做单位。
>
> 但是其实不是所有操作都是内存占用小，就效率高。比如CPU，可以在不同厂商的CPU里，他针对32位的int或者是64位的long类型的处理，性能更高。因为你占用内存小，在CPU的缓存行中会存储大量的数据，每次操作其中一个，可能都会导致缓存行失效，从而需要去主内存同步数据，这样反而会导致CPU的处理性能降低。。。