Java开发技术栈-文本

一.Java基础：

Java语法：

- 基础，基本类型，操作符，运算符，表达式；
- 面向对象，类，继承，多态，重写，重载；
- String，Object，Array，Enum；
- 集合，List，ArrayList，Set，HashSet，Map，HashMap，HashTable；
- File，IO，NIO，InputStream，OutputStream，Reader，Writer，Selector；
- 多线程，并发，Thread，Runnable，Future；
- 注解；
- 反射；
- JDBC；
- IOC依赖注入；
- AOP面向切面编程；

- 字符串常量池的迁移；
- 字符串KMP算法；
- equals和hashcode；
- 泛型、异常、反射；
- string的hash算法；
- hash冲突的解决办法：拉链法；
- foreach循环的原理；
- static、final、transient等关键字的作用；
- volatile关键字的底层实现原理；
- collections.sort方法使用的是那种排序方法；
- future接口，常见线程池中的FutureTask实现等；
- string的intern方法的内部细节，JDK1.6和JDK1.7的变化以及内部cpp代码StringTable的实现；

Java内存模型以及垃圾回收算法：

- 类加载机制，也就是双亲委派模型；
- Java内存分配模型（默认HotSpot）；线程共享的：堆区、永久区； 线程独占的：虚拟机栈、本地方法栈、程序计数器；
- 内存分配机制：年轻代（eden区、两个survivor区）、年老代、永久代以及它们的分

配过程；
- 强引用、软引用、弱引用、虚引用、GC；
- happens-before规则；
- 指令重排序、内存栅栏；
- Java8的内存分代改进；
- 垃圾回收算法：

标记-清除（不足之处：效率不高，内存碎片）
复制算法（解决了上述问题，但是内存只能使用一半，针对大部分对象存活时间短的场景，引出了一个默认8：1：1的改进，缺点是仍然需要借助外界来解决可能承载不下的问题）
标记整理

- 常用垃圾收集器

— 新生代：serial收集器、partNew收集器、parallel scavenge收集器
— 老年代：serial old收集器、parallel old收集器、cms（concurrent mark sweep）收集器、G1收集器（跨新生代和老年代）

- 常用GC的参数：-Xmn,-Xms, -Xmx，-XX:MaxPermSize，-XX:survivorRatio，-XX:-printGCDetails
- 常用工具：jps、jstat、jmap、jstack、图形工具jConsole、Visual VM、MAT

锁以及并发容器：

- synchronized和volatile理解；
- unsafe类的原理，使用它来实现CAS。因此诞生了AtomicInteger系列等；
- CAS可能产生的ABA问题的解决，如加入修改次数、版本号；
- 同步器AQS的实现原理；
- 独占锁、共享锁；可重入的独占锁reentrantlock、共享锁实现原理；
- 公平锁和非公平锁；
- 读写锁ReentrantReadWriteLock的实现原理；
- LockSupport工具；
- Condition接口及其实现原理；
- HashMap,HashSet,ArrayList,LinkedList,HashTable,ConcurrentHashMap,TreeMap实现原理；
- HashMap的并发问题；
- ConcurrentLinkedQueue的实现原理；
- Fork/Join框架；
- CountDownLatch和CyclicBarrier;

线程池源码：

- 内部执行原理；
- 各种线程池的区别；

正则表达式：

- 捕获组和非捕获组；
- 贪婪、勉强、独占模式；

二.Web方面：

- Servlet；
- JSP、JSTL、EL；
- Tomcat；

SpringMVC的架构设计：

- servlet开发存在的问题：映射问题、参数获取问题、格式化转换问题、返回值处理问题、试图渲染问题；
- SpringMVC为解决上述问题开发的几大组件及接口：HandlerMapping, HandlerAdapter, HandlerMethodArgumenntResolver,HttpMessageConverter, Converter, GenericConverter, HandlerMethodReturnValueHandler, ViewResolver, MultipartResolver;
- DispatcherServlet、容器、组件三者之间的关系；
- 叙述SpringMVC对请求的整体处理流程；
- SpringBoot；

SpringAOP源码：

- AOP的实现分类：编译期、字节码加载前、字节码加载后三种时机来实现AOP；
- 深刻理解其中的角色：AOP联盟、aspectj、JBoss AOP、Spring自身实现的AOP、Spring嵌入aspectj。特别是能用代码区分后两者。
- 接口设计

— AOP联盟定义的概念或接口：pointcut（概念，没有定义对应的接口）、joinpoint、advice、MethodInterceptor、MethodInvocation；
— SpringAOP针对上述Advice接口定义的接口及其实现类：BeforeAdvice,AfterAdvice, MethodBeforeAdvice, AfterReturningAdvice,针对aspectj对上述接口的实现AspectJMethodBeforeAdvice、AspectJAfterReturningAdvice、AspectJAfterThrowingAdvice、AspectJAfterAdvice；
— SpringAOP定义的pointcut接口：含有两个属性ClassFilter（过滤类）、MethodMatcher（过滤方法）；
— SpringAOP定义的ExpressionPointCut接口：实现中会引入aspectj的pointcut表达式；
— SpringAOP定义的PointcutAdvisor接口（将上述Advice接口和Pointcut接口结合起来）；

- SpringAOP的调用流程；
- SpringAOP自己的实现方式（ProxyFactoryBean）和借助aspectj实现方式区分；

Spring事务体系源码以及分布式事务Jotm Atomikos源码实现：

- JDBC事务存在的问题；
- Hibernate对事务的改进；
- 针对各种各样的事务，Spring如何定义事务体系的接口，以及如何融合JDBC事务和Hibernate事务的；
- 三种事务模型包含的角色以及各自的职责；
- 事务代码与业务代码分离的实现（AOP+ThreadLocal实现）；
- Spring事务拦截器TransactionInterceptor全景；
- X/Open DTP模型，两阶段提交，JTA接口定义；
- Jotm、Atomilos的实现原理；
- 事务的传播属性；
- PROPAGATION_REQUIRES_NEW、PROPAGATION_NESTED的实现原理以及区别；
- 事务的挂起和恢复的原理

数据库隔离级别：

- Read uncommitted:读未提交；
- Read committed:读已提交；
- Repeatable read：可重复度；
- Serializable：串行化；

数据库：

- 数据库性能的优化；
- 深入理解MySQL的Record Locks、Gap Locks、Next-Key Locks
- insert into select语句的加锁情况；
- 事务的ACID特性概念；
- innodb的MVCC理解；
- undo, redo, binlog

— undo redo都可以实现持久化，他们的流程是什么？为什么选用redo来做持久化；
— undo redo结合起来实现原子性和持久化，为什么undo log 要先于redo log持久化；
— undo为什么要依赖redo；
— 日志内容可以是物理日志，也可以是逻辑日志吗？他们各自的优点和缺点是什么？
— redo log最终采用的是物理日志加逻辑日志，物理到page,page内逻辑，还存在什么问题？怎么解决？Double Write;
— undo log为什么不采用物理日志而采用逻辑日志?
— 为什么要引入checkpoint?
— 引入checkpoint后为了保证一致性需要阻塞用户操作一段时间，怎么解决这个问题？（这个问题很有普遍性，redis、zookeeper都有类似的情况以及不同的应对策略），又有了同步checkpoint和异步checkpoint。
— 开启binlog的情况下，事务内部2PC的一般过程（含有2次持久化，redo log和binlog的持久化）
— 解释上述过程，为什么binlog的持久化要在redo log之后，在存储引擎commit之前；
— 为什么要保持事务之间写入binlog和执行存储引擎commit操作的顺序性？（即先写入binlog日志的事务一定先commit）；
— 为了保证上述顺序性，之前的办法是加锁prepare_commit_mutex，但是这极大的降低了事务的效率，怎么实现binlog的group commit?
— 怎么将redo log的持久化也实现group commit？至此事务内部2PC的过程，2次持久化的操作都可以group commit了，极大提高了效率；

ORM框架：mybatis、Hibernate

- 最原始的JDBC->Spring的JDBCTemplate->Hibernate->JPA->SpringDataJPA的演进之路；

SpringSecurity、shiro、SSO（单点登录）

- Session和cookie的区别和联系以及Session的实现原理；
- SpringSecurity的认证过程以及与Session的关系；
- CAS实现SSO；

日志：

- jdk自带的logging、log4j、log4j2、logback；
- 门面commons-logging、slf4j;
- 上述6种混战时的日志转换；

datasource：

- c3p0;
- druid;
- jdbcTemplate执行sql语句过程中对Connection的使用和管理；

框架与库：

- Spring；
- Hibernate，Ibatis；
- SpringMVC，Struts；
- Quartz；
- Ehcache；
- Apache commons

开发工具：

- Eclipse；
- IDEA；
- MVN；
- Ant，Ivy；

设计模式：

- 工厂模式；
- 工厂方法模式；
- 策略模式；

- 门面模式；
- 代理模式；
- 桥接模式；
- 单例模式；
- 多例模式；
- 装饰器模式；
- 迭代模式；

三.分布式、Java中间件、Web服务器等方面：

zookeeper源码：

- 客户端架构；
- 服务器端单机版和集群版，对应的请求处理器；
- 集群版session的建立和激活过程；
- leader选举过程；
- 事务日志和快照文件的详细解析；
- 实现分布式锁，分布式ID分发器；
- 实现leader选举；
- ZAB协议实现一致性原理；

序列化和反序列化框架：

- Avro研究；
- Thrift研究；
- Protobuf研究；
- Protostuff研究；
- Hessian

RPC框架dubbo源码：

- dubbo扩展机制的实现，对比SPI机制；
- 服务的发布过程；
- 服务的订阅过程；
- RPC通信的设计；

NIO模块以及对应的Netty和Mina、thrift源码：

- TCP握手和断开及有限状态机；
- backlog；
- BIO、NIO；
- 阻塞/非阻塞的区别、同步/异步的区别；
- 阻塞IO、非阻塞IO、多路复用IO、异步IO；
- reactor线程模型；
- jdk的poll、epoll、与底层poll、epoll的对接实现；
- Netty自己的epoll实现；
- 内核层poll、epoll的大致实现；
- epoll的边缘触发和水平触发；
- netty的EventLoopGroup设计；
- Netty的ByteBuf设计；
- Netty的ChannelHandler；
- Netty的零拷贝；
- Netty的线程模型，特别是与业务线程以及资源释放方面的理解；

消息队列kafka、RocketMQ、Notify、Hermes：

- kafka的文件存储设计；
- kafka的副本复制过程；
- kafka副本的leader选举过程；
- kafka的消息丢失问题；
- kafka的消息顺序性问题；
- kafka的isr设计和过半对比；
- kafka本身做的很轻量级来保持高效，很多高级特性没有：事务、优先级的消息、消息的过滤、更重要的是服务治理不健全、一旦出现问题，不能直观反映出来，不太适合对数据要求十分严格的企业级系统，而适合日志之类并发量大但是允许少量丢失或重复等场景；
- Notify、RocketMQ的事务设计；
- 基于文件的kafka、RocketMQ、和基于数据库的Notify和Hermes；
- 设计一个消息系统要考虑哪些方面；
- 丢失消息、消息重复、高可用等话题；

数据库的分库分表

NoSQL数据库MongoDB

KV键值系统memcached redis

- redis对客户端的维护和管理，读写缓冲区；
- redis事务的实现；
- jedis客户端的实现；
- jedisPool以及ShardedJedisPool的实现；
- redis epoll实现，循环中的文件事件和时间事件；
- redis的RDB持久化、save和bgsave；
- redis AOF命令追加、文件写入、文件同步到磁盘；
- redis AOF重写，为了减少阻塞时间采取的措施；
- redis的LRU内存回收算法；
- redis的master slave复制；
- redis的sentinel高可用方案；
- redis的cluster分片方案；

Web服务器Tomcat、nginx的设计原理

- Tomcat的整体构架设计；
- Tomcat对通信的并发控制；
- http请求到达Tomcat的整个处理流程；

ELK日志实时处理查询系统

- Elasticsearch, Logstash, Kibana

服务方面：

- SOA与微服务；
- 服务额合并部署、多版本自动快速切换和回滚；
- 服务的治理：限流和降级；

— 服务限流：令牌桶、漏桶；
— 服务降级：服务的熔断、服务的隔离、netflix的hystrix组件；

- 服务的线性扩展

— 无状态的服务如何做线性扩展：如一般的web应用，直接使用硬件或者软件做负载均衡，简单的轮询机制；
— 有状态的服务如何做线性扩展：如redis的扩展，一致性hash、迁移工具；
— 服务链路监控和报警：CAT、Dapper、Pinpoint；

Spring Cloud：

- Spring Cloud Zookeeper：用于服务注册和发现；
- Spring Cloud Config：分布式配置；
- Spring Cloud Netflix Eureka：用于rest服务的注册和发现；
- Spring Cloud Netflix Hystrix：服务的隔离、熔断和降级；
- Spring Cloud Netflix Zuul：动态路由、API Gateway；

分布式事务：

- JTA分布式事务接口定义，对此与Spring事务体系的整合；
- TCC分布式事务概念；
- TccCompensableAspect切面拦截创建ROOT事务；
- TccTransactionContexAspect切面使远程RPC调用资源加入到上述事务中，作为一个参与者；
- TccCompensableAspect切面根据远程RPC传递的TransactionContext的标记创建出分支事务；
- 全部RPC调用完毕，ROOT事务开始提交或者回滚，执行所有参与者的提交或者回滚；
- 所有参与者的提交或者回滚，还是通过远程RPC调用，provider端开始执行对应分支事务的confirm或者cancel方法；
- 事务的存储、集群共享问题；
- 事务的恢复、避免集群重复恢复；
- TCC分布式事务实现框架案例；
- JTA事务管理实现，类比Jotm、Atomikos等JTA实现；
- 事务的存储和恢复，集群是否共享问题调用方创建Compensable Transaction事务，并加入资源；
- CompensableMethodInterceptor拦截器向Spring事务注入CompensableInvocation;
- Spring的分布式事务管理器创建作为协调者CompensableTransaction类型事务，和当前线程进行绑定，同时创建一个JTA事务；

- 在执行SQL等操作的时候，所使用的JDBC等XAResource资源加入上述JTA事务；
- dubbo RPC远程调用前，CompensableDubboServiceFilter创建出一个代理XAResource，加入上述CompensableTransaction类型事务，并在RPC调用过程传递TransactionContext参与方创建分支的CompensableTransaction事务，并加入资源，然后提交JTA事务；
- RPC远程调用来到Provider端，CompensableDubboServiceFilter根据传过来的TransactionContext创建出对应的CompensableTransaction类型事务；
- provider端，执行时遇见@Transaction和@Compensable，作为一个参与者开启try阶段的事务，即创建了一个JTA事务；
- provider端try执行完毕开始准备try的提交，仅仅是提交上述JTA事务，返回结果到RPC调用端调用方法决定回滚还是提交；
- 全部执行完毕后开始事务的提交或者回滚，如果是提交则先对JTA事务进行提交（包含JDBC等XAResource资源的提交），提交成功后在对CompensableTransaction类型事务进行提交，如果JTA事务提交失败，则需要回滚CompensableTransaction类型事务。
- CompensableTransaction类型事务的提交就是对CompensableInvocation资源和RPC资源的提交，分别调用每一个CompensableInvocation资源的confirm，以及每一个RPC资源的提交CompensableInvocation资源的提交；
- 此时每一个CompensableInvocation资源的confirm开启的事务，又开始重复上述过程，对于JDBC等资源都加入新创建的JTA事务中，而RPC资源和CompensableInvocation资源仍然加入到当前线程绑定的CompensableTransaction;
- 当前CompensableInvocation资源的confirm开启的事务执行完毕后，开始执行commit,此时仍然是执行JTA事务的提交，提交完毕，一个CompensableInvocation资源的confirm完成，继续执行下一个CompensableInvocation资源的confirm，即又要重新开启一个新的JTA事务RPC资源的提交（参与方CompensableTransaction事务的提交）；
- 当所有CompensableInvocation资源的confirm执行完毕，开始执行RPC资源的commit，会运行远程调用，执行远程provider分支事务的提交，远程调用过程会传递事务id；
- provider端，根据传递过来的事务id找到对应的CompensableTransaction事务，开始执行提交操作，提交操作完成后返回响应结束；
- 协调者收到响应后继续执行下一个RPC资源的提交，当所有RPC资源也完成相应的提交，则协调者算是彻底完成该事务；

一致性算法：

- raft；

- zookeeper使用的ZAB协议；
- paxos

四.大数据方向

hadoop：

- UserGroupInfomation源码解读：JAAS认证，user和group关系的维护；
- RPC通信的实现；
- 代理用户的过程；
- kerberos认证；

MapReduce:

- MapReduce理论及其对应的接口定义；

HDFS：

- MapFile、SequenceFile；
- ACL；

YARN、Mesos资源调度

oozie：

- oozie XCommand设计；
- DagEngine的实现原理；

Hive：

- HiveServer2、metatore的thrift RPC通信设计；
- Hive的优化过程；
- HiveServer2的认证和授权；

- metastore的认证和授权；
- HiveServer2向metastore的用户传递过程；

Hbase：

- Hbase的整体架构图；
- Hbase的WAL和MVCC设计；
- client端的异步批量flush寻找RegionServer的过程；
- Zookeeper上Hbase节点解释；
- Hbase中的mini、major合并；
- Region的高可用问题对比kafka分区的高可用实现；
- RegionServer RPC调用的隔离问题；
- 数据从内存刷写到HDFS的粒度问题；
- rowKey的设计；
- MemStore与LSM；