# *JAVA开发必知会*

### PS:目标20K

# Spring 要点：

## 核心思想：

**IOC（Inversion of control）控制反转，也称为DI（Dependency Injection）依赖注入。**

把Bean的创建和调用交给Spring容器管理，目的降低项目耦合度。

**依赖注入方式：**

Setter注入和构造器注入

**Spring解决循环依赖：**

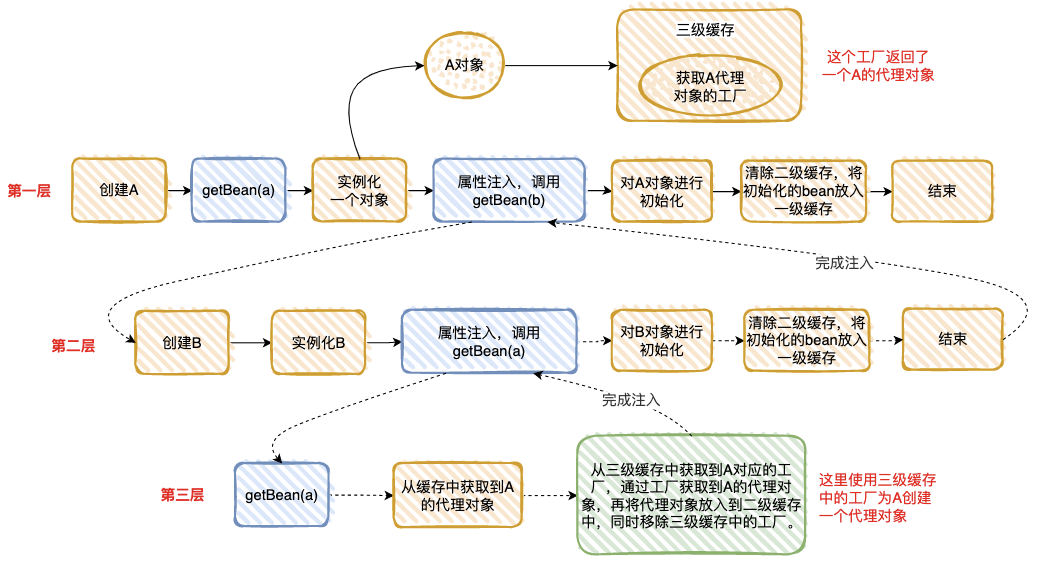
使用三级缓存

**三级缓存：**

SingletonFactories 单例对象工厂的cache

EarlySingletonObjects 提前曝光的单例对象的cache

SingletonObjects 单例对象的cache



1. 在第一层中，先去获取 A 的 Bean，发现没有就准备去创建一个，然后将 A 的代理工厂放入“三级缓存”（这个 A 其实是一个半成品，还没有对里面的属性进行注入），但是 A 依赖 B 的创建，就必须先去创建 B；
2. 在第二层中，准备创建 B，发现 B 又依赖 A，需要先去创建 A；

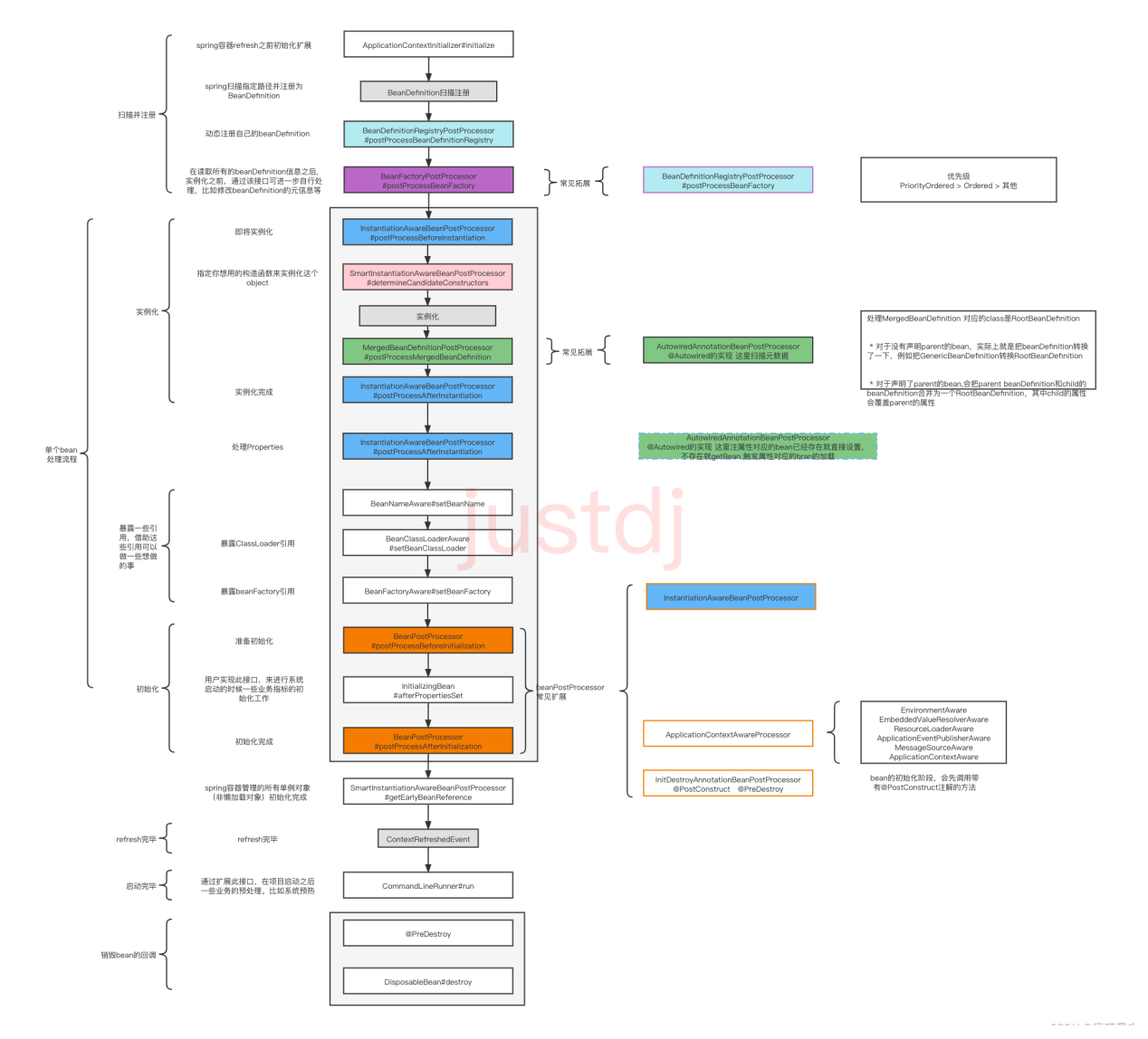
在第三层中，去创建 A，因为第一层已经创建了 A 的代理工厂，直接从“三级缓存”中拿到 A 的代理工厂，获取 A 的代理对象，放入“二级缓存”，并清除“三级缓存”；

1. 回到第二层，现在有了 A 的代理对象，对 A 的依赖完美解决（这里的 A 仍然是个半成品），B 初始化成功；
2. 回到第一层，现在 B 初始化成功，完成 A 对象的属性注入，然后再填充 A 的其它属性，以及 A 的其它步骤（包括 AOP），完成对 A 完整的初始化功能（这里的 A 才是完整的 Bean）。
3. 将 A 放入“一级缓存”。

**AOP（Aspect Oriented Programming）面向切面编程。**

这是对OO（面向对象编程）的一种补充，面向对象是把程序抽象成不同层次的对象进行编程，面向切面是把对象抽象成不同层次的切面进行编程，相当于细分后再细分。

## Spring管理Bean的生命周期：



**Bean生命周期：**

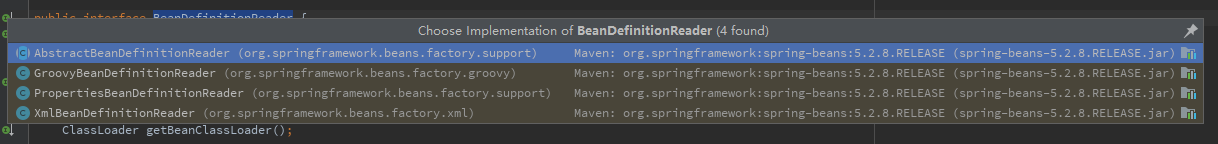
1. **BeanDefinitionReader读取器获取Bean的定义信息，并注册为BeanDefinition，里面包含了Bean实例化的各种条件和属性，作用域，是否懒加载等**
2. **通过BeanFactory实例化Bean，createBeanInstance（）**
3. **对Bean进行进行属性赋值，populateBean（）和invokeAwareMethods（）**
4. **Bean初始化，BeanPostProcessor执行前置处理，然后执行初始化方法invokeInitMethods（），最后执行后置处理器BeanPostProcessor**
5. **销毁Bean实例**

**扩展实现解释：**

1. Bean实现BeanNameAware接口，Spring会执行setBeanName方法，传入Bean的ID用于用户获取Bean在Spring容器中的名字，无实际意义
2. Bean实现BeanFactoryAware接口，Spring会执行setBeanFactory方法，传入BeanFactory实例给用户使用
3. Bean实现ApplicationContextAware接口，Spring会执行setApplicationContext方法传入应用上下文给用户使用
4. Bean实现BeanPostProcessor接口，Spring会执行postProcessBeforeInitialization方法，该执行在执行初始化前
5. Bean实现InitializingBean接口，Spring会执行afterPropertiesSet方法，该执行在属性初始化完成后执行，如果该Bean也设置了init-method，该初始化方法也会被执行
6. Bean实现BeanPostProcessor接口，Spring会执行postProcessAfterInitialization方法，该执行在执行初始化后
7. Bean实现DisposableBean接口，在Bean销毁时Spring会执行destory方法，同样，如果Bean也设置了destory-method，该销毁方法也会被执行

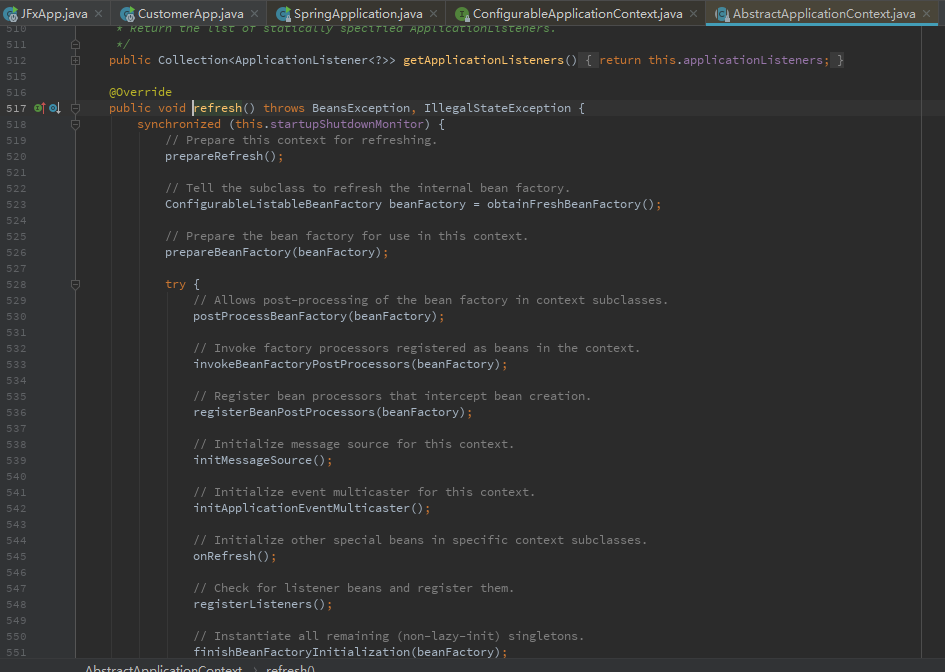
**Spring初始化解析：**

1. **初始化**，通过BeanDefinitionReader读取所有Bean的配置，比如XmlBeanDefinitionReader为解析XML文件，初始化加载所有的BEAN配置为BeanDefinition



1. **实例化**，从启动类run方法进去，往里找能找到refresh方法





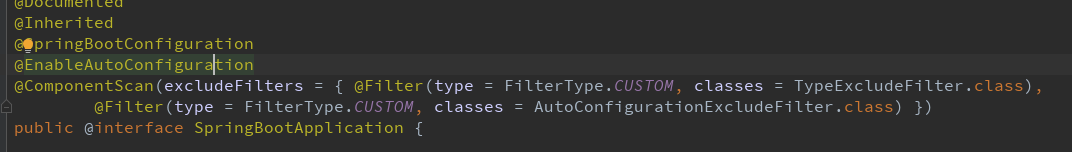
1. prepareRefresh()初始化spring容器属性设置，初始化属性设置，检查必备属性是否存在
2. obtainFreshBeanFactory()设置beanFactory序列号id和获取beanFactory
3. prepareBeanFactory()配置Bean工厂的属性，比如上下文类加载器，后置处理器，表达式解析器，Bean属性编辑器等
4. 模板方法，允许子类对BeanFactory做进一步设置
5. invokeBeanFactoryPostProcessors()调用BeanDefinitionRegistryPostProcessor实现向容器内添加bean的定义，调用BeanFactoryPostProcessor实现向容器内bean的定义的添加属性
6. registerBeanPostProcessors()，找到BeanPostProcessor后置处理器的实现，排序后注册进容器内
7. initMessageSource()设置国际化属性
8. initApplicationEventMulticaster()初始化时间广播器，并放入 applicationEventMulticaster bean 中
9. onRefresh()模板方法，在容器刷新的时候可以自定义逻辑，不同的Spring容器做不同的事情，比如：web环境中， 创建web容器
10. registerListeners()添加容器内事件监听器至事件广播器中，派发早期事件
11. finishBeanFactoryInitialization()实例化所有剩余的非懒加载单例,比如invokeBeanFactoryPostProcessors方法中根据各种注解解析出来的类，在这个时候都会被初始化。实例化的过程各种BeanPostProcessor开始起作用
12. finishRefresh()初始化生命周期处理器，调用生命周期处理器onRefresh方法，发布ContextRefreshedEvent事件，JMX相关处理
13. resetCommonCaches()清空产生的缓存数据

## Spring 自动装配

1. 启动类注解@SpringBootApplication

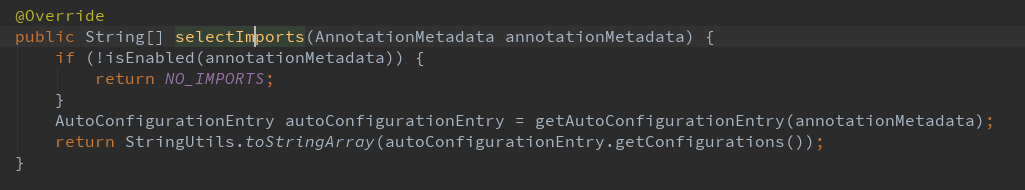


1. 该注解下@EnableAutoConfiguration使能自动配置

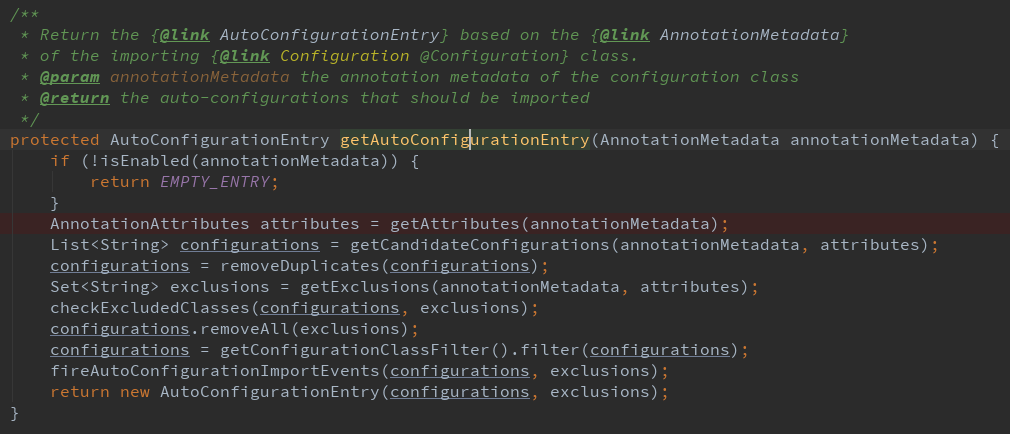


1. 引入自动配置选择器AutoConfigurationImportSelector,选择器下selectImports（）筛选自动装配的类

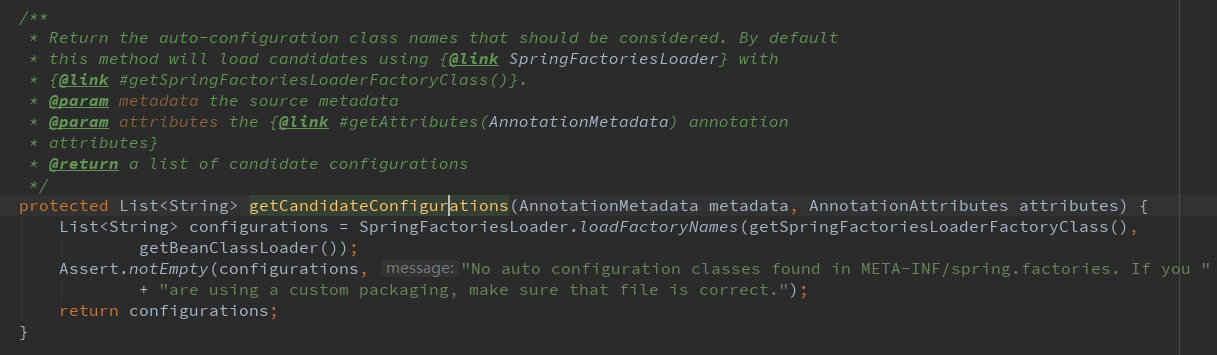




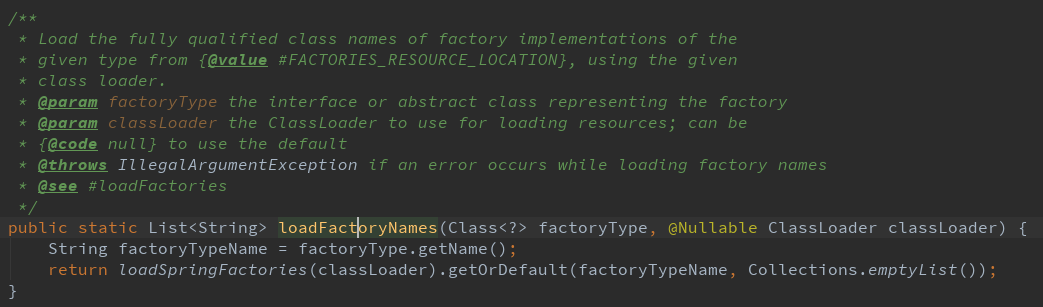
1. 筛选主方法getAutoConfigurationEntry（）

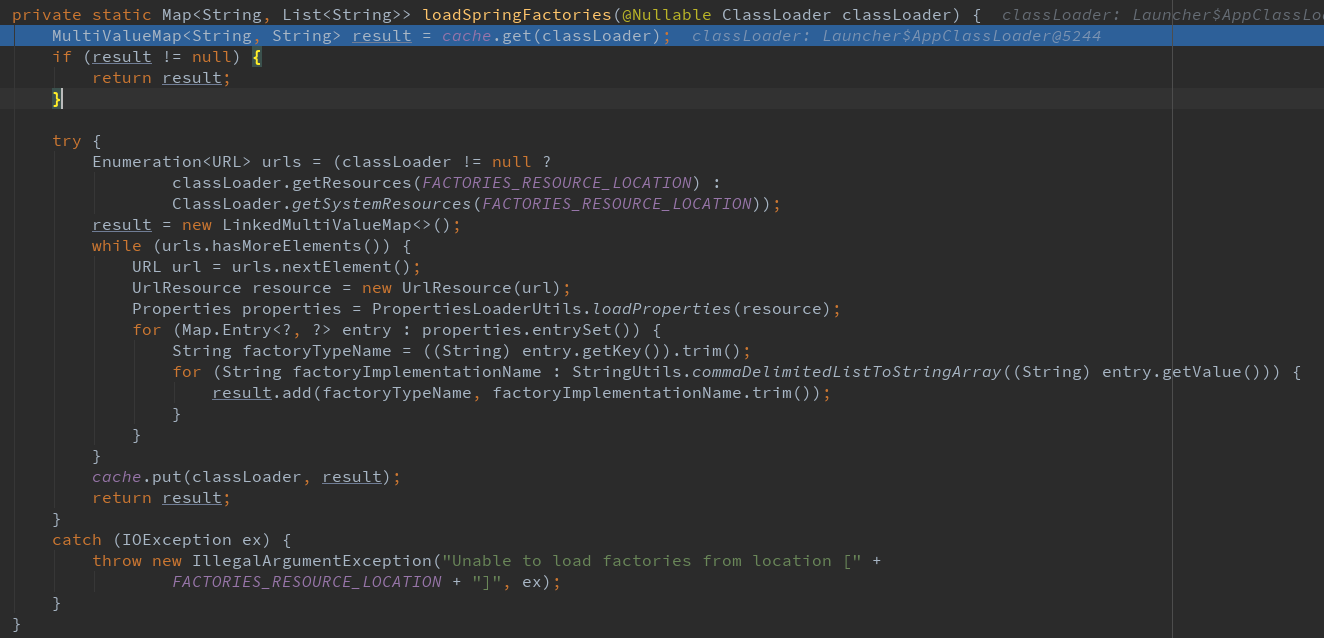


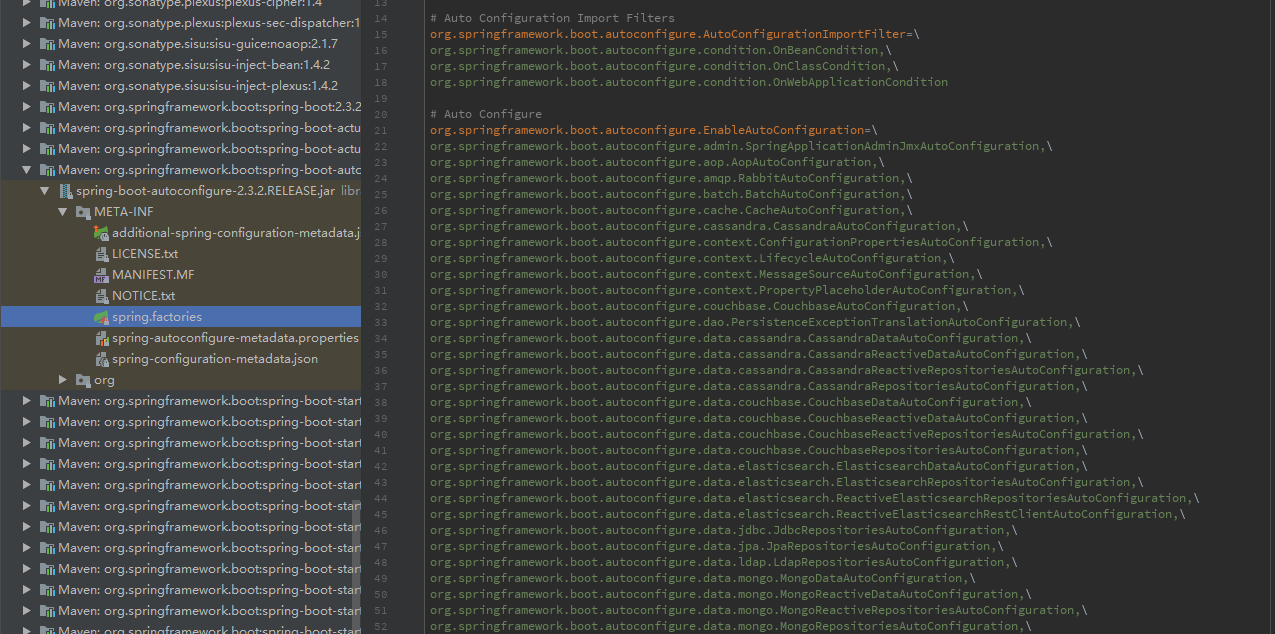
1. getCandidateConfigurations（）通过spring工厂加载器加载所有自动配置工厂



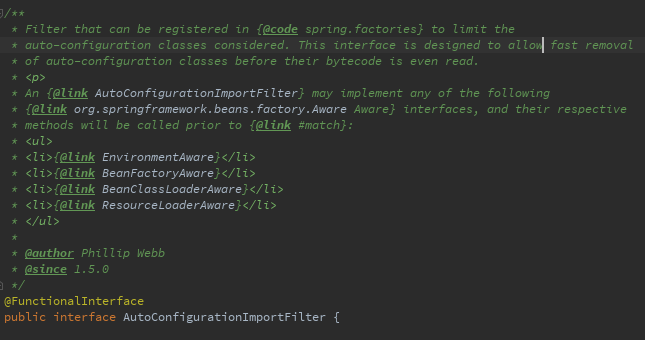
1. Spring工厂加载器加载所有工厂名loadFactoryNames（），这里就是加载META-INF/spring.factories路径下的所有自动装配的全限定类名，及用@EnableAutoConfiguration全限定类名org.springframework.boot.autoconfigure.EnableAutoConfiguration这个作为key获取下面所有的autoConfiguration







1. 加载 完成过后再通过自动装配过滤器过滤掉不需要自动装配的AutoConfigurationImportFilter，该接口有三个实现，**OnBeanCondition过滤器**判断Bean是否存在等等来控制是否自动装配，比如@ConditionalOnBean注解，存在某个Bean，或者@ConditionalOnMissingBean不存在某个Bean的情况下进行自动装配，**OnClassCondition过滤器**判断类路径来进行过滤，比如@ConditionalOnClass某个类存在时进行自动装配，**OnWebApplicationCondition过滤器**，判断是否是在web应用里进行加载

8. 筛选完成后触发自动装配监听器事件AutoConfigurationImportListener

## Spring事件监听器:

事件类：applicationEvent

事件监听器：applicationListener<T>

## BeanFactory和FactoryBean的区别：

1. BeanFactory以factory结尾，表明它是一个工厂，用来管理Bean的
2. FactoryBean以Bean结尾，表明是一个Bean，但是不是一个普通的Bean，它是实现了FactoryBean<T>的Bean，根据该Bean的Id从BeanFactory中获取的实际上是FactoryBean的getObject()返回的对象，而不是FactoryBean本身，如果要获取FactoryBean对象，可以在Id前面加一个&符号来获取

**AOP代理：**

**JDK动态代理和CGLib动态代理，Spring优先使用JDK。**

**JDK：**

利用拦截器（必须实现InvocationHandler接口）加上反射机制生成一个代理接口的匿名类，在调用具体方法前调用InvokeHandler来处理

**CGLib：**

利用ASM框架，对代理对象类生成的class文件加载进来，通过修改其字节码生成子类来进行代理

**Aspects（JDK动态代理）**

通知：

1. 前置通知（before advice），在某个连接点之前执行的通知，但这个通知不能阻止连接点的执行，除非它抛出一个异常
2. 返回后通知（after returning advice），在某个连接点正常完成后执行的通知
3. 抛出异常后通知（after throwing advice），再方法抛出异常退出时执行的通知
4. 后通知（after/finally advice）,当某个连接点退出的时候执行的通知，不论是正常退出还是抛出异常退出
5. 环绕通知（around advice），包围一个连接点的通知，这是最强大的一种通知类型，环绕通知可以在方法调用前后完成自定义的行为，他可以选择是否继续执行连接点或是直接返回自定义的返回值或者抛出异常来结束执行

**ArrayList：**

初始化容量为0，第一次扩容为10，后续以1.5倍扩容，为奇数时向下取整

**HashMap：**

扩容因子为0.75，目的：保证内存效率和减少hash冲突产生

**HashMap底层数据结构：**

**存储模式：**数组+链表/红黑树

**存储方式：**每次以Key和Value的形式存储，key通过hash算法获取，当产生hash冲突时，重复键值的数据以链表的形式存储，当链表长度大于8时，去检查hashMap的长度是否大于64，小于则对hashMap进行扩容，大于则把链表转换为红黑树形式存储。相反，当红黑树元素小于6时，并且进行resize时才会转换回链表

**升级：**在java1.7的时候链表插入数据时为头插法，每次从链表头部插入新的数据，这样在多线程的情况下可能造成链表循环，后续在java1.8之后采用尾插法，避免的这一类问题

**扩容：**map默认长度为16，创建一个新的数组，长度为原来长度的两倍，把原来的数据重新hash后存入新的数组

**ConcurrentHashMap：**

1.7版本锁的粒度是基于Segment的，一个Segment会包含多个hashEntry，锁的的粒度较大，效率低，后续1.8版本锁的粒度直接精细到hashEntry，并发效率提升到o（logn）

**Cas乐观锁：**

· CAS（Compare And Swap）比较并替换，是线程并发运行时用到的一种技术

· CAS是原子操作，保证并发安全，而不能保证并发同步

· CAS是CPU的一个指令（需要JNI调用Native方法，才能调用CPU的指令）

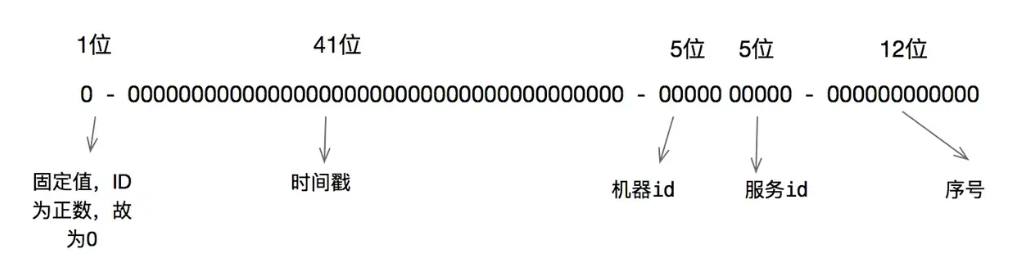
· CAS是非阻塞的、轻量级的乐观锁

**线程：**

**CountDowmLatch 程序计数器**

**CylicBarrier 同步处理**

**雪花算法64位作用：**



**线程池的属性：**

**corePoolSize：核心线程数，**这个参数跟后面讲述的线程池的实现原理有非常大的关系。在创建了线程池后，默认情况下，线程池中并没有任何线程，而是等待有任务到来才创建线程去执行任务，除非调用了prestartAllCoreThreads()或者prestartCoreThread()方法（从这2个方法的名字就可以看出，是预创建线程的意思，即在没有任务到来之前就创建corePoolSize个线程或者一个线程）。当线程池中的线程数目达到corePoolSize后，就会把到达的任务放到缓存队列当中。

**maximumPoolSize：线程池最大线程数，**这个参数也是一个非常重要的参数，它表示在线程池中最多能创建多少个线程。

**keepAliveTime：表示线程没有任务执行时最多保持多久时间会终止。**默认情况下，只有当线程池中的线程数大于corePoolSize时，keepAliveTime才会起作用，直到线程池中的线程数不大于corePoolSize，即当线程池中的线程数大于corePoolSize时，如果一个线程空闲的时间达到keepAliveTime，则会终止，直到线程池中的线程数不超过corePoolSize。但是如果调用了allowCoreThreadTimeOut(boolean)方法，在线程池中的线程数不大于corePoolSize时，keepAliveTime参数也会起作用，直到线程池中的线程数为0。

**unit：参数keepAliveTime的时间单位，**有7种取值，在TimeUnit类中有7种静态属性：

1. 天：TimeUnit.DAYS;

2. 小时：TimeUnit.HOURS;

3. 分钟：TimeUnit.MINUTES;

4. 秒：TimeUnit.SECONDS;

5. 毫秒：TimeUnit.MILLISECONDS;

6. 微秒：TimeUnit.MICROSECONDS;

7. 纳秒：TimeUnit.NANOSECONDS;

**Quene：任务队列**

1. ArrayBlockingQueue：基于数组结构的有界阻塞队列，按FIFO排序任务；

2. LinkedBlockingQuene：基于链表结构的阻塞队列，按FIFO排序任务，吞吐量通常要高于ArrayBlockingQuene；

3. SynchronousQuene：一个不存储元素的阻塞队列，每个插入操作必须等到另一个线程调用移除操作，否则插入操作一直处于阻塞状态，吞吐量通常要高于LinkedBlockingQuene；

4. priorityBlockingQuene：具有优先级的无界阻塞队列；

**handler：拒绝策略，**表示当拒绝处理任务时的策略，有以下四种取值：

1. ThreadPoolExecutor.AbortPolicy：丢弃任务并抛出RejectedExecutionException异常。

2. ThreadPoolExecutor.DiscardPolicy：也是丢弃任务，但是不抛出异常。

3. ThreadPoolExecutor.DiscardOldestPolicy：丢弃队列最前面的任务，然后重新尝试执行任务（重复此过程）

4. ThreadPoolExecutor.CallerRunsPolicy：由调用线程处理该任务

## #{}和${}

#{}预编译，防止sql注入，自动添加双引号

${}字符串替换，可能会sql注入，不安全

# Redis要点:

## Redis(Remote Dictionary Server)核心数据类型：

String字符串，hash散列映射（map,空间占用最小），list列表，set无序集合，zset有序集合

## Redis删除策略：

1. 定时删除，设置过期时间，到期了自动删除
2. 惰性删除，到期了，等下次访问再删除
3. 定期删除，周期轮询删除，定期抽取一定量的key检测是否过期了，过期了的就删除，最后判断这轮删除的Key数量是否超过指定比例，超过就继续轮询检查

## Redis淘汰策略：

1. volatile-lru，删除设置了过期时间的，且最近最不常使用的key
2. Allkeys-lru，删除所有的key中最近最不常使用的key，这是应用最广泛的策略
3. Volatile-random，随机删除一个设置了过期时间的key
4. Allkeys-random，随即删除一个key
5. Volatile-ttl，删除设置了过期时间的，且即将过期的key
6. Noevication，不删除任何key，如果内存溢出则报错，默认值

Redis4.0后新增的策略

1. Volatile-lfu，删除设置了过期时间的且最不常用的key
2. Allkeys-lfu，删除所有key中最不常用的key

## Redis持久化：

RDB（存快照）

占用存储空间：小，压缩级别

存储速度：慢

恢复速度：快

数据安全性：会丢失数据

资源消耗：高/重量级

启动优先级：低

AOF（存日志）

占用存储空间：大，指令级别

存储速度：快

恢复速度：慢

数据安全性：依据策略决定

资源消耗：低/轻量级

启动优先级：高

## Redis缓存问题：

缓存雪崩：较短时间内较多key同时过期，导致大量请求到达DB

解决：错峰设置key过期时间

缓存穿透：出现大量不存在的key访问redis，请求直接到达DB

解决：布隆过滤器

缓存击穿：请求高热key过期，请求直接到达DB

解决：提前预热高热key

## Redis官方为什么不提供Windows版本

因为目前Linux版本已经相当稳定，而且用户量很大，无需开发windows版本，反而会带来兼容性等问题.

## Redis一个字符串类型的值能存储最大容量是多少

512M

## Redis集群怎么实现

1.twemproxy，大概概念是，它类似于一个代理方式，使用方法和普通redis无任何区别，设置好它下属的多个redis实例后，使用时在本需要连接redis的地方改为连接twemproxy，它会以一个代理的身份接收请求并使用一致性hash算法，将请求转接到具体redis，将结果再返回twemproxy。使用方式简便(相对redis只需修改连接端口)，对旧项目扩展的首选。 问题：twemproxy自身单端口实例的压力，使用一致性hash后，对redis节点数量改变时候的计算值的改变，数据无法自动移动到新的节点。

2.codis，目前用的最多的集群方案，基本和twemproxy一致的效果，但它支持在 节点数量改变情况下，旧节点数据可恢复到新hash节点。

3.redis cluster3.0自带的集群，特点在于他的分布式算法不是一致性hash，而是hash槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

## Redis集群方案什么情况下会导致整个集群不可用

有A，B，C三个节点的集群,在没有复制模型的情况下,如果节点B失败了，那么整个集群就会以为缺少5501-11000这个范围的槽而不可用.

## Redis有哪些适合的场景

1. session会话缓存,分布式系统也可以用
2. 消息队列,因为提供了list和set数据类型的操作
3. 排行榜/计数器,因为提供了list和set数据类型的操作
4. 发布/订阅场景

## Redis和Redisson有什么关系

Redisson是一个高级的分布式协调Redis客服端，能帮助用户在分布式环境中轻松实现一些Java的对象

## Redis哈希槽的概念

Redis集群没有使用一致性hash,而是引入了哈希槽的概念，Redis集群有16384个哈希槽，每个key通过CRC16校验后对16384取模来决定放置哪个槽，集群的每个节点负责一部分hash槽。

## Redis集群的主从复制模型是怎样的

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型,每个节点都会有N-1个复制品.

## Redis集群会有写操作丢失

Redis并不能保证数据的强一致性，这意味这在实际中集群在特定的条件下可能会丢失写操作。

## Redis集群之间是如何复制的

异步复制

## Redis集群最大节点个数是多少

16384个

## 为什么要做Redis分区

分区可以让Redis管理更大的内存，Redis将可以使用所有机器的内存。如果没有分区，你最多只能使用一台机器的内存。分区使Redis的计算能力通过简单地增加计算机得到成倍提升,Redis的网络带宽也会随着计算机和网卡的增加而成倍增长。

## Twemproxy是什么

Twemproxy是Twitter维护的（缓存）代理系统，代理Memcached的ASCII协议和Redis协议。

## Redis常见性能问题和解决方案

(1) Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件

(2) 如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一次

(3) 为了主从复制的速度和连接的稳定性，Master和Slave最好在同一个局域网内

(4) 尽量避免在压力很大的主库上增加从库

(5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...

这样的结构方便解决单点故障问题，实现Slave对Master的替换。如果Master挂了，可以立刻启用Slave1做Master，其他不变。

# 数据库要点(mySql为例):

## 什么是索引

索引是一种数据结构，可以帮助我们快速的进行数据的查找.

## 索引是个什么样的数据结构

索引的数据结构和具体存储引擎的实现有关，在MySQL中使用较多的索引有Hash索引，B+树索引等，而我们经常使用的InnoDB存储引擎的默认索引实现为:B+树索引。

## Hash索引和B+树所有有什么区别或者说优劣

**Hash**:使用hash函数获取到对应的键值,再回表查询对应的数据即可,查询很快,但是它是无序的,并且不能进行范围查询,因为存在hash碰撞,所以查询效率其实是不确定的

**B+树**:使用的平衡查找树,是对B树的一种优化,每次都从根节点往叶子结点查找到键值后,根据查询判断是否要回表查询,天然支持范围和有序查询,效率是非常稳定的

## 聚簇索引和非聚簇索引

在B+树的索引中，叶子节点可能存储了当前的key值，也可能存储了当前的key值以及整行的数据，这就是聚簇索引和非聚簇索引。 在InnoDB中,只有主键索引是聚簇索引，如果没有主键，则挑选一个唯一键建立聚簇索引。如果没有唯一键，则隐式的生成一个键来建立聚簇索引.当查询使用聚簇索引时,在对应的叶子节点,可以获取到整行数据,因此不用再次进行回表查询.

## 在建立索引的时候，都有哪些需要考虑的因素

1. 索引字段使用的频率
2. 联合索引需要考虑索引的顺序
3. 存储占用和数据库查询压力,索引也不能过多

## 联合索引是什么

联合索引:使用多个字段一起建立索引

最左原则:查询如果不包含最左索引字段,会导致索引失效,不走索引而进行全表查询

## 检查慢SQL

使用解释器解释sql语句(explain命令),解释后能看到sql执行的顺序,是否使用索引,执行时间等

## 哪些会导致索引失效

模糊查询 like

或者 or

使用联合索引不遵循最左匹配原则等

## 数据库事务的特性：

原子性:要么全部执行成功,要么全部执行失败

一致性:事务的执行只会从一个一致性到达另一个一致性,不存在中间状态转换

持久性:事务提交后,结果能保持不变

隔离性:每个操作(事务)之间应该互不影响

## 事务的传播行为：

1. 当前没有事务，新建一个事务，如果有就加入当前事务
2. 加入当前的事务，如果没有就以无事务的方式进行
3. 加入当前的事务，如果没有事务就抛出异常
4. 新建一个事务，如果当前存在事务就把当前的事务先挂起
5. 以非事务的方式进行，如果当前存在事务则先把当前事务先挂起
6. 以非事务的方式进行，如果当前存在事务就抛出异常
7. 如果当前存在事务就嵌套在事务里进行操作，如果没有就按照1方式进行

## 事务的隔离级别

1. 读已提交
2. 读未提交
3. 可重复读
4. 串行

## Innodb使用的是哪种隔离级别

InnoDB默认使用的是可重复读隔离级别.

## MySQL都有哪些锁

**锁的类别分类:**

共享锁: 又叫做读锁. 当用户要进行数据的读取时,对数据加上共享锁.共享锁可以同时加上多个.

排他锁: 又叫做写锁. 当用户要进行数据的写入时,对数据加上排他锁.排他锁只可以加一个,他和其他的排他锁,共享锁都相斥

**锁的作用范围分类:**

行级锁,页级锁,表级锁

## 为什么要尽量设定一个主键

1. 保证数据的完整性
2. 加快查询效率

## 字段为什么要求定义为not null

null值会占用更多的字节，且会在程序中造成很多与预期不符的情况。

## 如果要存储用户的密码散列,应该使用什么字段进行存储

密码散列,盐,用户身份证号等固定长度的字符串应该使用char而不是varchar来存储,这样可以节省空间且提高检索效率。

## MySQL支持哪些存储引擎

MySQL支持多种存储引擎,比如InnoDB,MyISAM,Memory,Archive等等.在大多数的情况下,直接选择使用InnoDB引擎都是最合适的,InnoDB也是MySQL的默认存储引擎.InnoDB和MyISAM有什么区别?InnoDB支持事物，而MyISAM不支持事物InnoDB支持行级锁，而MyISAM支持表级锁InnoDB支持MVCC, 而MyISAM不支持InnoDB支持外键，而MyISAM不支持InnoDB不支持全文索引，而MyISAM支持。

## 横向分表和纵向分表

横向分表:数据覆盖范围:行,适合数据量比较大的情况

纵向分表:数据覆盖范围:列,适合字段关联关系比较清晰,数据量不能太大,比较占用空间

## 存储过程

存储过程是一个预编译的代码块，执行效率比较高,一个存储过程替代大量SQL语句 ，可以降低网络通信量，提高通信速率,可以一定程度上确保数据安全.

但是维护成本高,并不建议使用.

## 三个范式

第一范式: 每个列都不可以再拆分.

第二范式: 非主键列完全依赖于主键,而不能是依赖于主键的一部分.

第三范式: 非主键列只依赖于主键,不依赖于其他非主键

在设计数据库结构的时候,要尽量遵守三范式,如果不遵守,必须有足够的理由.比如性能. 事实上我们经常会为了性能而妥协数据库的设计.

# Spring Cloud Alibaba 分布式项目技术要点：

**Spring cloud alibaba 注册中心 nacos**

**Spring cloud alibaba 负载均衡ribbon**

**Spring cloud alibaba 远程调用 openFeign**

**Spring cloud alibaba 配置中心 nacos-config**

**Spring cloud alibaba 分布式事务 seata**

**Spring cloud alibaba 网关 gateWay**

**Spring cloud alibaba 链路追踪 skyWalking**

**Spring cloud alibaba 分布式日志处理方案 ELK**

**Spring cloud alibaba 流量控制熔断降级 Sentinel**

**Spring cloud alibaba 安全框架 Spring Security Oauth2**

# 其他知识点：

**Linux系统：**

**常用命令：**

Ls 列出该目录的目录和文件，以及其他信息

Pwd 显示用户当前所在的目录

Cd 跳转到某个目录

Tree 查看目录下的文件或者子目录，不包含影藏文件

Stat 查看文件或者目录属性

Mkdir 在当前目录下创建一个目录

Rm 删除文件或者目录

Cp 复制文件或者目录

Mv 移动文件或者目录

Cat 查看目标文件的内容

More 查看文件，单位页

Less 查看文件，单位行

Head 查看文件头部，默认前10行

Tail 查看文件尾部，默认后10行

Find 查找文件的位置

Grep 查找字符串或者文件

Zip/unzip 压缩/解压文件或文件夹，格式zip

Tar 压缩/解压文件或文件夹

Ps 展示所有线程的状态

Kill 杀掉某个线程

Netstat 查看网络状态

IfConfig 查看网路信息

Ping ping网络

Route 查看路由列表

**Linux安装MySql**

**Linux配置nginx+tomcat集群**

**Linux配置Redis**

**RabbitMq 消息处理：**

**五种消息模型：**

1. 简单模型Simple

一对一模式



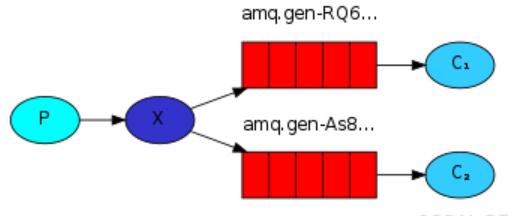
1. 工作队列 Work

消息分发给多个消费者，但每条消息只能消费一次



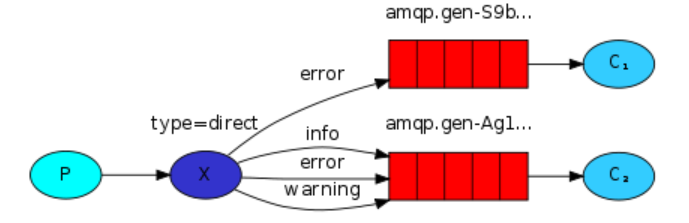
1. 广播模式 Fanout

多个消费者，每个消费者有自己的quene队列，每个队列绑定到交换机Exchange，生产者只能把消息发送到交换机，交换机转发到所有已绑定的所有队列，实现一条消息可以被多个消费者消费



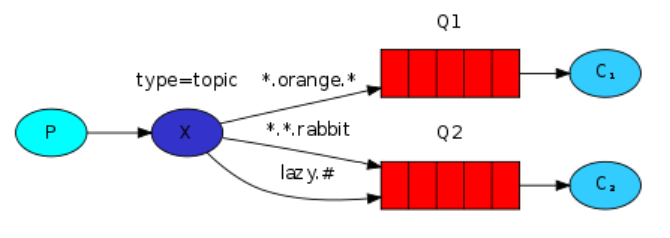
1. 定向模型 Direct

在广播模式基础上，队列和交换机的绑定增加了一个参数，RoutingKey（路由Key），在生产者向交换机发送消息时也必须指定消息的RoutingKey



1. 主题模型 Topic

在定向模型的基础上，路由Key的设置方式改成了通配符的匹配方式



**数据库分库分表：**

纵向拆分和垂直拆分

**Docker 部署项目：**

**单点登录：**

**父域cookie：**

把需要验证的sessionId或者token保存到父域里面，父域的cookie是被子域共享的，这样就实现了单点登录，比如设置到父域baidu.com后，tieba.baidu.com和map.baidu.com都能共享了，但是不支持跨主域名。

**认证中心（标准做法）：**

单独设置一个认证中心，第一次访问时，应用系统判断用户没有登录，于是跳转到认证中心让用户进行登录，登陆成功后返回用户登录的token和访问链接，并且认证中心也记住了该用户登录状态，用户拿到新的链接并携带token重新访问应用系统，应用系统拿着携带的token请求认证中心，验证token合法性，通过后及可正常访问了。

**使用前端LocalStorage实现：**

前端把sessionId或者token存放在前端的LocalStorage下面，如果有多个域则同时存放在多个下面，就能实现共享了。

## JVM内存结构：

方法区：类的属性、成员变量、构造函数，常量等

堆空间：创建的对象

本地方法栈：管理了native方法的一些执行细节

虚拟机栈：保存每一次方法调用的信息

程序计数器：记录线程执行的字节码行号

**JWT：**

用户登录鉴权技术

组成： 标头+有效载荷+签名

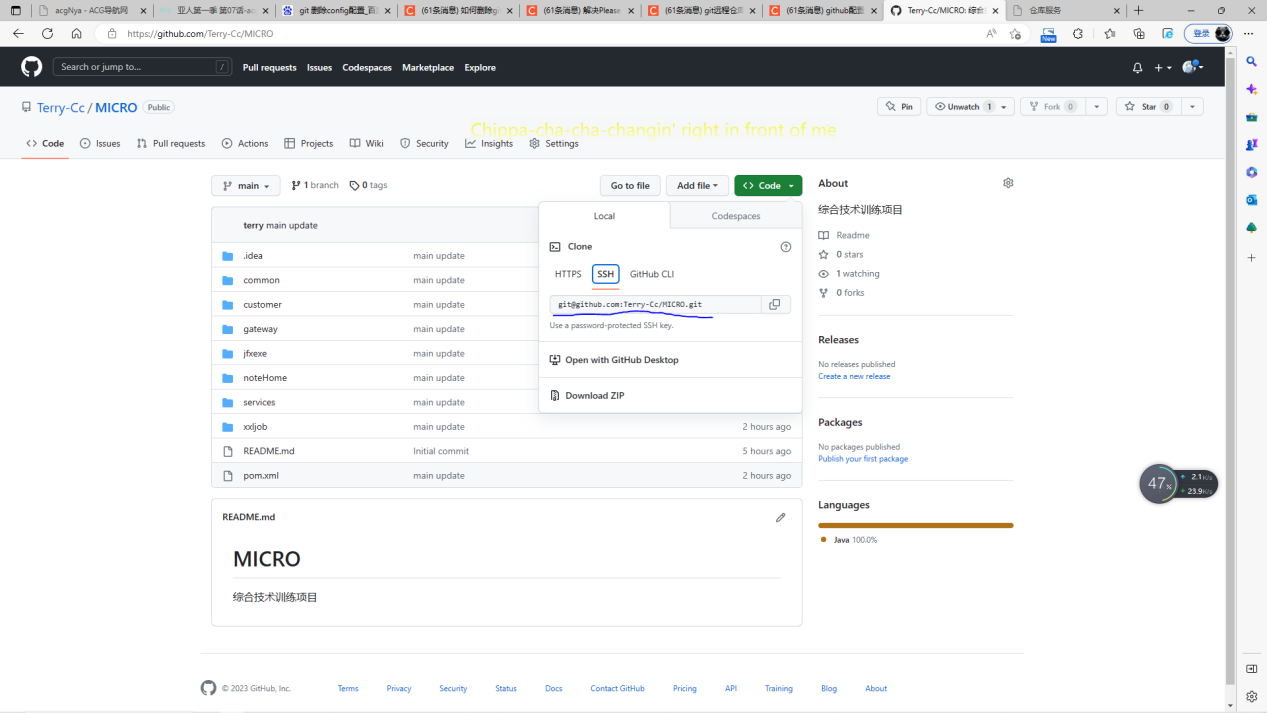
## GITHub配置SSH提交代码到仓库：

**项目Git bash进去：**

1. 移除https链接访问模式

Git remote rm origin

1. 添加SSH配置，用户名加gitHub ssh链接



Git remote add Terry-Cc git@github.com:Terry-Cc/MICRO.git

1. 配置git同户名和邮箱

Git config --global user.name Terry-Cc

Git config --global user.email [terry180219@163.com](mailto:terry180219@163.com)

1. 生成ssh秘钥

ssh-keygen -t rsa -C ["terry180219@163.com"](mailto:\"terry180219@163.com\")

秘钥生成后会显示生成后文件的位置，比如公钥：

Your public key has been saved in /c/Users/HP\_USER/.ssh/id\_rsa.pub

1. 登录github官网，配置SSH，把刚才生成的公钥文件内容配置到github SSH

