1.MYSQL

存储引擎，事务，锁，索引

Feature	MyISAM	Memory	InnoDB	Archive	NDB
B-tree indexes	Yes	Yes	Yes	No	No
Backup/point-in-time recovery(note 1)	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Clustered indexes	No	No	Yes	No	No
Compressed data	Yes (note 2)	No	Yes	Yes	No
Data caches	No	N/A	Yes	No	Yes
Encrypted data	Yes (note 3)	Yes (note 3)	Yes (note 4)	Yes (note 3)	Yes (note 3)
Foreign key support	No	No	Yes	No	Yes (note 5)
Full-text search indexes	Yes	No	Yes (note 6)	No	No
Geospatial data type support	Yes	No	Yes	Yes	Yes
Geospatial indexing support	Yes	No	Yes (note 7)	No	No
Hash indexes	No	Yes	No (note 8)	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Locking granularity	Table	Table	Row	Row	Row
MVCC	No	No	Yes	No	No
Replication support (note 1)	Yes	Limited (note 9)	Yes	Yes	Yes
Storage limits	256TB	RAM	64TB	None	384EB
T-tree indexes	No	No	No	No	Yes
Transactions	No	No	Yes	No	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

NDB 用的不多

INNODB支持外键，支持事务

事务的特性

- 原子性
- 一致性
- 隔离性
- 持久性

事务的隔离性

- 第一类丢失更新、第二类丢失更新、脏读、不可重复读、幻影读。

spring事务

声明式事务，编程式事务

mysql的锁

- 表级锁 开销小、加锁快、发生锁冲突的概率高，并发度低，不会出现死锁
- 行级锁 开销大、加锁慢、发生锁冲突的概率低，并发度高，会出现死锁

类型：

- 共享锁 行级，读取一行
- 排他锁 行级 更新一行
- 意向共享锁 表级，准备加共享锁
- 意向排他锁 标记，准备加排他锁
- 间隙锁 行级，使用范围条件时，对范围内不存在的记录加锁，一是为了防止幻读，二是为了满足回复和复制的需要。

	IS	IX	S	X
IS				×
IX			×	×
S		×		×
X	×	×	×	×

增加行级锁之前，会自动给表加意向锁

执行DML语句时，Innodb会自动给数据加排他锁。

执行DQL语句时查询语句

- 共享锁　lock in ...
- 排他锁　for update
- 间隙锁　采用范围条件是，innoDB对不存在的记录自动增加间隙锁。

事务有可能产生死锁。

事务1：　update t set ... where id = 1; update t set ... where id =2;

事务2：　update t set... where id = 2; update t set ... where id = 1;

解决方案

- 一般InnoDB会自动检测到，并使一个事务回滚，另一个事务继续。
- 设置超时等待参数 innodb_lock_wait_timeout

避免死锁

- 不同的业务并发访问多个表时，应约定以相同的顺序来访问这些表。
- 以批量的方式处理数据时，应事先对数据排序，保证线程按固定的顺序来处理数据。
- 在事务中，如果要更新事务，应直接申请足够级别的锁。

乐观锁

1.版本号机制

2.cas算法 也叫自旋锁

MYSQL的索引，

- 数据分块存储，每一块称为一页
- 所有的值都是按顺序存储的，并且每一个叶子到根的距离相同
- 非叶节点存储数据的边界，叶子节点存储指向数据行的指针。
- 通过边界缩小数据的范围，从而避免全表扫描，加快了查找的速度。

redis的数据类型

key

string

hash

list

set

sorted set

bitmap

hyperloglog

常见面试题/redis/过期策略

- 惰性删除，客户端访问某个key时，redis会检查该key是否过期，若过期则删除。
- 定期扫描
    - 1.从过期字典中随机选择20个key
    - 2.删除这20个key中已过期的key
    - 3.如果过期的key的比例超过25%，则重复步骤1

当redis占用内存超出最大限制时，可采用如下策略，让redis淘汰一些数据，以腾出空间继续提供读写服务。

- 对可能增大内存的命令返回错误
- 在设置了过期时间的key中，选择剩余寿命(TTL)最短的key，将其淘汰；
- 在设置了过期时间的key中，选择最少使用的key，将其淘汰。
- 在设置了过期时间的key中，随机选择一些key，将其淘汰。
- 在所有的key中，选择最少使用的key，将其淘汰。
- 在所有的key中，选择最少使用的key，将其淘汰。
- 在所有的key中，所及选择一些key，将其淘汰。

近似lru算法

给每个key维护一个时间戳，淘汰时随机采样5个key，从中淘汰最旧的key，如果还是超出内存限制，则继续随机采样淘汰。

优点：比lru节约内存，却可以取得非常近似的效果。

redis/缓存穿透

故意查不存在的数据，使得请求直达存储层，使其负载过大，甚至泵机。

解决方案

- 缓存空对象，存储层未命中之后，仍然将空值存入缓存层。再次访问该数据时，缓存层会直接返回空值。
- 布隆过滤器，将所有存在的key提前存入布隆过滤器，在访问缓存层之前，先通过过滤器拦截，若请求的是不存在的key，则直接返回空值。

缓存击穿

一份热点数据，访问量非常大，在其缓存失效瞬间，大量请求直达存储层，导致服务崩溃。

解决方案

1.加互斥锁

对数据的访问加互斥锁，当一个线程访问该数据时，其他数据只能等待。

这个线程访问过后，缓存中的数据将被重建，届时其他线程就可以直接从缓存取值。

2.永不过期

不设置过期时间，所以不会出现上述问题，这是“物理”上的不过其。

为每个value设置逻辑过期时间，当发现该值逻辑过期时，使用单独的线程重建缓存。

缓存雪崩

由于某种原因，缓存层不能提供服务，导致所有的请求直达存储层，造成存储层泵机。

- 1.避免同时过期
- 2.构建高可用的redis缓存
- 3.构建多级缓存
- 4.启用限流和降级措施。

redis分布式锁

修改数据时，经常需要先把数据读入内存中，在内存中修改后再存回去，在分布式应用中，可能多个进程同时执行上述操作，而读取和修改非原子操作，所以会产生冲突，增加分布式锁，可以解决此类问题。

基本原理

同步锁：在多个进程都能访问到的地方，做一个标记，标识该数据的访问权限。

分布式锁：在多个进程都能访问到的地方，做一个标记，标识该数据的访问权限。

实现方式

- 基于数据库实现分布式锁
- 基于redis实现分布式锁
- 基于zookeeper实现分布式锁

redis实现分布式锁的原则

1.安全属性：独享。在任意时刻，只有一个客户端持有锁。

2.活性A:无死锁。即便持有锁的客户端崩溃或者网络被分裂，锁仍然可以被获取。

3.活性B，容错。只要大部分redis节点都活着，客户端就可以获取和释放锁。

单实例实现方法

1.获取锁使用命令

2.通过lua脚本释放锁-可以避免删除别的客户端获取成功的锁。用lua脚本不走客户端可以避免这个问题。

多实例实现分布式锁。

redlock算法

- 获取当前unix时间，以毫秒为单位
- 依次尝试从N个实例，使用相同的key和随机值获取锁，并设置响应超时时间。如果服务器没有在规定时间内响应，客户端应该尽尝试另外一个redis实例。
- 客户端使用当前时间减去开始获取锁的时间，得到获取锁使用的时间，当且仅当大多数的redis节点都取到锁，并且使用的时间小于锁失效时间时，锁才算取得成功。
- 如果取到了锁，key的真正有效时间等于有效时间减去获取锁使用的时间。
- 如果锁获取失败，客户端应该在所有的redis实例上进行解锁。

考虑大部分节点都成功。

Bean的作用域,通过scope注解改变

singleton

prototype

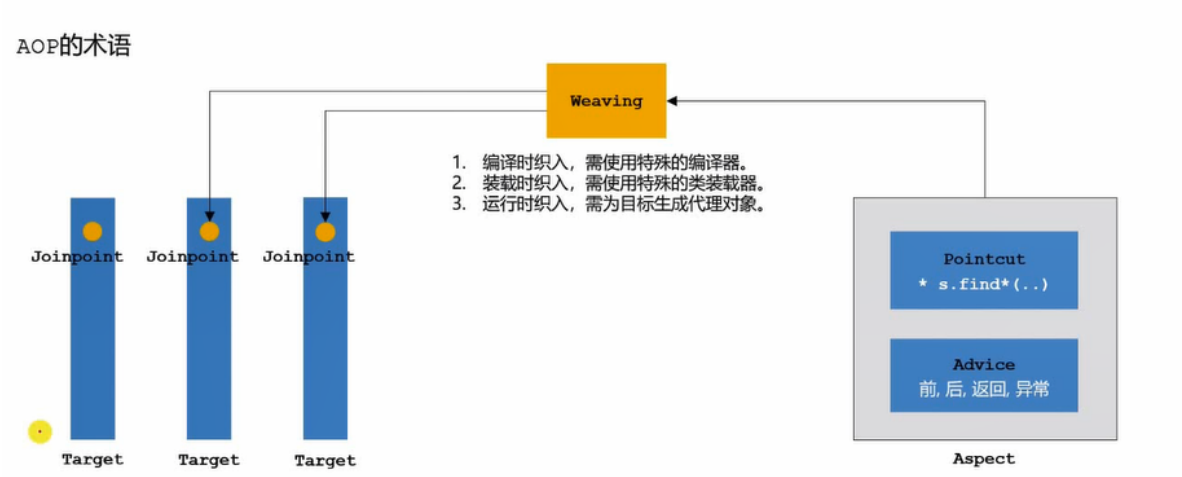
request

session

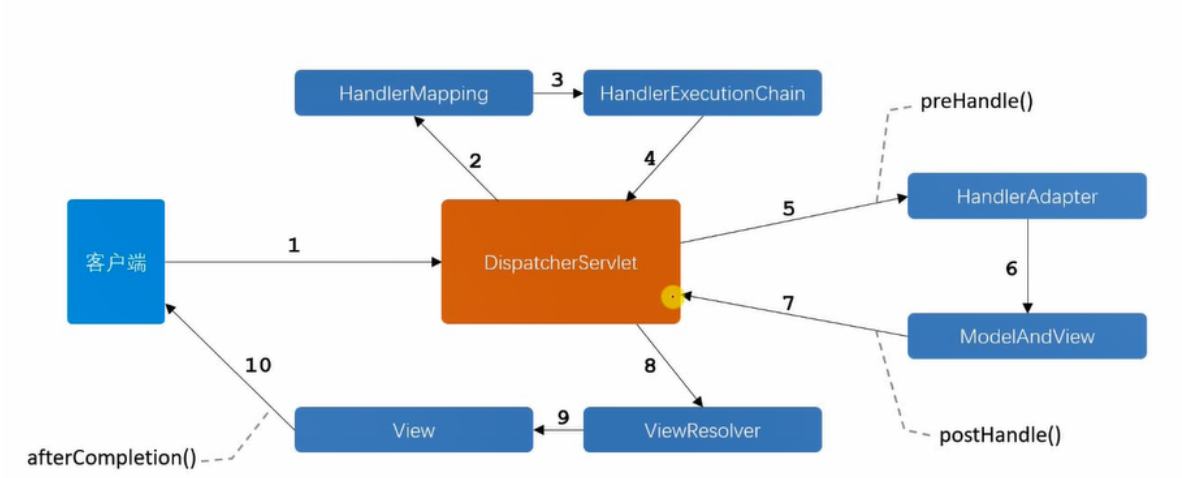
globalsession

application

AOP


AOP的术语
Weaving
1. 编译时织入，需使用特殊的编译器。
2. 装载时织入，需使用特殊的类装载器。
3. 运行时织入，需为目标生成代理对象。
Joinpoint
Joinpoint
Joinpoint
Target
Target
Target
Pointcut
* s.find*(..)
Advice
前, 后, 返回, 异常
Aspect

spring mvc


HandlerMapping
3
HandlerExecutionChain
preHandle()
2
4
HandlerAdapter
5
客户端
1
DispatcherServlet
6
7
ModelAndView
10
8
View
9
ViewResolver
postHandle()
afterCompletion()