

项目发布与总结

牛客Java高级工程师 第八章

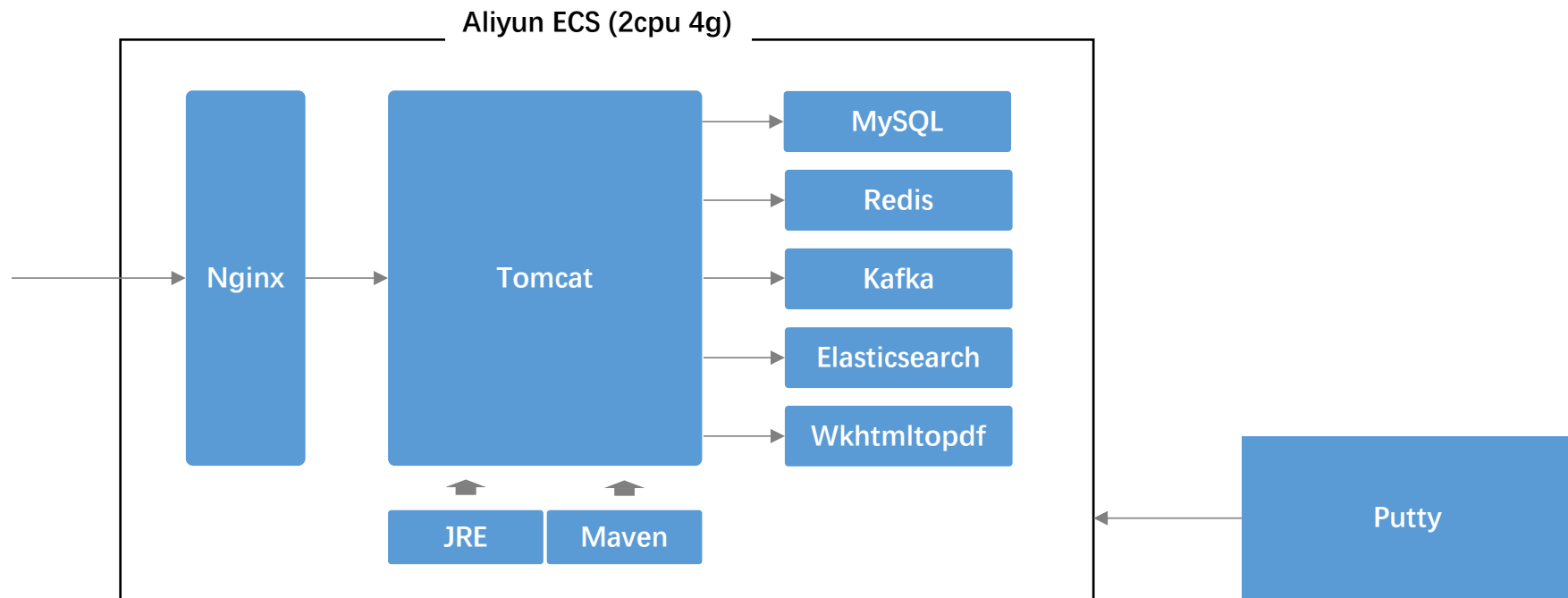
► 1. 单元测试

- Spring Boot Testing
 - 依赖: `spring-boot-starter-test`
 - 包括: Junit、Spring Test、AssertJ、...
- Test Case
 - 要求: 保证测试方法的独立性。
 - 步骤: 初始化数据、执行测试代码、验证测试结果、清理测试数据。
 - 常用注解: `@BeforeClass`、`@AfterClass`、`@Before`、`@After`。

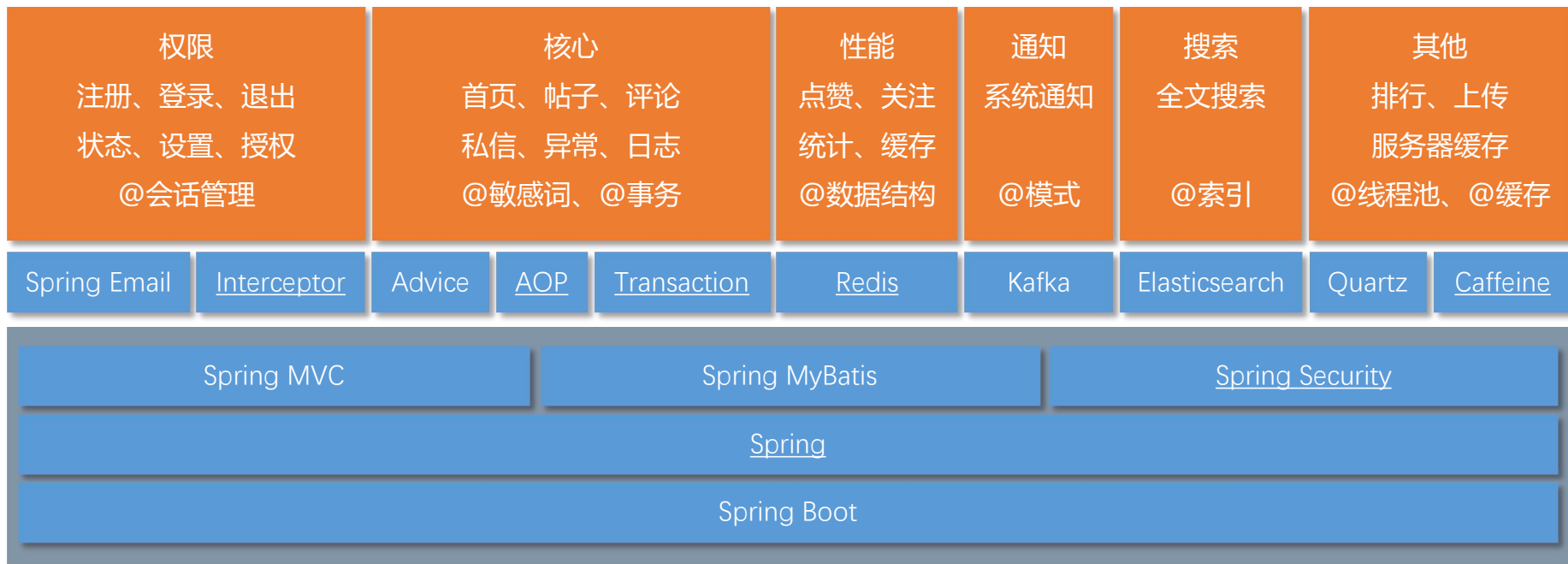
► 2. 项目监控

- Spring Boot Actuator
 - Endpoints: 监控应用的入口, Spring Boot内置了很多端点, 也支持自定义端点。
 - 监控方式: HTTP 或 JMX。
 - 访问路径: 例如 `"/actuator/health"` 。
 - 注意事项: 按需配置暴露的端点, 并对所有端点进行权限控制。

► 3. 项目部署



► 4. 项目总结



► 5. 常见面试题

1. MySQL

存储引擎、事务、锁、索引

2. Redis

数据类型、过期策略、淘汰策略、缓存穿透、缓存击穿、缓存雪崩、分布式锁

3. Spring

Spring IoC、Spring AOP、Spring MVC

► 5. 常见面试题 / MySQL / 存储引擎

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

► 5. 常见面试题 / MySQL / 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

- 并发异常：第一类丢失更新、第二类丢失更新、脏读、不可重复读、幻读
- 隔离级别：Read Uncommitted、Read Committed、Repeatable Read、Serializable

Spring事务管理

- 声明式事务
- 编程式事务

► 5. 常见面试题 / MySQL / 锁

范围

- **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 (InnoDB)

- 共享锁 (S)：行级，读取一行；
- 排他锁 (X)：行级，更新一行；
- 意向共享锁 (IS)：表级，准备加共享锁；
- 意向排他锁 (IX)：表级，准备加排他锁；
- 间隙锁 (NK)：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，二是为了满足恢复和复制的需要。

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

► 5. 常见面试题 / MySQL / 锁

加锁

- 增加行级锁之前，InnoDB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

共享锁 (S) : `SELECT ... FROM ... WHERE ... LOCK IN SHARE MODE;`

排他锁 (X) : `SELECT ... FROM ... WHERE ... FOR UPDATE;`

间隙锁 (NK) : 上述SQL采用范围条件时，InnoDB对不存在的记录自动增加间隙锁。

► 5. 常见面试题 / MySQL / 锁

死锁

- 场景

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

1. 一般InnoDB会自动检测到, 并使一个事务回滚, 另一个事务继续;
2. 设置超时等待参数 `innodb_lock_wait_timeout`;

- 避免死锁

1. 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;
2. 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;
3. 在事务中, 如果要更新记录, 应直接申请足够级别的锁, 即排他锁;

► 5. 常见面试题 / MySQL / 锁

悲观锁（数据库）

乐观锁（自定义）

1. 版本号机制

```
UPDATE ... SET ..., VERSION=#{version+1} WHERE ... AND VERSION=${version}
```

2. CAS算法 (Compare and swap)

是一种无锁的算法，该算法涉及3个操作数（内存值V、旧值A、新值B），当V等于A时，采用原子方式用B的值更新V的值。该算法通常采用自旋操作，也叫自旋锁。它的缺点是：

- ABA问题：某线程将A改为B，再改回A，则CAS会误认为A没被修改过。
- 自旋操作采用循环的方式实现，若加锁时间长，则会给CPU带来巨大的开销。
- CAS只能保证一个共享变量的原子操作。

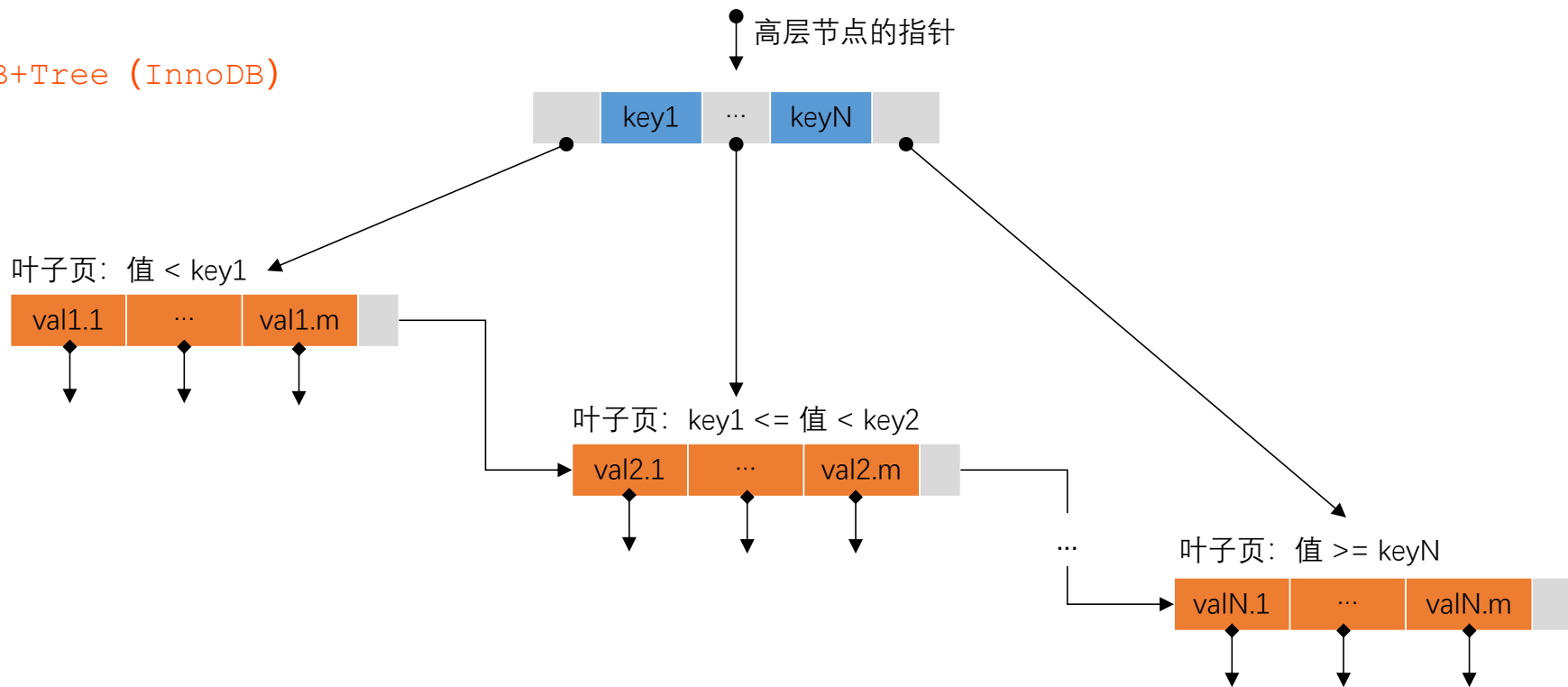
► 5. 常见面试题 / MySQL / 索引

B+Tree (InnoDB)

- 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。

► 5. 常见面试题 / MySQL / 索引

B+Tree (InnoDB)



► 5. 常见面试题 / Redis / 数据类型

数据类型	最大存储数据量
key	512M
string	512M
hash	$2^{32} - 1$
list	$2^{32} - 1$
set	$2^{32} - 1$
sorted set	
bitmap	512M
hyperloglog	12K

► 5. 常见面试题 / Redis / 过期策略

Redis会把设置了过期时间的key放入一个独立的字典里，在key过期时并不会立刻删除它。

Redis会通过如下两种策略，来删除过期的key：

- 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

Redis默认每秒执行10次过期扫描（配置hz选项），扫描策略如下：

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；

► 5. 常见面试题 / Redis / 淘汰策略

当Redis占用内存超出最大限制 (`maxmemory`) 时, 可采用如下策略 (`maxmemory-policy`) , 让Redis淘汰一些数据, 以腾出空间继续提供读写服务:

- `noeviction`: 对可能导致增大内存的命令返回错误 (大多数写命令, `DEL`除外) ;
- `volatile-ttl`: 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;
- `volatile-lru`: 在设置了过期时间的key中, 选择最少使用的key (LRU) , 将其淘汰;
- `volatile-random`: 在设置了过期时间的key中, 随机选择一些key, 将其淘汰;
- `allkeys-lru`: 在所有的key中, 选择最少使用的key (LRU) , 将其淘汰;
- `allkeys-random`: 在所有的key中, 随机选择一些key, 将其淘汰;

► 5. 常见面试题 / Redis / 淘汰策略

- LRU算法

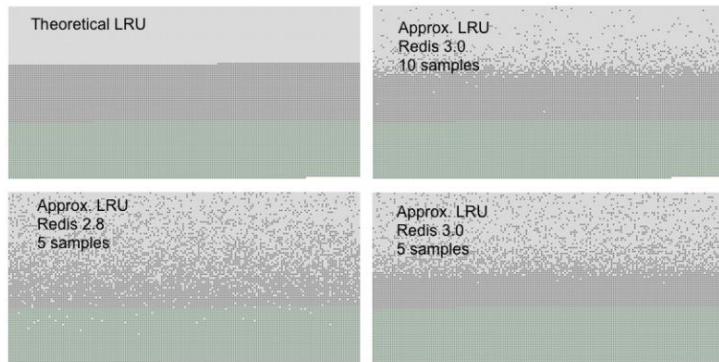
维护一个链表，用于顺序存储被访问过的key。

在访问数据时，最新访问过的key将被移动到表头，
即最近访问的key在表头，最少访问的key在表尾。

- 近似LRU算法 (Redis)

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

优点：比LRU算法节约内存，却可以取得非常近似的效果。



► 5. 常见面试题 / Redis / 缓存穿透

- 场景

查询根本不存在的`数据`，使得请求直达存储层，导致其负载过大，甚至宕机。

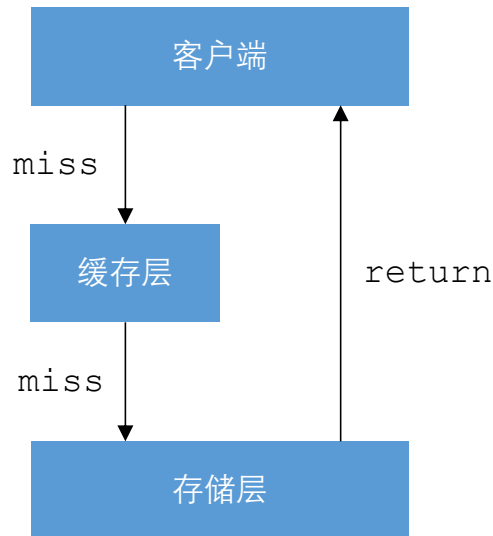
- 解决方案

1. 缓存空对象

存储层未命中后，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。

2. 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。



► 5. 常见面试题 / Redis / 缓存击穿

- 场景

一份热点数据，它的访问量非常大。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，其他线程只能等待。

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”上的不过期。

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。

► 5. 常见面试题 / Redis / 缓存雪崩

- 场景

由于某些原因，缓存层不能提供服务，导致所有的请求直达存储层，造成存储层宕机。

- 解决方案

1. 避免同时过期

设置过期时间时，附加一个随机数，避免大量的key同时过期。

2. 构建高可用的Redis缓存

部署多个Redis实例，个别节点宕机，依然可以保持服务的整体可用。

3. 构建多级缓存

增加本地缓存，在存储层前面多加一级屏障，降低请求直达存储层的几率。

4. 启用限流和降级措施

对存储层增加限流措施，当请求超出限制时，对其提供降级服务。

► 5. 常见面试题 / Redis / 分布式锁

场景

修改时，经常需要先将数据读取到内存，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：在多个线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

1. 基于数据库实现分布式锁；
2. 基于Redis实现分布式锁；
3. 基于Zookeeper实现分布式锁；

► 5. 常见面试题 / Redis / 分布式锁

Redis实现分布式锁的原则

1. 安全属性：独享。在任一时刻，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。

► 5. 常见面试题 / Redis / 分布式锁

单Redis实例实现分布式锁

1. 获取锁使用命令：

```
SET resource_name my_random_value NX PX 30000
```

NX：仅在key不存在时才执行成功。PX：设置锁的自动过期时间。

2. 通过Lua脚本释放锁：

```
if redis.call("get",KEYS[1]) == ARGV[1] then
    return redis.call("del", KEYS[1])
else return 0 end
```

可以避免删除别的客户端获取成功的锁：

A加锁 -> A阻塞 -> 因超时释放锁 -> B加锁 -> A恢复 -> 释放锁

► 5. 常见面试题 / Redis / 分布式锁

多Redis实例实现分布式锁

Redlock算法，该算法有现成的实现，其Java版本的库为Redisson。

1. 获取当前Unix时间，以毫秒为单位。
2. 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，客户端应该尽快尝试另外一个Redis实例。
3. 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。
4. 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。
5. 如果获取锁失败，客户端应该在所有的Redis实例上进行解锁。

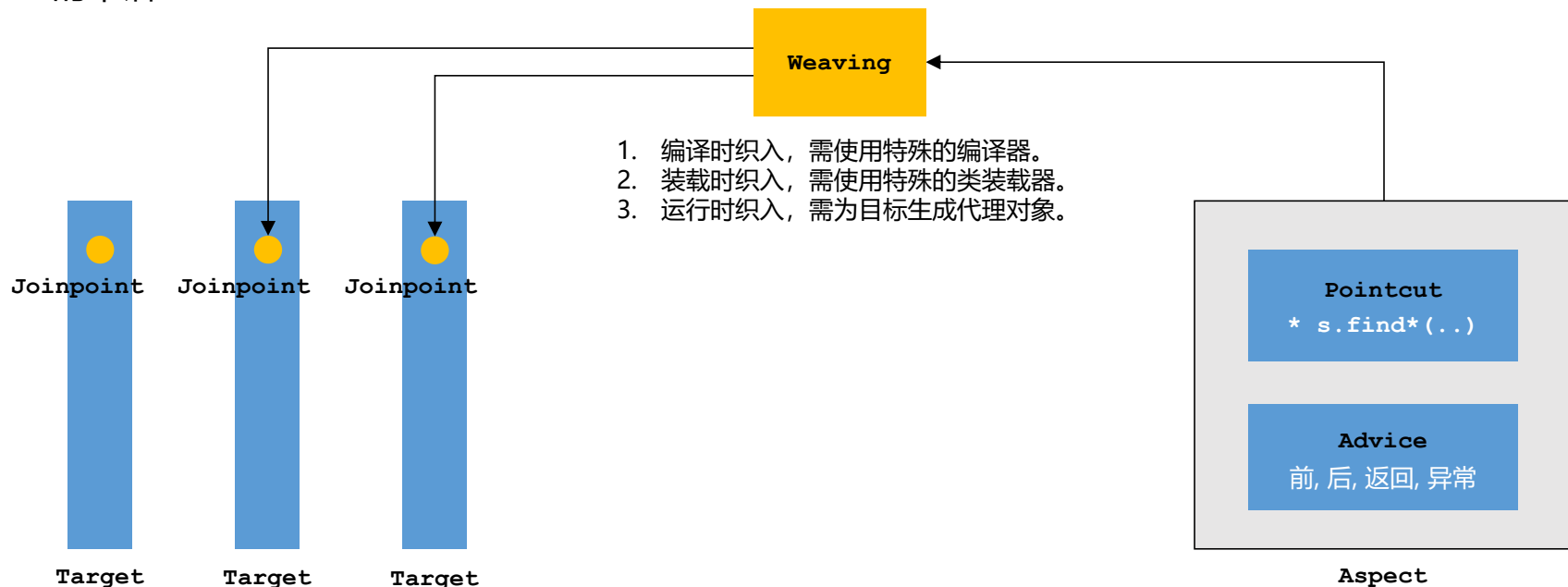
► 5. 常见面试题 / Spring / Spring IoC

Bean的作用域

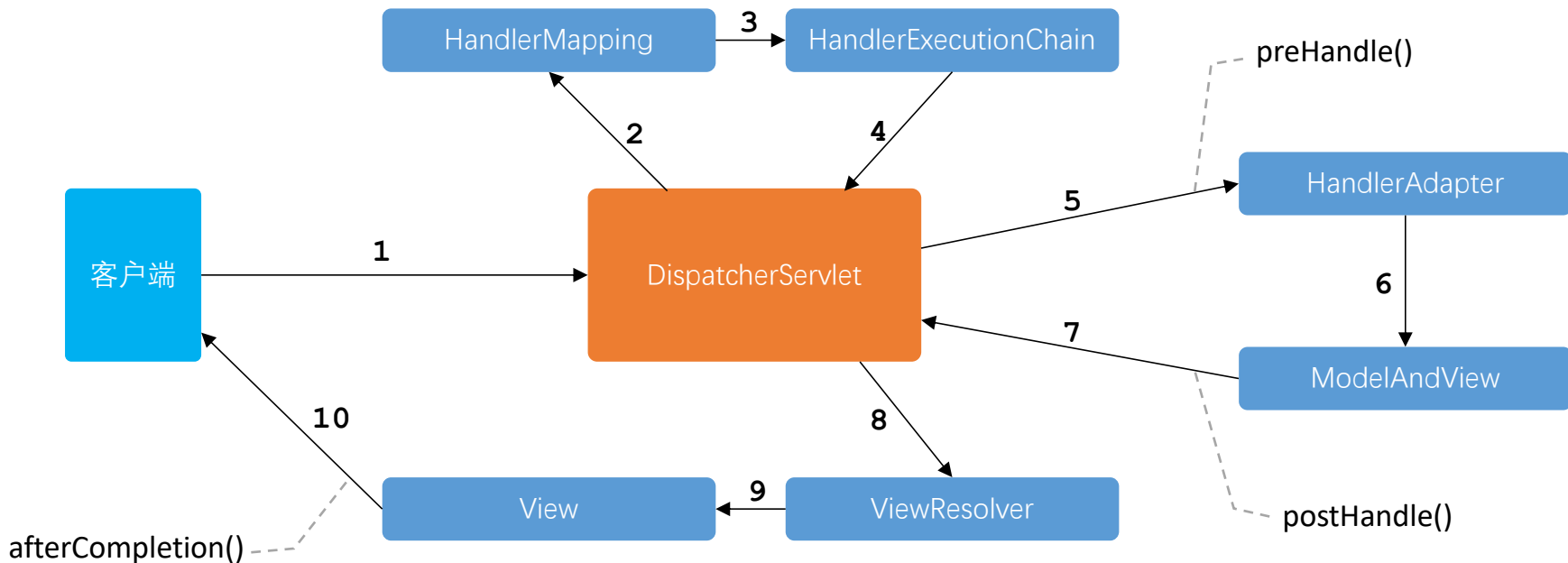
作用域	使用范围	描述
singleton	所有Spring 应用	在容器中只存在一个实例，默认值。
prototype	所有Spring 应用	在容器中存在多个实例，即每次获取该Bean时，都会创建一个新实例。
request	Spring Web 应用	为每个请求创建一个新的实例。
session	Spring Web 应用	为每个会话创建一个新的实例。
globalSession	Spring Web 应用	为全局的session创建一个实例，只在Portlet上下文中有有效。
application	Spring Web 应用	为整个Web应用创建一个新的实例。

► 5. 常见面试题 / Spring / Spring AOP

AOP的术语



► 5. 常见面试题 / Spring / Spring MVC



Thanks

