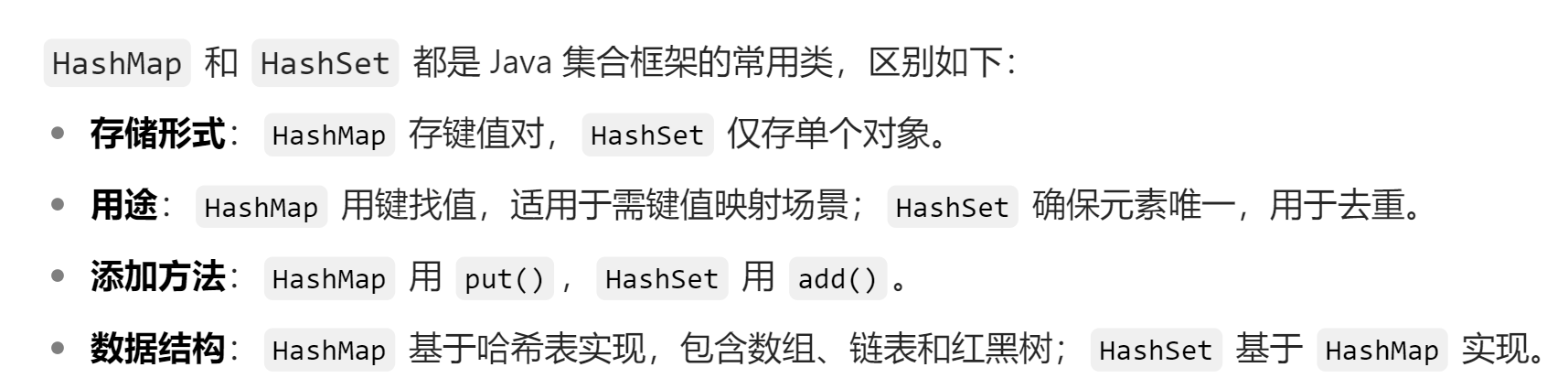
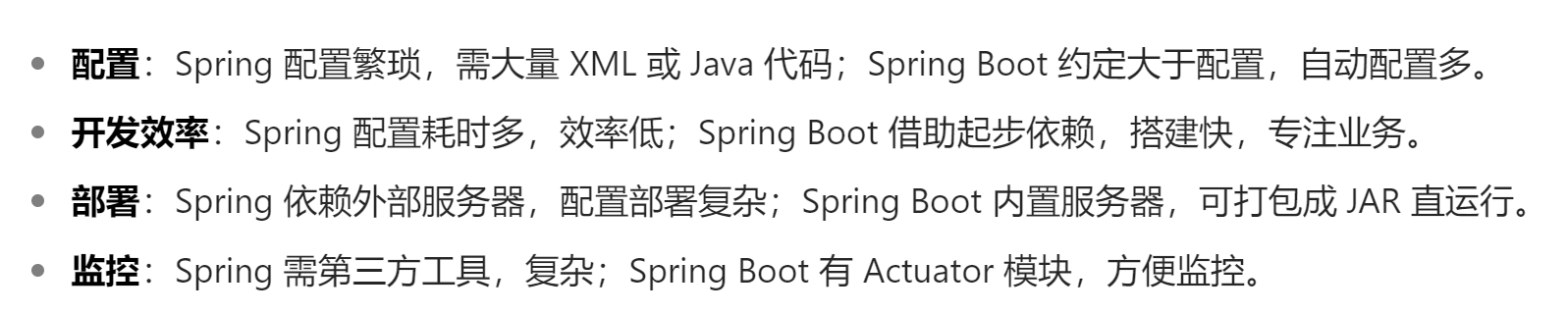
1. hashmap跟hashset的区别



2,spring和springboot的区别



3. spring-boot-starter 是什么 为什么可以做到开箱即用

spring-boot-starter 是 Spring Boot 提供的一种依赖模块，其作用是将开发特定功能所需的一系列依赖整合在一起，就像一个功能完备的 “工具包”，

依赖管理自动化

开发中手动管理多个依赖及其版本易出错。spring-boot-starter 把特定功能依赖打包，添加对应依赖后，Maven 或 Gradle 自动下载相关依赖并确保版本兼容。如添加 spring-boot-starter-web，Spring MVC、嵌入式 Tomcat 等依赖会自动添加。

自动配置机制

Spring Boot 有自动配置特性，引入 spring-boot-starter 后，会依据类路径下的依赖自动配置。如引入 spring-boot-starter-data-jpa，会自动配置数据源、实体管理器等，无需手动编写大量配置代码。

约定大于配置

Spring Boot 遵循此原则，设定默认配置约定，包括项目结构、配置文件位置与命名。默认配置文件为 application.properties 或 application.yml，放于 src/main/resources 目录。多数情况可直接用默认配置，少量修改就能满足个性化需求。

4. yml yaml还有properties的优先级

Properties的优先级最低 yml yaml的优先级一样

5. 二叉树 红黑树 平衡二叉树 二叉排序树

二叉树

定义：是一种每个节点最多有两个子节点的树状数据结构，这两个子节点通常被称作左子节点和右子节点。

特点：结构简单且灵活，应用场景广泛。

用途：常用于表示具有层次关系的数据，像文件系统的目录结构。

二叉排序树

定义：也叫二叉搜索树，对于树中的每个节点，其左子树中所有节点的值都小于该节点的值，而右子树中所有节点的值都大于该节点的值。

特点：中序遍历能得到有序序列，方便进行查找、插入和删除操作。

用途：在需要快速查找、插入和删除数据的场景中应用较多，如数据库索引。

平衡二叉树

定义：属于特殊的二叉排序树，它要求每个节点的左右子树的高度差不超过 1。

特点：通过保持树的平衡，避免树退化为链表，保证了操作的时间复杂度稳定在 。

用途：在对查找效率要求较高的场景中表现出色，例如数据库查询优化。

红黑树

定义：是一种自平衡的二叉排序树，每个节点都带有颜色属性（红色或黑色），通过对节点颜色的约束来维持树的平衡。

特点：相比于普通平衡二叉树，红黑树在插入和删除操作时的旋转操作更少，性能更优。

用途：在 Java 的 TreeMap、TreeSet 以及 Linux 内核的进程调度等场景中广泛应用。

6. mysql四种隔离 级别是怎么实现的

MySQL 四种隔离级别实现依赖锁机制与多版本并发控制（MVCC）：

读未提交：基本不用锁，允许读未提交数据，有脏读问题。

读已提交：用行级锁和 MVCC。读加共享锁，读完即放；写用排他锁。提供数据快照，避免脏读，但有不可重复读问题。

可重复读：MySQL 默认级别，用行级锁、MVCC、间隙锁和临键锁。事务开始分配 ID 获取快照，期间读此快照数据。避免不可重复读，InnoDB 能防幻读。

串行化：最高级别，强制事务串行。读加共享锁，写加排他锁，事务结束才释放，避免所有并发问题，但并发性能差。

7.restful风格

RESTful 是设计网络应用 API 的风格，遵循 REST 原则：

核心：把事物抽象为资源，有唯一 URL，通过 HTTP 方法操作实现状态转移，资源有 JSON 等多种表现形式。

设计准则：URL 用名词；用 GET（获取）、POST（创建）、PUT（更新）、DELETE（删除）操作资源；用 HTTP 状态码表示结果；服务器无状态，请求自带信息；可对不常变资源设缓存。

8.mq中异步操作的好处和坏处

好处

性能响应佳：生产者发消息到 MQ 后无需等待处理结果，可立即继续执行，提升系统响应速度与处理能力，如电商下单场景。

资源利用高：高并发时，生产者快速存消息，消费者按需处理，避免资源闲置，适合 I/O 密集型任务。

扩展性强：分布式系统中，各组件通过 MQ 异步通信，可独立运行扩展，一个服务瓶颈不影响其他。

系统解耦好：生产者和消费者通过 MQ 间接通信，互不依赖，某服务变更或故障不严重影响其他。

坏处

复杂度提升：需处理消息顺序、重复消费、丢失等问题，增加开发和维护难度。

一致性挑战：难以保证数据实时一致，需复杂机制实现最终一致性。

调试困难：异步流程复杂，定位问题和排查故障难度大。

增加延迟：消息在 MQ 中排队、传输会带来额外延迟。

9.雪花算法及其原理

雪花算法（Snowflake）是 Twitter 开源的分布式唯一 ID 生成算法，生成 64 位长整型 ID，在分布式系统应用广泛。

ID 结构

1 位符号位：固定为 0，保证 ID 为正数。

41 位时间戳：记录从纪元起的毫秒数，支持约 69 年，使 ID 趋势递增，利于数据库操作。

5 位数据中心 ID：支持 32 个数据中心，区分不同地理位置或功能的数据中心。

5 位机器 ID：每个数据中心内支持 32 台机器，结合数据中心 ID 唯一标识每台机器。

12 位序列号：同一毫秒内从 0 递增，最多生成 4096 个 ID，超量则等下一毫秒。

工作原理

确定纪元：设定起始时间，时间戳记录从纪元到当前的毫秒数。

分配 ID：为每台机器配置唯一的数据中心 ID 和机器 ID，运行中一般不变。

生成序列号：同一毫秒内序列号递增，超 4096 则阻塞线程等下一毫秒。

组合 ID：按规则将各部分组合成 64 位长整型的唯一 ID。

雪花算法能在分布式系统高效生成全局唯一且递增的 ID，常用于分布式数据库等场景。

10.雪花算法跟UUID的区别

雪花算法和 UUID 都用于生成唯一标识，二者区别如下：

生成方式

雪花算法：结合时间戳、数据中心 ID、机器 ID 和序列号生成 64 位长整型。

UUID：有多种算法，如基于时间、随机数等，常见为 36 字符字符串。

ID 特性

雪花算法：趋势递增，利于数据库索引排序。

UUID：大多无序，用于数据库可能导致索引碎片化。

性能

雪花算法：生成快、资源消耗低。

UUID：部分版本生成慢、资源消耗高。

适用场景

雪花算法：适用于分布式系统生成订单号、消息 ID 等。

UUID：适用于对唯一性要求高、无需排序的场景，如文件存储。