# Spring/Spring MVC

## 为什么要使用spring？

* spring提供ioc技术，容器来管理依赖的对象，不需要自己创建和管理依赖对象了，更轻松的实现了程序的解耦。
* spring 提供了事务支持，使得事务操作变的更加方便。
* spring 提供了AOP面向切面编程，这样可以更方便的处理某一类的问题。
* 更方便的框架集成，spring可以很方便的集成其他框架，比如MyBatis、hibernate等。

## 解释一下什么是aop？

* aop是面向切面编程，属于对OOP的扩展。其实现是基于动态代理设计模式，在IoC基础上实现的，

可以实现在不修改原有代码的情况下，做额外扩展。

* 简单来说就是统一处理某一“切面”（类）的问题的编程思想，比如统一处理日志、异常等。

## 解释一下什么是ioc？

ioc：Inversion of Control（中文：控制反转）是spring的核心，对于spring框架来说，就是由spring来负责控制对象的生命周期和对象间的关系。

简单来说，控制指的是当前对象对内部成员的控制权；控制反转指的是，这种控制权不由当前对象管理了，由其他（类，第三方容器）来管理。

## spring有哪些主要模块？

* core：提供了框架的基本组成部分包括控制反转和依赖注入功能。
* context：Spring上下文是一个配置文件，主要向框架提供上下文信息。。
* dao：提供了单独的支持JDBC操作的API。
* aop：提供了面向切面的编程实现，让你可以自定义拦截器、切点等。
* Web：提供了针对Web开发的集成特性 。
* **WebMvc**：基于 MVC 设计模式的一个轻量级 Web 开发框架。

## spring常用的注入方式有哪些？

* setter属性注入
* 构造方法注入
* 注解方式注入

## spring中的bean是线程安全的吗？

Spring中的Bean是否线程安全，跟Spring容器无关，只是交由Spring容器托管而已。

那么，在Spring容器中，什么样的Bean会存在线程安全问题呢？回答，这个问题之前我们得先回顾一下Spring Bean的作用域。在Spring定义的作用域中，其中有 prototype（ 多例Bean ）和 singleton （ 单例Bean）。那么，定义为 prototype 的Bean，是在每次getBean 的时候都会创建一个新的对象。定义为 singleton 的Bean，在Spring容器中只会存在一个全局共享的实例。

我们已经知道，多例Bean每次都会新创建新实例，也就是说线程之间不存在Bean共享的问题。因此，多例Bean是不存在线程安全问题的。

而单例Bean是所有线程共享一个实例，因此，就可能会存在线程安全问题。但是单例Bean又分为无状态Bean和有状态Bean。在多线程操作中只会对Bean的成员变量进行查询操作，不会修改成员变量的值，这样的Bean称之为无状态Bean。所以，可想而知，无状态的单例Bean是不存在线程安全问题的。但是，在多线程操作中如果需要对Bean中的成员变量进行数据更新操作，这样的Bean称之为有状态Bean，所以，有状态的单例Bean就可能存在线程安全问题。

所以，最终我们得出结论，在Spring中，只有有状态的单例Bean才会存在线程安全问题。

## spring支持几种bean的作用域？

spring支持5种作用域，如下：

* singleton：单例模式，spring ioc容器中只存在一个bean实例，bean以单例模式存在，是系统默认值；
* prototype：原型模式或叫多例模式，每次从容器调用bean时都会创建一个新的实例，既每次getBean()相当于执行new Bean()操作；

Web 环境下的作用域：

* request：每次http请求都会创建一个bean；
* session：同一个http session共享一个 bean 实例；
* global-session：全局作用域，用于portlet容器，因为每个portlet有单独的 session，globalsession提供一个全局性的 http session。

**注意：**使用prototype作用域需要慎重的思考，因为频繁创建和销毁bean会带来很大的性能开销。

## spring自动装配bean有哪些方式？

* no：默认值，表示没有自动装配，应使用显式bean引用进行装配。
* byName：它根据bean的名称注入对象依赖项。
* byType：它根据类型注入对象依赖项。
* 构造函数：通过构造函数来注入依赖项，需要设置大量的参数。
* autodetect：容器首先通过构造函数使用autowire装配，如果不能，则通过byType自动装配。

## spring事务实现方式有哪

* **声明式事务：**声明式事务也有两种实现方式，基于xml配置文件的方式和注解方式（在类上添加@Transaction注解）。
* **编码方式：**提供编码的形式管理和维护事务。

## 说一下spring的事务隔离？

多个事务同时操作数据库时，允许多个事务操作的方式就是事务隔离级别。事务隔离级别主要是通过添加锁操作实现的。事务隔离级别主要是解决高并发下脏读、幻读、不可重复读问题的。

isolation: 设置隔离级别 默认为数据库的级别

级别越低 并发性能越高 安全性差 级别越高 并发性能越低 安全性高

READ\_UNCOMMITTED 读未提交 可能会产生脏读、不可重复读、幻读。（最低级别）

READ\_COMMITTED 读已提交 可能会产生不可重复读、幻读（Oracle 默认级别）

REPEATABLE\_READ （锁行） 可重复读 可能会产生幻读（MYSQL默认级别）

SERIALIZABLE（锁表） 串行化 多个并行事务串行化执行，不会产生安全性问题（最高级别）

* **脏读：**

事务A没有提交事务，事务B读取到事务A未提交的数据，这个过程称为脏读。读取到的数据叫做脏数据。

脏读发生在一个事务读取了被另一个事务改写但尚未提交的数据时。如果这些改变在稍后被回滚了，那么第一个事务读取的数据就会是无效的。

* **不可重复读：**

当事务A读取到表中一行数据时，同时另一个事务修改这行数据，事务A读取到的数据和表中真实数据不一致。

不可重复读发生在一个事务执行相同的查询两次或两次以上，但每次查询结果都不相同时。这通常是由于另一个并发事务在两次查询之间更新了数据。

* **幻读：**

幻读是事务非独立执行时发生的一种现象。例如事务T1对一个表中所有的行的某个数据项做了从“1”修改为“2”的操作，这时事务T2又对这个表中插入了一行数据项，而这个数据项的数值还是为“1”并且提交给数据库。而操作事务T1的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务T2中添加的，就好像产生幻觉一样，这就是发生了幻读。

## 说一下spring mvc运行流程？

* spring mvc先将请求发送给DispatcherServlet。
* DispatcherServlet查询一个或多个 HandlerMapping，找到处理请求的Controller。
* DispatcherServlet再把请求提交到对应的Controller。
* Controller 进行业务逻辑处理后，会返回一个 ModelAndView。
* Dispathcher查询ViewResolver视图解析器，找到ModelAndView对象指定的视图对象。
* 视图对象负责渲染返回给客户端视图。

## spring mvc 有哪些组件？

* 前置控制器DispatcherServlet。
* 映射控制器HandlerMapping。
* 处理器Controller。
* 模型和视图ModelAndView。
* 视图解析器ViewResolver。

## @RequestMapping 的作用是什么？

将http请求映射到相应的类/方法上。

## @Autowired 的作用是什么？

@Autowired是由Spring提供的，它可以用来对构造方法、成员变量以及方法参数上进行标注，能够根据对象类型完成自动注入

# Spring Boot

## 什么是 spring boot？

spring boot是为spring服务的，是用来简化新spring应用的初始搭建以及开发过程的。

## 为什么要用 spring boot？

* 配置简单
* 独立运行
* 自动装配
* 无代码生成和xml配置
* 提供应用监控
* 易上手
* 提升开发效率

## spring boot 核心配置文件是什么？

spring boot 核心的两个配置文件：

* bootstrap(.yml 或者.properties)：boostrap由父ApplicationContext加载的，比applicaton优先加载，且boostrap里面的属性不能被覆盖； 1
  + `application (.yml 或者.properties)：用于spring boot项目的自动化配置。

## .spring boot 配置文件有哪几种类型？它们有什么区别？

配置文件有：

.properties格式和.yml格式，它们主要的区别是书法风格不同。

.properties配置如下：

spring.RabbitMQ.port=5672

.yml配置如下：

spring:

RabbitMQ:

port:

5672

## spring boot 有哪些方式可以实现热部署？

使用devtools启动热部署，添加devtools库，勾上自动编译。

## 什么是 ORM 框架？

ORM（Object Relation Mapping）对象关系映射，是把数据库中的关系数据映射成为程序中的对象。

使用ORM 的优点：提高了开发效率降低了开发成本、开发更简单更对象化、可移植更强。

# MyBatis

## MyBatis 中 #{}和 ${}的区别是什么？

#{}是预编译处理，${}是字符替换。

#号，等同于JDBC里面的?号（占位符），可以防止SQL注入。

$号，相当于直接把参数拼接到了原始的SQL里面，MyBatis不会对它进行特殊处理。

## MyBatis 有几种分页方式？

分页方式：逻辑分页和物理分页。

* 逻辑分页：使用 MyBatis 自带的RowBounds进行分页，它是一次性查询很多数据，然后在数据中再进行检索。
* 物理分页：自己手写SQL分页或使用分页插件PageHelper，去数据库查询指定条数的分页数据的形式。

## MyBatis 逻辑分页和物理分页的区别是什么？

* 逻辑分页是一次性查询很多数据，然后再在结果中检索分页的数据。这样做弊端是需要消耗大量的内存、有内存溢出的风险、对数据库压力较大。
* 物理分页是从数据库查询指定条数的数据，弥补了一次性全部查出的所有数据的种种缺点，比如需要大量的内存，对数据库查询压力较大等问题。

## MyBatis 是否支持延迟加载？延迟加载的原理是什么？

MyBatis支持延迟加载，设置lazyLoadingEnabled=true即可。延迟加载的原理的是调用的时候触发加载，而不是在初始化的时候就加载信息。比如调用a.getB().getName()，这个时候发现a.getB()的值为null，此时会单独触发事先保存好的关联B对象的SQL，先查询出来B，然后再调用a.setB(b)，而这时候再调用a.getB().getName() 就有值了，这就是延迟加载的基本原理。

## 说一下 MyBatis 的一级缓存和二级缓存？

Mybatis里面设计了两级缓存来提升数据的检索效率，避免每次数据的访问都需要去查询数据库。

* 一级缓存：它是SqlSession级别的缓存，也叫本地缓存，因为每个用户在执行查询的时候都需要使用SqlSession来执行，为了避免每次都去查数据库，MyBatis把查询出来的数据保存到SqlSession的本地缓存中，后续的SQL如果命中缓存，就可以直接从本地缓存读取。
* 二级缓存：当多个用户在查询数据的时候，只要有任何一个SqlSession拿到了数据就会放入到二级缓存里面，其他的SqlSession就可以从二级缓存加载数据。

开启二级缓存数据查询流程：二级缓存->一级缓存->数据库。

缓存更新机制：当某一个作用域(一级缓存Session/二级缓存Mapper)进行了C/U/D操作后，默认该作

用域下所有select中的缓存将被clear。

## MyBatis 有哪些执行器（Executor）？

MyBatis有三种基本的Executor执行器：

* SimpleExecutor：每执行一次update或select就开启一个Statement对象，用完立刻关闭Statement对象；
* ReuseExecutor：执行update或select，以SQL作为key查找Statement对象，存在就使用，不存在就创建，用完后不关闭Statement对象，而是放置于Map内供下一次使用。简言之，就是重复使用Statement对象；
* BatchExecutor：执行update（没有 select，jdbc 批处理不支持 select），将所有SQL都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个Statement对象，每个Statement对象都是 addBatch()完毕后，等待逐一执行 executeBatch()批处理，与jdbc批处理相同。

## MyBatis 分页插件的实现原理是什么？

分页插件的基本原理是使用MyBatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的SQL，然后重写SQL，根据dialect方言，添加对应的物理分页语句和物理分页参数。

# MySQL

## 数据库的三范式是什么？

1NF：原子性，每个属性不可再拆分。

2NF：唯一性，数据库表中的每个记录必须可以被唯一地区分，即表中需要有主键

3NF：一张表中不能出现其他表的非主键字段。

## 一张自增表里面总共有 7 条数据，删除了最后 2 条数据，重启 MySQL 数据库，又插入了一条数据，此时 id 是几？

表类型如果是MyISAM，那id就是8。

表类型如果是InnoDB，那id就是6。

InnoDB表只会把自增主键的最大 id 记录在内存中，所以重启之后会导致最大id丢失。

## 说一下 ACID 是什么？

* 原子性：事务是一个整体，事务中的操作要不全都完成，要不全都失败。
* 一致性：事务开始前和事务结束后，数据库的完整性没有被破坏。
* 隔离性：数据库允许多个并发事务同时对其数据进行读写和修改。
* 持久性：事务结束后，对数据的修改就是永久的。

## char 和 varchar 的区别是什么？

char：固定长度类型

varchar：可变长度类型

## float 和 double 的区别是什么？

float 最多可以存储8位的十进制数，并在内存中占4字节。

double 最多可以存储16位的十进制数，并在内存中占8字节。

## MySQL 的内连接、左连接、右连接有什么区别？

内连接关键字：inner join；

左连接关键字：left join；

右连接关键字：right join。

内连接是把匹配的关联数据显示出来；左连接是左边的主表全部显示出来，右边的表显示 出符合条件的数据；右连接正好相反。

## MySQL 索引是怎么实现的？

索引是一种特殊的数据结构，用于快速查找数据。

MySQL中的索引实现方式是使用B+树结构来组织索引

## 说一下 MySQL 常用的引擎？

**InnoDB引擎：是事务型数据库的首选引擎，支持事务ACID**

**MyISAM引擎：存储引擎不支持事务，所以对事务有要求的业务场景不能使用**

## 说一下 MySQL 的行锁和表锁？

MyISAM 只支持表锁，InnoDB 支持表锁和行锁，默认为行锁。

表级锁：每次操作锁住整张表。锁定粒度大，发生锁冲突的概率最高，并发度最低。

行级锁：每次操作锁住对应的行数据。锁定粒度最小，发生锁冲突的概率最纸，并发度最高。

## 说一下乐观锁和悲观锁？

**乐观锁：**每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在提交更新的时候会判断一下在此期间别人有没有去更新这个数据。

**悲观锁：**每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻止，直到这个锁被释放。

数据库的乐观锁需要自己实现，在表里面添加一个version字段，每次修改成功值加1，这样每次修改的时候先对比一下，自己拥有的version和数据库现在的version是否一致，如果不一致就不修改，这样就实现了乐观锁。

## 如何做 MySQL 的性能优化？

* 为搜索字段创建索引。
* 避免使用select \*，列出需要查询的字段。
* 引入分库分表机制
* 针对热点数据，可以引入更为高效的分布式数据库，比如Redis

# Redis

## Redis 是什么？都有哪些使用场景？

Redis是一个开源的、高性能的、键值对内存数据库。

Redis 使用场景：

缓存、分布式锁、排行榜、消息队列、自增id

## Redis 有哪些功能？

* 数据缓存功能
* 分布式锁的功能
* 支持数据持久化
* 支持事务
* 支持消息队列

## Redis 为什么是单线程的？

因为cpu不是 Redis的瓶颈，Redis的瓶颈最有可能是机器内存或者网络带宽。既然单线程容易实现，而且cpu又不会成为瓶颈，那就顺理成章地采用单线程的方案了。

关于Redis的性能，官方网站也有，普通笔记本轻松处理每秒几十万的请求。 而且单线程并不代表就慢 nginx 和nodejs也都是高性能单线程的代表。

## 什么是缓存穿透？怎么解决？

**缓存穿透：**指查询一个一定不存在的数据，由于缓存是不命中时需要从数据库查询，查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，造成缓存穿透。

**解决方案：**最简单粗暴的方法如果一个查询返回的数据为空，我们就把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

## Redis 支持的数据类型有哪些？

Redis支持的数据类型：string（字符串）、list（列表）、hash（哈希）、set（集 合）、 zset（有序集合）。

## Redis 支持的 Java 客户端都有哪些？

支持的Java客户端有Redisson、jedis、lettuce 等。 官方推荐Redisson

## 怎么保证缓存和数据库数据的一致性？

合理设置缓存的过期时间。

新增、更改、删除数据库操作时同步更新 Redis，可以使用事物机制来保证 数据的一致性。

## Redis 持久化有几种方式？

Redis 的持久化有两种方式，或者说有两种策略：

* RDB（Redis Database）：指定的时间间隔对数据进行快照存储。
* AOF（Append Only File）：每一个写命令都通过write函数追加到文件中。

## Redis 淘汰策略有哪些？

volatile-lru：从已设置过期时间的数据中挑选最近最少使用的数据淘汰。

volatile-ttl：从已设置过期时间的数据集中挑选将要过期的数据淘汰。

volatile-random：从已设置过期时间的数据集中任意选择数据淘汰。

allkeys-lru：从数据集中挑选最近最少使用的数据淘汰。

allkeys-random：从数据集中任意选择数据淘汰。

no-enviction（驱逐）：禁止驱逐数据。

## Redis 常见的性能问题有哪些？该如何解决？

主服务器写内存快照，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以主服务器最好不要写内存快照。

Redis主从复制的性能问题，为了主从复制的速度和连接的稳定性，主从库最好在同一个局域网内。