**实现GoodsVo中ProductVo的嵌套校验：**

**1.如何启用GoodsVo中ProductVo的校验注解：**

为字段开启校验：

@Valid

@ApiModelProperty(value = "商品规格")

private List<ProductVo> productList;

**2.可控的合法性检查**：

1.添加分组接口类

2.启用注解处增加分组标记

3.注解处增加分组标记



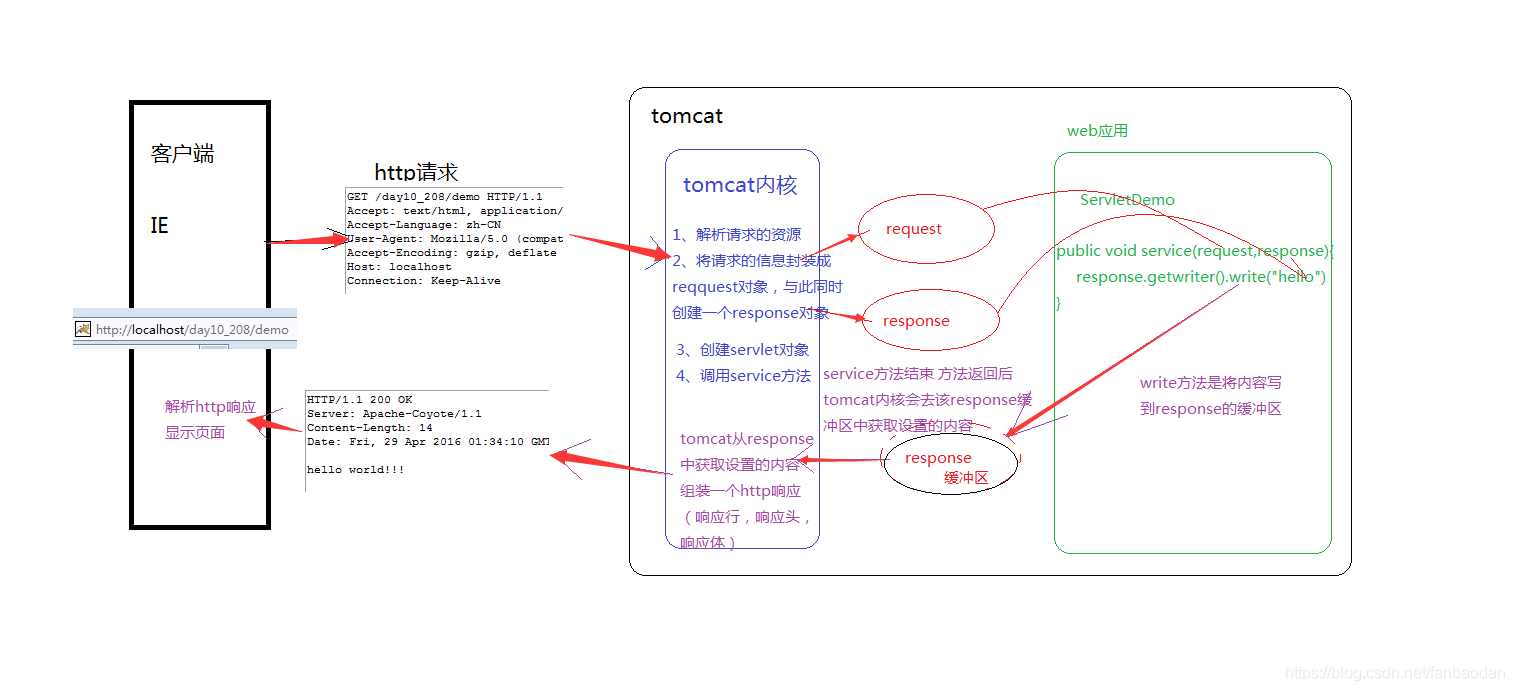
**ControllerExceptionHandler中注入的httpServletResponse**

Controller的作用域是单例的，但是其成员变量httpServletResponse的作用域却是原型/请求的

HttpServletResponse从何而来：从RequestContextHolder的ThreadLocal<RequestAttributes>

HttpServletResponse何时更新：在接收到HTTP请求后更新

ThreadLocal提供了线程内存储变量的能力，这些变量不同之处在于每一个线程读取的变量是对应的互相独立的。通过get和set方法就可以得到当前线程对应的值。



## Singleton类型的Spring容器Bean对象是如何实现并发的

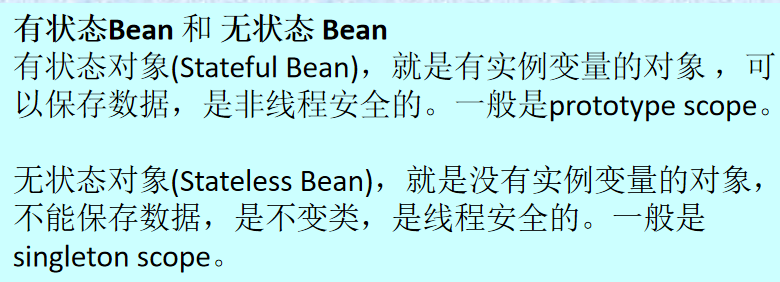
线程安全问题都是由成员变量及静态变量引起的。

若每个线程中对成员变量、静态变量只有读操作，而无写操作，一般来说，这个成员变量、静态变量是线程安全的；

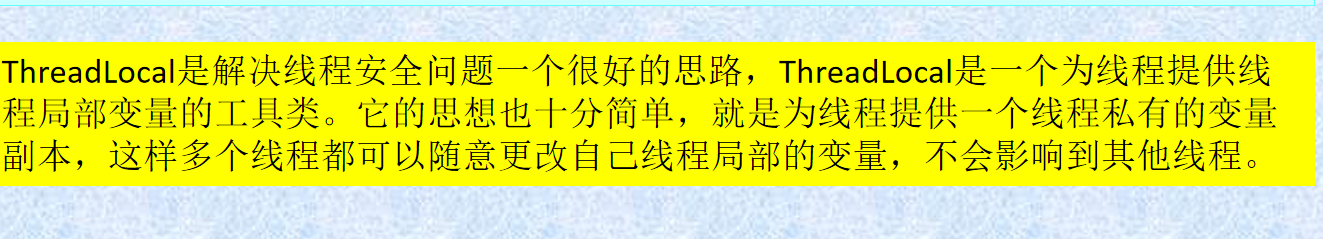
若有多个线程同时执行写操作，一般都需要考虑线程同步，否则就可能影响线程安全。

Spring 下并发访问的线程安全性问题 ：



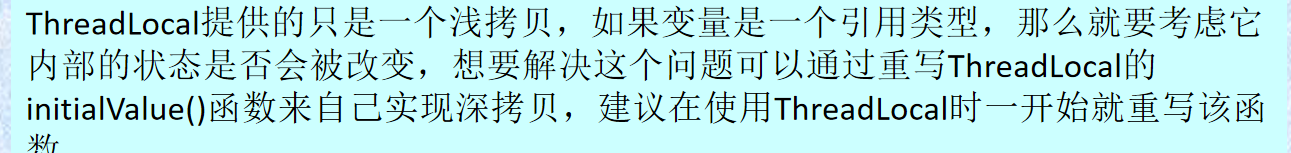


就是因为Spring对一些Bean（如RequestContextHolder、TransactionSynchronizationManager、LocaleContextHolder等）中非线程安全状态采用**ThreadLocal**进行处理，让它们也成为线程安全的状态，因此有状态的Bean就可以在多线程中共享了。



ThreadLocal为解决多线程程序的并发问题提供了一种新的思路

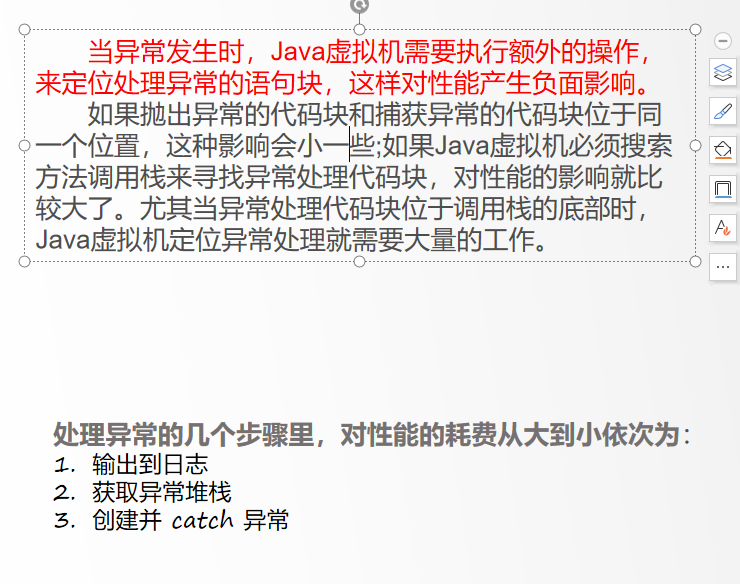
对于多线程资源共享的问题，**同步机制**采用了“**以时间换空间**”的方式，而**ThreadLocal**采用了“**以空间换时间**”的方式。前者仅提供一份变量，让不同的线程排队访问，而后者为每一个线程都提供了一份变量，因此可以同时访问而互不影响。



## 低效的异常



Java中异常为什么低效



因此我们可以关闭追溯运行堆栈信息。关闭追溯运行堆栈信息有两种方法：

1. 覆写fillInStackTrace 方法

2. 在构造方法中关闭

## 哪个更快

可管理性比较

经过分析，我们认为方案2即在dao层完成对象关联的方案，可管理性要大大优于方案1.

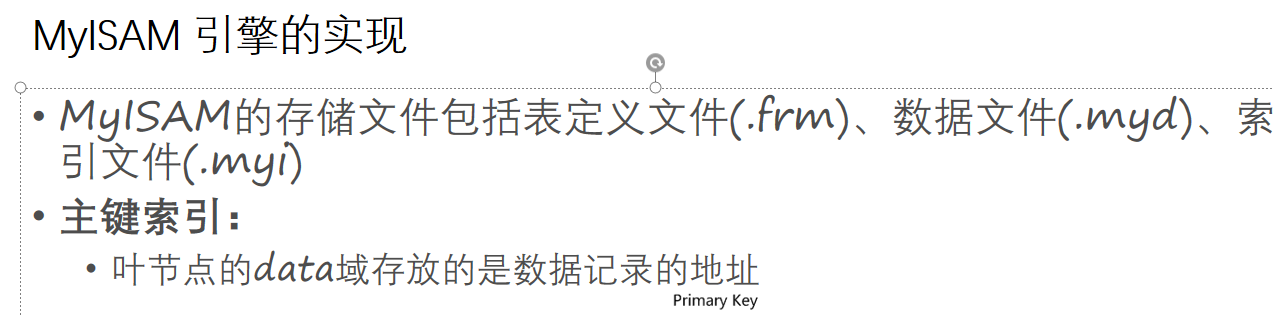
为什么呢？

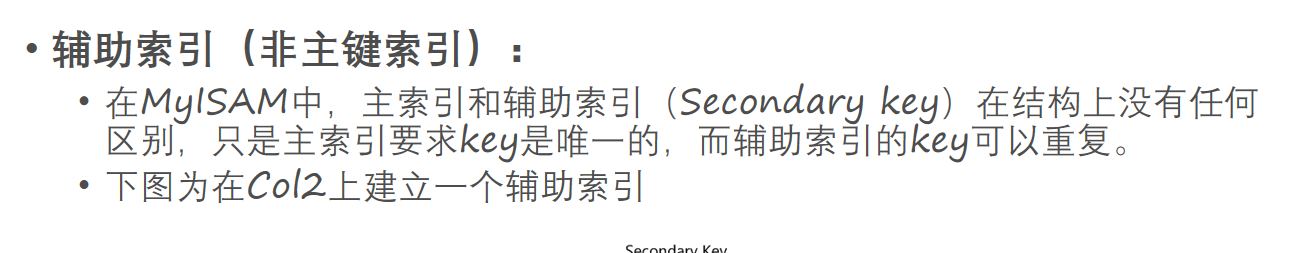
理由1：单机数据库计算资源很贵，数据库同时要服务写和读，都需要消耗CPU，为了能让数据库的吞吐变得更高，而我们的业务又不在乎那几百微妙到毫秒级的延时差距，业务会把更多计算放到dao层或者service层做，毕竟计算资源很好水平扩展，而数据库很难啊，所以大多数业务会把纯计算操作放到dao层做，而将数据库当成一种带事务能力的kv系统（key-value以键值对存储数据的一种数据库）来使用，这是一种重业务，轻DB的架构思路。

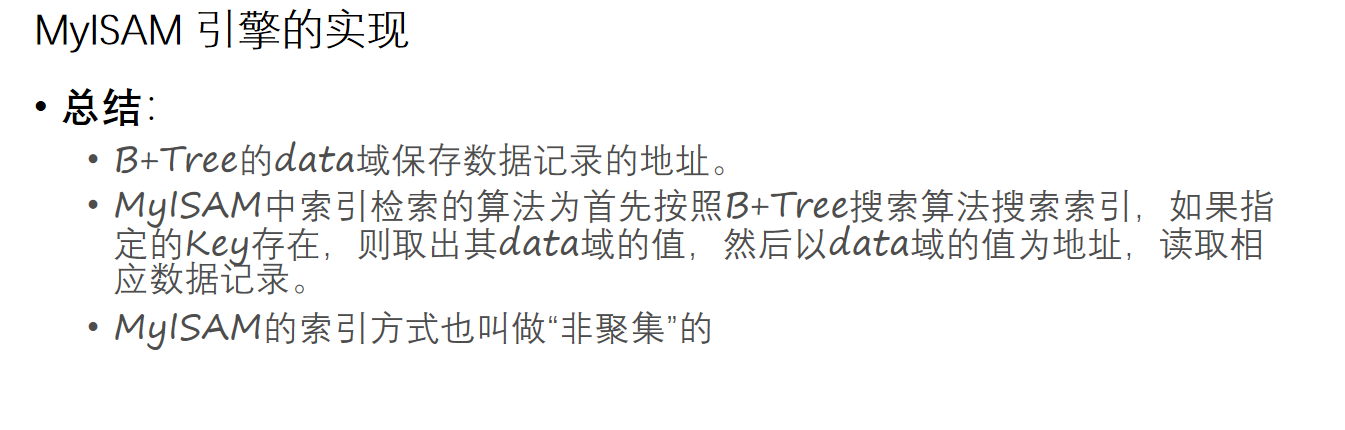
理由2：很多复杂的业务可能会由于发展的历史原因，一般不会只用一种数据库，一般会在多个数据库上加一层中间件，这样的情况下做join显得很困难。

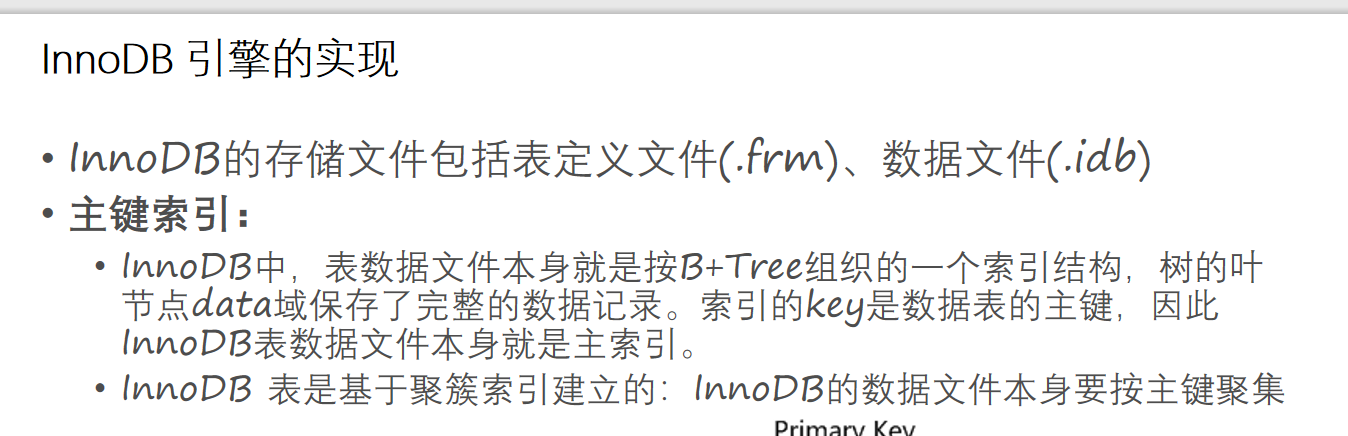
理由3：对于一些大型公司由于数据规模庞大，不得不对数据库进行分库分表。对于分库分表的应用，使用join也受到了很多限制，除非业务能够很好的根据sharding key明确要join的两个表在同一个物理库中。

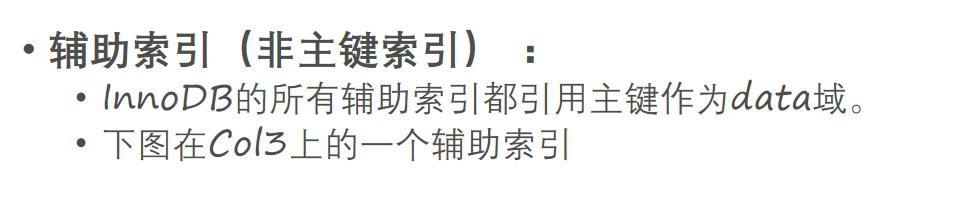
## MySQL的索引

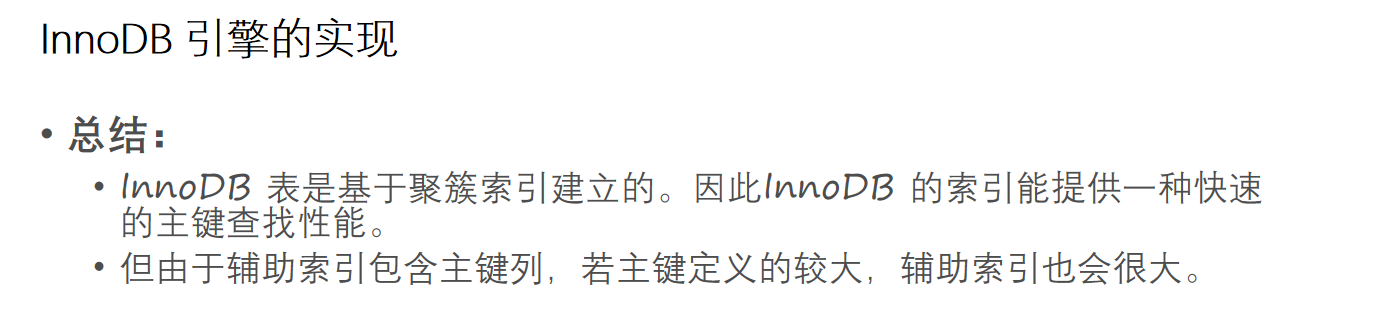














## MySQL的MVCC多版本并发机制

MVCC在MYSQL InnoDB中的实现主要是为了提高数据库并发性能，用更好的方式去处理读-写冲突，做到即使有读写冲突时，也能做到不加锁，非阻塞并发读