### Java 8

#### Lambda表达式简介。

要理解Lambda表达式首先必须明白行为参数化的意义；在软件工程中，用户的需求随时会改变，行为参数化可以帮助您应对不断变化的需求。Lambda表达式就是行为参数化概念的一种解决方案。

#### 什么是函数式接口。

只定义了一个抽象方法的接口，称作函数式接口，函数式接口可以描述Lambda表达式的签名；常用的函数式接口有Predicate — 定义了一个test方法、Consumer — 接收一个泛型参数没有任何返回值、

Funtion — 接收一个泛型T对象，返回一个泛型R对象。

#### 什么方法引用。

方法引用可以看做是Lambda表达式的一种简写方式。

#### 什么是默认方法。

接口中使用关键字 default 修饰的方法，称作默认方法。我们知道向已经发布的接口添加一个方法是非常麻烦的事情，因为所有的子类都必须实现这个方法；有了默认方法之后，我们在已经发布的接口中添加默认方法，则之前改接口的子类不会受到任何影响。

#### 流是什么。

通过声明的方式来处理集合，我们可以把它看做是遍历数据集的高级迭代器，此外流是可以并行的处理的。

#### 流和集合的区别。

集合和流之间的差异就在于什么时候进行计算，集合中的元素必须先计算出来才能添加到集合中。与之相比，流则是概念上的数据结构(不能添加或者删除)，其元素是按需计算的，只有在需要的时候才会计算值。

打个比方吧，流好比就是DVD里面的电影；而流就好比在互联网上观看电影，只需要提前下载用户观看位置的几帧就可以了。

流只能遍历一次，如果重新遍历一次，则抛出流已经被操作或者关闭的异常。

Java流采用了内部迭代的方式(集合采用的是外部迭代的方式)，内部迭代的优点在于可以在内部做很多优化处理。

#### 流提供了那些功能。

流提供了诸如筛选、匹配、映射、分组、分区等等功能。

#### 构建流的方式。

1. 通过值创建流。
2. 通过数组和集合创建流。
3. 通过文件创建流。

### MyBatis

### Spring

#### Spring 概念。

1. 尽可能的降低侵入：程序不依赖Spring API，完全通过XML配置、注解开发；
2. 切面编程：通过切面编程消除了很多重复的代码；
3. 模板：通过模板整合各个技术框架；如比如支持 Hibernat的 HibernateTemplate 、支持 MyBatis的SqlSessionTemplate 、支持 Redis的 RedisTemplate 等。

#### Spring IOC简介。

首先我们应该搞清楚一个问题，什么是依赖注入？ 通过描述(XML配置文件)，由第三方创建对象的方式就是Spring IOC(依赖注入)；Spring 认为一切JAVA资源都是JAVA BEAN，都可以通过Spring IOC容器来进行控制。

#### Spring IOC容器的大致设计。

Spring IOC容器的设计主要是基于 BeanFactory、ApplicationContext 两个接口，其中ApplicationContext是BeanFactory 子接口之一，换句话说 BeanFactory 是Spring IoC 容器所定义的最底层接口，而 ApplicationContext 是其高级接口之一，并且对 BeanFactory 功能做了许多有用的扩展，所以在绝大部分的工作场景下 都会使用 ApplicationContext 作为Spring IoC 容器。

#### Spring IOC容器的实现过程。

1. 资源定位：通过XML配置文件，定位类的位置；
2. BeanDefinition的载入：生成实例应用的过程；
3. BeanDefinition的注册：把第二步骤生成的Bean注册到Sping IOC容器中。

需要注意的是Sping Bean的初始化有两种方式。一种是懒加载方式，这种方式在第一次getBean的时候进行初始化；第二种方式是在创建bean的时候初始化；默认为第二种方式。

#### Spring Bean的 3中依赖注入的方式。

1. 构造器注入
2. Setter注入
3. 接口注入：有些时候资源并非来自于系统本身，而是来自于外界；比如完全可以将数据库连接配置到tomcat中，在程序中采用接口注入的方式获取数据库连接。

#### 使用注解装配bean。

基于“约定优先于配置”的开发原则，我们大多数时候推荐使用注解的方式来装配bean,下面是对Spring中一些常见注解的介绍：

1. ＠Component：装配bean;
2. ＠Autowired: 自动装配，次注解是按照类型的方式去寻找Spring IOC容器中的bean,所以当一个接口存在多个实现类的时候，它无法成功装配。
3. @Primary: 注解＠Primary代表首要的，当 Spring IoC 通过一个接口或者抽象类注入对象的时候，由于存在多个实现类或者具体类，就会犯糊涂，不知道采用哪个类注入为好。注解＠Primary则是告诉 Spring IoC 容器，请优先使用该类注入。
4. ＠Qualifier：按照名称去寻找Spring IOC容器中的bean。
5. ＠Bean：@Bean可以注解在方法上面，表示将该方法返回的类注册到Spring IOC容器中。
6. ＠ PropertySource: 装载配置文件。

#### bean的作用域。

1. 单例(singleton)：它是默认选项，在整个应用中，Spring只会为其生成一个bean的实例。
2. 原型(prototype)：每次注入，Spring都会为其创建一个新的实例。
3. 会话(session)：在Web应用中，在会话过程中只为它创建一个实例。
4. 请求(request)：在Web应用中，在一次请求中只为它创建一个实例。

#### Spring EL的简易使用。

Spring还提供了更加灵活的注入方式，那就是Spring EL表达式；Spring El表达式使用 “#”获取值，它还能够进行简单的运算。

#### Spring AOP实现原理。

Sping通过动态代理来实现AOP，Spring提供了两种动态代理方式，一种是JDK动态代理(必须使用接口)，二是cglib动态代理(可以不需要接口)。

#### Aop术语概念和约定流程。

1. 切面：就是在一个什么样的环境中工作，如数据库事务就是一个切面；我们可以把它理解成一个拦截器。
2. 通知(Adice):切面的方法，包括前置通知、后置通知、返回通知、异常通知、环绕通知。
3. 引入(Introduction):允许我们在现有的类中添加自定义的方法。
4. 切点(Pointcut）：在动态代理中，被切面拦截的方法就是一个切点。
5. 连接点：连接点是一个判断条件，由它指定哪些是切点。

#### 如何使用Spring进行AOP开发。

首先我们需要知道的是Spring是方法级别的AOP框架，而我们主要也是以某个类的某个方法来作为切点。

1. 创建切面：Spring使用@Aspect 来创建切面。
2. 连接点：Spring 使用 execution 表达式来判断连接点。

#### 多个切面的执行顺序。

Spring 可以通过＠Order注解来控制多个切面的执行顺序，如果没有@Order注解，则多个切面的情况下，其执行顺序是随机的。

#### 传统JDBC的弊端和Sping的解决方案。

1. 过程复杂，需要建立连接、组装SQL、关闭连接资源等等步骤；
2. Try.. catch .. finally 语句的频繁使用，造成代码泛滥；

Spring提供了jdbcTemplate解决方案，而且Spring集成了第三方高效的数据库连接管理框架。

#### Sping 事务管理器基础知识。

Spring 中数据库事务是通过 PlatformTransactionManager 进行管理的；

#### Sping 事务管理器提交和回滚规则。

事务的创建、提交和回滚是通过PlatformTransactionManager接口来完成的；当事务产生异常时会回滚事务，在默认的实现中所有的异常都会回滚，我们可以通过配置去修改当发生某些异常的时候不回滚事务；当无异常时会提交事务。

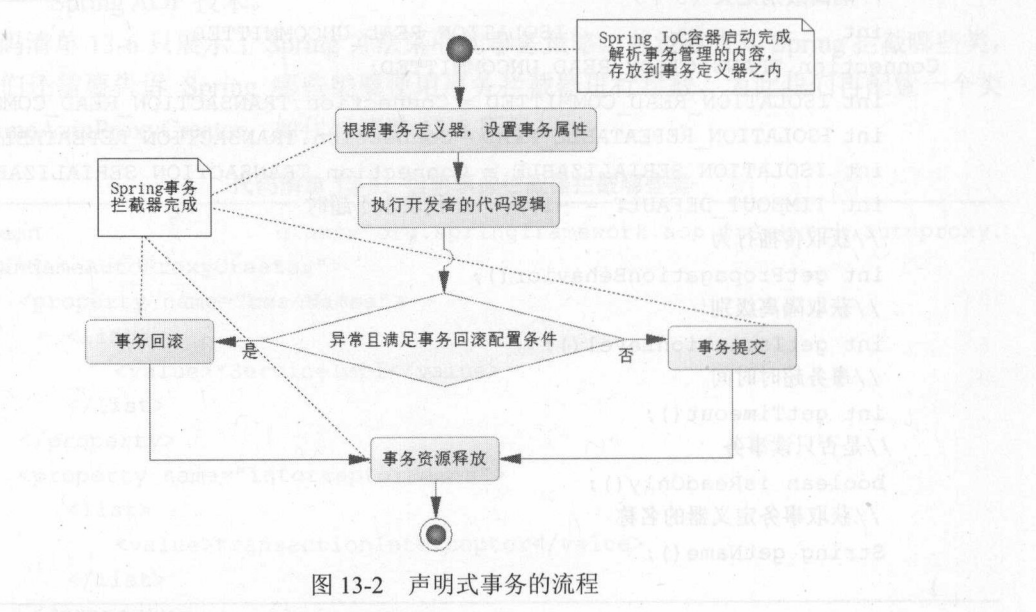
#### Sping 中常用的事务管理器有哪些？

DataSourceTransactionManager、JPATransactionManager、HibernateTransactionManager等等。

#### Sping 事务管理器如何配置？

1. 配置数据源事务管理器；
2. 配置事务拦截器指明拦截哪些类；

#### Sping声明式事务的流程？



1. Spring IOC初始化的时候，读取XML配置或者@Transction注解的类的信息，并且将事务配置信息保存到事务定义类中(TransactionDefinition 接口的子类)；
2. Spring拦截需要事务管理的方法；
3. Spring通过事务管理器创建事务，并且获取事务定义类(TransactionDefinition )中的隔离及级别、传播属性等等配置采取一种特定的策略。
4. 通过反射的方式调用开发者的业务，发生异常，并且符合事务定义类回滚条件的，Spring就会将数据库事务回滚，否则提交数据库事务。

#### Sping 数据库ACID特性。

1. 原子性：要么全部完成，要么全部不完成。
2. 一致性：两个事物并行操作的结果必须和串行操作的结果保持一致。
3. 隔离性：两个事务之间的隔离性。
4. 持久性：在事务完成以后，该事务对数据库所做的更改便持久保存在数据库之中，并不会被回滚。

#### 数据库的隔离级别。

**1、Read uncommitted 读未提交：**公司发工资了，领导把5000元打到singo的账号上，但是该事务并未提交，而singo正好去查看账户，发现工资已经到账，是5000元整，非常高 兴。可是不幸的是，领导发现发给singo的工资金额不对，是2000元，于是迅速回滚了事务，修改金额后，将事务提交，最后singo实际的工资只有 2000元，singo空欢喜一场。

**2、Read committed 读提交：**singo拿着工资卡去消费，系统读取到卡里确实有2000元，而此时她的老婆也正好在网上转账，把singo工资卡的2000元转到另一账户，并在 singo之前提交了事务，当singo扣款时，系统检查到singo的工资卡已经没有钱，扣款失败，singo十分纳闷，明明卡里有钱，为 何......。出现上述情况，即我们所说的不可重复读 。大多数数据库的默认级别就是Read committed，比如Sql Server , Oracle。

**3、Repeatable read 重复读：**当隔离级别设置为Repeatable read 时，可以避免不可重复读。当singo拿着工资卡去消费时，一旦系统开始读取工资卡信息（即事务开始），singo的老婆就不可能对该记录进行修改(但是可以读)，也就是singo的老婆不能在此时转账。虽然Repeatable read避免了不可重复读，但还有可能出现幻读 。singo的老婆工作在银行部门，她时常通过银行内部系统查看singo的信用卡消费记录。有一天，她正在查询到singo当月信用卡的总消费金额 （select sum(amount) from transaction where month = 本月）为80元，而singo此时正好在外面胡吃海塞后在收银台买单，消费1000元，即新增了一条1000元的消费记录（insert transaction ... ），并提交了事务，随后singo的老婆将singo当月信用卡消费的明细打印到A4纸上，却发现消费总额为1080元，singo的老婆很诧异，以为出 现了幻觉，幻读就这样产生了。Mysql的默认隔离级别就是Repeatable read。

**4、Serializable 序列化：**Serializable 是最高的事务隔离级别，同时代价也花费最高，性能很低，一般很少使用，在该级别下，事务顺序执行，不仅可以避免脏读、不可重复读，还避免了幻像读。

#### Sping 提供的事务传播行为。

Sping提供了7中事务传播行为，下面我们列举3中：

1. REQUIRED ：默认的事务传播行为，当前方法不存在事务，则创建事务；当前方法已经存在事务，则沿用当前的事务。
2. SUPPORT：当方法调用时，如果不存在当前事务，那么不启用事务；如果存在当前事务，那么就沿用当前事务。
3. REQUIRES\_NEW：无论是否存在当前事务，方法都会在新的事务中运行。
4. NEVER：不支持事务，只有在没有事务的环境中才能运行它，如果当前方法存在当前事务，则抛出异常。

#### 正确使用注解＠Transactional。

1、对于静态（ static ）方法和public方法，注解＠Transactional 是失效，因为spring事务是通过动态代理实现的。

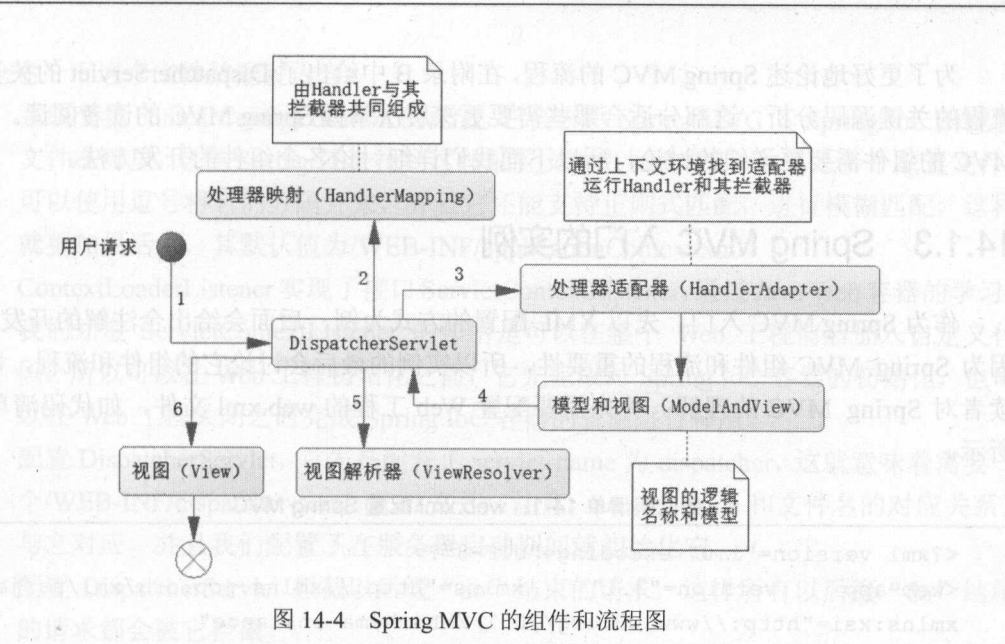
2、自调用事务失效，即同一个类中方法1调用方法2，则方法2配置的事务属性失效；这是因为自调用过程并没有产生代理对象的原因所致。

### Spring MVC

#### Spring MVC 的组件和流程。

Spring MVC是一种基于Servlet的技术，它提供了核心控制器DispatcherServlet; HandlerMapping、HandlerAdapter、ModelAndView、ViewResolver、View等相关组件。

Spring MVC 流程如下：



1. 初始化的时候根据配置文件生成HandlerMapping。
2. DispatcherServlet 首先通过请求和事先解析好的 HandlerMapping配置，找到对应的处理器（ Handler），这样就准备开始运行处理器和拦截器组成的执行链而运行处理器需要有个对应的环境这样它就有了一个处理器的适配器（HandlerAdapter ），通过这个适配器就能运行对应的处理器及其拦截器，这里的处理器包含了控制器的内容和其他增强的功能。
3. 处理器返回ModelAndView给DispacherServlet 。
4. 通过视图解析器得到View，默认返回的是jstlView视图。

#### 2.Spring MVC 接收请求参数的方式有哪些？

1、 不需要任何注解，只需要前、后的参数名保持一致即可接收参数。这正是Spring MVC的智能之处。

2、 使用＠RequestParam注解接收参数。

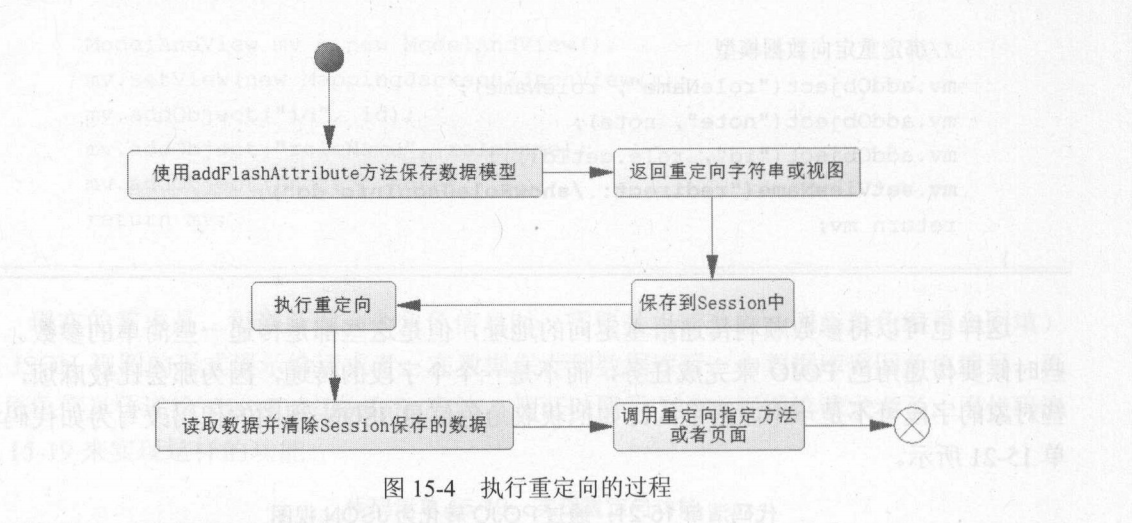
3、 使用 URL 传递参数（@PathVariable）。

4、 使用JSON传递参数（＠RequestBody）。

#### 3.Spring MVC 重定向和重定向时的数据传输？

Spring MVC有一个约定，当返回的字符串中有一个redirect的时候，它就会认为需要的是一个重定向。

在URL重定向的时候，并不能有效传递对象，Spring MVC为我们提供了flash属性，用来在重定向的时候传递对象。其原理如下：

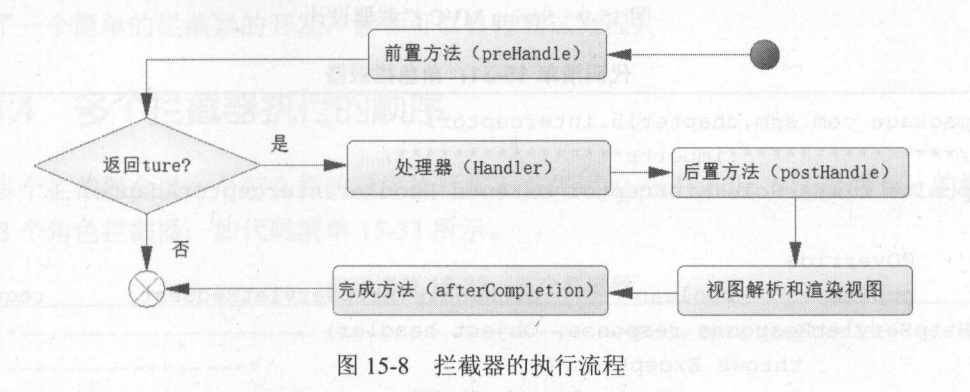


#### 4.Spring MVC 保存并获取属性参数？

Spring MVC 提供了5个注解，从request、session中读取数据或者保存数据到request、session中；它们分别是@requestAttribute(获取http请求中的属性值)、 @SessionAttribute（获取session中的属性值）、 @SessionAttributes(保存属性值到session中)、＠CookieValue(获取Cookie中的属性值)、＠RequestHeader(获取request头中的属性值)。

#### 5.Spring MVC 拦截器？

Spring MVC 拦截器需要实现HandlerinterceptorAdapter接口，下面是单个拦截器的执行流程（拦截器可以有多个）：



#### Spring MVC 表单校验？

Spring MVC提供了两种校验方式，一是使用 JSR 303 注解验证输入内容，二是使用Spring MVC提供的验证器（自定义的验证器必须实现Validator接口）。

#### Spring MVC 数据模型？

所谓数据模型，实际上就是需要展示在前端页面上的数据，Spring MVC 通过 Model或ModelMap来构建数据模型； Model 和 ModelMap在本质上并没有区别。

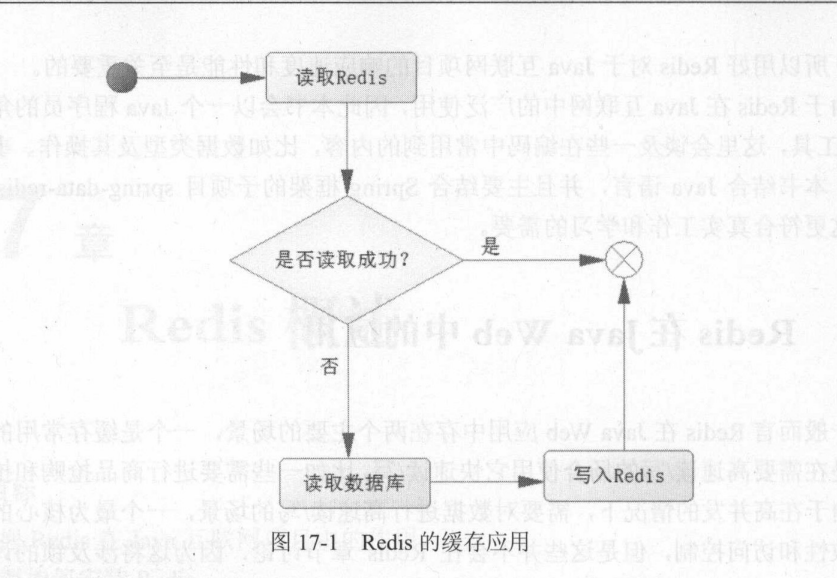
#### Spring MVC 有哪些视图？

Spring MVC视图可以分作逻辑视图、非逻辑视图；JstlView、 IntemalResourceView（**逻辑视图**） 是父子类关系，所以它们可以归为一类，它们主要是为了 JSP 渲染而服务的；MappingJackson2J son View （**非逻辑视图**）则是一个 JSON 视图 ；当然Spring MVC还有其它等等一些不常用的视图。

### Reids3.0

#### Redis在Java Web中的应用。

* 1. **缓存。**

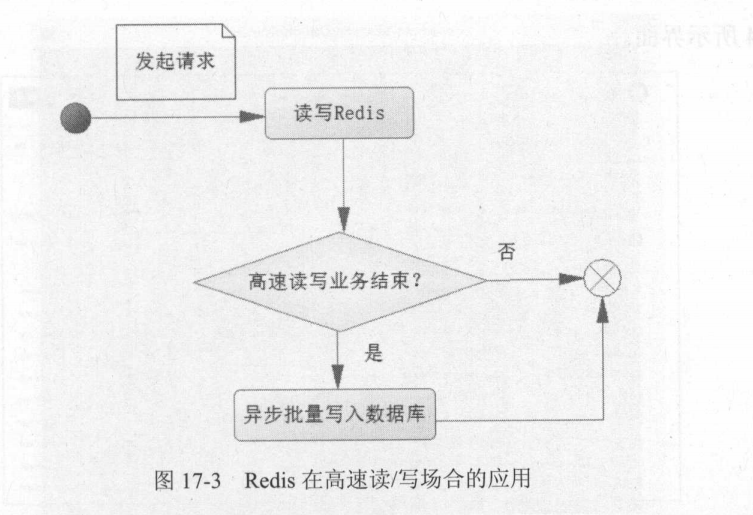


第一次读取数据的时候，读取redis的数据就会失败，此时会触发程序读取数据库，把数据读取出来，写入redis。

第二次及以后读取数据时，就直接读取redis.

* 1. **高速读/写场合。**

在互联网应用中往往存在着一些需要高速读/写的场合，比如商品的秒杀、抢红包、春运抢票等等。



假设面对的是一个商品秒杀的场景，从上面的流程来看，一个用户抢购商品，绝大部分的场合都是在操作内存数据库，所以性能会更加优越。只有在商品抢购一空后才会触发系统把Redis缓存的数据写入数据库磁盘中。

而在现实中需要高速响应的系统比上面的分析更加复杂，需要解决高并发下数据的安全性和一致性、有效请求、无效请求、事务一致性等诸多问题；

我们可以通过验证码、短信、单位时间内限制用户的访问量等形式过滤无效的请求。

#### Redis 的优点。

速度快、高可用和分布式(redis3.0 提供了redis的集群实现)、客户端语言多、简单稳定(单线程)、功能丰富（支持5种数据结构、提供发布订阅功能、流水线PipeLine、Lua脚本）。

#### Redis 能代替传统关系型数据库吗？

暂时无法代替，Redis是基于内存的数据库，无法支持大容量存储、没有像SQL那样强大的脚本语言、数据结构比较简单等等这些原因，注定其无法替代传统的关系型数据库。

#### Redis 的数据类型？

字符串、哈希表、列表、集合、有序集合、基数六种数据类型。

#### Redis 事务简单介绍

redis事务执行分作三步。开启事务（multi）、命令进入队列、执行事务（exec）。

redis可以使用watch命令监控事务，通过判断监控的key值是否一致，从而决定是否回滚事务。

redis的事务回滚只是对命令是否正确做判断，对于命令执行过程中产生的异常无法回滚，这正是redis事务的缺陷。

Redis事务不支持集群操作。

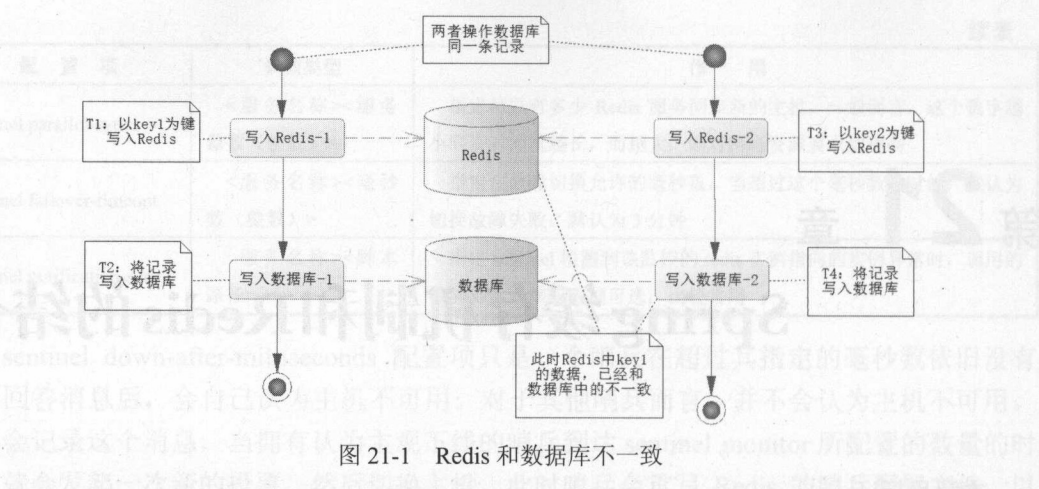
#### Redis 流水线（pipelined）

使用multi...exec事务命令是有系统开销的，因为它会检测对应的锁和序列化命令。有时候我们希望在没有任何附加条件的场景下去使用队列批量执行一系列命令，从而提高系统的性能，这就是流水线（pipelined）技术。

#### Redis 和 数据库结合会产生什么问题。

会产生数据不一致的情况，比如下面两种情况：

1. 对于Redis的事务，通过学习应该清楚它并不是很严格的，如果发生异常回滚事件，那么Redis的数据就可能和数据库不太一致。
2. 如下图：

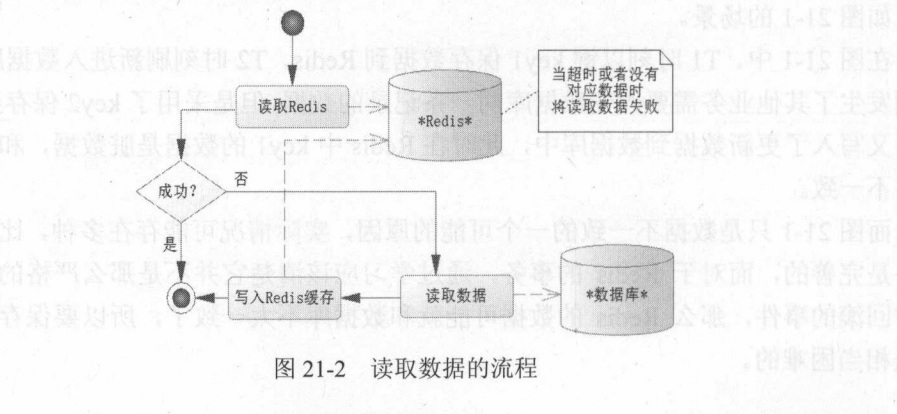


Tl 时刻以键 keyl 保存数据到 Redis, T2 时刻刷新进入数据库，但是 T3时刻发生了其他业务需要改变数据库同一条记录的数据，但是采用了 key2 保存到 Redis 中，然后又写入了更新数据到数据库中，此时在 Redis keyl 的数据是脏数据，和数据库的数据并不一致。

#### 如何解决Redis和数据库结合产生的不一致的问题。

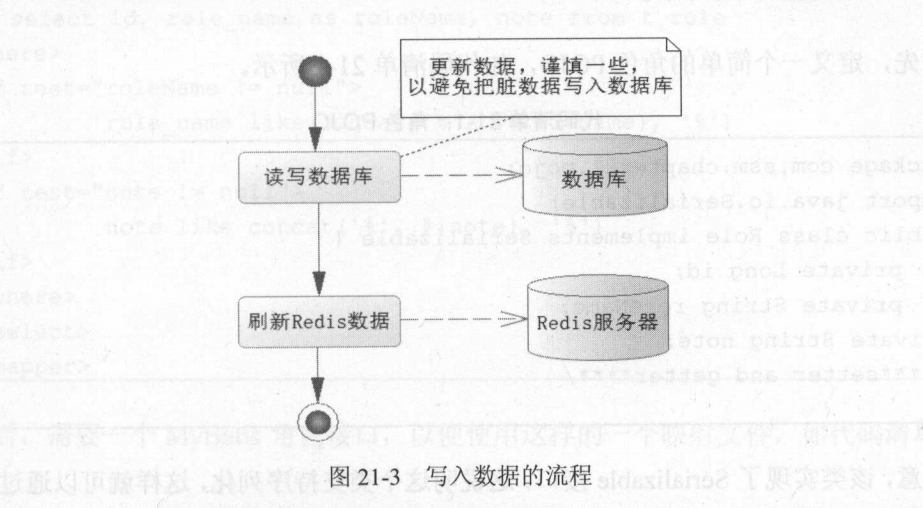
在读、写的时候制定相应的规则，是可以有效的避免这种情况的发生的：

1. 读操作



数据缓存往往会在Redis上设置超时时间，当设置Redis的数据超时后，Redis就没法读出数据了，这个候就会触发程序读取数据库 然后将读取的数据库数据写入Redis（此时会给 Redis 重设超时时间 ），这样程序在读取 过程中就能按一定的时间间隔刷新数据了。

1. 写操作



写入业务数据，先从数据库中读取最新数据，然后进行业务操作，更新业务数据到数据库后，再将数据刷新到 Redis 缓存中，这样就完成了一次写操作。

### JAVA设计模式