**缓存**

本文所有截图和一些基本概念都是来自尚硅谷培训机构SpringBoot或大数据模块讲师的PPT，特别鸣谢。（以下内容只针对Java开发领域）

# 一、数据库连接

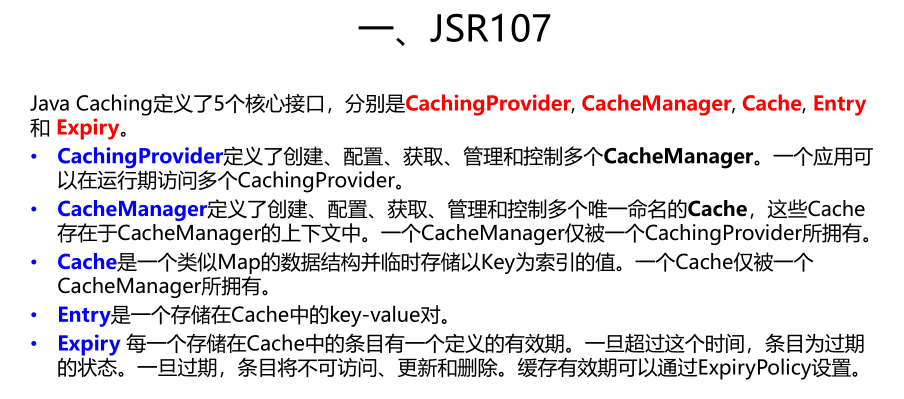
数据库在项目中是一个核心资源，频繁的连接而不进行管理必然导致资源崩溃，良好的数据库连接管理可以帮助项目健壮、安全。在目前成熟的连接池技术中，做得比较好的两个技术是dpcp和c3p0（其实就是两个Java的Jar包，只要在连接数据源的时候填写几个参数就可以帮助管理数据库连接资源），它们能很好的分配、管理、释放数据库连接。具体常用的几个参数是：最大连接数量，最大空闲等待时间，最小连接数量等。当然，现在的Java项目中是基本看不到这种东西了，看不见不代表没有，现在流行的MyBatis、Hibernate、JPA等等一系列耳熟能详的ORM框架已经把这些东西写在了底层，所以在具体的编程过程中已经不需要程序员去关心这些。

# 二、缓存的前世今生

最初缓存出现的原因就是：将频繁使用的数据查询出来放到一个地方，给数据库降压，简单来说就是保护数据库。但是现在的缓存已经不单单只是缓存了，它甚至改变了传统的数据库（一家之言，数据库这种东西，短时间之内肯定是不会淘汰的，未来谁也说不准，但是Redis所代表的NoSQL系列，以及区块链“去中心化”的思想出来，的的确确让我们看到了对于数据的其他解决方案，我们应该以一种发展的眼光去审视它们）。

再回到正题来，说说缓存的前世今生吧：

最早其实Java的JDK里面有对缓存的支持，也就是JSR107。



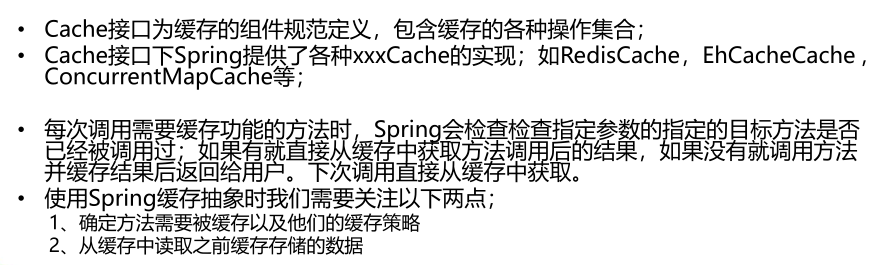
为什么不用呢？其实不是JDK提供的不好，而是显得有些笨重，这一点和java序列化很类似，序列化等会再说。

那目前市面上使用什么作为缓存工具呢？（我比较喜欢说“市面上”这个词，比较贴切，菜市场下一个季节什么菜卖的好谁也说不好，也是一种隐喻要一直关注前沿技术吧），主要有三个：redis、memcached、ehcache，也可以称缓存“中间件”。但是使用最广泛的是大名鼎鼎的Redis。

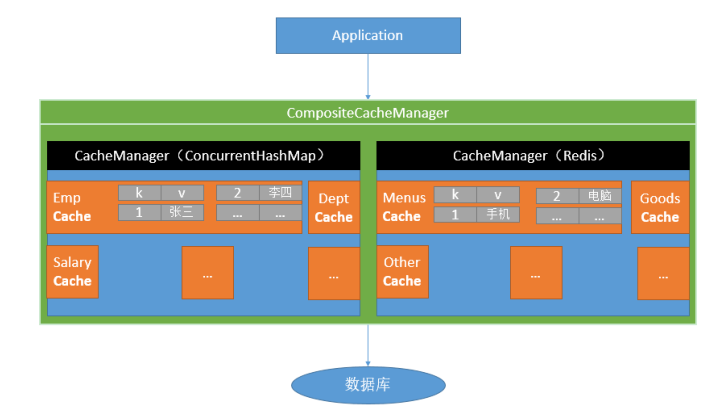
在说Redis之前要说一个东西，那就是”Spring缓存抽象”

# 三、Spring缓存抽象

现在越来越多的Java项目都是使用” Spring全家桶”，许多的程序员的确用起来得心应手（但还是要熟练啊），Spring从3.1开始定义了Cache和CacheManager接口来统一不同的缓存技术,并支持使用JCache（JSR-107）注解简化我们开发。



这是比较官方的说法，我这枚小白的理解就是，只要你项目中使用了Spring并且你想在你的项目中使用缓存工具，你用Redis也好，用ehcache，甚至你硬是想手撸原生的Java缓存，都可以从这个接口下面开始你的表演，大大小小的东西都塞进了一个口袋里面，添加了新一个层次的抽象。具体使用可以到spring官网查看官方文档，推荐去SpringBoot模块去看。



# 四、Redis，HashMap与哈希表

回头来说Redis,它是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。它存储数据是以K,V的形式存储，增删改查语句都比较简单。

多说一句，学过数据结构的对哈希表肯定不会对这种KV陌生，Java里面对哈希表比较完美的实现是HashMap这个接口，里面存储的就是kv键值对，在极度极端的情况下，也就是K值都一样时候，HashMap会变成线性表，效率低下，但是出现的概率极低，听说在设计这个接口的时候K值有x个相同的概率在数学上是服从泊松分布的（我也没去考证，但是这个接口写的是真的很棒）。但是当每一个线性表长度达到6之后又会变成红黑树，对于HashMap的知识是比较有意思的，对了，HashMap线程不安全，但另外一个接口HasnTable线程安全但是效率低下，所以在高并发的环境下要使用ConcurrentHashMap,面试经常问，要记住哦，如果面试额时候再问道这些，就把哈希表这种数据结构好好看一下，面试官绝对没话说。Redis底层其实也是对哈希表的实现，只是解决办法和Java的又不太一样，但还是保证了它的**高可靠性，线程安全性**。（由于对Redis知之甚少，只能通过哈希表说一下了，其实只要知道它存储的是kv键值对，会增删改查，项目中主要的应用还是对**Redis的整合**，这才是难点重点）

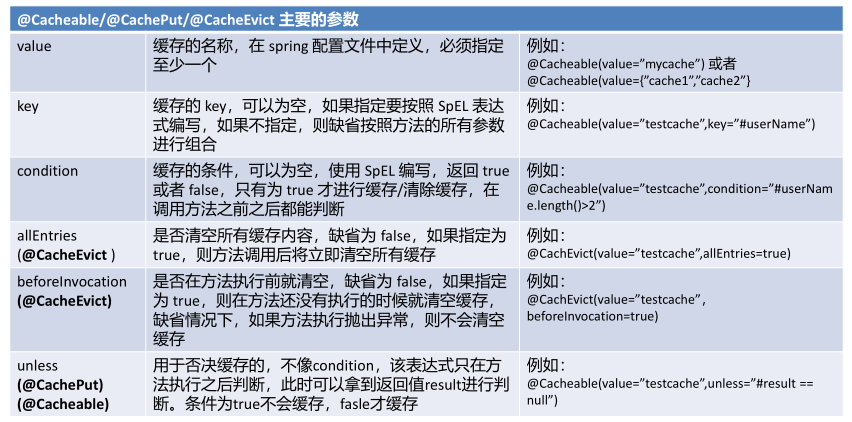
那么使用缓存之后，针对数据库的增删改查就不太一样了。

**查询**时会先去查询缓存，有就返回。没有才会去查询数据库，查询到了先新建缓存并且存储到缓存中以后再返回；**增加**的时候先在数据库新增数据，再去开辟缓存空间将数据存储到缓存；**删除**的时候先删除数据库再删除缓存；**修改**的时候先修改数据库再修改缓存。

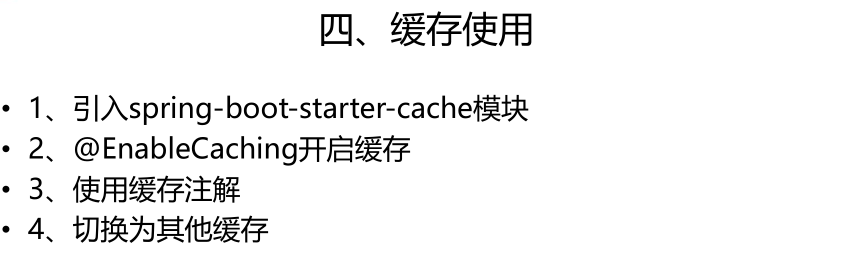
很多时候这是程序员自己要写的逻辑，我们维护的就不单单是数据库，还增加了缓存，变成了维护两项。很多时候为了避免忘记或者是顺序不对，比如说涉及到事务可能会“脏读”，“幻读”，“不可重复读”等情况的时候（这些都是数据库事务里面必然会出现的逻辑漏洞），那么很多公司就会把这一块加一层数的封装，把业务逻辑封装好，不需要程序员去考虑缓存，程序员面对的还是原来的增删改查，但是其实是对数据库和缓存都进行了增删改查，只是降低了出错的风险而对程序员不可见了而已。（不知道有没有说明白，反正很多时候你写的增删改查已经不单单是对数据库的增删改查了，而是包含了数据库和缓存，并且如果不去跟代码，获取你还不知道，但是那些大牛就能一眼看出很多门道）。因为现在程序员如果不是针对于技术开发，而是专注于业务的话，他们是不需要知道这些的，越来越多时候只有去到公司研发中心，或者是核心团队，或者是没有做成“产品”的公司，才会考虑到这些。

给出一些SpringBoot整合缓存时候常用的注解，方法（不需要手撸缓存代码的可以忽略）





在SpringBoot项目里面使用缓存的步骤：



在项目里面整合Redis的步骤



# 五、序列化

序列化：说简单一点就是在你电脑上的对象信息当你电脑关机了就没了，下次启动要重新实例化，可不可以想办法把它存起来，下次不想在重新实例化它了，或者我想把它发送给另外一台电脑上。设计模式里面有一种模式叫做单例模式，举个例子，在中国只能娶一个妻子，这就是单例模式，不允许实例化第二个。在JavaEE的项目中也有啊，一个session，当你关闭浏览器，这个session就不见了，在这个session里面的操作存储下来就需要系列化。尤其是现在大数据、分布式系统已经成为主流、微服务的概念深入人心，序列化就越来越重要。

序列化就是把内存中的对象，转换成字节序列（或其他数据传输协议）以便于存储（持久化）和网络传输。反序列化就是将收到字节序列（或其他数据传输协议）或者是硬盘的持久化数据，转换成内存中的对象。

Java 的序列化是一个重量级序列化框架（Serializable），一个对象被序列化后，会附带很多额外的信息（各种校验信息，header，继承体系等），不便于在网络中高效传输。所以，很多现在流行的框架都会有自己的序列化机制。

比如hadoop 自己开发了一套序列化机制（Writable），精简、高效。

序列化和反序列化在分布式数据处理领域经常出现：进程通信和永久存储。然而 Hadoop中各个节点的通信是通过远程调用（RPC）实现的，那么 RPC 序列化要求具有以下特点：1）紧凑：紧凑的格式能让我们充分利用网络带宽，而带宽是数据中心最稀缺的资源2）快速：进程通信形成了分布式系统的骨架，所以需要尽量减少序列化和反序列化的性能开销，这是基本的；3）可扩展：协议为了满足新的需求变化，所以控制客户端和服务器过程中，需要直接引进相应的协议，这些是新协议，原序列化方式能支持新的协议报文；4）互操作：能支持不同语言写的客户端和服务端进行交互；

# 六、远程过程调用（RPC）

谈到序列化就不得不说RPC，对Dubbo这个框架熟悉的肯定都知道RPC（Dubbo是阿里的，2016年SpringCloud抢占全球分布式市场之际，在2017年农历春节的时候免费贡献给了Apache组织，是国内分布式系统领域比较权威且使用广泛的框架）：

RPC （Remote procedure Call）远程过程调用，是一种计算机通讯协议。

计算机A上的一个进程，调用另外一台计算机B上的进程，其中A上的调用被挂起，而B上的被调用进程开始执行，当B返回给A结果的时候，A

继续执行。调用方法可以通过使用参数将信息传送给被调用方。而这个过程对开发人员来说是透明的。简单来说就是我是服务提供者，我就等着别调用我，我跟他不在一台机器上。我是服务使用者，我就掉别人的服务，不需要我自己有。

RPC框架的三个主要角色是：Provider、Consumer和Registry

Server是暴露服务的服务提供方

Client是调用远程服务的服务消费方

Registry服务注册与服务发现的注册中心 还负责负载均衡

服务提供者启动之后向注册中心注册机器的ip、port以及提供的服务列表：服务消费者启动后会向注册中心获取服务提供方的地址列表，可以实现负载均衡和Failover，一个成熟的RPC框架需要用到的技术：

1、动态代理：生成client stub 和 server stub 需要用到java的动态代理技术。

可以选用JDK的动态代理，也可以使用比较成熟的开源字节码工具框架 Cglib、Javassist等。其中，推荐使用Cglib因为它跟Spring是完美兼容的。如果不想用这些，可以使用AcpectJ去面向切面编程，Spring两大神器，IOC和AOP。

2、序列化：为了能在互联网上传输和接收java对象，需要对他进行序列化与反序列化操作。可以选用的Java的序列化机制，因为原生Java的序列化做得太强大了，造成效率低的问题，反而不推荐使用。推荐使用一些开源的、成熟的序列化技术，有：protobuf Thrift hessian Kryo Msgpack等。

3、NIO：当前很多RPC框架都直接基于netty这种IO通讯框架，这是计算机网络的知识了。

4、服务注册中心：可以选用的技术有Redis，Zookeeper，Consul，Etcd

这里面大名鼎鼎的就是Redis，Zookeeper这两个了。

其他开源的RPC框架有：Motan gRPC Thrift

其实，在Java的JDK里面就有一个概念叫RMC，叫做远程方法调用，可以这么理解，RPC是在RMC上的一层封装。调用一个方法变成了调用一个服务。

最后说一下抽象：“抽象”是从始至终都贯彻在计算机科学道路上的一个概念。

从最底层的将电路的开闭抽象为0和1，将01抽象为cpu、内存、指令。将一条条指令抽象为条语句i=i+1,又将一条条语句抽象为一段面向过程的C语言代码，又抽象为面向过程、面向对象、面向切面等等一系列语言。再将这些语言精巧的设计成框架、设计模式等等等等。所以学习是要学习的是这个抽象的思维，遇到问题是怎样抽象出解决思路的。通常抽象的结果都是将更加简单方便的解决方法提供给使用者，将一些实现的细节都封装在内部，一方面是降低了难度，另外一方面是处于安全考虑。