一、Redis

1、什么是redis

简单点来说redis就是一个基于内存的高性能的键值对（key-value）数据库（非关系型数据库）；

2、简介

redis本质上是一个key-value类型的内存数据库，整个数据库全部加载在内存当中进行操作，定期通过异步操作把数据库数据刷新到硬盘上进行保存。由于操作是全部在内存中，Redis的性能很优越，每秒大概可以处理超过10万次读写操作，是目前已知的性能最快的key-value数据库；而Redis最大的魅力是支持存储多种数据结构，此外单个value的最大限制是1GB，因此Redis可以用来实现很多功能，比如可以使用redis的List来做一个FIFO双向链表，实现一个轻量级的高性能的消息队列服务等等。另外Redis也可以对存入的key-value设置expire时间，因此也可以被当作一个功能加强版的memcached（Memcached 是一个高性能的分布式内存对象缓存系统，用于动态web应用以减轻数据库负载。它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提高动态、数据库驱动网站的速度。memcached基于一个存储键/值对的hashmap）来用。它的主要缺点就是数据库容量受到物理内存的限制，不能用做海量数据的高性能读写，所以Redis的应用场景局限在较小数据量的高性能操作和运算上；

3、redis支持的数据格式

string，list，set，sorted set（zset），hash

4、redis的使用好处

* 速度快，因为数据存在内存中，类似于HashMap；
* 支持丰富数据类型，支持string，list，set，sorted set，hash；
* 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么是全部执行，要么就全部不执行；
* 丰富的特性：可用于缓存，消息，按key设置的过期时间 ，过期后将自动删除；

5、redis中数据淘汰策略

* 从已设置的过期时间的数据集中挑选最近最少使用的数据淘汰；
* 从已设置过期时间的数据集中挑选将要过期的数据淘汰；
* 从已设置的过期时间的数据集中任意选择数据淘汰；
* 从数据集中挑选最近最少使用的数据淘汰；
* 从数据集中任意选择数据淘汰

6、Redis的使用场景

Redis最适合所有数据in-momory的场景。

* 会话缓存：用Redis缓存会话比其他存储的优势在于Redis提供持久化
* 队列：Redis在内存存储引擎领域的一大优点是提供list和set操作，所以redis可以作为一个很好的队列平台来使用；
* 排行榜/计数器：Redis在内存中对数字进行递增或递减的操作实现非常好。可以使用Redis的set和sorted set来实现；

7、MongoDB也是一种可以用来做缓存的NoSQl缓存数据库；他们之间的区别在于：（1）、MongoDB建议集群部署，更多的考虑集群方案，Redis更偏重于进程顺序写入，虽然支持集群，但也仅限于主-从模式；（2）、Redis是完全在内存中保存数据的数据库，使用磁盘知识为了持久性目的，Redis数据全部存在内存，定期写入磁盘；MongoDB是文档行的非关系型数据库，更加类似于MySQL，支持字段索引、游标操作，其优势在于查询功能比较强大，擅长查询JSON数据，能存储海量数据，但是不支持事务；MOngoDB的所有书库都是存放在硬盘上的，所有操作的数据通过mmap的方式映射到内存某个区域，然后mongoDB就在这块区域进行数据修改，避免了零碎的硬盘操作；如果mongoDB在内存中修改了数据，mmap数据flush到硬盘之前，系统宕机了，数据就会丢失；Redis只能使用单线程，性能受限于CPU的性能；mongoDB内置了数据分析的功能，Redis不支持；Redis对事务的支持比较弱，只能保证事务中每个操作连续执行，mongoDB不支持事务；

二、FastDFS

1、简介

FastDFS是一个开源的轻量级分布式文件系统，由跟踪服务器（tracker server）、存储服务器（storage server）和客户端（client）三个部分组成，主要解决了海量数据存储问题，特别适合以中小文件（4KB～500MB）为载体的在线服务。

2、Tracker server

Tracker是FastDFS的协调者（调度服务器），负责管理所有的storage server和group，每个storage在启动后会连接tracker，告知自己所属的group等信息，并保持周期性的心跳，tracker根据storage的心跳信息，简历group==》storage server的映射表；tracker的集群中松油的tracker是平等的，客户端请求tracker server采用轮询的方法，如果请求的tracker不能提供服务就换林谷一个tracker。

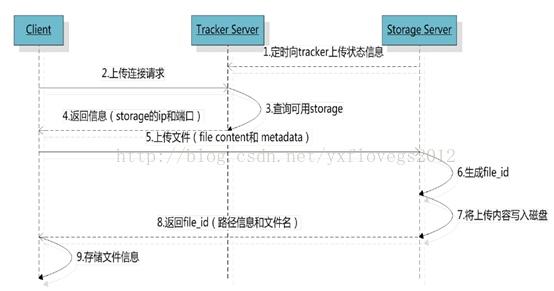
3、storage server

storage server是存储服务器，主要用来做文件存储，storage做集群时，以组为单位。一个组内的storage server之间相互通信，文件进行同步，保证组内storage完全一致，不同组之间不能相互通信。

4、文件上传流程

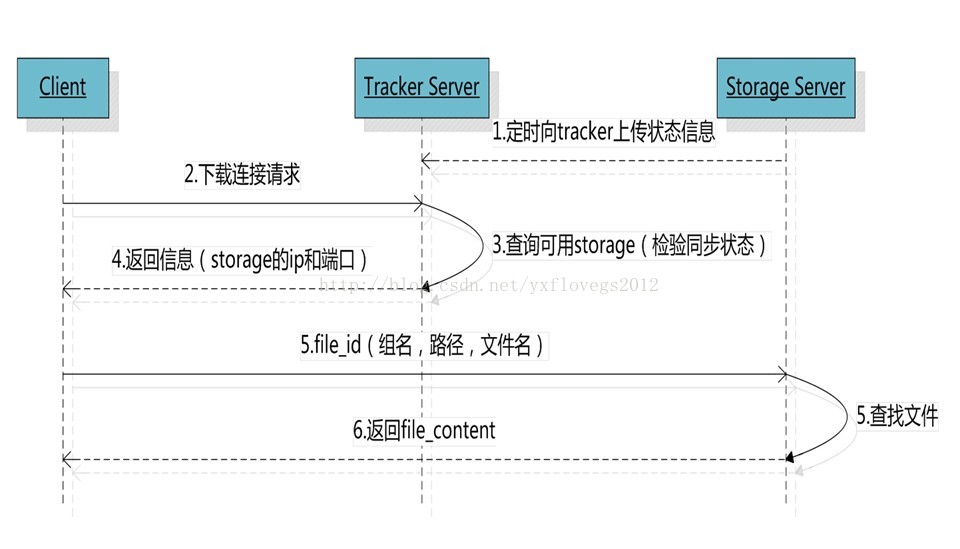
* 1. 选择tracker server：当集群中不止一个tracker server时，由于tracker之间是完全对等的关系，客户端在upload文件时可以任意选择一个tracker。
  2. 选择存储的group，当tracker接受到upload file请求时，会为该文件分配一个存储该文件的group。
  3. 选择storage server，当选定group后，tracker会在group内选择一个storage server给客户端。
  4. 选择storage path，当分配好storage server后，客户端将向storage发送写文件请求，storage将会为文件分配一个数据存储目录；
  5. 生成fileid，选定存储目录之后，storage会为文件生成一个fileid；
  6. 选择两级目录，每个存储目录下有两级256\*256个字目录，storage会按照文件的fileid进行两次hash，路由到其中一个子目录，然后将文件以fileid为文件名存储到带子目录下。
  7. 客户端upload file成功后，会拿到一个storage生成的文件名，接下来客户端根据这个文件名即可访问到该文件。

文件结构： 组名/磁盘/目录/fileid（文件名）



5、文件下载流程

和文件上传的流程反之看图体会



三、ActiveMQ

前言：

消息中间件就是指利用高效可靠的消息传递机制进行与平台无关的数据交流，并基于数据通信来进行分布式系统的集成。通过提供消息传递和消息队列模型，他可以在分布式环境下提供应用解耦、弹性伸缩、冗余存储、异步通信、数据同步的等功能。有ActiveMQ、RabbitMQ、Kafka、ZeroMQ等；

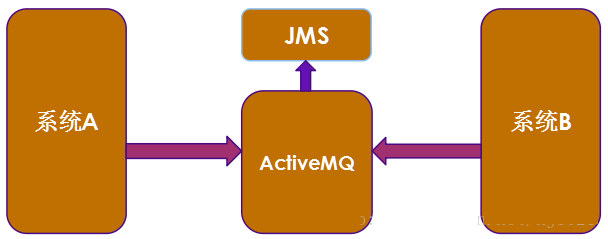
1、背景和简介：

不同系统之间的信息交换，是我们开发中比较常见的场景，比如系统A要把数据发送给系统B。于此SUN公司提出了一种面向消息的中间件服务—JMS规范，产雇佣的集中信息交互技术（httpClient、hession、dubbo、jms、webservice五种）；JMS（JMS就是Java消息服务，是JavaEE的标准之一，这种规范指出：消息的发送应该是异步的、非阻塞的。也就是说消息的发送者发送完消息就直接返回了，不需要等待接受着返回后才能返回。发送者和接受者应该是互不影响的。所以这种规范能够减轻活着消除系统瓶颈，实现系统之间的解耦，提高系统的整体伸缩性和灵活性。JMS知识JavaEE中定义的一组标准API，它自身并不是一个消息服务系统，他是消息传送服务的一个抽象，也就是定义了消息传送的接口而并没有其具体实现）。

1）、ActiveMQ就是JMS规范的具体实现，他是Apache下的一个项目，采用Java语言开发，是一款非常流行的为应用程序提供高效的可扩展的、稳定的和安全的企业级消息通信的开源消息服务器。

2）、RabbitMQ是采用Erlang语言实现的AMQP协议的消息中间件，最初起源于金融系统，用于分布式系统存储转发消息。

3）、Kafka起初是由LinkedIn公司采用Scala语言开发的一个分布式、多分区、多副本且基于zookeeper协调的分布式消息系统，现已经捐献给Apache基金会。他是一种高吞吐量的分布式发布订阅消息系统，以可水平扩展和高吞吐率而被广泛使用。



2、Java消息队列JMS整体设计结构

* 基本要素：生产者（producer）、消费者（consumer），消息服务（broker）
* JMS两种消息传送方式：1）点对点：专门用于使用队列Queue传送消息；基于队列的点对点消息只能被一个消费者消费，如多个消费者都注册到同一个消息队列上，当生产着发送一条消息后，只会有其中的一个消费者接收到该消息，而不是所有的消费者都能够收到该消息；2）发布/订阅：专门用于使用主题topic传送消息，基于主题的发布和订阅消息能被多个消费者消费，生产者发送的所有的消息，所有订阅了该topic的消费者都能接收到；
* JMS API：1）公共API，ConnectionFactory/Connection/Message/Destination/MessageProducer/MessageConsumer.总的来说就是游客Connectionfactory就可以创建Connection，有了Connection就可以创建Session，有了Session就可以创建Message、MessageProducer和MessageConsumer；2）点对点API：QueueConnectionFactory/QueueConnection/QueueSession/Message/Queue/QueueSender/QueueReceiver. 3）发布/订阅APITopicConnectionfactory/TopicConnection/TopicSession/Topic/Mession/TopicPublisher/TopicSubscriber;

四、Solr

1、简介

solr是一种开放源代码的、基于Lucene Java独立的企业级搜索应用服务器，易于加入web应用程序中。它对外提供类似web-service的api接口。用户可以通过http请求，向搜索引擎服务器提交一定格式的xml文件，生成索引；也可以通过http get操作题书查询的请求，得到xml/json格式的返回结果；要了解solr，先了解lucene

1. Lucene是apche软件基金会的一个开放源代码的全文检索引擎工具包，是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，布冯文本分析引擎。他不是一个完整的搜索应用程序，而是为你的应用程序提供索引和搜索功能。Lucene能够为文本类型的数据库建立索引，所以只要能够把需要索引的数据格式转换为文本类型，Lucene就可以对你的文档进行索引和搜索。
2. 什么是索引？为什么要索引？
   1. 对非结构化数据的检索，比如要找内容中包含一个字符串的文件，就需要对文档依次检索，从头扫到尾，如果此文档包含此字符串，则这个文档就是我们要找的文件，接着再找下一个文档，指导扫面完所有的文档。然后有一种想法就出来了，我们把非结构化数据弄的结构话一点，这样搜索就会变快很多了。这也就是全文检索服务实现的基本思路。即是将非结构化数据中的一部分信息提取出来，重新组织，让它变得结构化，然后再对有结构的数据进行搜索，从而达到搜索相对较快的目的；
   2. 索引就是从非结构化数据中提取出来的然后进行重新组织的信息；
   3. 说到索引，再说说反向索引。非结构化数据所存储的信息包含哪些字符串，就是文件到字符串的映射；反之如果我们已知字符串，要查询包含此字符串的文件，即是字符串到文件的映射。而这种从字符串到文件的映射事文件到字符串的反向过程，称之为反向索引。

3）、索引以及搜索过程：

* 首先存在一系列被索引文件；
* 被索引文件经过语法分析和语言处理形成一系列词（Term）；
* 经过索引创建形成词典和反向索引表；
* 通过索引存储将索引写入硬盘；
* 搜索过程；
* 用户输入查询语句；
* 对查询语句经过语法分析和语言分析得到一系列词；
* 通过语法分析得到一个查询树；
* 通过索引存储将索引读入到内存；
* 利用查询树搜索索引，从而得到每个词（term）的文档链表，对文档表进行交，差，并得到结果文档；
* 将搜索到的结果文档对查询的相关性进行排序；（这里可能会问到怎么对查询到的结果集进行排序？ 根据权重进行排序，权重就是词（Term）对于文档的重要性，越重要的词有越大的权重）
* 返回查询结果给用户。

3）、Solr和lucene的区别

solr依存于Lucene，因为Solr底层的核心技术使用的是Lucene来实现，Solr和Lucene的本质区别是：（1）、Lucene本质上是搜索库，不是独立的应用程序，而搜索是；（2）、Lucene专注于搜索底层的建设，而Solr专注于企业应用；（3）、Lucene不负责支撑搜索服务所必需的管理，而Solr负责；总的来说就是Solr是Lucene面向企业搜索应用的扩展。

4）、技术选型Solr和Elasticsearch

a）、Elasticsearch是分布式的；Elasticsearch完全支持Apache Lucene的接近实时的搜索；处理多租户不需要特殊配制，而Solr则需要更多的高级设置；Elasticsearch采用Gateway的概念，是的备份更加简单；各节点组成对等的网络结构，某些节点出现故障时会自动分配其他节点代替其进行工作；

b）、Solr有一个更大、更成熟的用户、开发和贡献者社区；支持添加多种格式的索引，如html、pdf、json、xml等；Solr比较成熟、稳定；不考虑建索引的同时进行搜索，速度更快；

c）、比较：当单纯的对已有数据进行搜索时，solr更快；当实时建立索引时，solr会产生io阻塞，查询性能较差，Elasticsearch具有明显的优势；随着数据量的增加，solr的搜索效率会变得更低，而Elasticsearch却没有明显的变化；solr利用zookeeper进行分布式管理，而Elasticsearch自身带有分布式协调管理功能；solr支持更多格式的数据，而elasticsearch仅支持json格式；solr官方提供的功能更多，而Elasticsearch本身更注重于核心功能，高级功能多有第三方插件提供；solr是传统搜索应用中表现好于Elasticsearch，但在传统实时搜索应用时效率明显低于elasticsearch；

五、JVM垃圾回收机制

1、什么是垃圾回收

* 垃圾回收就是回收内存中不再使用的对象；
  + 引用计数法：如果一个对象没有被任何引用指向，就可以视之为垃圾；
  + 根搜索算法（可达性分析算法）：通过一系列名为GC Roots的对象（可作为GC Roots的对象：虚拟机栈中引用的对象；方法区中静态属性引用的对象；方法区中常量引用的对象）作为起始点，从这些节点往下搜索，搜索所走过的路程称之为引用链，当一个对象到GC Roots没有任何引用链相连时，则证明此对象是不可用的；
* 步骤：查找内存汇总不再使用的对象；释放这些对象占用的内存；

3、分代

* 新生（年轻）代：所有新生成的对象都是放在年轻代的，年轻代的目标就是尽可能的快速收集掉那些生命周期短的对象；其中内存按8:1:1的比例氛围一个Eden区和两个Survivor区，大部分对象都是在Eden区生成，回收时首先将Eden区存活的对象复制到Survivor(1)的一个区域中，然后清空eden区；当这个Survivor(1)区满了时，就将Eden区和这个Survivor(1)区中的存活对象复制到另一个Survivor(2)区，然后清空Eden区和这个Survivor(1)区，此时Survivor(1)区时空的，然后将Survivor(1)和Survivor(2)区交换，就是一直保持Survivor(2)区为空，如此往复。当Survivor(2)区不足以存放Eden区和Survivor(1)的存活对象时，九江存活对象直接存放到年老代。若是年老代也满了就触发一次Full GC；
* 年老代：在年轻代中经历了N次垃圾回收后仍然存活的对象，就会放到年老代；存放的对象都是一些生命周期较长的对象；内存比起年轻代大很多（大概比例时1:2），当年老代内存也满时触发Full GC；
* 持久代：用于存放静态文件，如Java类、方法等

3、垃圾收集算法

* 标记-清除算法：标记出所有需要回收的对象，在标记完成后统一回收被标记的对象；缺点是：效率不高，再者就是清除之后会产生大量的不连续的内存碎片，可能会导致后续程序运行过程中需要分配较大对象时，无法找到足够的连续内存而不得不提前出发另一个垃圾收集的动作；
* 复制算法：为了解决上述效率问题。它将可用的内存按容量进行划分为大小相等的两块，每次只使用其中一块，当用完这一块，就将还存活着的对象复制到另外一块内存上，然后再把已经使用过的内存空间一次清理掉。依次往复操作，这样按顺序进行内存分配，实现简单，运行高效；但是本身可用内存会缩小为原先的一半；
* 标记-整理算法：标记过程仍然和标记-清除一样，但是后续步骤不是直接对可回收对象进行清理，而是让所有的存活对象都向一端移动，然后直接清理端边界之外的内存；
* 分代收集算法：当前商业虚拟机的垃圾回收都采用分代回收。一般将Java堆内存氛围新生代和老年代，这样就可以根据各个年代的特点采用最适应的收集算法。在新生代，每次垃圾收集时都发现大批对象死去，只有少量存活，那就选用复制算法，只需要付出少量存活对象的复制成本就可以完成收集，而老年代对象存活率较高，没有额外的空间对它进行分配担保，就必须使用标记-清理和标记-整理算法；

4.GC是什么时候触发的

GC有两种类型Scavenge GC和Full GC

* Scavenge GC：当新对象生成，并且在Eden区申请空间失败时就会触发Scavenge GC，对Eden区域进行GC，清除非存活对象，并且把尚且存活的对象移动到Survivor区，然后整理Survivor的两个区。这种发誓的GC是对年轻代的Eden区进行，不会影响到年老代。由于大部分的对象都是从Eden区开始跌，但是Eden区不会分配的很大，所以Eden区的GC会很频繁的进行。
* Full GC：对整个堆内存进行整理。Full GC因为需要对整个堆内存进行回收，所以比起Scavenge GC要慢，因此应该尽可能的减少Full GC的调节；一般年老代被写满、持久代被写满，System.gc（）被显示调用等会触发Full GC；

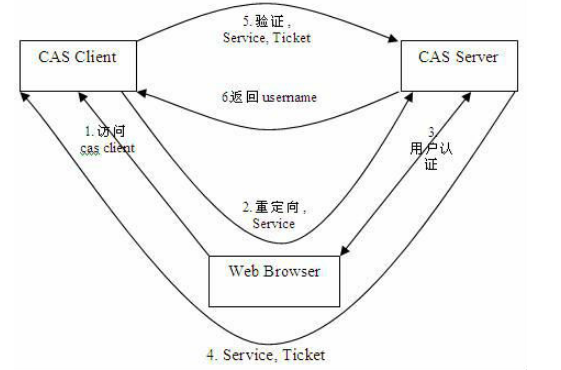
六、单点登录（SSO-CAS）

1、简介

SSO（single sign on），是指在多系统应用群众登陆一个系统，便可在其它所有系统中得到授权而无需再次登陆，包括单点登录和单点注销两部分；

CAS（Central Authentication Service）是一款正对web应用的单点登陆框架，CAS包含两个部分，CAS server 和CAS client。CAS server需要独立部署，主要负责对用户的认证工作，CAS Client负责处理客户端受保护资源的访问请求，需要登陆时，重定向到CAS server；

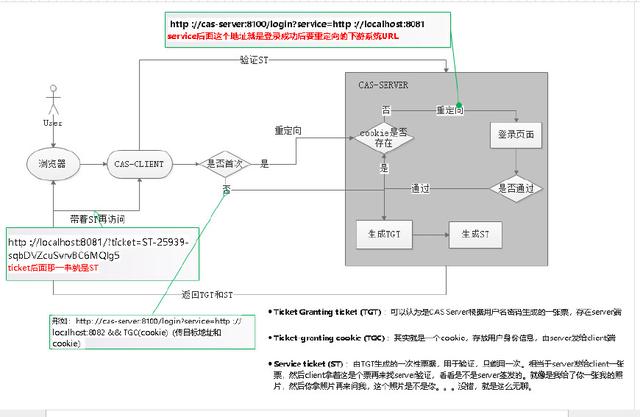
2、流程图



根据上述流程图，单点登录的访问流程步骤如下：

* 访问服务：SSO客户端发送请求访问应用系统提供的服务资源；
* 定向认证：SSO客户端会重定向用户的请求到SSO服务器；
* 用户认证：用户身份认证；
* 发放票据：SSO服务器会产生一个随机的Service Ticket；
* 验证票据：SSO服务器验证票据Service ticket的合法性，验证通过后，允许客户端访问服务；
* 传输用户信息：SSO服务器验证票据通过后，传输用户认证结果信息给客户端；

3、CAS的处理流程



CAS处理步骤：

* 用户访问网站，第一次来，重定向到CASServer，发现没有cookie，所以再重定向到CAS Server端的登陆页面，并且URL带有网站地址，便于认证成功后跳转，形如：http：//cas-server:9100/login?service=http://localhost:9003；
* 在登陆页面输入用户名密码认证，认证成功后cas server 生成TGT，再用TGT僧成一个ST，然后再第三次重定向并返回ST和cookie（TGT）到浏览器；
* 浏览器呆着ST再次访问要访问的地址http：//localhost:9003/?ticket=ST-435243-sqbDVZcuSDFdfgFds4;
* 浏览器的服务器收到ST后再去cas-server验证一下是否自己签发的，验证通过后就会显示页面信息，也就是重定向到service后面的URL，首次登陆完毕；
* 再登陆另一个接入CAS的网站，重定向到CAS Server，server判断是第一次来（但是此时有TGC，也就是cookie，所以不用重新登陆），但是此时没有ST，去CAS server申请一个，于是重定向到cas server(如http:cas-server:9100/login?service=http:localhost:9005&TGC(cookie)目标地址和cookie);
* cas server生成了ST后重定向给浏览器http：//localhost:9005?ticket=ST-23434-sqasdfdassdfsdf34;
* 浏览器的服务器收到ST后再cas server验证一下是否为自己签发的，验证通过后就会显示页面信息；

七、Spring Security权限管理

1、简介

Spring Security 是基于当前非常流行和成功的Java应用开发框架Spring的，它提供了一套Web应用安全性的完整解决方案。Web应用的安全性包括用户认证（Authentication）和用户授权（Authorization）两部分。用户认证指的是某个用户是否是系统中的合法主体，也就是说用户是否能够访问该系统。用户认证要求用户提供用户名和密码。系统通过校验用户名和密码来完成认证过程；用户授权指的是验证某个用户是否有权限执行某个操作。

Spring Security在用户认证方面，包括Http基本认证、HTTP表单验证、HTTP摘要认证、OpenID等；在用户授权方面，提供了基于角色的访问控制和访问控制列表，可以对应用中的领域对象进行细粒度的控制。

2、Spring Security用户认证授权过程

用户的相关信息是通过UserDetailService接口来加载的，该接口的唯一方法loadUserByUsername（String username），用来根据用户名加载相关的信息，方法的返回值是UserDetails接口，其中包含了用户的信息，包括用户名、密码、权限、是否启用、是否绑定、是否过期等，其中最重要的是用户权限，由GrantedAuthority接口来表示；

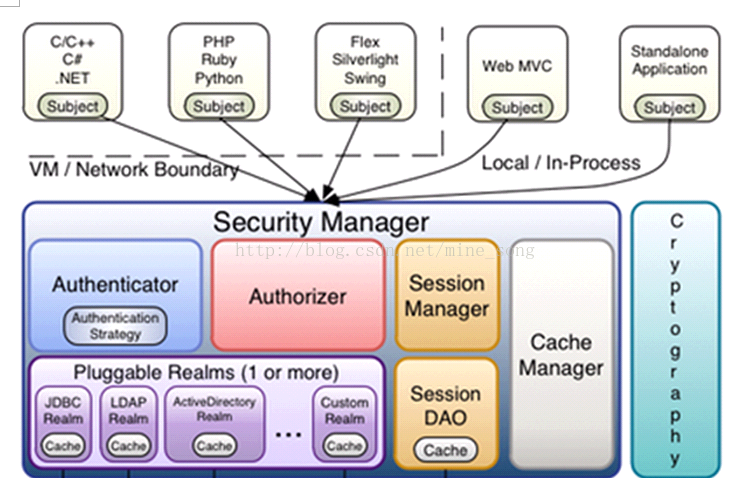
用户一次完整的登陆验证和授权，是一个请求经过层层过滤器从而实现权限控制，整个web端的配置为DelegatingFilterProxy（Spring security的委托过滤其代理类），它并不是实现真正的过滤，而是所有过滤器链的代理类，真正执行拦截处理的是由spring容器管理的一个个filter bean组成的filterChain。

三个重要的过滤器：

* UsernamePasswordAuthenticationFilter：这个过滤器用于拦截我们表单提交的请求（默认为/login），进行用户的认证；
* ExceptionTranslationFilter：这个过滤器主要是用来补货处理Spring security抛出的异常，异常的主要来源filterSecurityIntercaptor；
* FilterSecurityInterceptor：这个过滤器主要进行授权判断；

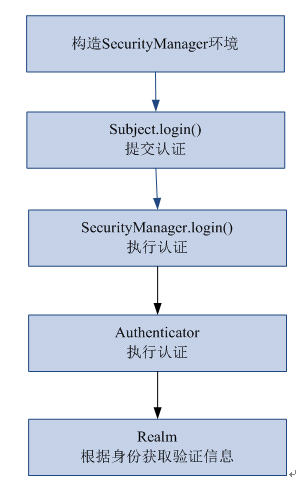
3、shiro框架

1. 简介：shiro是Apache 下的一款强大的易于使用的的安全框架，提供了用户认证、用户授权、加密、会话管理等；



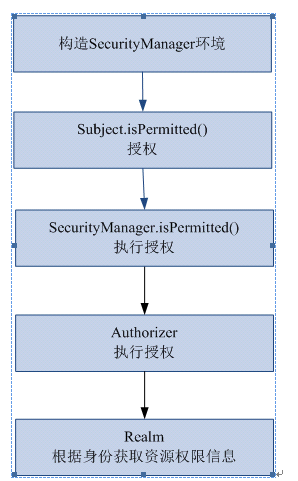
* 1. 三大核心组件及其他组件：
     + subject：表示当前用户。Shiro中通过subject来提供基础的当前用户信息，Subject不仅仅表示某个用户，与当前应用交互的任何东西都是Subject，所有的Subjec都要绑定到SecurityManager上，与Subject交互实际上就是被转换为SecurityManager的交互；
     + SecurityManager：是Subject的管理者，也是shiro框架的核心组件，是shiro框架的全局管理组件，用于调度各种shiro框架的服务。作用类似于springMVC中的DispacherServlet，用于拦截请求并处理；
     + Realm：Realm是用户的信息认证器和用户的权限认证器，我们需要自己来实现Realm来实现Realm来自定义的管理我们自己系统内部的权限规则。Securitymanager要验证用户，需要从Realm中获取用户。可以Realm看作是数据源；
     + authenticator：认证器，主题进行认证最终通过authentication进行的；
     + authorizer：授权器，主题进行授权最终通过authorizer进行；
     + 。。。。。。。。

b） 认证过程：



* 通过配置文件（ini或者xml）创建securityManager；
* 调用subject.login()方法主体提交认证，提交的token
* securityManager进行认证，securityManager最终由ModularRealmAuthenticaticator进行认证；
* ModularRealmAuthenticaticator调用Realm查询用户信息；如果查询到用户信息，就返回用户信息，否则就返回null；

c） 授权过程：



* 对subject急性授权，调用方法ispermitted；
* Securitymanager执行授权，通过ModularRealmAuthorizer执行授权；
* ModularRealmAuthorizer执行realm（自定义的实现Realm接口的类）从数据库查询权限数据，调用realm的授权方法：dogetAuthorizationInfo；
* Realm查询到权限数据，返回ModularRealmAuthoizer；

八、多线程问题

九、数据库(sql语句)优化

十、SpringBoot