# Java知识点总结

目录

[Java知识点总结 1](#_Toc523235127)

[**1.** **mybatis是如何管理session和cache的。** 3](#_Toc523235128)

[2.常见序列化协议及其优缺点 5](#_Toc523235129)

[**3.网站高并发大流量访问的处理及解决方法** 5](#_Toc523235130)

[4.Reactor多线程模型的特点 7](#_Toc523235131)

[5.阻塞、非阻塞、同步、异步区别 11](#_Toc523235132)

[6.堆（heap）与栈（stack） 11](#_Toc523235133)

[7.mybatis数据库数据处理 14](#_Toc523235134)

[8.java反射机制 15](#_Toc523235135)

[9.高级工程师：微服务与分布式 15](#_Toc523235136)

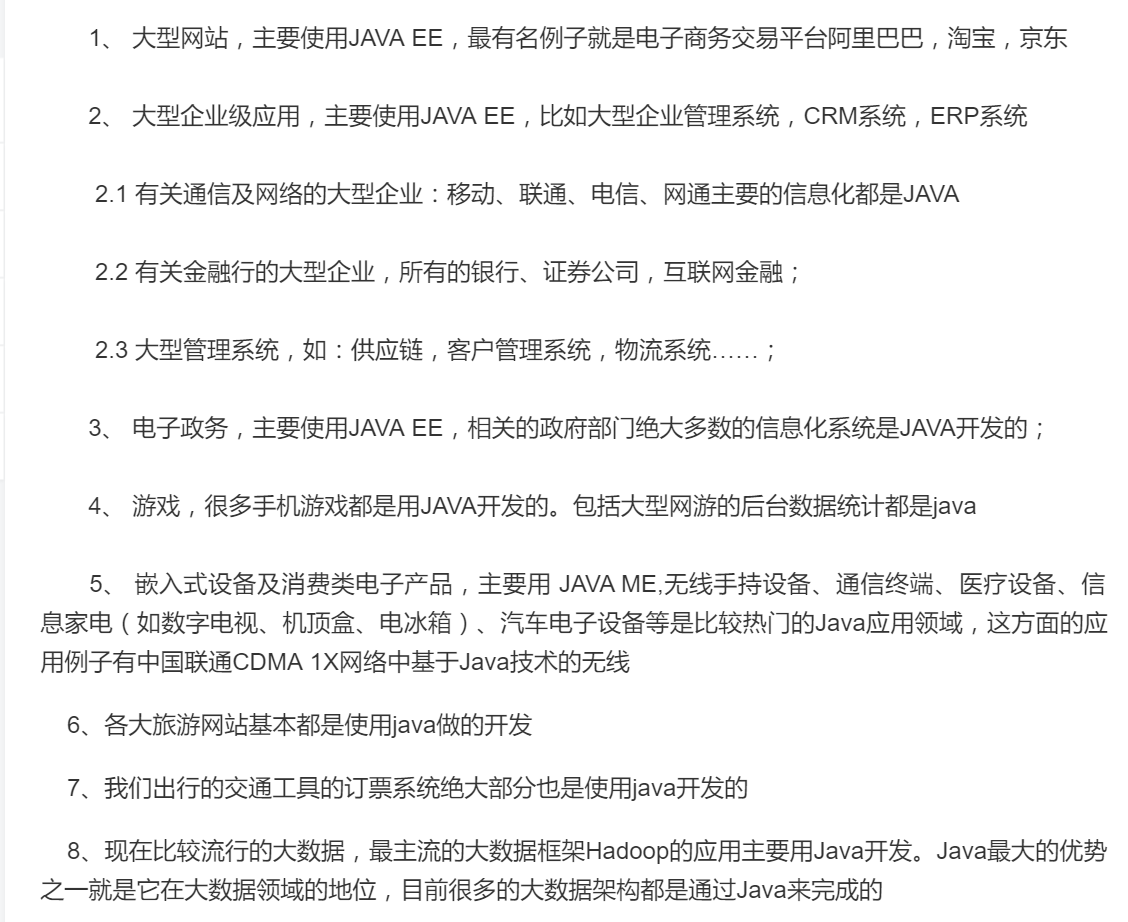
[10.什么是RPC？ 17](#_Toc523235137)

[11.什么是微服务 19](#_Toc523235138)

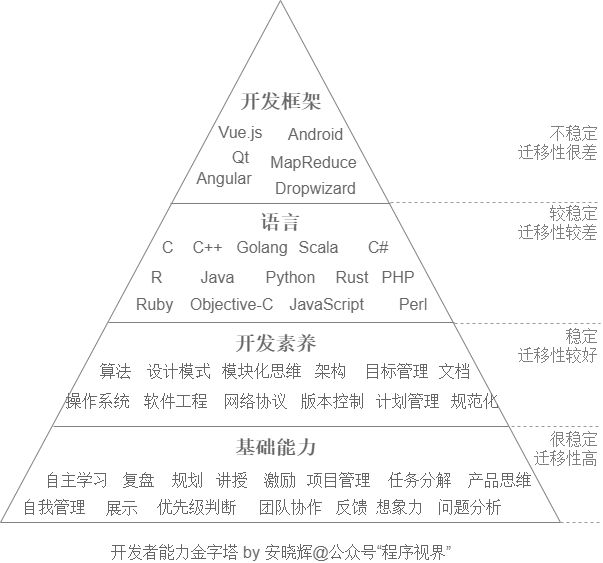
[12.jms通信原理？ 19](#_Toc523235139)

[13.java中的数据结构有哪些？ 20](#_Toc523235140)

[14.微服务架构设计图 21](#_Toc523235141)



**·**Java发展方向表



·语言学习主要程度表

1. **mybatis是如何管理session和cache的。**

   answer:  
        每当我们使用MyBatis开启一次和数据库的会话，MyBatis会创建出一个SqlSession对象表示一次数据库会话。

      在对数据库的一次会话中，我们有可能会反复地执行完全相同的查询语句，如果不采取一些措施的话，每一次查询都会查询一次数据库,而我们在极短的时      
      间内做了完全相同的查询，那么它们的结果极有可能完全相同，由于查询一次数据库的代价很大，这有可能造成很大的资源浪费。

          为了解决这一问题，减少资源的浪费，MyBatis会在表示会话的SqlSession对象中建立一个简单的缓存，将每次查询到的结果结果缓存起来，当下次查询  
    的时候，如果判断先前有个完全一样的查询，会直接从缓存中直接将结果取出，返回给用户，不需要再进行一次数据库查询了。  
  
        一个SqlSession对象中创建一个本地缓存(local cache)，对于每一次查询，都会尝试根据查询的条件去本地缓存中查找是否在缓存中，如果在缓存中，就  
    直接从缓存中取出，然后返回给用户；否则，从数据库读取数据，将查询结果存入缓存并返回给用户。  
        实际上, MyBatis只是一个MyBatis对外的接口，SqlSession将它的工作交给了Executor执行器这个角色来完成，负责完成对数据库的各种操作。当创建了  
    一个SqlSession对象时，MyBatis会为这个SqlSession对象创建一个新的Executor执行器，而缓存信息就被维护在这个Executor执行器中，MyBatis将缓存  
    和对缓存相关的操作封装成了Cache接口中。由于Session级别的一级缓存实际上就是使用PerpetualCache维护的,其内部就是通过一个简单的HashMap<k,v  
    > 来实现的，没有其他的任何限制。  
        a. MyBatis在开启一个数据库会话时，会 创建一个新的SqlSession对象，SqlSession对象中会有一个新的Executor对象，Executor对象中持有一个新的Pe  
    rpetualCache对象；当会话结束时，SqlSession对象及其内部的Executor对象还有PerpetualCache对象也一并释放掉。

        b. 如果SqlSession调用了close()方法，会释放掉一级缓存PerpetualCache对象，一级缓存将不可用；

        c. 如果SqlSession调用了clearCache()，会清空PerpetualCache对象中的数据，但是该对象仍可使用；

**d.SqlSession中执行了任何一个update操作(update()、delete()、insert()) ，都会清空PerpetualCache对象的数据，但是该对象可以继续使用；**

# 2.常见序列化协议及其优缺点

    **answer:  
     <1>protobuf——protobuf的优点是占用空间小，序列化反序列化快；缺点是可读性不行；而网络之间数据通信数据越少越好，响应速度越快越好；反而对这个可读性要求不高；所以我认为用于跨网络之间数据通信比较合适；**

    <2>json——json的优点是可读性比xml稍差比protobuf好，空间和速度比protobuf差，比xml好；而局域网和同机器不同系统之间通信对性能要求不是特别高，对可读性有一定要求，这样可以方便调试；所以我任务用于局域网和同机器不同系统之间通信比较合适，或者跨语言接口数据封装；

     <3>xml——xml的优点是可读性不错，非常符合人类思维；但是效率不好；而存储的数据对可读性要求比较高，对效率不高，因为不会经常读写；所以我认为xml用于数据存储比较合适；

**3.网站高并发大流量访问的处理及解决方法**

**1.硬件升级**

普通的P4服务器一般最多能支持每天10万独立IP，如果访问量比这个还要大， 那么必须首先配置一台更高性能的专用服务器才能解决问题 ，否则怎么优化都不可能彻底解决性能问题。

**2.负载均衡**

它是根据某种负载策略把请求分发到集群中的每一台服务器上，让整个服务器群来处理网站的请求。   
公司比较有钱的，可以购买专门负责负载均衡的硬件（如：F5）,效果肯定会很好。对于大部分公司，会选择廉价有效的方法扩展整个系统的架构，来增加服务器的吞吐量和处理能力，以及承载能力。

**3.服务器集群**

服务器集群就是指将N台服务器集中起来一起进行同一种服务，它们之间通过网络实现通信。让N台服务器之间相互协作，共同承载一个网站的请求压力。   
在客户端看来就像是只有一个服务器。集群可以利用多个计算机进行并行计算从而获得很高的计算速度，也可以用多个计算机做备份，从而使得任何一个机器坏了整个系统还是能正常运行。

**4.数据库读写分离**

基本的原理是让主数据库处理事务性增、改、删操作（INSERT、UPDATE、DELETE），而从数据库处理SELECT查询操作。

**5.数据库分表技术（垂直分割，水平分割）**

当一张的数据达到几百万时，你查询一次所花的时间会变多，如果有联合查询的话，很有可能会死在那儿了。分表的目的就在于此，减小数据库的负担，缩短查询时间。

**6.表建立相应的索引**

使用索引可快速访问数据库表中的特定信息。

**7.页面静态化**

前台实现完全的静态化最好，可以完全不用访问数据库。

**8.缓存技术（MemCache、Redis）**

缓存技术就是另一个解决方案，就是将动态数据存储到缓存文件中，动态网页直接调用 这些文件，而不必再访问数据库

**9.禁止外部盗链**

外部网站的图片或者文件盗链往往会带来大量的负载压力，因此应该严格限制外部对于自身的图片或者文件盗链

**10.控制大文件的下载**

**大文件的下载会占用很大的流量，并且对于非SCSI硬盘来说，大量文件下载会消耗 CPU，使得网站响应能力下降。因此，尽量不要提供超过2M的大文件下载，如果需要提供，建议将大文件放在另外一台服务器上。**

# 4.Reactor多线程模型的特点

1.2.1. 单线程模型

Reactor单线程模型，指的是所有的IO操作都在同一个NIO线程上面完成，NIO线程的职责如下：

1）作为NIO服务端，接收客户端的TCP连接；

2）作为NIO客户端，向服务端发起TCP连接；

3）读取通信对端的请求或者应答消息；

4）向通信对端发送消息请求或者应答消息。

Reactor单线程模型示意图如下所示：

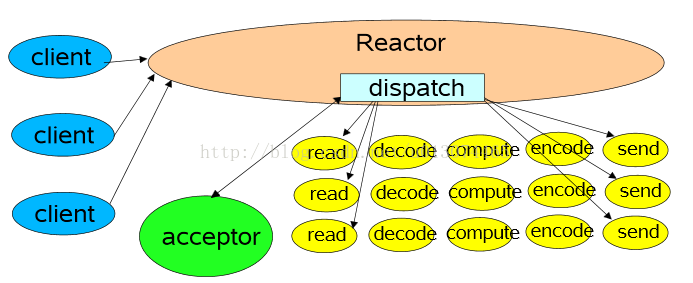


图1-1 Reactor单线程模型

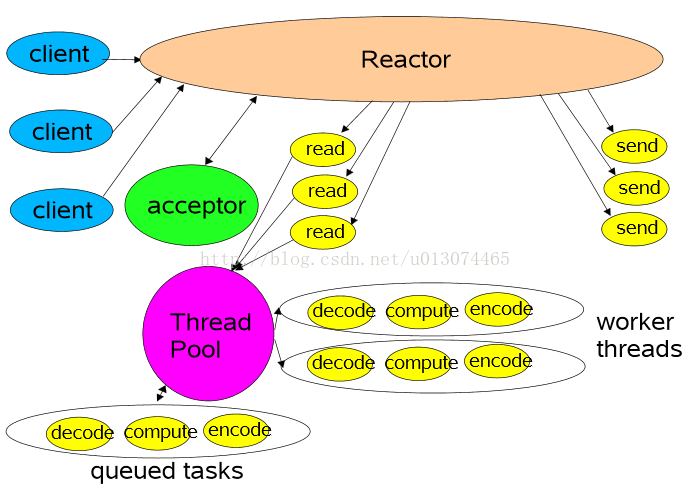
由于Reactor模式使用的是异步非阻塞IO，所有的IO操作都不会导致阻塞，理论上一个线程可以独立处理所有IO相关的操作。从架构层面看，一个NIO线程确实可以完成其承担的职责。例如，通过Acceptor类接收客户端的TCP连接请求消息，链路建立成功之后，通过Dispatch将对应的ByteBuffer派发到指定的Handler上进行消息解码。用户线程可以通过消息编码通过NIO线程将消息发送给客户端。

Reactor多线程模型的特点：

1）有专门一个NIO线程-Acceptor线程用于监听服务端，接收客户端的TCP连接请求；

2）网络IO操作-读、写等由一个NIO线程池负责，线程池可以采用标准的JDK线程池实现，它包含一个任务队列和N个可用的线程，由这些NIO线程负责消息的读取、解码、编码和发送；

3）1个NIO线程可以同时处理N条链路，但是1个链路只对应1个NIO线程，防止发生并发操作问题。

在绝大多数场景下，Reactor多线程模型都可以满足性能需求；但是，在极个别特殊场景中，一个NIO线程负责监听和处理所有的客户端连接可能会存在性能问题。例如并发百万客户端连接，或者服务端需要对客户端握手进行安全认证，但是认证本身非常损耗性能。在这类场景下，单独一个Acceptor线程可能会存在性能不足问题，为了解决性能问题，产生了第三种Reactor线程模型-主从Reactor多线程模型。  


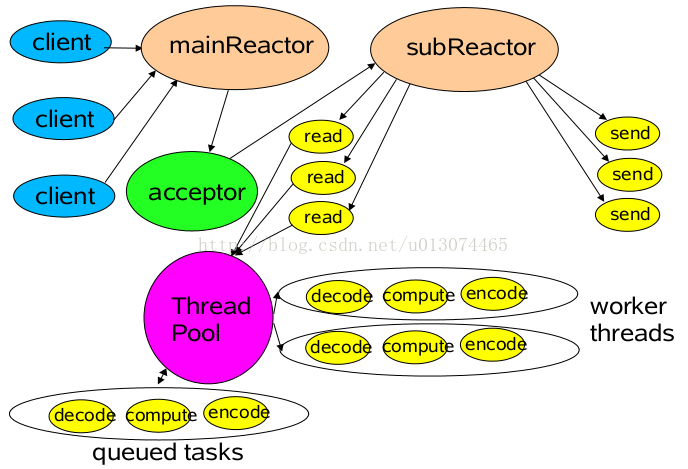
主从多线程模型  
主从Reactor线程模型的特点是：服务端用于接收客户端连接的不再是个1个单独的NIO线程，而是一个独立的NIO线程池。Acceptor接收到客户端TCP连接请求处理完成后（可能包含接入认证等），将新创建的SocketChannel注册到IO线程池（sub reactor线程池）的某个IO线程上，由它负责SocketChannel的读写和编解码工作。Acceptor线程池仅仅只用于客户端的登陆、握手和安全认证，一旦链路建立成功，就将链路注册到后端subReactor线程池的IO线程上，由IO线程负责后续的IO操作。

它的工作流程总结如下：

从主线程池中随机选择一个Reactor线程作为Acceptor线程，用于绑定监听端口，接收客户端连接；

Acceptor线程接收客户端连接请求之后创建新的SocketChannel，将其注册到主线程池的其它Reactor线程上，由其负责接入认证、IP黑白名单过滤、握手等操作；

步骤2完成之后，业务层的链路正式建立，将SocketChannel从主线程池的Reactor线程的多路复用器上摘除，重新注册到Sub线程池的线程上，用于处理I/O的读写操作

1. 

# 5.阻塞、非阻塞、同步、异步区别

 1.同步与异步

同步和异步关注的是消息通信机制 (synchronous communication/ asynchronous communication)  
所谓同步，就是在发出一个\*调用\*时，在没有得到结果之前，该\*调用\*就不返回。但是一旦调用返回，就得到返回值了。  
换句话说，就是由\*调用者\*主动等待这个\*调用\*的结果。

而异步则是相反，\*调用\*在发出之后，这个调用就直接返回了，所以没有返回结果。换句话说，当一个异步过程调用发出后，调用者不会立刻得到结果。而是在\*调用\*发出后，\*被调用者\*通过状态、通知来通知调用者，或通过回调函数处理这个调用。

2. 阻塞与非阻塞  
阻塞和非阻塞关注的是程序在等待调用结果（消息，返回值）时的状态.

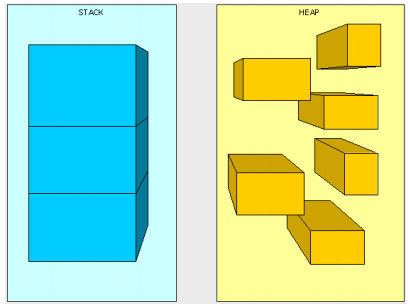
阻塞调用是指调用结果返回之前，当前线程会被挂起。调用线程只有在得到结果之后才会返回。

**非**阻塞调用指在不能立刻得到结果之前，该调用不会阻塞当前线程。 

# 6.堆（heap）与栈（stack）

    堆和栈是程序运行的关键，很有必要它他们的关系说清楚。

    在java中，Main函数就是栈的起始点，也是程序的起始点。程序要运行总是有一个起点的（程序执行的入口）。



**概括：**

**1  栈是运行时的单位 , 而堆是存储的单元**。

 2 **栈解决程序的运行问题，即程序如何执行，或者说如何处理数据，**

**堆解决的是数据存储的问题，即数据怎么放，放在哪儿**。

在**java**中一个线程就会相应有一个线程栈与之对应，这点很容易理解，因为不同的线程执行逻辑有所不同，因此需要一个独立的线程栈。

而堆则是所有线程共享的。

**疑问一：为什么要把堆和栈区分出来呢？栈中不是也可以存储数据吗？**

     1. 从软件设计的角度看，栈代表了处理逻辑，而堆代表了数据。这样分开，使得处理逻辑更为清晰。分而治之的思想。

        这种隔离、模块化的思想在软件设计的方方面面都有体现。

     2.堆与栈的分离，使得堆中的内容可以被多个栈共享（也可以理解为多个线程访问同一个对象）。

        好处:  a.提供了一种有效的数据交互方式（如：共享内存）

                 b.堆中的共享常量和缓存可以被所有栈访问，节省了空间。

     3. 栈因为运行时的需要，比如保存系统运行的上下文，需要进行地址段的划分。

        由于栈只能向上增长，因此就会限制住栈存储内容的能力，

        而堆不同，堆中的对象是可以根据需要动态增长的，

        因此栈和堆的拆分使得动态增长成为可能，相应栈中只需记录堆中的一个地址即可。

     4. 面向对象就是堆和栈的完美结合。

        其实，面向对象方式的程序与以前结构化的程序在执行上没有任何区别。

        但是，面向对象的引入，使得对待问题的思考方式发生了改变，而更接近于自然方式的思考。

        当我们把对象拆开，你会发现，对象的属性其实就是数据，存放在堆中；

        而对象的行为（方法），就是运行逻辑，放在栈中。

        我们在编写对象的时候，其实就是编写了数据结构，也编写了处理数据的逻辑。不得不承认，面向对象的设计，确实很美。

**疑问二：**  **堆中存什么？栈中存什么？**

      1. 栈存储的信息都是跟当前线程（或程序）相关的信息。(局部变量、程序运行状态、方法、方法返回值)等，

         栈中存的是基本数据类型和堆中对象的引用。一个对象的大小是不可估计的，或者说是可以动态变化的，但是

         在栈中，一个对象只对应了一个4byte的引用（堆栈分离的好处）。

     2. 堆只负责存储对象信息。

**疑问三：  为什么不把基本类型放堆中呢？**

     1. 其占用的空间一般是1~8个字节---需要空间比较少，

     2.而且因为是基本类型，所以不会出现动态增长的情况---长度固定，因此栈中存储就够了，如果把它存在堆中是没有什么意义的（还会浪费空间，后面说明??）。

**疑问四：**  **java中的参数传递是传值呢？还是传引用？**

     对象传递是引用值传递，原始类型数据传递是值传递

     实际上这个传入函数的值是对象引用的拷贝，即传递的是引用的地址值，所以还是按值传递

     tips：

      堆和栈中，栈是程序运行最根本的东西。程序运行可以没有堆，但是不能没有栈。

      而堆是为栈进行数据存储服务的，说白了堆就是一块共享的内存。

      不过，正是因为堆和栈的分离的思想，才使得**java**的垃圾回收成为可能。

**java**中，栈的大小通过-Xss来设置，当栈中存储的数据比较多时，需要适当调大这个值，否则会出现 **java**.lang.StackOverflowError异常。

  常见的出现这个异常的是无法返回的递归，因为此时栈中保存的信息都是方法返回的记录点。

# 7.mybatis数据库数据处理

        映射方式：List<hashmap<String,Object>> list<map1>  
        字段字符串：{a,b,c} sql=a,b,c  
        生成查询：select sql from table\_name  
        获取字段值：val1 = map1.get(a)  
        生成map：Map.put(sql1,val1)

# 8.java反射机制

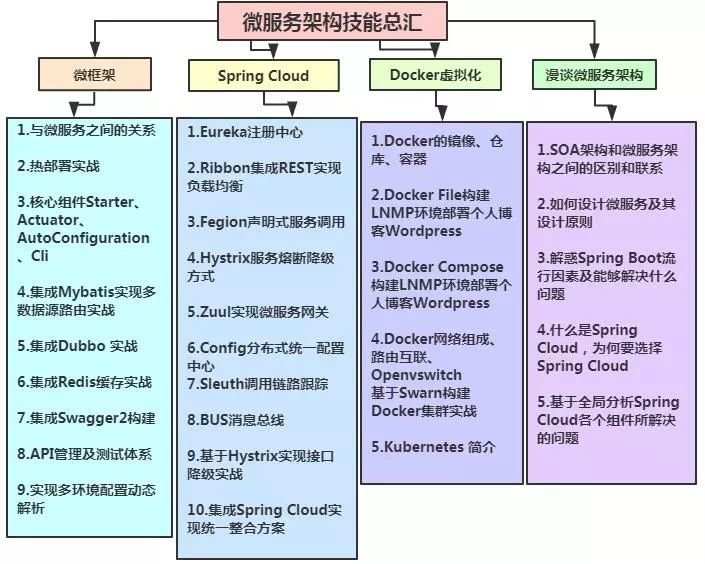
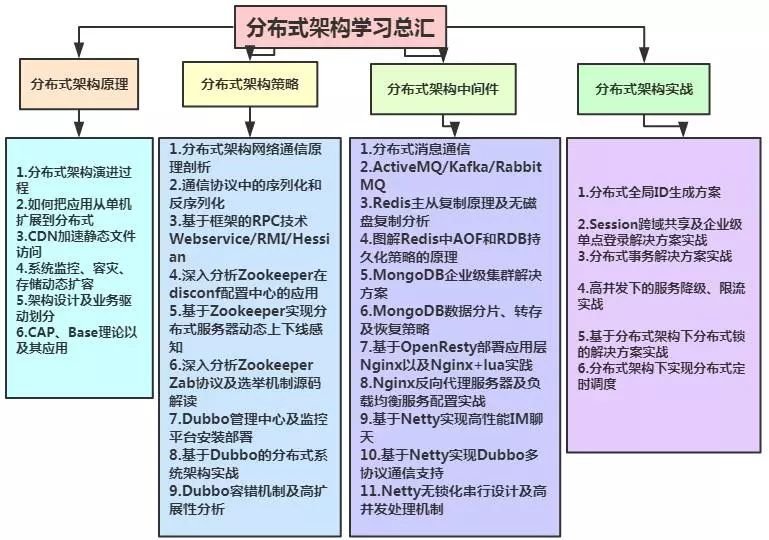
 Java反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为Java语言的反射机制。  
1.关于Class

        1、Class是一个类，一个描述类的类（也就是描述类本身），封装了描述方法的Method，描述字段的Filed，描述构造器的  
            Constructor等属性  
        2、对象照镜子后（反射）可以得到的信息：某个类的数据成员名、方法和构造器、某个类到底实现了哪些接口。  
        3、对于每个类而言，JRE 都为其保留一个不变的 Class 类型的对象。  
            一个 Class 对象包含了特定某个类的有关信息。  
        4、Class 对象只能由系统建立对象

    5、一个类在 JVM 中只会有一个Class实例   
2.获取类的三种方法  
    通过类名: class com.java.reflection.Person无参构造器

        通过getClass(): class com.java.reflection.Person  
        通过全类名获取: class com.java.reflection.Person  
     3.利用newInstance创建对象：调用的类必须有无参的构造器  
    4.ClassLoader类加载器

# 9.高级工程师：微服务与分布式

# 10.什么是RPC？

Remote Procedure Calls  
远程过程调用 (RPC) 是一种协议，程序可使用这种协议向网络中的另一台计算机上的程序请求服务。由于使用 RPC 的程序不必了解支持通信的网络协议的情况，因此 RPC 提高了程序的互操作性。在 RPC 中，发出请求的程序是客户程序，而提供服务的程序是服务器。  
  
RPC（远程过程调用）是一项广泛用于支持分布式应用程序（不同组件分布在不同计算机上的应用程序）的技术。RPC 的主要目的是为组件提供一种相互通信的方式，使这些组件之间能够相互发出请求并传递这些请求的结果。

|  |  |  |
| --- | --- | --- |
| 技术 | 简介 | 是否支持跨平台 |
| Corbra | 90年代产物，已被淘汰 | 不支持 |
| RMI | EJB时代产物，已逐渐被淘汰 | 不支持 |
| WebService | 基于Http SOAP，效率低，逐渐被淘汰 | 支持 |
| Hessain | 基于Http，二进制序列化，效率高，使用广泛 | 支持 |
| Rest（spring mvc等） | 支持Http Rest，广泛应用于无线API，开放平台等 | 支持 |
| JMS、开源MQ | Java消息服务（消息中间件），使用广泛 | 支持 |
| Socket | 基于Mina、Netty（NIO、AIO高效通信） | 理论上支持 |

**互联网时代常见的RPC技术和框架**

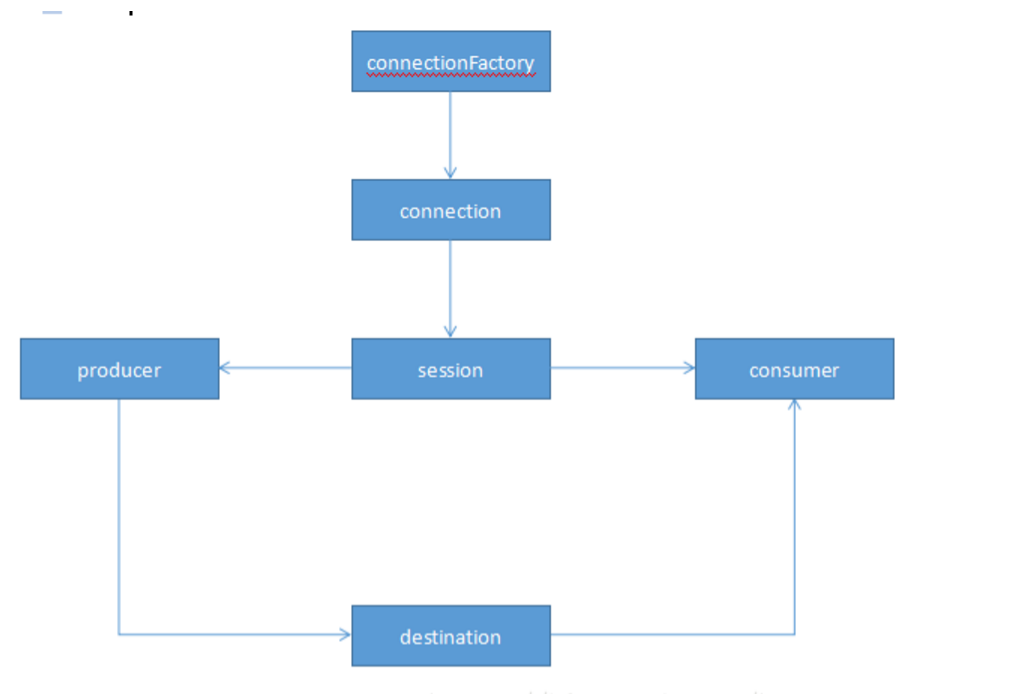
应用级的服务框架：  
Dubbo/Dubbox  
ZeroICE  
GRpc  
Spring Boot/Spring Cloud  
  
基础通信框架：  
Protocol Buffers  
Thrift

远程通信协议：  
RMI  
Socket  
SOAP(HTTP XML)  
REST(HTTP JSON)

# 11.什么是微服务

**微服务英文名称Microservice，Microservice架构模式就是将整个Web应用组织为一系列小的Web服务。这些小的Web服务可以独立地编译及部署，并通过各自暴露的API接口相互通讯。它们彼此相互协作，作为一个整体为用户提供功能，却可以独立地进行扩。**

# 12.jms通信原理？



JMS即[Java消息服务](https://baike.baidu.com/item/Java%E6%B6%88%E6%81%AF%E6%9C%8D%E5%8A%A1" \t "_blank)（Java Message Service）应用程序接口，是一个[Java平台](https://baike.baidu.com/item/Java%E5%B9%B3%E5%8F%B0" \t "_blank)中关于面向[消息中间件](https://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E4%B8%AD%E9%97%B4%E4%BB%B6/5899771)（MOM）的[API](https://baike.baidu.com/item/API/10154" \t "_blank)，用于在两个应用程序之间，或[分布式系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E7%B3%BB%E7%BB%9F/4905336" \t "_blank)中发送消息，进行异步通信。Java消息服务是一个与具体平台无关的API，绝大多数MOM提供商都对JMS提供支持（摘自百度百科） 。

connectionFactory：连接工厂，用于创建连接

connection：连接，由连接工厂创建，表示客户端与服务端的连接

session：会话，客户端与服务端的一次连接称为一次会话

producer：生产者，一次会话中，生产者发送消息到destination

consumer：消费者，一次会话中，消费者从destination中获取消息

destination：目的地，存储生产者和消费者之间传递的消息，分为队列queue和主题topic。

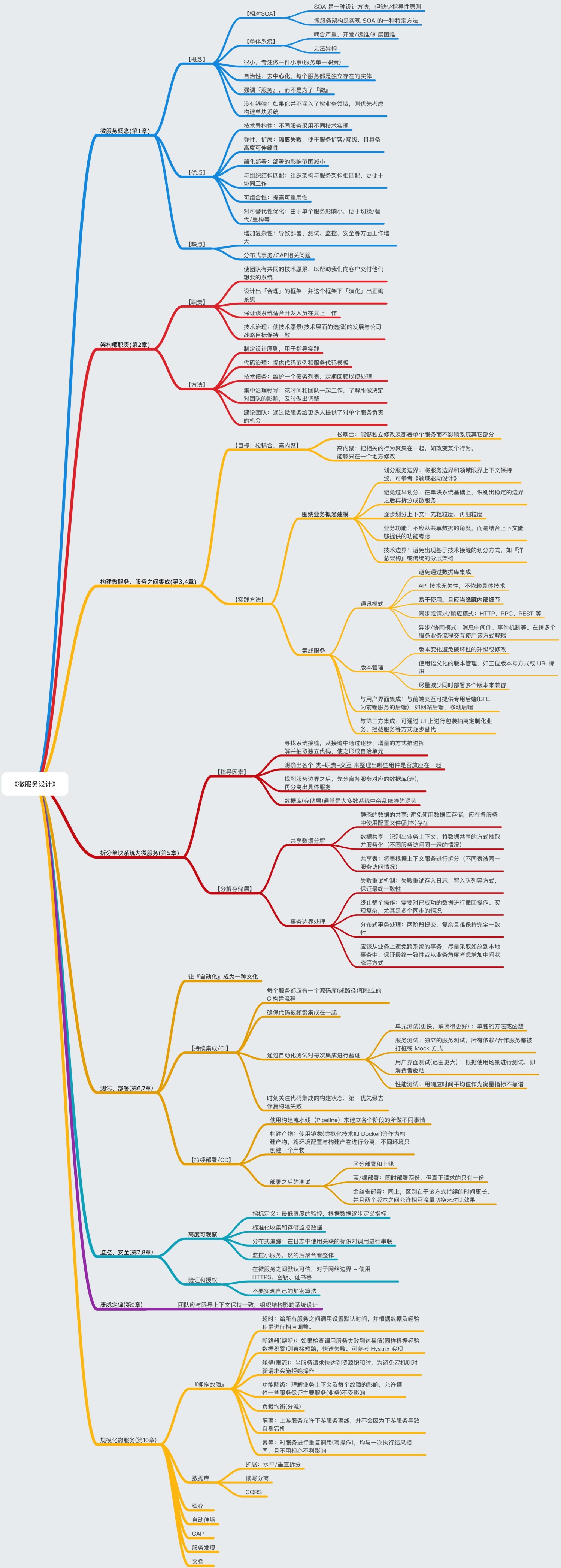
点对点（queue）：一个生产者发送消息，只有一个消费者消费消息，发送或接收时，对方都不需要保持在线。

订阅发布（pub/sub）：一个生产者发送消息，只要订阅了该主题的消费者都可以接收到消息，但是在生产者发送消息的时候，消费者要保持在线状态。

# 13.java中的数据结构有哪些？



# 14.微服务架构设计图



# 15.