A:rdd五大特性

Q:分区列表（getPartitions），每一个分区都有一个计算函数（compute），依赖于其他rdd的列表（getDependecies），分区器（针对于key-value形式，对应属性为partitioner），每个分区都有一个优先位置列表（getPreferLocation）（即为4方法一属性）

A:RDD的弹性性质体现在哪些方面？

Q:1.自动进行内存和磁盘的数据存储的切换2.基于Lineage的高效容错机制3.task如果失败会自动进行重试4,stage失败会自动进行重试5.checkpoint和persist，可主动或被动触发6，数据调度弹性7数据分片的高度弹性。总结起来，所谓弹性就是固定，多种选择，主主要体现在**数据的存储，数据的计算，数据的调度，数据的获取，任务的容错，存储方式多种选择，数据调度的多种选择比如分片，容错，节点失败后自动重试或者切换节点继续计算。数据的获取比如创建rdd 的方式也是多种的（集合，文件，父rdd）**

Q:join是shuffle操作么？

A:如果每个分区仅仅和特定的分区进行join，那么就是窄依赖

Q:为什么要使用lazy特性

A:因为lazy特性可以使spark观察到很多运算步骤，观察到的越多，对其进行优化的越多，比如DAG生成就对容错和重计算提供了很大的帮助。

Q:哪些操作发生Driver上？

A:DAG划分，一些元信息的管理，通信，优化操作

Q:为什么action操作可以触发job

A:

Q：宽依赖和窄依赖

A：最简单直接的方法就是父子分区，如果子分区对于多个不同父分区就是宽，反之亦然，可以有没有想过为什么要分窄依赖和宽依赖，没有行不行？如果从计算的角度上来说，我认为是行的，它也是可以计算出结果，但是从容错上以及性能上会大大的打折扣，甚至违背了spark的初衷，为什么？宽依赖和窄依赖在代码里是作为stage划分的依据，而stage是什么？他就像一个大函数，把一些列map操作，或者无需shuffle的操作进行封装，实现最大化pipline，除此之外，在容错上也可以实现较粗粒度的划分，不至于牵一发而动全身。其实spark在计算环节做的很多工作都是围绕一个原则，就是最大化的pipline，知道我为什么说行么，其实仅仅考虑计算的话，确实可以计算一步通信一次，但是然后持久化话一次，下一步计算然后在通信。。。这渐渐的不就是还原成mapreduce了么？MapReduce当然是可行的，可是好不好呢，不好。频繁的网络IO和磁盘IO会造成资源利用不高。

Q:窄依赖又分什么？

A:窄依赖又分为RangeDep和OneToOneDep，分别对应union操作和普通map操作。宽依赖及时shuffledep

Q:DAG的生成机制

A:对于源代码层面，可以关注DAGSchuedular的getParentStages方法。用文字描述就是：生成关键在于回溯，程序提交后，所有的rdd都是被看成一个stage的，注意最开始只有一个stage，在回溯过程中，遇到shuffle就断开，遇到在窄依赖就归并到同一个同一个stage，最后就生成了一个stage划分。

Q:task有几种类型

A:shuffleMaptask和resultmaptask，分别对应shufflemapstage和resultmapstage

Q:rdd的计算过程

1. Driver向executor发送launchTask的通知
2. Exector接收到通知后，会反序列化接收到的taskdes,并去拉去需要的依赖和jar包
3. 创建taskRunner对象，并放入线程池
4. 通过线程池运行任务
5. 其实task本质就是一个线程，运行任务就是调用task的run方法
6. 线程真正的业务逻辑是要落实到rdd上，最终要调用rdd的iterator方法，在iteratior方法会调用真正的计算业务

7.对于ResultMapTask，MapOutputTracker会把情面shffleMapTask的计算结果给ResultMapTask,便于finalStage的计算和最终结果的输出

Q:缓存级别是如何影响计算的

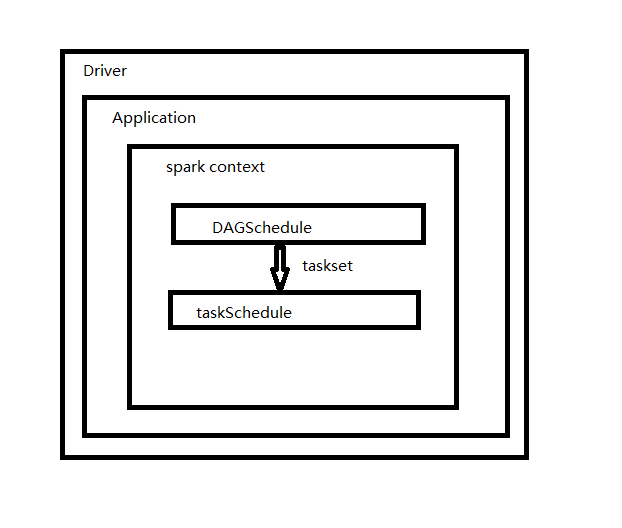
A:这个其实可以看iterator方法就一目了然，他会根据根据存储级别来判定是要重新计算或者从缓存里获取数据又或者使用checkpoint

Q:总结一下rdd的容错原理

A：stage和task失败，依托lineage进行重算，缓存和checkpoint的使用

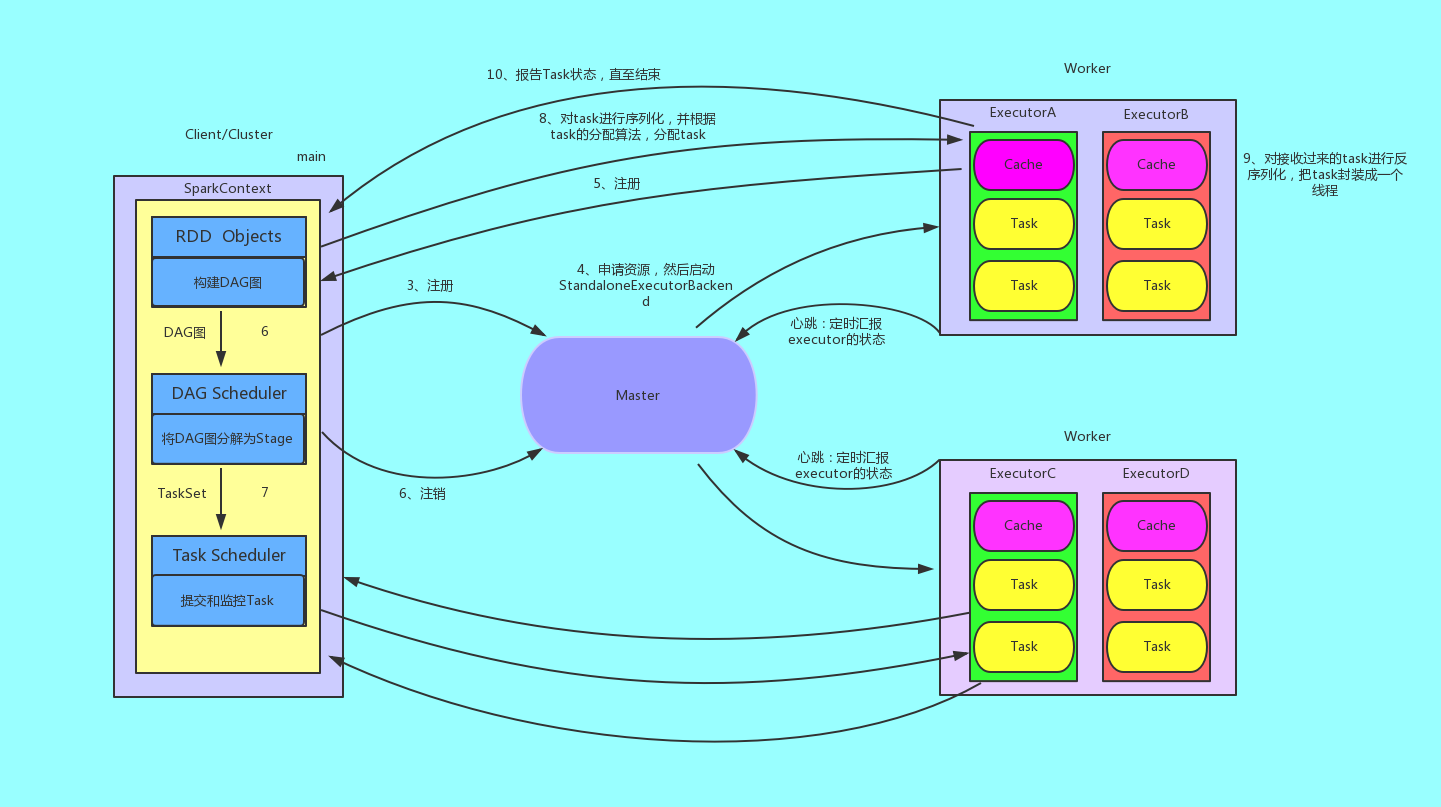
### Q: Spark on Standalone运行过程

A:首先需要去除driver在哪？首先需要知道有client和master的概念，client不一定在master上，一般的如果不是在master上提交，一般提交的那一台机器就是client，对于spark-shell，client即是master



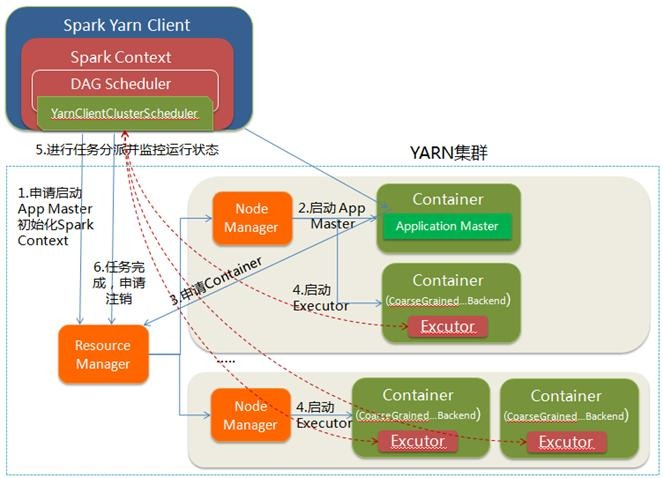
一般而言master机器做管理和资源调度，比如存储一些数据的元信息，分配节点，维护节点（心跳）启动executor

具体任务的分配还是交由driver端

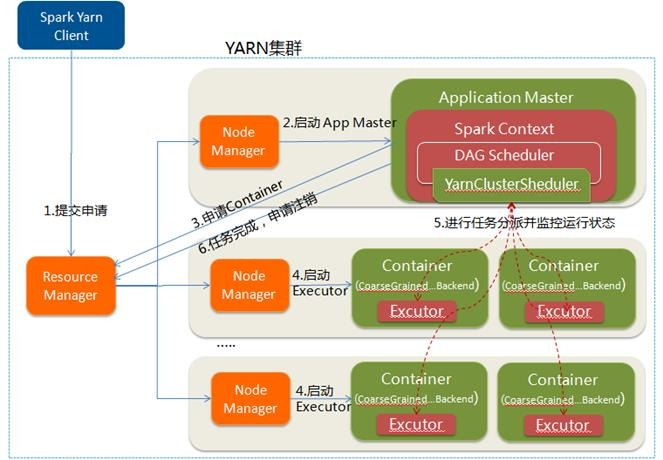


**对于yarn集群，关键是要对yarn调度需要熟悉，运行流程其实还是跟standlone很像的。清楚client模式和cluster模式在什么情况下使用。**

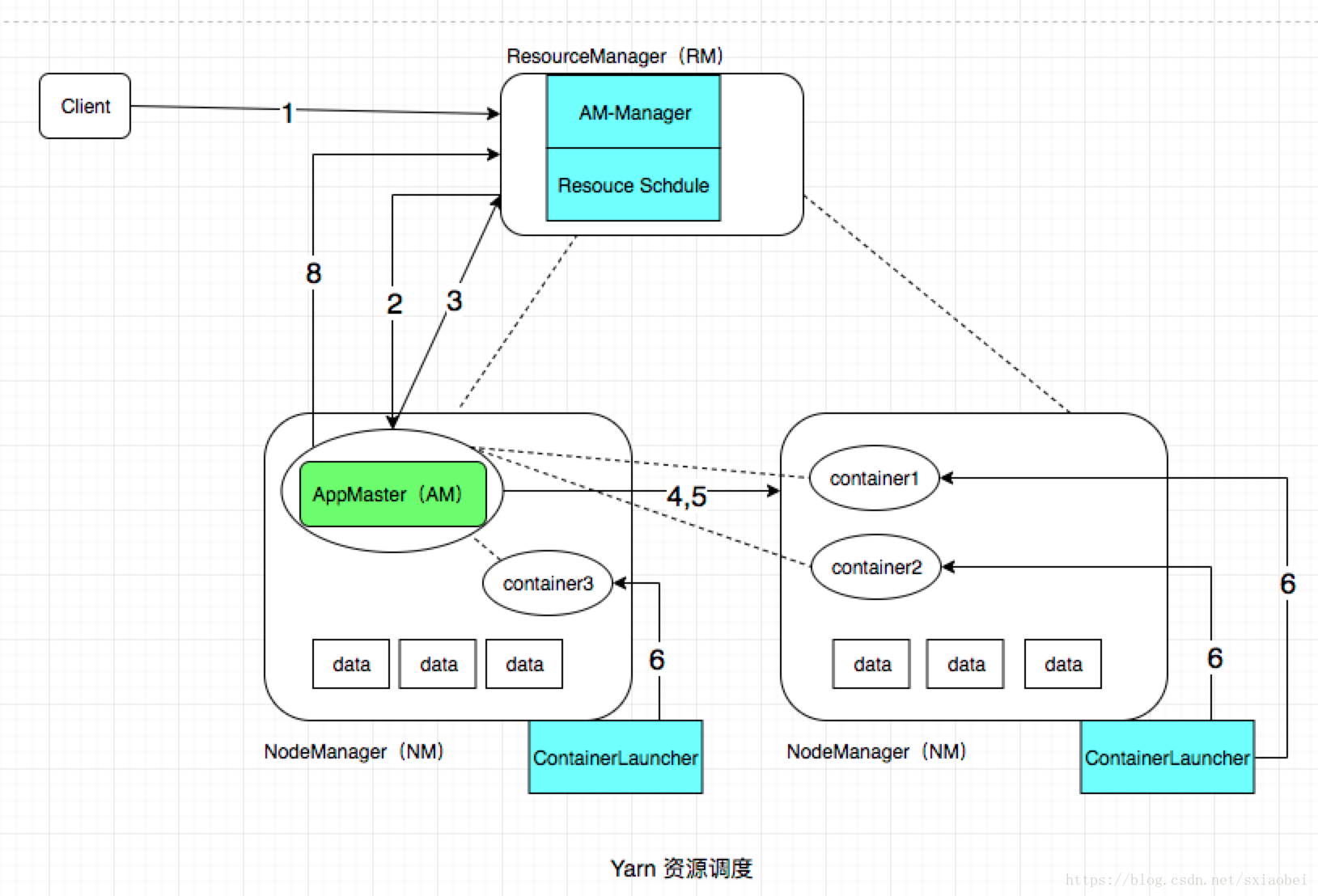
**对于yarn来说有ResourceManager和NodeManager，spark on yarn的client模式比较大的区别就是需要先想resoucemanager申请一个节点充当master的作用，也就是application master，有这个master想resource manger去申请分配资源然后告诉driver，合格driver还在你提交的机器上（强调这句话的目的是因为集群模式下，这里就会不同）**



**集群模式区别在于，driver端不会在你提交的额机器上，而是有resourcemanager为drvier分配一个节点作为driver，所有的程序都跑在集群上，有点像yarn的spark-shell模式：master和driver在一起了**



**如果真的要细细研究的话**



Resource manager里面还可以在分出application master manager负责application master的

1. 提交任务到resource manager
2. Rs收到请求之后，调用application master manager，向node manager发送请求，申请资源并且要求container启动application master
3. application master 启动成功后，然后为自己的task的申请资源，这个时候只要依赖resource shed
4. task资源申请ok后，am会要求地洞task
5. task开始根据参数启动任务，拉去相应的资源文件和数据
6. 任务进行，task不断向am报告状态

Q：Driver到底是什么？

A:其实Driver就是主函数main

Q:为什么Key不可以为Array等复杂类型？

A:因为难以在复杂类型上定义分区函数（这个可能要从分区函数自定义上去研究哈）

Q:Job触发流程

A：首先需要一个actio=n算子，为什么呢，因为action算子有个方法叫做runJob，这就是为什么只有在进行action操作的时候才会触发job

Q:DAGScheduler,SchedulerBackend和底层调度器TaskScheduler的初始化顺序

A:DAG是最后初始化的

Q:stage划分算法

A:在正式划分之前看成只有一个rdd和一个stage，就叫他finalrdd和finalstage，根据finalrdd的依赖判断是否是宽依赖还是窄依赖，如果是宽依赖就是看这个stage是否已经存在，如果存在就取出来放大一个set里面，如果不存在就创建一个，如果是窄依赖就把该rdd的父rdd放入等待检查的栈里面，继续这个过程，最终这个set里面就是所有的stage，然后转换为列表返回（至于为什么转化为列表返回，我的猜想是，还是考虑到性能，因为set里面用的是链表等对象，占用内存比较大，而且这个对象应该会进行网络传输，相比之下转为list在性能上会比较好）

Q:任务默认的调度模式是

A:FIFO

Q:TaskScheduler原理

A:在DAG划分完stage后，会按照stage的先后顺序依次提交给底层的TaskScheduler，一个stage就是一个taskSet啦，也是用到了pipline的思想。TaskSchedulerImpl会初始化一个TaskSetManager对其生命周期进行管理，负责发送具体的任务到executor上，处理task任务失败的情况，比如将失败的任务重新加入到队列中（默认是四次），这里面还有两种调度模式不知道重不重要，FIFO和Fair模式

Q:spark通信机制

A:有所了解，但是研究不深，大概知道用到了netty，rpc等等，而且版本上有点不同，早的版本大概用的是akka，现在好像不用了。

Q:Executor启动有哪几种策略

两种，一种是轮流均摊策略，采用圆桌算法依次轮流均摊，直到满足资源需求，轮流均摊策略通常会有更好的数据本地性，因此它是默认的选择策略。第二种是依次全占，在可用的worker上一次获取每个worker的全部资源。

Q:可以启动多个executor么？

**A:理论上可以配置executor核数和worker核心数来实现，但实际情况不一定能够实现，这个要了解executor启动流程。（这个暂时放一下，因为感觉好像不是很重要）**

Q：shuffle有几种方式

A:两种，基于Hash和基于sort

Q:shuffled 对比？

A:虽然在最开始的时候spark使用hash作为默认的实现方式，但是这种方式会产生文件数量不可控的情况，最初会为reduce端每一个任务创建一个文件，后来优化引入了文件合并机制，仍然是治标不治本，文件的数量还是会跟reduce端有关，无奈，开始研究基于sort的shuffle机制，这种机制下只会生成两个文件，一个数据文件，一个索引文件，（其实这样的话会不会存在单个文件过大的呢？）

Q:Tungsten

A:没有银弹，Tungsten计划启动，发现spark的瓶颈在于内存和cpu，tungsten计划主要是从内存管理模型比如对象的存储方式，对象头什么的

 **Memory Management and Binary Processing**: off-heap管理内存，降低对象的开销和消除JVM GC带来的延时。

 **Cache-aware computation**: 优化存储，提升CPU L1/ L2/L3缓存命中率。

 **Code generation**: 优化Spark SQL的代码生成部分，提升CPU利用率。

性能优化

Q:配置的生效机制

A:代码》submit》参数文件》默认值

Q:有哪些编码技巧

A:重复使用的rdd一定要缓存复用，尽早过滤掉不需要的数据，减少在shuffle阶段的数据量，尽量用高效率算子如sortBykey，groupBykey,他们会进行本地combine，mapPartition，可以一个分区共用一个对象，特别在使用数据库连接的时候。还有reparationAndSortWithinpartition,广播变量的使用，序列化，使用优化的数据结构，比如字符串来表示对象

参数，虚拟机之类的，就要根据实际情况来调整了。