问题一：RabbitMQ 中的 broker 是指什么？cluster 又是指什么？

答：broker 是指一个或多个 erlang node 的逻辑分组，且 node 上运行着 RabbitMQ 应用程序。cluster 是在 broker 的基础之上，增加了 node 之间共享元数据的约束。

问题二：什么是元数据？元数据分为哪些类型？包括哪些内容？与 cluster 相关的元数据有哪些？元数据是如何保存的？元数据在 cluster 中是如何分布的？

答：在非 cluster 模式下，元数据主要分为 Queue 元数据（queue 名字和属性等）、Exchange 元数据（exchange 名字、类型和属性等）、Binding 元数据（存放路由关系的查找表）、Vhost 元数据（vhost 范围内针对前三者的名字空间约束和安全属性设置）。在 cluster 模式下，还包括 cluster 中 node 位置信息和 node 关系信息。元数据按照 erlang node 的类型确定是仅保存于 RAM 中，还是同时保存在 RAM 和 disk 上。元数据在 cluster 中是全 node 分布的。

下图所示为 queue 的元数据在单 node 和 cluster 两种模式下的分布图。

问题三：RAM node 和 disk node 的区别？

答：RAM node 仅将 fabric（即 queue、exchange 和 binding 等 RabbitMQ 基础构件）相关元数据保存到内存中，但 disk node 会在内存和磁盘中均进行存储。RAM node 上唯一会存储到磁盘上的元数据是 cluster 中使用的 disk node 的地址。要求在 RabbitMQ cluster 中至少存在一个 disk node 。

问题四：RabbitMQ 上的一个 queue 中存放的 message 是否有数量限制？

答：可以认为是无限制，因为限制取决于机器的内存，但是消息过多会导致处理效率的下降。

问题五：RabbitMQ 概念里的 channel、exchange 和 queue 这些东东是逻辑概念，还是对应着进程实体？这些东东分别起什么作用？

答：queue 具有自己的 erlang 进程；exchange 内部实现为保存 binding 关系的查找表；channel 是实际进行路由工作的实体，即负责按照 routing_key 将 message 投递给 queue 。由 AMQP 协议描述可知，channel 是真实 TCP 连接之上的虚拟连接，所有 AMQP 命令都是通过 channel 发送的，且每一个 channel 有唯一的 ID。一个 channel 只能被单独一个操作系统线程使用，故投递到特定 channel 上的 message 是有顺序的。但一个操作系统线程上允许使用多个 channel 。channel 号为 0 的 channel 用于处理所有对于当前 connection 全局有效的帧，而 1-65535 号 channel 用于处理和特定 channel 相关的帧。AMQP 协议给出的 channel 复用模型如下

其中每一个 channel 运行在一个独立的线程上，多线程共享同一个 socket。

问题六：vhost 是什么？起什么作用？

答：vhost 可以理解为虚拟 broker ，即 mini-RabbitMQ server。其内部均含有独立的 queue、exchange 和 binding 等，但最最重要的是，其拥有独立的权限系统，可以做到 vhost 范围的用户控制。当然，从 RabbitMQ 的全局角度，vhost 可以作为不同权限隔离的手段（一个典型的例子就是不同的应用可以跑在不同的 vhost 中）。

【cluster 相关】

问题七：在单 node 系统和多 node 构成的 cluster 系统中声明 queue、exchange ，以及进行 binding 会有什么不同？

答：当你在单 node 上声明 queue 时，只要该 node 上相关元数据进行了变更，你就会得到 Queue.Declare-ok 回应；而在 cluster 上声明 queue ，则要求 cluster 上的全部 node 都要进行元数据成功更新，才会得到 Queue.Declare-ok 回应。另外，若 node 类型为 RAM node 则变更的数据仅保存在内存中，若类型为 disk node 则还要变更保存在磁盘上的数据。

问题八：客户端连接到 cluster 中的任意 node 上是否都能正常工作？

答：是的。客户端感觉不到有何不同。

问题九：若 cluster 中拥有某个 queue 的 owner node 失效了，且该 queue 被声明具有 durable 属性，是否能够成功从其他 node 上重新声明该 queue ？

答：不能，在这种情况下，将得到 404 NOT_FOUND 错误。只能等 queue 所属的 node 恢复后才能使用该 queue 。但若该 queue 本身不具有 durable 属性，则可在其他 node 上重新声明。

问题十：cluster 中 node 的失效会对 consumer 产生什么影响？若是在 cluster 中创建了 mirrored queue ，这时 node 失效会对 consumer 产生什么影响？

答：若是 consumer 所连接的那个 node 失效（无论该 node 是否为 consumer 所订阅 queue 的 owner node），则 consumer 会在发现 TCP 连接断开时，按标准行为执行重连逻辑，并根据“Assume Nothing”原则重建相应的 fabric 即可。若是失效的 node 为 consumer 订阅 queue 的 owner node，则 consumer 只能通过 Consumer Cancellation Notification 机制来检测与该 queue 订阅关系的终止，否则会出现傻等却没有任何消息来到的问题。

问题十一：能够在地理上分开的不同数据中心使用 RabbitMQ cluster 么？

答：不能。第一，你无法控制所创建的 queue 实际分布在 cluster 里的哪个 node 上（一般使用 HAProxy + cluster 模型时都是这样），这可能会导致各种跨地域访问时的常见问题；第二，Erlang 的 OTP 通信框架对延迟的容忍度有限，这可能会触发各种超时，导致业务疲于处理；第三，在广域网上的连接失效问题将导致经典的“脑裂”问题，而 RabbitMQ 目前无法处理（该问题主要是说 Mnesia）。

【综合问题】

问题十二：为什么 heavy RPC 的使用场景下不建议采用 disk node ？

答：heavy RPC 是指在业务逻辑中高频调用 RabbitMQ 提供的 RPC 机制，导致不断创建、销毁 reply queue ，进而造成 disk node 的性能问题（因为会针对元数据不断写盘）。所以在使用 RPC 机制时需要考虑自身的业务场景。

问题十三：向不存在的 exchange 发 publish 消息会发生什么？向不存在的 queue 执行

consume 动作会发生什么？
答：都会收到 Channel.Close 信令告之不存在（内含原因 404 NOT_FOUND）。

问题十四：routing_key 和 binding_key 的最大长度是多少？
答：255 字节。

问题十五：RabbitMQ 允许发送的 message 最大可达多大？
答：根据 AMQP 协议规定，消息体的大小由 64-bit 的值来指定，所以你就可以知道到底能发多大的数据了。

问题十六：什么情况下 producer 不主动创建 queue 是安全的？
答：1.message 是允许丢失的；2.实现了针对未处理消息的 republish 功能（例如采用 Publisher Confirm 机制）。

问题十七：“dead letter”queue 的用途？
答：当消息被 RabbitMQ server 投递到 consumer 后，但 consumer 却通过 Basic.Reject 进行了拒绝时（同时设置 requeue=false），那么该消息会被放入“dead letter”queue 中。该 queue 可用于排查 message 被 reject 或 undeliver 的原因。

问题十八：为什么说保证 message 被可靠持久化的条件是 queue 和 exchange 具有 durable 属性，同时 message 具有 persistent 属性才行？
答：binding 关系可以表示为 exchange – binding – queue 。从文档中我们知道，若要求投递的 message 能够不丢失，要求 message 本身设置 persistent 属性，要求 exchange 和 queue 都设置 durable 属性。其实这问题可以这么想，若 exchange 或 queue 未设置 durable 属性，则在其 crash 之后就会无法恢复，那么即使 message 设置了 persistent 属性，仍然存在 message 虽然能恢复但却无处容身的问题；同理，若 message 本身未设置 persistent 属性，则 message 的持久化更无从谈起。

问题十九：什么情况下会出现 blackholed 问题？
答：blackholed 问题是指，向 exchange 投递了 message ，而由于各种原因导致该 message 丢失，但发送者却不知道。可导致 blackholed 的情况：1.向未绑定 queue 的 exchange 发送 message；2.exchange 以 binding_key key_A 绑定了 queue queue_A，但向该 exchange 发送 message 使用的 routing_key 却是 key_B。

问题二十：如何防止出现 blackholed 问题？
答：没有特别好的办法，只能在具体实践中通过各种方式保证相关 fabric 的存在。另外，如果在执行 Basic.Publish 时设置 mandatory=true ，则在遇到可能出现 blackholed 情况时，服务器会通过返回 Basic.Return 告之当前 message 无法被正确投递（内含原因 312 NO_ROUTE）。

问题二十一：Consumer Cancellation Notification 机制用于什么场景？
答：用于保证当镜像 queue 中 master 挂掉时，连接到 slave 上的 consumer 可以收到自身 consume 被取消的通知，进而可以重新执行 consume 动作从新选出的 master 出获得消息。若不采用该机制，连接到 slave 上的 consumer 将不会感知 master 挂掉这个事

情，导致后续无法再收到新 master 广播出来的 message 。另外，因为在镜像 queue 模式下，存在将 message 进行 requeue 的可能，所以实现 consumer 的逻辑时需要能够正确处理出现重复 message 的情况。

问题二十二：Basic.Reject 的用法是什么？
答：该信令可用于 consumer 对收到的 message 进行 reject 。若在该信令中设置 requeue=true，则当 RabbitMQ server 收到该拒绝信令后，会将该 message 重新发送到下一个处于 consume 状态的 consumer 处（理论上仍可能将该消息发送给当前 consumer）。若设置 requeue=false ，则 RabbitMQ server 在收到拒绝信令后，将直接将该 message 从 queue 中移除。
另外一种移除 queue 中 message 的小技巧是，consumer 回复 Basic.Ack 但不对获取到的 message 做任何处理。
而 Basic.Nack 是对 Basic.Reject 的扩展，以支持一次拒绝多条 message 的能力。

问题二十三：为什么不应该对所有的 message 都使用持久化机制？
答：首先，必然导致性能的下降，因为写磁盘比写 RAM 慢的多，message 的吞吐量可能有 10 倍的差距。其次，message 的持久化机制用在 RabbitMQ 的内置 cluster 方案时会出现“坑爹”问题。矛盾点在于，若 message 设置了 persistent 属性，但 queue 未设置 durable 属性，那么当该 queue 的 owner node 出现异常后，在未重建该 queue 前，发往该 queue 的 message 将被 blackholed ；若 message 设置了 persistent 属性，同时 queue 也设置了 durable 属性，那么当 queue 的 owner node 异常且无法重启的情况下，则该 queue 无法在其他 node 上重建，只能等待其 owner node 重启后，才能恢复该 queue 的使用，而在这段时间内发送给该 queue 的 message 将被 blackholed 。所以，是否要对 message 进行持久化，需要综合考虑性能需要，以及可能遇到的问题。若想达到 100,000 条/秒以上的消息吞吐量（单 RabbitMQ 服务器），则要么使用其他的方式来确保 message 的可靠 delivery ，要么使用非常快速的存储系统以支持全持久化（例如使用 SSD）。另外一种处理原则是：仅对关键消息作持久化处理（根据业务重要程度），且应该保证关键消息的量不会导致性能瓶颈。

问题二十四：RabbitMQ 中的 cluster、mirrored queue，以及 warrens 机制分别用于解决什么问题？存在哪些问题？
答：cluster 是为了解决当 cluster 中的任意 node 失效后，producer 和 consumer 均可以通过其他 node 继续工作，即提高了可用性；另外可以通过增加 node 数量增加 cluster 的消息吞吐量的目的。cluster 本身不负责 message 的可靠性问题（该问题由 producer 通过各种机制自行解决）；cluster 无法解决跨数据中心的问题（即脑裂问题）。另外，在 cluster 前使用 HAProxy 可以解决 node 的选择问题，即业务无需知道 cluster 中多个 node 的 ip 地址。可以利用 HAProxy 进行失效 node 的探测，可以作负载均衡。下图为 HAProxy + cluster 的模型。

Mirrored queue 是为了解决使用 cluster 时所创建的 queue 的完整信息仅存在于单一 node 上的问题，从另一个角度增加可用性。若想正确使用该功能，需要保证：1.consumer 需要支持 Consumer Cancellation Notification 机制；2.consumer 必须能够正确处理重复 message 。

Warrens 是为了解决 cluster 中 message 可能被 blackholed 的问题，即不能接受 producer 不停 republish message 但 RabbitMQ server 无回应的情况。Warrens 有两种构成方式，一种模型是两台独立的 RabbitMQ server + HAProxy ，其中两个 server 的状态分别为 active 和 hot-standby 。该模型的特点为：两台 server 之间无任何数据共享和协议交互，两台 server 可以基于不同的 RabbitMQ 版本。如下图所示

另一种模型为两台共享存储的 RabbitMQ server + keepalived ，其中两个 server 的状态分别为 active 和 cold-standby 。该模型的特点为：两台 server 基于共享存储可以做到完全恢复，要求必须基于完全相同的 RabbitMQ 版本。如下图所示

Warrens 模型存在的问题：对于第一种模型，虽然理论上讲不会丢失消息，但若在该模型上使用持久化机制，就会出现这样一种情况，即若作为 active 的 server 异常后，持久化在该 server 上的消息将暂时无法被 consume ，因为此时该 queue 将无法在作为 hot-standby 的 server 上被重建，所以，只能等到异常的 active server 恢复后，才能从其上的 queue 中获取相应的 message 进行处理。而对于业务来说，需要具有：a.感知 AMQP 连接断开后重建各种 fabric 的能力；b.感知 active server 恢复的能力；c.切换回 active server 的时机控制，以及切回后，针对 message 先后顺序产生的变化进行处理的能力。对于第二种模型，因为是基于共享存储的模式，所以导致 active server 异常的条件，可能同样会导致 cold-standby server 异常；另外，在该模型下，要求 active 和 cold-standby 的 server 必须具有相同的 node 名和 UID ，否则将产生访问权限问题；最后，由于该模型是冷备方案，故无法保证 cold-standby server 能在你要求的时限内成功启动。