BAT 高级面试题

答案

1、多线程的应用

一、共享资源

共享资源：就是内存中的一块资源同时被多个进程所访问，而每个进程可能会对该资源的数据进行修改

问题：如果 线程 A 访问了某块资源 C，并且修改了其中的数据，此时 线程 B 也访问了 资源 C，并且也对 C 中的数据进行了修改；那么等到 线程 A 和 线程 B 执行结束后，此时，资源 C 中的数据就并不是最初的设置了

二、线程通信

通常， 一个线程不应该单独存在，应该和其他线程之间有关系

例如：一个线程完成了自己的任务后需要切换到另一个线程完成某个任务；或者一个线程将数据传递给另一个线程

三、线程的状态

1. 当一个线程对象创建并开启后，它就会被放到线程调度池中，等待系统调度；如图
1. 当正在运行的线程被阻塞时，就会被移出 可调度线程池，此时不可再调度它
1. 当线程正常结束，异常退出，强制退出时都会导致该线程死

亡，死亡的线程会从内存中移除，无法调度

2、GCD 实现多个请求都完成之后返回结果

- 同步堵塞
- 栅栏函数
- 调度组

3、A、B 两个 int 数组，得到 A 数组中 B 数组不包含的元素

```
NSArray *arr1 = @[@1,@2,@3,@4,@5,@6,@7,@8];
NSArray *arr2 = @[@4,@5,@6,@7,@8,@9,@10,@11];
NSMutableArray        *mArray        =        [NSMutableArray
arrayWithCapacity:1];
for (int i=0; i<arr1.count; i++) {
    for (int j=0; j<arr2.count; j++) {
        if (arr1[i]==arr2[j]) {
            [mArray addObject:arr1[i]];
        }
    }
}
```

4、事件传递链，页面上一个按钮，按钮和它的 superView 有一样的 action,为什么只执行 button 的 action?

- hitTest 方法:首先会通过调用自身的 pointInside 方法判断用户触摸的点是否在当前对象的响应范围内,如果 pointInside 方法返回 NO hitTest 方法直接返回 nil
- 如果 pointInside 方法返回 YES hitTest 方法接着会判断自身是否有子视图.如果有则调用顶层子视图的 hitTest 方法 直到有子视图返回 View
- 如果所有子视图都返回 nil hitTest 方法返回自身.

5、runtime 的应用

- 具体应用拦截系统自带的方法调用（Method Swizzling 黑魔法）
- 实现给分类增加属性
- 实现字典的模型和自动转换
- JSPatch 替换已有的 OC 方法实行等
- aspect 切面编程

6、array 中加入对象，对象的 retainCount 会加 1 如何是的对象自己管理自己的生命周期
[obj autoreleasepool] 将对象加到自动释放池

7、bugly 的卡顿监控原理
runloop 的两次 source 的监控
渲染界面的频率来监控帧率

8、如何架构一个 app
架构 app 方式方法有很多： MVC MVP MVVM 组件化 路由

9、c 中，malloc 对象，传入了 size，free 只需要指针，这是为什么？

总体上说，ptmalloc 的内存管理是基于内存池的，而它的内存来源有两种：
1 通过 brk()获得
2 通过 mmap()匿名映射获得

当用户向 ptmalloc 请求内存时：
1 首先查找定长内存分配池，如果查找到则返回
2 如果没有空闲内存可供使用，则向操作系统申请一块 64Mb 的内存，从中切出用户需要的内存，返回

当用户调用 free 释放内存时：

1 直接将内存放入适当的定长内存池队列

2 如果触发了一定的条件，则将所有空闲内存合并，如果满足释放条件，将内存全部还给操作系统

当然了，上面的描述中省略了太多的细节。比如什么时候走 brk 什么时候走 mmap， 再比如当请求的内存大于一个阈值时，ptmalloc 将会变成一个 mmap 的简单封装，还有触发内存归还操作系统的条件等等。
不过已经足够回答题目中的问题了：因为 malloc 的时候记录了大小。
这里还可以得出另一个结论：由于 malloc 的时候记录了大量的状态，所以在频繁使用 malloc 分配小内存时，会造成大量的内存浪费。举例来说，当反复 malloc(1)时，每一次分配的内存在 32 字节：包括 size of previous chunk，size of chunk，bk_chunk_pointer，fd_chunk_pointer 共 4 个指针，合计 4 * 8 = 32 字节....

10、如何管理移动端团队，包括帮助大家提高技术

这个题没有固定答案，看你个人的经验，团队管理能力

11、数据库选择原因（realm 、coreData、FMDB、Sqlite）

- SQLite

SQLite 是在世界上使用的最多的数据库引擎，并且还是开源的。它实现了无配置，无服务要求的事务数据库引擎。SQLite 可以在 Mac OS-X, iOS, Android, Linux, 和 Windows 上使用.

由于它是使用 ANSI-C 开发的，因此它提供了一个简单的，方便使用的编程接口。SQLite 也是一个小的，轻量级的，可以被存储在跨平台磁盘文件的完善的数据库。

SQLite 之所以这么流行的原因是：

- 独立于服务器
- 零配置
- 多进程和线程下安全访问。

- 在表中使用含有特殊数据类型的一列或多列存储数据。

- Core Data

Core Data 是 App 开发者可以使用的第二大主要的 IOS 存储技术。你需要根据数据类型和数据量进行管理和存储，SQLite 和 Core Data 都有它们各自的优缺点。Core Data 更加关注于对象而不是传统的表数据库方法。使用 Core Data，你可以存储一个 Objective－C 类的对象。

- 比 SQLite 使用更多的内存。
- 比 SQLite 使用更多的存储空间。
- 比 SQLite 在取数据方面更快。

- fmdb

FMDB 框架其实只是一层很薄的封装，主要的类也就两个：FMDatabase 和 FMResultSet。在使用 fmdb 的时候还需要导入 libsqlite3.0.dylib。

core data 允许用户使用代表实体和实体间关系的高层对象来操作数据。它也可以管理串行化的数据，提供对象生存期管理与 object_graph 管理，包括存储。Core Data 直接与 Sqlite 交互，避免开发者使用原本的 SQL 语句.

- Realm

Realm 是个新技术。Realm 天生比前面提到的数据库解决方案更快，更高效。新的解决方案就叫做 Realm,它是一个跨平台的移动数据库。它可以在 Objective-C 和 Swift 中使用，并且它是专门为 iOS 和 Android 设计的数据库。

Realm 最主要的优势是：

- 绝对免费
- 快速，简单的使用
- 没有使用限制

- 为了速度和性能，运行在自己的持久化引擎上。

12、数据库做过哪些优化

- 1、选取最适用的字段属性

MySQL 可以很好的支持大数据量的存取，但是一般说来，数据库中的表越小，在它上面执行的查询也就会越快。因此，在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小。

例如，在定义邮政编码这个字段时，如果将其设置为 CHAR(255),显然给数据库增加了不必要的空间，甚至使用 VARCHAR 这种类型也是多余的，因为 CHAR(6)就可以很好的完成任务了。同样的，如果可以的话，我们应该使用 MEDIUMINT 而不是 BIGIN 来定义整型字段。

另外一个提高效率的方法是在可能的情况下，应该尽量把字段设置为 NOT NULL，这样在将来执行查询的时候，数据库不用去比较 NULL 值。

对于某些文本字段，例如“省份”或者“性别”，我们可以将它们定义为 ENUM 类型。因为在 MySQL 中，ENUM 类型被当作数值型数据来处理，而数值型数据被处理起来的速度要比文本类型快得多。这样，我们又可以提高数据库的性能。

- 2、使用连接（JOIN）来代替子查询(Sub-Queries)

MySQL 从 4.1 开始支持 SQL 的子查询。这个技术可以使用 SELECT 语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。例如，我们要将客户基本信息表中没有任何订单的客户删除掉，就可以利用子查询先从销售信息表中将所有发出订单的客户 ID 取出来，然后将结果传递给主查询，如下所示：

DELETEFROMcustomerinfo

WHERECustomerIDNOTin(SELECTCustomerIDFROMsalesinfo)

使用子查询可以一次性的完成很多逻辑上需要多个步骤才能完成的 SQL 操作，同时也可以避免事务或者表锁死，并且写起来也很容易。但是，有些情况下，子查询可以被更有效率的连接（JOIN）..替代。例如，假设我们要将所有没有订单记录的用户取出来，可以用下面这个查询完成：

```
SELECT*FROMcustomerinfo
WHERECustomerIDNOTin(SELECTCustomerIDFROMsalesinfo)
```

如果使用连接（JOIN）..来完成这个查询工作，速度将会快很多。尤其是当 salesinfo 表中对 CustomerID 建有索引的话，性能将会更好，查询如下：

```
SELECT*FROMcustomerinfo
```

LEFTJOINsalesinfoONcustomerinfo.CustomerID=salesinfo.CustomerID
WHEREsalesinfo.CustomerIDISNULL 连接（JOIN）..之所以更有效率一些，是因为 MySQL 不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作。

- 3、使用联合(UNION)来代替手动创建的临时表

MySQL 从 4.0 的版本开始支持 union 查询，它可以把需要使用临时表的两条或更多的 select 查询合并的一个查询中。在客户端的查询会话结束的时候，临时表会被自动删除，从而保证数据库整齐、高效。使用 union 来创建查询的时候，我们只需要用 UNION 作为关键字把多个 select 语句连接起来就可以了，要注意的是所有 select 语句中的字段数目要想同。下面的例子就演示了一个使用 UNION 的查询。

```
SELECTName,PhoneFROMclientUNION
SELECTName,BirthDateFROMauthorUNION
SELECTName,SupplierFROMproduct
```

- 4、事务

尽管我们可以使用子查询（Sub-Queries）、连接（JOIN）和联合（UNION）来创建各种各样的查询，但不是所有的数据库操作都可以只用一条或少数几条 SQL 语句就可以完成的。更多的时候是需要用到一系列的语句来完成某种工作。但是在这种情况下，当这个语句块中的某一条语句运行出错的时候，整个语句块的操作就会变得不确定起来。设想一下，要把某个数据同时插入两个相关联的表中，可能会出现这样的情况：第一个表中成功更新后，数据库突然出现意外状况，造成第二个表中的操作没有完成，这样，就会造成数据的不完整，甚至会破坏数据库中的数据。要避免这种情况，就应该使用事务，它的作用是：要么语句块中每条语句都操作成功，要么都失败。换句话说，就是可以保持数据库中数据的一致性和完整性。事物以 BEGIN 关键字开始，COMMIT 关键字结束。在这之间的一条 SQL 操作失败，那么，ROLLBACK 命令就可以把数据库恢复到 BEGIN 开始之前的状态。

```
BEGIN;
INSERTINTOsalesinfoSETCustomerID=14;UPDATEinventorySET
Quantity=11WHEREitem='book';COMMIT;
```

事务的另一个重要作用是当多个用户同时使用相同的数据源时，它可以利用锁定数据库的方法来为用户提供一种安全的访问方式，这样可以保证用户的操作不被其它的用户所干扰。

- 5、锁定表

尽管事务是维护数据库完整性的一个非常好的方法，但却因为它的独占性，

有时会影响数据库的性能，尤其是在很大的应用系统中。由于在事务执行的过程中，数据库将会被锁定，因此其它的用户请求只能暂时等待直到该事务结束。如果一个数据库系统只有少数几个用户来使用，事务造成的影响不会成为一个太大的问题；但假设有成千上万的用户同时访问一个数据库系统，例如访问一个电子商务网站，就会产生比较严重的响应延迟。
其实，有些情况下我们可以通过锁定表的方法来获得更好的性能。下面的例子就用锁定表的方法来完成前面一个例子中事务的功能。

```
LOCKTABLEinventoryWRITESELECTQuantityFROMinventoryWHERE
EItem='book';
...
UPDATEinventorySETQuantity=11WHEREItem='book';UNLOCKTA
BLES
```

这里，我们用一个 *select* 语句取出初始数据，通过一些计算，用 *update* 语句将新值更新到表中。包含有 *WRITE* 关键字的 *LOCKTABLE* 语句可以保证在 *UNLOCKTABLES* 命令被执行之前，不会有其它的访问来对 *inventory* 进行插入、更新或者删除的操作。

- 6、使用外键

锁定表的方法可以维护数据的完整性，但是它却不能保证数据的关联性。这个时候我们就可以使用外键。
例如，外键可以保证每一条销售记录都指向某一个存在的客户。在这里，外键可以把 *customerinfo* 表中的 *CustomerID* 映射到 *salesinfo* 表中 *CustomerID*，任何一条没有合法 *CustomerID* 的记录都不会被更新或插入到 *salesinfo* 中。

```
CREATETABLEcustomerinfo( CustomerIDINTNOTNULL,PRIMARYK
EY(CustomerID))TYPE=INNODB;
CREATETABLEsalesinfo( SalesIDINTNOTNULL,CustomerIDINTN
OTNULL,
PRIMARYKEY(CustomerID,SalesID),
FOREIGNKEY(CustomerID)REFERENCEScustomerinfo(CustomerI
D)ONDELETECASCADE)TYPE=INNODB;
```

注意例子中的参数“*ONDELETECASCADE*”。该参数保证当 *customerinfo* 表中的一条客户记录被删除的时候，*salesinfo* 表中所有与该客户相关的记录也会被自动删除。如果要在 *MySQL* 中使用外键，一定要记住在创建表的时候将表的类型定义为事务安全表 *InnoDB* 类型。该类型不是 *MySQL* 表的默认类型。定义的方法是在 *CREATETABLE* 语句中加上 *TYPE=INNODB*。如例中所示。

- 7、使用索引

索引是提高数据库性能的常用方法，它可以令数据库服务器以比没有索引快

得多的速度检索特定的行，尤其是在查询语句当中包含有 MAX(),MIN()和 ORDERBY 这些命令的时候，性能提高更为明显。

那该对哪些字段建立索引呢？

一般说来，索引应建立在那些将用于 JOIN,WHERE 判断和 ORDERBY 排序的字段上。尽量不要对数据库中某个含有大量重复的值的字段建立索引。对于一个 ENUM 类型的字段来说，出现大量重复值是很有可能的情况例如 customerinfo 中的“province”..字段，在这样的字段上建立索引将不会有什么帮助；相反，还有可能降低数据库的性能。我们在创建表的时候可以同时创建合适的索引，也可以使用 ALTERTABLE 或 CREATEINDEX 在以后创建索引。此外，MySQL 从版本 3.23.23 开始支持全文索引和搜索。全文索引在 MySQL 中是一个 FULLTEXT 类型索引，但仅能用于 MyISAM 类型的表。对于一个大的数据库，将数据装载到一个没有 FULLTEXT 索引的表中，然后再使用 ALTERTABLE 或 CREATEINDEX 创建索引，将是非常快的。但如果将数据装载到一个已经有 FULLTEXT 索引的表中，执行过程将会非常慢。

- 8、优化的查询语句

绝大多数情况下，使用索引可以提高查询的速度，但如果 SQL 语句使用不恰当的话，索引将无法发挥它应有的作用。

下面是应该注意的几个方面。

- 首先，最好是在相同类型的字段间进行比较的操作。

在 MySQL3.23 版之前，这甚至是一个必须的条件。例如不能将一个建有索引的 INT 字段和 BIGINT 字段进行比较；但是作为特殊的情况，在 CHAR 类型的字段和 VARCHAR 类型字段的字段大小相同的时候，可以将它们进行比较。

- 其次，在建有索引的字段上尽量不要使用函数进行操作。

例如，在一个 DATE 类型的字段上使用 YEAE()函数时，将会使索引不能发挥应有的作用。所以，下面的两个查询虽然返回的结果一样，但后者要比前者快得多。

- 第三，在搜索字符型字段时，我们有时会使用 LIKE 关键字和通配符，这种做法虽然简单，但却也是以牺牲系统性能为代价的。

例如下面的查询将会比较表中的每一条记录。

```
SELECT*FROMbooks

WHEREnamelike"MySQL%"
```

但是如果换用下面的查询，返回的结果一样，但速度就要快上很多：

```
SELECT*FROMbooks

WHEREname>="MySQL"andname<"MySQM"
```

最后，应该注意避免在查询中让 MySQL 进行自动类型转换，因为转换过程也会使索引变得不起作用。

13、arc 情况下，编译的时候，系统是怎么添加相关内存管理的代码

14、脚本打包原理

本脚本主要作用为代替人工打包 app，导出 ipa 包并安装的过程，如果是 AppStore 方式，会自动上传 AppStore，不需要手动管理。 如需使用自动安装 ipa 功能，需要进行一些额外的环境配置。
利用 xcode 的一些命令： xcodebuild - archive -

15、app 运行过程中，同时最多有几个线程，怎么实现的高并发

同时最多有几个线程： 根据 cpu 的能力，目测：50 个

生活中遇到的很多场景，多是 IO 密集型。解决这类问题的核心思想就是减少 cpu 空转的时间，增加 CPU 的利用率。具体有下面两种方法：

- 限制活动线程的个数不超过硬件线程的个数

  活动线程指 Runnable 状态的线程。

  Blocked 状态的线程个数不在限制内。Blocked 状态的线程都在等待外部事件触发，比如鼠标点击、磁盘 IO 操作事件，操作系统会将他们移除到调度队列外，所以它们不会消耗 cpu 时间片。程序可以有很多的线程，但是只要保证活动的线程个数小于硬件线程的个数，运行效率是可以保证的。

  计算密集型和 IO 密集型线程是要分开看待的。计算密集型线程应该永远不被 block，大部分时间都要保证 runnable 状态。要有效的利用处理器资源，可以让计算密集型的线程个数跟处理器个数匹配。而 IO 密集型线程大部分时间都在等待 IO 事件，不需要太多的线程。

- 基于任务的编程（协程）

线程个数跟硬件线程一致。任务调度器把对应的任务放入跟线程做一个映射，放入到相应的线程执行。有几个明显的优势：

  - 按需调度。

    线程调度器的时间片是公平的分配给各个线程的，因为它不不理解程序的业务逻辑。这就跟计划经济样的，极度的公平就是不公平，市场经济这种按需分配才能提高效率。任务调度器是理解任务信息的，可以更有效的调度任务。

  - 负载均衡。

    将程序分成一个个小的任务，让调度器来调度，不让所有的线程空跑，保证线程随时有活干。有效的利用计算资源、平衡计算资源。

  - 更易编程。

    以线程为基础的编程，要提高效率，经常要考虑到底层的硬件线程，考虑线程调度受到的影响。但是如果基于任务来编程，只要集中注意力在任务之间的逻辑关系上，处理好任务之间的关系。调度效率可以交给调度器来管控。