初识MySQL

标签： MySQL是怎样运行的

MySQL的客户端 / 服务器架构

以我们平时使用的微信为例，它其实是由两部分组成的，一部分是客户端程序，一部分是服务器程序。客户端可能有很多种形式，比如手机APP，电脑软件或者是网页版微信，每个客户端都有一个唯一的用户名，就是你的微信号，另一方面，腾讯公司在他们的机房里运行着一个服务器软件，我们平时操作微信其实都是用客户端来和这个服务器来打交道。比如狗哥用微信给猫爷发了一条消息的过程其实是这样的：

1. 消息被客户端包装了一下，添加了发送者和接受者信息，然后从狗哥的微信客户端传送给微信服务器；
2. 微信服务器从消息里获取到它的发送者和接收者，根据消息的接受者信息把这条消息送达到猫爷的微信客户端，猫爷的微信客户端里就显示出狗哥给他发了一条消息。

`MySQL`的使用过程跟这个是一样的，它的服务器程序直接和我们存储的数据打交道，然后可以有好多客户端程序连接到这个服务器程序，发送增删改查的请求，然后服务器就响应这些请求，从而操作它维护的数据。和微信一样，`MySQL`的每个客户端都需要提供用户名密码才能登录，登录之后才能给服务器发请求来操作某些数据。我们日常使用`MySQL`的情景一般是这样的：

1. 启动`MySQL`服务器程序。
2. 启动`MySQL`客户端程序并连接到服务器程序。
3. 在客户端程序中输入一些命令语句作为请求发送到服务器程序，服务器程序收到这些请求后，会根据请求的内容来操作具体的数据并向客户端返回操作结果。

我们知道计算机很牛逼，在一台计算机上可以同时运行多个程序，比如微信、QQ、音乐播放器、文本编辑器啥的，每一个运行着的程序也被称为一个`进程`。我们的`MySQL`服务器程序和客户端程序本质上都算是计算机上的一个`进程`，这个代表着`MySQL`服务器程序的进程也被称为`MySQL数据库实例`，简称`数据库实例`。

每个进程都有一个唯一的编号，称为`进程ID`，英文名叫`PID`，这个编号是在我们启动程序的时候由操作系统随机分配的，操作系统会保证在某一时刻同一台机器上的进程号不重复。比如你打开了计算机中的QQ程序，那么操作系统会为它分配一个唯一的进程号，如果你把这个程序关掉了，那操作系统就会把这个进程号回收，之后可能会重新分配给别的进程。当我们下一次再启动 QQ程序的时候分配的就可能是另一个编号。每个进程都有一个名称，这个名称是编写程序的人自己定义的，比如我们启动的`MySQL`服务器进程的默认名称为`mysqld`， 而我们常用的`MySQL`客户端进程的默认名称为`mysql`。

MySQL的安装

不论我们通过下载源代码自行编译安装的方式还是直接使用官方提供的安装包进行安装之后，`MySQL`的服务器程序和客户端程序都会被安装到我们的机器上。不论使用上述两者的哪种安装方式，一定一定一定（重要的话说三遍）要记住你把`MySQL`安装到哪了，换句话说，一定要记住`MySQL`的安装目录。

> 小贴士： `MySQL`的大部分安装包都包含了服务器程序和客户端程序，不过在Linux下使用RPM包时会有单独的服务器RPM包和客户端RPM包，需要分别安装。

另外，`MySQL`可以运行在各种各样的操作系统上，我们后边会讨论在类`UNIX`操作系统和`Windows`操作系统上使用的一些差别。为了方便大家理解，我在`macOS`操作系统（苹果电脑使用的操作系统）和`Windows`操作系统上都安装了`MySQL`，它们的安装目录分别是：

- `macOS`操作系统上的安装目录：

```
/usr/local/mysql/
```

- `Windows`操作系统上的安装目录：

```
C:\Program Files\MySQL\MySQL Server 5.7
```

下边我会以这两个安装目录为例来进一步扯出更多的概念，不过一定要注意，这两个安装目录是我的运行不同操作系统的机器上的安装目录，一定要记着把下边示例中用到安装目录的地方替换为你自己机器上的安装目录。

> 小贴士： 类UNIX操作系统非常多，比如FreeBSD、Linux、macOS、Solaris等都属于UNIX操作系统的范畴，我们这里使用macOS操作系统代表类UNIX操作系统来运行MySQL。

bin目录下的可执行文件

在`MySQL`的安装目录下有一个特别特别重要的`bin`目录，这个目录下存放着许多可执行文件，以`macOS`系统为例，这个`bin`目录的绝对路径就是（在我的机器上）：

```
/usr/local/mysql/bin
```

我们列出一些在`macOS`中这个`bin`目录下的一部分可执行文件来看一下（文件太多，全列出来会刷屏的）：

```
.
├── mysql
├── mysql.server -> ../support-files/mysql.server
├── mysqladmin
├── mysqlbinlog
├── mysqlcheck
├── mysqld
├── mysqld_multi
├── mysqld_safe
├── mysqldump
├── mysqlimport
├── mysqlpump
... (省略其他文件)
0 directories, 40 files
```

`Windows`中的可执行文件与`macOS`中的类似，不过都是以`.exe`为扩展名的。这些可执行文件都是与服务器程序和客户端程序相关的，后边我们会详细唠叨一些比较重要的可执行文件，现在先看看执行这些文件的方式。

对于有可视化界面的操作系统来说，我们拿着鼠标点点点就可以执行某个可执行文件，不过现在我们更关注在命令行环境下如何执行这些可执行文件，命令行通俗的说就是那些黑框框，这里的指的是类`UNIX`系统中的`Shell`或者`Windows`系统中的`cmd.exe`，如果你现在还不知道怎么启动这些命令行工具，网上搜搜吧～下边我们以`macOS`系统为例来看看如何启动这些可执行文件（`Windows`中的操作是类似的，依葫芦画瓢就好了）

- 使用可执行文件的相对/绝对路径

  假设我们现在所处的工作目录是`MySQL`的安装目录，也就是`/usr/local/mysql`，我们想启动`bin`目录下的`mysqld`这个可执行文件，可以使用相对路径来启动：

  ```
  ./bin/mysqld
  ```

  或者直接输入`mysqld`的绝对路径也可以：

  ```
  /usr/local/mysql/bin/mysqld
  ```

- 将该`bin`目录的路径加入到环境变量`PATH`中

  如果我们觉得每次执行一个文件都要输入一串长长的路径名贼麻烦的话，可以把该`bin`目录所在的路径添加到环境变量`PATH`中。环境变量`PATH`是一系列路径的集合，各个路径之间使用冒号`:`隔离开，比方说我的机器上的环境变量`PATH`的值就是：

  ```
  /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
  ```

  我的系统中这个环境变量`PATH`的值表明：当我在输入一个命令时，系统便会在`/usr/local/bin`、`/usr/bin:`、`/bin:`、`/usr/sbin`、`/sbin`这些目录下依次寻找是否存在我们输入的那个命令，如果寻找成功，则执行该目录下对应的可执行文件。所以我们现在可以修改一下这个环境变量`PATH`，把`MySQL`安装目录下的`bin`目录的路径也加入到`PATH`中，在我的机器上修改后的环境变量`PATH`的值为：

  ```
  /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/mysql/bin
  ```

  这样现在不论我们所处的工作目录是啥，我们都可以直接输入可执行文件的名字就可以启动它，比如这样：

  ```
  mysqld
  ```

  方便多了哈～

  > 小贴士：关于啥是环境变量以及如何在当前系统中添加或修改系统变量不是我们唠叨的范围，大家找本相关的书或者上网查一查哈～

启动MySQL服务器程序

UNIX里启动服务器程序

在类 UNIX 系统中用来启动 MySQL 服务器程序的可执行文件有很多，大多在 MySQL 安装目录的 bin 目录下，我们一起来瞅瞅。

mysqld

mysqld 这个可执行文件就代表着 MySQL 服务器程序，运行这个可执行文件就可以直接启动一个服务器进程。但这个命令不常用，我们继续往下看更牛逼的启动命令。

mysqld_safe

mysqld_safe 是一个启动脚本，它会间接的调用 mysqld ，而且还顺便启动了另外一个监控进程，这个监控进程在服务器进程挂了的时候，可以帮助重启它。另外，使用 mysqld_safe 启动服务器程序时，它会将服务器程序的出错信息和其他诊断信息重定向到某个文件中，产生出错日志，这样可以方便我们找出发生错误的原因。

mysql.server

mysql.server 也是一个启动脚本，它会间接的调用 mysqld_safe ，在调用 mysql.server 时在后边指定 start 参数就可以启动服务器程序了，就像这样：

```
mysql.server start
```

需要注意的是，这个 ***mysql.server*** 文件其实是一个链接文件，它的实际文件是 ***../support-files/mysql.server***。我使用的 macOS 操作系统会帮我们在 bin 目录下自动创建一个指向实际文件的链接文件，如果你的操作系统没有帮你自动创建这个链接文件，那就自己创建一个呗～ 别告诉我你不会创建链接文件，上网搜搜呗～

另外，我们还可以使用 mysql.server 命令来关闭正在运行的服务器程序，只要把 start 参数换成 stop 就好了：

```
mysql.server stop
```

mysqld_multi

其实我们一台计算机上也可以运行多个服务器实例，也就是运行多个 MySQL 服务器进程。 mysql_multi 可执行文件可以对每一个服务器进程的启动或停止进行监控。这个命令的使用比较复杂，本书主要是为了讲清楚 MySQL 服务器和客户端运行的过程，不会对启动多个服务器程序进行过多唠叨。

Windows里启动服务器程序

Windows 里没有像类 UNIX 系统中那么多的启动脚本，但是也提供了手动启动和以服务的形式启动这两种方式，下边我们详细看。

mysqld

同样的，在 MySQL 安装目录下的 bin 目录下有一个 mysqld 可执行文件，在命令行里输入 mysqld ，或者直接双击运行它就算启动了 MySQL 服务器程序了。

以服务的方式运行服务器程序

首先看看啥是个 `Windows` 服务？如果无论是谁正在使用这台计算机，我们都需要长时间的运行某个程序，而且需要在计算机启动的时候便启动它，一般我们都会把它注册为一个 `Windows 服务`，操作系统会帮我们管理它。把某个程序注册为 `Windows` 服务的方式挺简单，如下：

```
"完整的可执行文件路径" --install [-manual] [服务名]
```

其中的 `-manual` 可以省略，加上它的话表示在 `Windows` 系统启动的时候不自动启动该服务，否则会自动启动。`服务名` 也可以省略，默认的服务名就是 `MySQL`。比如我的 `Windows` 计算机上 `mysqld` 的完整路径是：

```
C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld
```

所以如果我们想把它注册为服务的话可以在命令行里这么写：

```
"C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld" --install
```

在把 `mysqld` 注册为 `Windows` 服务之后，我们就可以通过下边这个命令来启动 `MySQL` 服务器程序了：

```
net start MySQL
```

当然，如果你喜欢图形界面的话，你可以通过 `Windows` 的服务管理器通过用鼠标点点点的方式来启动和停止服务（作为一个程序猿，还是用黑框框吧～）。

关闭这个服务也非常简单，只要把上边的 `start` 换成 `stop` 就行了，就像这样：

```
net stop MySQL
```

启动MySQL客户端程序

在我们成功启动 `MySQL` 服务器程序后，就可以接着启动客户端程序来连接到这个服务器喽，`bin` 目录下有许多客户端程序，比方说 `mysqladmin`、`mysqldump`、`mysqlcheck` 等等等等（好多呢，就不一一列举了）。这里我们重点要关注的是可执行文件 `mysql`，通过这个可执行文件可以让我们和服务器程序进程交互，也就是发送请求，接受服务器的处理结果。启动这个可执行文件时一般需要一些参数，格式如下：

```
mysql -h主机名   -u用户名 -p密码
```

各个参数的意义如下：

参数名

含义

`-h`

表示服务器进程所在计算机的域名或者IP地址，如果服务器进程就运行在本机的话，可以省略这个参数，或者填`localhost`或者`127.0.0.1`。也可以写作`--host=主机名`的形式。

`-u`

表示用户名。也可以写作`--user=用户名`的形式。

`-p`

表示密码。也可以写作`--password=密码`的形式。

> 小贴士： 像 h、u、p 这样名称只有一个英文字母的参数称为短形式的参数，使用时前边需要加单短划线，像 host、user、password 这样大于一个英文字母的参数称为长形式的参数，使用时前边需要加双短划线。后边会详细讨论这些参数的使用方式的，稍安勿躁~

比如我这样执行下边这个可执行文件(用户名密码按你的实际情况填写)，就可以启动`MySQL`客户端，并且连接到服务器了。

```
mysql -hlocalhost -uroot -p123456
```

我们看一下连接成功后的界面：

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.21 Homebrew

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

最后一行的`mysql>`是一个客户端的提示符，之后客户端发送给服务器的命令都需要写在这个提示符后边。

如果我们想断开客户端与服务器的连接并且关闭客户端的话，可以在`mysql>`提示符后输入下边任意一个命令：

1. `quit`
2. `exit`
3. `\q`

比如我们输入`quit`试试：

```
mysql> quit
Bye
```

输出了`Bye`说明客户端程序已经关掉了。注意注意注意，这是关闭客户端程序的方式，不是关闭服务器程序的方式，怎么关闭服务器程序上一节里唠叨过了。

如果你愿意，你可以多打开几个黑框框，每个黑框框都使用`mysql -hlocahhost -uroot -p123456`来运行多个客户端程序，每个客户端程序都是互不影响的。如果你有多个电脑，也可以试试把它们用局域网连起来，在一个电脑上启动`MySQL`服务器程序，在另一个电脑上执行`mysql`命令时使用`IP`地址作为主机名来连接到服务器。

连接注意事项

- 最好不要在一行命令中输入密码。

  我们直接在黑框框里输入密码很可能被别人看到，这和你当着别人的面输入银行卡密码没啥区别，所以我们在执行`mysql`连接服务器的时候可以不显式的写出密码，就像这样：

  ```
  mysql -hlocahhost -uroot -p
  ```

  点击回车之后才会提示你输入密码：

  ```
  Enter password:
  ```

  不过这回你输入的密码不会被显示出来，心怀不轨的人也就看不到了，输入完成点击回车就成功连接到了服务器。

- 如果你非要在一行命令中显式的把密码输出来，那`-p`和密码值之间不能有空白字符（其他参数名之间可以有空白字符），就像这样：

  ```
  mysql -h localhost -u root -p123456
  ```

  如果加上了空白字符就是错误的，比如这样：

  ```
  mysql -h localhost -u root -p 123456
  ```

- `mysql`的各个参数的摆放顺序没有硬性规定，也就是说你也可以这么写：

  ```
  mysql -p  -u root -h localhost
  ```

- 如果你的服务器和客户端安装在同一台机器上，`-h`参数可以省略，就像这样：

  ```
  mysql -u root -p
  ```

- 如果你使用的是类`UNIX`系统，并且省略`-u`参数后，会把你登陆操作系统的用户名当作`MySQL`的用户名去处理。

  比方说我用登录操作系统的用户名是`xiaohaizi`，那么在我的机器上下边这两条命令是等价的：

```
mysql -u xiaohaizi -p
mysql -p
```

对于`Windows`系统来说，默认的用户名是`ODBC`，你可以通过设置环境变量`USER`来添加一个默认用户名。

客户端与服务器连接的过程

我们现在已经知道如何启动`MySQL`的服务器程序，以及如何启动客户端程序来连接到这个服务器程序。运行着的服务器程序和客户端程序本质上都是计算机上的一个进程，所以客户端进程向服务器进程发送请求并得到回复的过程本质上是一个进程间通信的过程！`MySQL`支持下边三种客户端进程和服务器进程的通信方式。

TCP/IP

真实环境中，数据库服务器进程和客户端进程可能运行在不同的主机中，它们之间必须通过网络来进行通讯。`MySQL`采用`TCP`作为服务器和客户端之间的网络通信协议。在网络环境下，每台计算机都有一个唯一的`IP地址`，如果某个进程有需要采用`TCP`协议进行网络通信方面的需求，可以向操作系统申请一个`端口号`，这是一个整数值，它的取值范围是`0~65535`。这样在网络中的其他进程就可以通过`IP地址 + 端口号`的方式来与这个进程连接，这样进程之间就可以通过网络进行通信了。

`MySQL`服务器启动的时候会默认申请`3306`端口号，之后就在这个端口号上等待客户端进程进行连接，用书面一点的话来说，`MySQL`服务器会默认监听`3306`端口。

> 小贴士：`TCP/IP`网络体系结构是现在通用的一种网络体系结构，其中的`TCP`和`IP`是体系结构中两个非常重要的网络协议，如果你并不知道协议是什么，或者并不知道网络是什么，那恐怕兄弟你来错地方了，找本计算机网络的书去瞅瞅吧！什么？计算机网络的书写的都贼恶心，看不懂？没关系，等我～

如果`3306`端口号已经被别的进程占用了或者我们单纯的想自定义该数据库实例监听的端口号，那我们可以在启动服务器程序的命令行里添加`-P`参数来明确指定一下端口号，比如这样：

```
mysqld -P3307
```

这样`MySQL`服务器在启动时就会去监听我们指定的端口号`3307`。

如果客户端进程想要使用`TCP/IP`网络来连接到服务器进程，比如我们在使用`mysql`来启动客户端程序时，在`-h`参数后必须跟随`IP地址`来作为需要连接的服务器进程所在主机的主机名，如果客户端进程和服务器进程在一台计算机中的话，我们可以使用`127.0.0.1`来代表本机的`IP地址`。另外，如果服务器进程监听的端口号不是默认的`3306`，我们也可以在使用`mysql`启动客户端程序时使用`-P`参数（大写的`P`，小写的`p`是用来指定密码的）来指定需要连接到的端口号。比如我们现在已经在本机启动了服务器进程，监听的端口号为`3307`，那我们启动客户端程序时可以这样写：

```
mysql -h127.0.0.1 -uroot -P3307 -p
```

不知大家发现了没有，我们在启动服务器程序的命令`mysqld`和启动客户端程序的命令`mysql`后边都可以使用`-P`参数，关于如何在命令后边指定参数，指定哪些参数我们稍后会详细唠叨的，稍微等等哈～

命名管道和共享内存

如果你是一个`Windows`用户，那么客户端进程和服务器进程之间可以考虑使用`命名管道`或`共享内存`进行通信。不过启用这些通信方式的时候需要在启动服务器程序和客户端程序时添加一些参数：

- 使用`命名管道`来进行进程间通信

  需要在启动服务器程序的命令中加上`--enable-named-pipe`参数，然后在启动客户端程序的命令中加入`--pipe`或者`--protocol=pipe`参数。

- 使用`共享内存`来进行进程间通信

  需要在启动服务器程序的命令中加上`--shared-memory`参数，在成功启动服务器后，`共享内存`便成为本地客户端程序的默认连接方式，不过我们也可以在启动客户端程序的命令中加入`--protocol=memory`参数来显式的指定使用共享内存进行通信。

  不过需要注意的是，使用`共享内存`的方式进行通信的服务器进程和客户端进程必须在同一台`Windows`主机中。

> 小贴士： 命名管道和共享内存是Windows操作系统中的两种进程间通信方式，如果你没听过的话也不用纠结，并不妨碍我们介绍MySQL的知识～

Unix域套接字文件

如果我们的服务器进程和客户端进程都运行在同一台操作系统为类`Unix`的机器上的话，我们可以使用`Unix域套接字文件`来进行进程间通信。如果我们在启动客户端程序的时候指定的主机名为`localhost`，或者指定了`--protocal=socket`的启动参数，那服务器程序和客户端程序之间就可以通过`Unix`域套接字文件来进行通信了。`MySQL`服务器程序默认监听的`Unix`域套接字文件路径为`/tmp/mysql.sock`，客户端程序也默认连接到这个`Unix`域套接字文件。如果我们想改变这个默认路径，可以在启动服务器程序时指定`socket`参数，就像这样：

```
mysqld --socket=/tmp/a.txt
```

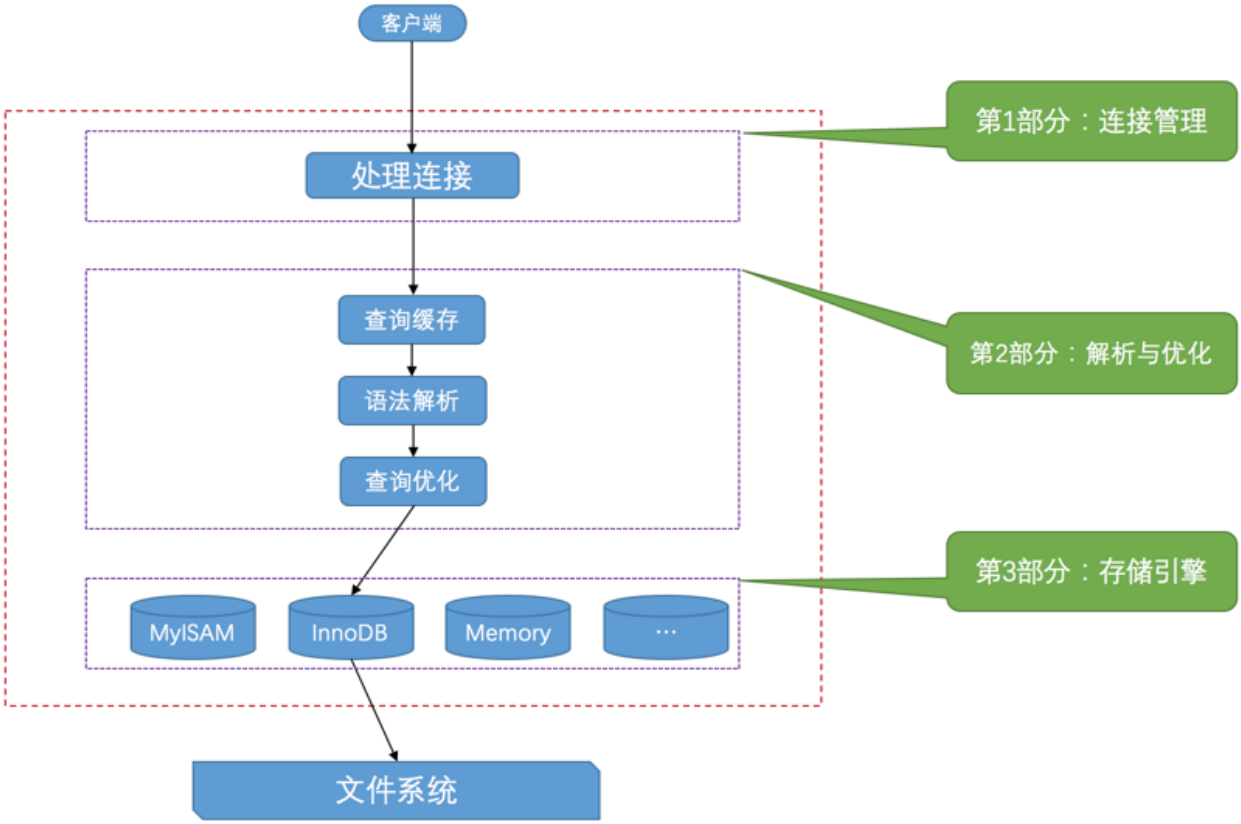
这样服务器启动后便会监听`/tmp/a.txt`。在服务器改变了默认的`UNIX`域套接字文件后，如果客户端程序想通过`UNIX`域套接字文件进行通信的话，也需要显式的指定连接到的`UNIX`域套接字文件路径，就像这样：

```
mysql -hlocalhost -uroot --socket=/tmp/a.txt -p
```

这样该客户端进程和服务器进程就可以通过路径为`/tmp/a.txt`的`Unix`域套接字文件进行通信了。

服务器处理客户端请求

其实不论客户端进程和服务器进程是采用哪种方式进行通信，最后实现的效果都是：客户端进程向服务器进程发送一段文本（MySQL语句），服务器进程处理后再向客户端进程发送一段文本（处理结果）。那服务器进程对客户端进程发送的请求做了什么处理，才能产生最后的处理结果呢？客户端可以向服务器发送增删改查各类请求，我们这里以比较复杂的查询请求为例来画个图展示一下大致的过程：



从图中我们可以看出，服务器程序处理来自客户端的查询请求大致需要经过三个部分，分别是`连接管理`、`解析与优化`、`存储引擎`。下边我们来详细看一下这三个部分都干了什么。

连接管理

客户端进程可以采用我们上边介绍的`TCP/IP`、`命名管道或共享内存`、`Unix域套接字`这几种方式之一来与服务器进程建立连接，每当有一个客户端进程连接到服务器进程时，服务器进程都会创建一个线程来专门处理与这个客户端的交互，当该客户端退出时会与服务器断开连接，服务器并不会立即把与该客户端交互的线程销毁掉，而是把它缓存起来，在另一个新的客户端再进行连接时，把这个缓存的线程分配给该新客户端。这样就起到了不频繁创建和销毁线程的效果，从而节省开销。从这一点大家也能看出，`MySQL`服务器会为每一个连接进来的客户端分配一个线程，但是线程分配的太多了会严重影响系统性能，所以我们也需要限制一下可以同时连接到服务器的客户端数量，至于怎么限制我们后边再说哈~

在客户端程序发起连接的时候，需要携带主机信息、用户名、密码，服务器程序会对客户端程序提供的这些信息进行认证，如果认证失败，服务器程序会拒绝连接。另外，如果客户端程序和服务器程序不运行在一台计算机上，我们还可以采用使用了`SSL`（安全套接字）的网络连接进行通信，来保证数据传输的安全性。

当连接建立后，与该客户端关联的服务器线程会一直等待客户端发送过来的请求，`MySQL`服务器接收到的请求只是一个文本消息，该文本消息还要经过各种处理，预知后事如何，继续往下看哈~

解析与优化

到现在为止，`MySQL`服务器已经获得了文本形式的请求，接着 还要经过九九八十一难的处理，其中的几个比较重要的部分分别是`查询缓存`、`语法解析`和`查询优化`，下边我们详细来看。

查询缓存

如果我问你 `9+8×16-3×2×17` 的值是多少，你可能会用计算器去算一下，或者牛逼一点用心算，最终得到了结果 `35`，如果我再问你一遍 `9+8×16-3×2×17` 的值是多少，你还用再傻呵呵的算一遍么？我们刚刚已经算过了，直接说答案就好了。 `MySQL` 服务器程序处理查询请求的过程也是这样，会把刚刚处理过的查询请求和结果 `缓存` 起来，如果下一次有一模一样的请求过来，直接从缓存中查找结果就好了，就不用再傻呵呵的去底层的表中查找了。这个查询缓存可以在不同客户端之间共享，也就是说如果客户端A刚刚查询了一个语句，而客户端B之后发送了同样的查询请求，那么客户端B的这次查询就可以直接使用查询缓存中的数据。

当然， `MySQL` 服务器并没有人聪明，如果两个查询请求在任何字符上的不同（例如：空格、注释、大小写），都会导致缓存不会命中。另外，如果查询请求中包含某些系统函数、用户自定义变量和函数、一些系统表，如 mysql 、information_schema、 performance_schema 数据库中的表，那这个请求就不会被缓存。以某些系统函数举例，可能同样的函数的两次调用会产生不一样的结果，比如函数 `NOW` ，每次调用都会产生最新的当前时间，如果在一个查询请求中调用了这个函数，那即使查询请求的文本信息都一样，那不同时间的两次查询也应该得到不同的结果，如果在第一次查询时就缓存了，那第二次查询的时候直接使用第一次查询的结果就是错误的！

不过既然是缓存，那就有它缓存失效的时候。MySQL的缓存系统会监测涉及到的每张表，只要该表的结构或者数据被修改，如对该表使用了 `INSERT` 、 `UPDATE` 、 `DELETE` 、 `TRUNCATE TABLE` 、 `ALTER TABLE` 、 `DROP TABLE` 或 `DROP DATABASE` 语句，那使用该表的所有高速缓存查询都将变为无效并从高速缓存中删除！

> 小贴士： 虽然查询缓存有时可以提升系统性能，但也不得不因维护这块缓存而造成一些开销，比如每次都要去查询缓存中检索，查询请求处理完需要更新查询缓存，维护该查询缓存对应的内存区域。从MySQL 5.7.20 开始，不推荐使用查询缓存，并在MySQL 8.0中删除。

语法解析

如果查询缓存没有命中，接下来就需要进入正式的查询阶段了。因为客户端程序发送过来的请求只是一段文本而已，所以 `MySQL` 服务器程序首先要对这段文本做分析，判断请求的语法是否正确，然后从文本中将要查询的表、各种查询条件都提取出来放到 `MySQL` 服务器内部使用的一些数据结构上来。

> 小贴士： 这个从指定的文本中提取出我们需要的信息本质上算是一个编译过程，涉及词法解析、语法分析、语义分析等阶段，这些问题不属于我们讨论的范畴，大家只要了解在处理请求的过程中需要这个步骤就好了。

查询优化

语法解析之后，服务器程序获得到了需要的信息，比如要查询的列是哪些，表是哪个，搜索条件是什么等等，但光有这些是不够的，因为我们写的 `MySQL` 语句执行起来效率可能并不是很高， `MySQL` 的优化程序会对我们的语句做一些优化，如外连接转换为内连接、表达式简化、子查询转为连接吧啦吧啦的一堆东西。优化的结果就是生成一个执行计划，这个执行计划表明了应该使用哪些索引进行查询，表之间的连接顺序是啥样的。我们可以使用 `EXPLAIN` 语句来查看某个语句的执行计划，关于查询优化这部分的详细内容我们后边会仔细唠叨，现在你只需要知道在 `MySQL` 服务器程序处理请求的过程中有这么一个步骤就好了。

存储引擎

截止到服务器程序完成了查询优化为止，还没有真正的去访问真实的数据表， `MySQL` 服务器把数据的存储和提取操作都封装到了一个叫 `存储引擎` 的模块里。我们知道 `表` 是由一行一行的记录组成的，但这只是一个逻辑上的概念，物理上如何表示记录，怎么从表中读取数据，怎么把数据写入具体的物理存储器上，这都是 `存储引擎` 负责的事情。为了实现不同的功能， `MySQL` 提供了各式各样的 `存储引擎` ，不同 `存储引擎` 管理的表具体的存储结构可能不同，采用的存取算法也可能不同。

> 小贴士： 为什么叫`引擎`呢？因为这个名字更拉风～其实这个存储引擎以前叫做`表处理器`，后来可能人们觉得太土，就改成了`存储引擎`的叫法，它的功能就是接收上层传下来的指令，然后对表中的数据进行提取或写入操作。

为了管理方便，人们把`连接管理`、`查询缓存`、`语法解析`、`查询优化`这些并不涉及真实数据存储的功能划分为`MySQL server`的功能，把真实存取数据的功能划分为`存储引擎`的功能。各种不同的存储引擎向上边的`MySQL server`层提供统一的调用接口（也就是存储引擎API），包含了几十个底层函数，像"读取索引第一条内容"、"读取索引下一条内容"、"插入记录"等等。

所以在`MySQL server`完成了查询优化后，只需按照生成的执行计划调用底层存储引擎提供的API，获取到数据后返回给客户端就好了。

常用存储引擎

`MySQL`支持非常多种存储引擎，我这先列举一些：

存储引擎

描述

`ARCHIVE`

用于数据存档（行被插入后不能再修改）

`BLACKHOLE`

丢弃写操作，读操作会返回空内容

`CSV`

在存储数据时，以逗号分隔各个数据项

`FEDERATED`

用来访问远程表

`InnoDB`

具备外键支持功能的事务存储引擎

`MEMORY`

置于内存的表

`MERGE`

用来管理多个MyISAM表构成的表集合

`MyISAM`

主要的非事务处理存储引擎

`NDB`

MySQL集群专用存储引擎

这么多我们怎么挑啊，哈哈，你多虑了，其实我们最常用的就是 `InnoDB` 和 `MyISAM`，有时会提一下 `Memory`。其中 `InnoDB` 是 `MySQL` 默认的存储引擎，我们之后会详细唠叨这个存储引擎的各种功能，现在先看一下一些存储引擎对于某些功能的支持情况：

Feature

MyISAM

Memory

InnoDB

Archive

NDB

B-tree indexes

yes

yes

yes

no

no

Backup/point-in-time recovery

yes

yes

yes

yes

yes

Cluster database support

no

no

no

no

yes

Clustered indexes

no

no

yes

no

no

Compressed data

yes

no

yes

yes

no

Data caches

no

N/A

yes

no

yes

Encrypted data

yes

yes

yes

yes

yes

Foreign key support

no

no

yes

no

yes

Full-text search indexes

yes

no

yes

no

no

Geospatial data type support

yes

no

yes

yes

yes

Geospatial indexing support

yes

no

yes

no

no

Hash indexes

no

yes

no

no

yes

Index caches

yes

N/A

yes

no

yes

Locking granularity

Table

Table

Row

Row

Row

MVCC

no

no

yes

no

no

Query cache support

yes

yes

yes

yes

yes

Replication support

yes

Limited

yes

yes

yes

Storage limits

256TB

RAM

64TB

None

384EB

T-tree indexes

no

no

no

no

yes

Transactions

no

no

yes

no

yes

Update statistics for data dictionary

yes

yes

yes

yes

yes

密密麻麻列了这么多，看的头皮都发麻了，达到的效果就是告诉你：这玩意儿很复杂。其实这些东西大家没必要立即就给记住，我列出来的目的就是想让大家明白不同的存储引擎支持不同的功能，有些重要的功能我们会在后边的唠叨中慢慢让大家理解的～

关于存储引擎的一些操作

查看当前服务器程序支持的存储引擎

我们可以用下边这个命令来查看当前服务器程序支持的存储引擎：

```
SHOW ENGINES;
```

来看一下调用效果：

```
mysql> SHOW ENGINES;
+--------------------+---------+----------------------------------------------------------------
-------+--------------+------+------------+
| Engine             | Support | Comment
| Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------
-------+--------------+------+------------+
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign
keys     | YES          | YES  | YES        |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables
| NO           | NO   | NO         |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary
tables      | NO           | NO   | NO         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it
disappears) | NO           | NO   | NO         |
| MyISAM             | YES     | MyISAM storage engine
| NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine
| NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine
| NO           | NO   | NO         |
| PERFORMANCE_SCHEMA | YES     | Performance Schema
| NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine
| NULL         | NULL | NULL       |
```

```
+--------------------+---------+----------------------------------------------------------
-------+--------------+------+------------+
9 rows in set (0.00 sec)

mysql>
```

其中的 `Support` 列表示该存储引擎是否可用，`DEFAULT` 值代表是当前服务器程序的默认存储引擎。`Comment` 列是对存储引擎的一个描述，英文的，将就着看吧。`Transactions` 列代表该存储引擎是否支持事务处理。`XA` 列代表着该存储引擎是否支持分布式事务。`Savepoints` 代表着该列是否支持部分事务回滚。

> 小贴士：好吧，也许你并不知道什么是个事务、更别提分布式事务了，这些内容我们在后边的章节会详细唠叨，现在瞅一眼看个新鲜就得了。

设置表的存储引擎

我们前边说过，存储引擎是负责对表中的数据进行提取和写入工作的，我们可以为不同的表设置不同的存储引擎，也就是说不同的表可以有不同的物理存储结构，不同的提取和写入方式。

创建表时指定存储引擎

我们之前创建表的语句都没有指定表的存储引擎，那就会使用默认的存储引擎 `InnoDB`（当然这个默认的存储引擎也是可以修改的，我们在后边的章节中再说怎么改）。如果我们想显式的指定一下表的存储引擎，那可以这么写：

```
CREATE TABLE 表名(
    建表语句;
) ENGINE = 存储引擎名称;
```

比如我们想创建一个存储引擎为 `MyISAM` 的表可以这么写：

```
mysql> CREATE TABLE engine_demo_table(
    ->     i int
    -> ) ENGINE = MyISAM;
Query OK, 0 rows affected (0.02 sec)

mysql>
```

修改表的存储引擎

如果表已经建好了，我们也可以使用下边这个语句来修改表的存储引擎：

```
ALTER TABLE 表名 ENGINE = 存储引擎名称;
```

比如我们修改一下 `engine_demo_table` 表的存储引擎：

```
mysql> ALTER TABLE engine_demo_table ENGINE = InnoDB;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

这时我们再查看一下`engine_demo_table`的表结构：

```
mysql> SHOW CREATE TABLE engine_demo_table\G
*************************** 1. row ***************************
       Table: engine_demo_table
Create Table: CREATE TABLE `engine_demo_table` (
  `i` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.01 sec)

mysql>
```

可以看到该表的存储引擎已经改为`InnoDB`了。