ASCII字符集：

收录128个字符，包括空格，标点符号，数字，大小写字母和一些不可见的。所以可以使用1个字节来进行编码

‘L’ -> 01001100(十六进制：0x4c，十进制：76)

‘M’->01001101(十六进制：0x4d，十进制：77)

ISO-8859-1字符集：

又称为latin1字符集。共收录256个字符，是在ASCII字符集的基础上又扩充了128西欧常用字符（包括德法两个的字母），也可以使用1个字节来进行编码。

GB2312字符集：

收录了汉字以及拉丁字母，希腊字母，日文平假名及片假名字母，俄语西里尔字母。其中收录了汉字6763个，其他文字符号682个。兼容ASCII字符集，所以在编码格式上有些奇怪：

* + 如果该字符在ASCII字符集中，则采用1字节编码
  + 否则采用2字节编码

这种表示一个字符需要的字节数可能不同的编码方式称为变长编码方式

 我们怎么区分某个字节代表一个单独的字符还是代表某个字符的一部分呢？别忘了`ASCII`字符集只收录128个字符，使用0～127就可以表示全部字符，所以如果某个字节是在0～127之内的，就意味着一个字节代表一个单独的字符，否则就是两个字节代表一个单独的字符。

GBK字符集：

只是在收录字符范围上对GB2312字符集做了扩充，编码方式上兼容GB2312

UTF-8字符集：

收录了地球上能想到的所有字符，而且还在不断扩充。这种字符集兼容ASCII字符集，采用变长编码方式，编码一个字符需要使用1~4个字节，例如：

‘L’->01001100(十六进制：0x4c)

‘啊’->1100111011010010(2个字节，十六进制表示：0xCED2)

MySQL中的utf8和utf8mb4

在MySQL中字符集表示一个字符所用最大字节长度在某些方面会影响系统的存储和性能

所用MySQL中定义了两个概念：

Utf8mb3:------阉割过的utf8字符集，只使用1~3个字节表示字符

Utf8mb4-------正宗的utf8字符集，使用1~4个字节表示字符

MySQL中，utf8是utf8mb3的别名，所以在MySQL中提到utf8就意味着使用1~3个字节来表示一个字符。若要用4个字节编码一个字符的情况，就用utf8mb4

字符集的查看：

查看当前mysql中支持的字符集可以用下边的语句：

Show (character set|charset) [like 匹配的模式];

Character set和carset是同义词，用任意一个都可以

例： show charset;

Default Collation列表示字符集中一种默认的比较规则

Maxlen列表示该字符集表示一个字符需要几个字节

一些常用字符集的Maxlen:

|  |  |
| --- | --- |
| 字符集名称 | Maxlen |
| ASCII | 1 |
| Latin1 | 1 |
| Gb2312 | 2 |
| Gbk | 2 |
| Utf8 | 3 |
| Utf8mb4 | 4 |

比较规则查看：

Show collation [like 匹配的模式]

一种字符集可能对应着若干种比较规则

例：show collation like ‘utf8\\_%’

比较规则的规律：

比较规则的名称与其关联的字符集的名称开头。

后面紧跟着该比较规则主要作用于哪种语言，比如utf8\_polish\_ci 表示以波兰语的规则比较

Utf8\_general\_ci是一种通用的比较规则

名称后缀意味着该比较规则是否区分语言中的重音，大小写等等

|  |  |  |
| --- | --- | --- |
| 后缀 | 英文释义 | 描述 |
| \_ai | Accent insensitive | 不区分重音 |
| \_as | Accent sensitive | 区分重音 |
| \_ci | Case insensitive | 不区分大小写 |
| \_cs | Case sensitive | 区分大小写 |
| \_bin | Binary | 以二进制方式比较 |

字符集和比较规则的应用：

各级别的字符集和比较规则:

Mysql有4个级别的字符集和比较规则

* + - * 服务器级别
      * 数据库级别
      * 表级别
      * 列级别

服务器级别：

MySQL提供的两个系统变量来表示服务器级别的字符集和比较规则：

Character\_set\_server 服务器级别的字符集

Collation\_server 服务器级别的比较规则

例：show variables like ‘character\_set\_server’;

Show variables like ‘collation\_server’;

计算机中服务器级别默认的字符集是utf8，默认比较规则utf8\_general\_ci

可以在启动服务器或者在服务器运行中使用set语句修改这两个变量的值：

可以在配置文件中这样写：

[server]

Character\_set\_server=gbk

Collation\_serer=gbk\_chinese\_ci

当服务器启动的时候读取这个配置文件后这两个系统变量的就修改了

数据库级别：

在创建和修改数据库的时候可以指定该数据库的字符集和比较规则

例：

Create database 数据库名

[[default] character set 字符集名称]

[[default] collate 比较规则名称];

Alter database 数据库名

[[default] character set 字符集名称]

[[default] collate 比较规则名称];

实例：

Create database demo\_db;

Character set gb2312;

Collate gb2312\_chinese\_ci;

查看当前数据库使用的字符集和比较规则：

Use 数据库名;

Show variables like ‘character\_set\_database’;

Show variables like ‘collation\_database’;

注：数据库级别的这两个系统变量是只读的，不能用这两个变量修改数据库的字符集和比较规则

创建数据库时，若不指定字符集和比较规则，则使用服务器级别的字符集和比较规则

表级别：

在创建和修改表的时候指定表的字符集和比较规则：

Create table 表名(列的信息)

[[default] character set 字符集名称]

[collation 比较规则名称]

Alter table 表名

[[default] character set 字符集名称]

[collate 比较规则名称]

例：

Create table t(

Col VARCHAR(10)

)character set utf8 collate utf8\_general\_ci;

若创建和修改表的语句中没有指明字符集和比较规则，将使用该表所在数据库的字符集和比较规则

列级别：

对于存储字符串的列，同一个表中的不同的列也可以有不同的字符集和比较规则。

在创建和修改列定义的时可以指定该列的字符集和比较规则

Create table 表名(

列名 字符串类型[character set 字符集名称] [collate 比较规则名称],

其他列…

);

Ater table 表名 modify 列名 字符串类型 [character set 字符集名称] [collate 比较规则名称];

例：

Alter table t modify col varchar(10) character set gbk collate gbk\_chinese\_ci;

仅修改字符集或者仅修改比较规则：

只修改字符集，则比较规则将变为修改后的字符集默认的比较规则

只修改比较规则，则字符集将变为修改后的比较规则对应的字符集

MySQL中的字符集转换：

从客户端发往服务器的请求本质是一个字符串，服务器向客户端返回的结果本质也是一个字符串，而字符串其实是使用某种字符集编码的二进制数据。从发送请求到返回结果这个过程伴随着多次字符集的转换。这个过程会用到3个系统变量：

|  |  |
| --- | --- |
| 系统变量 | 描述 |
| Character\_set\_client | 服务器解码请求时使用的字符集 |
| Character\_set\_connection | 服务器处理 请求时会把请求字符串从character\_set\_client转为character\_set\_connection |
| Character\_set\_results | 服务器向客户端返回数据时使用的字符集 |

Show variables like ‘character\_set\_client’;

Show variables like ‘character\_set\_connection’;

Show variables like ‘character\_set\_results’;

1. 客户端发送 请求所使用的字符集

一般情况，客户端使用的字符集和当前操作系统一致，操作系统不同字符集也不同：

* + 类Unix系统：utf8
  + Windows系统：gbk

1. 服务器接收到客户端发送来的请求其实是穿二进制的字节，它会认为这串字节采用的字符集是character\_set\_client，然后把这串字节转为character\_set\_connection字符集编码的字符
2. 服务器响应的时候按照character\_set\_results的字符集编码格式发送给客户端

需要注意的地方：

* 服务器认为客户端发送过来的请求是用character\_set\_client编码的
* 服务器将得到的结果集使用character\_set\_results编码后发送给客户端
* Character\_set\_connection只是服务器在将请求的字节串从character\_set\_client转为character\_set\_connection时使用，它是什么其实没多重要，但需要该字符集包含的字符范围一定涵盖请求中的字符，不然会导致有的字符无法使用character\_set\_connection代表队字符集进行编码。 如chracter\_set\_client字符集为utf8, character\_set\_connection字符集为ASCII，若客户端发送一个汉字到服务器，那服务器这边无法使用ASCII来编码这个汉字。

通常我们会把character\_set\_client character\_set\_connection character\_set\_results这三个系统变量设置成和客户端使用的字符集一致的情况，这样减少了无谓的字符集转换

Set names 字符集名;

这条命令产生的效果和执行这三条的效果一样：

Set character\_set\_client=字符集名;

Set character\_set\_connection=字符集名;

Set character\_set\_results=字符集名;

若想在启动客户端时就把这三个系统变量设置成一样，可以在配置文件中：

[client]

Default-character-set=utf8

比较规则的应用：

比较规则的作用通常比较字符串大小的表达式以及对某个字符串列进行排序中。也称为排序规则

在对字符串做比较或者对某个字符串列做排序操作时没有得到想要的结果，需要考虑下是不是比较规则的问题