MySQL プロトコル詳説

Perl からみた MySQL の通信プロトコル

小山浩之

oyama@cpan.org

Shibuya Perl Mongers

おしながき

- MySQL とのコミュニケーション
- 送受信するパケットの構造
- Perl での実装と Perl 特有の問題

おしながき

- MySQL とのコミュニケーション
- 送受信するパケットの構造
- Perl での実装と Perl 特有の問題

MySQL の通信プロトコルを解説し、いかに Perl で処理をおこなうか

MySQL とのコミュニケーション

- MySQL は Server と Client で構成。
- 両者は Socket で通信。
  - /tmp/mysql.sock *(UNIX domain socket)*
  - 3306/tcp *(INET socket)*
- Client は Server に Socket を繋ぎメッセージを交換して DB を操作。

MySQL とのコミュニケーション

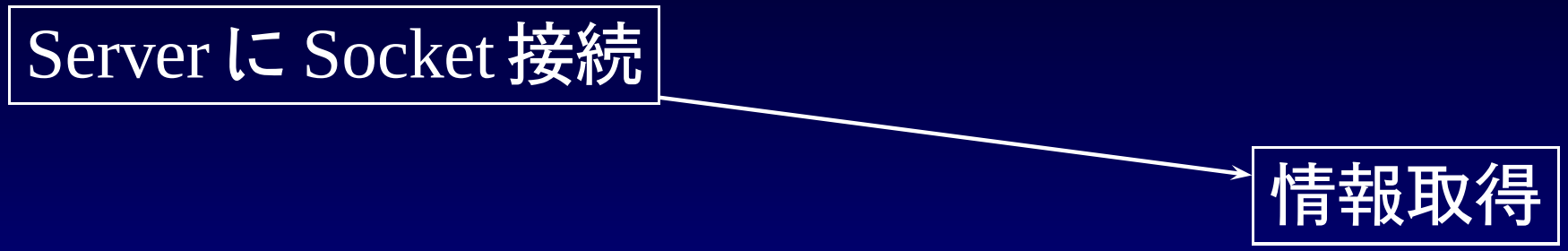
- MySQL は Server と Client で構成。
- 両者は Socket で通信。
  - /tmp/mysql.sock *(UNIX domain socket)*
  - 3306/tcp *(INET socket)*
- Client は Server に Socket を繋ぎメッセージを交換して DB を操作。

なんら特別なことはなくフツーに Socket を使った通信。→ *HTTP* や *SMTP* などと同列

Clientから見た通信シーケンス

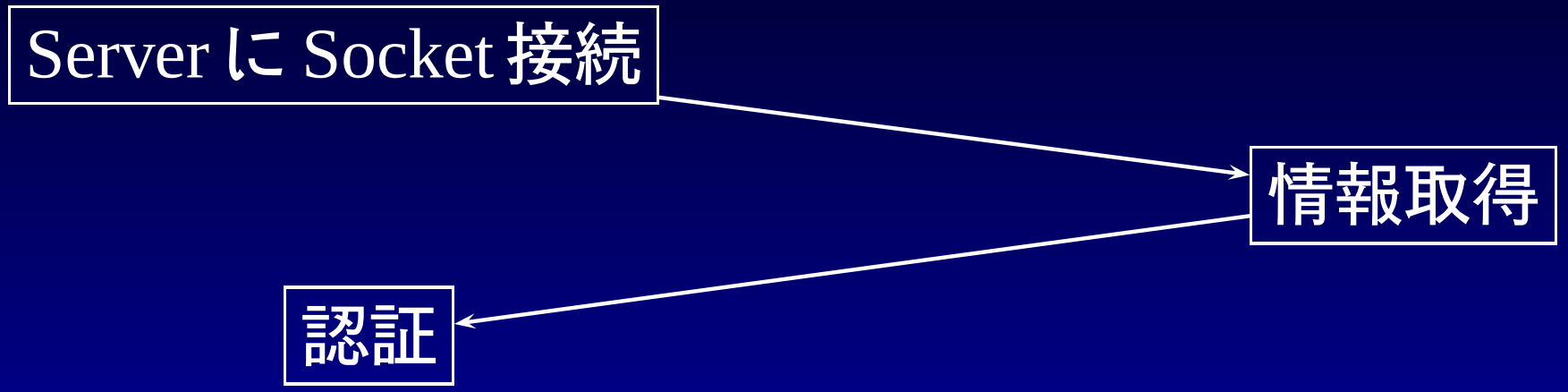
Server に Socket 接続

Clientから見た通信シーケンス



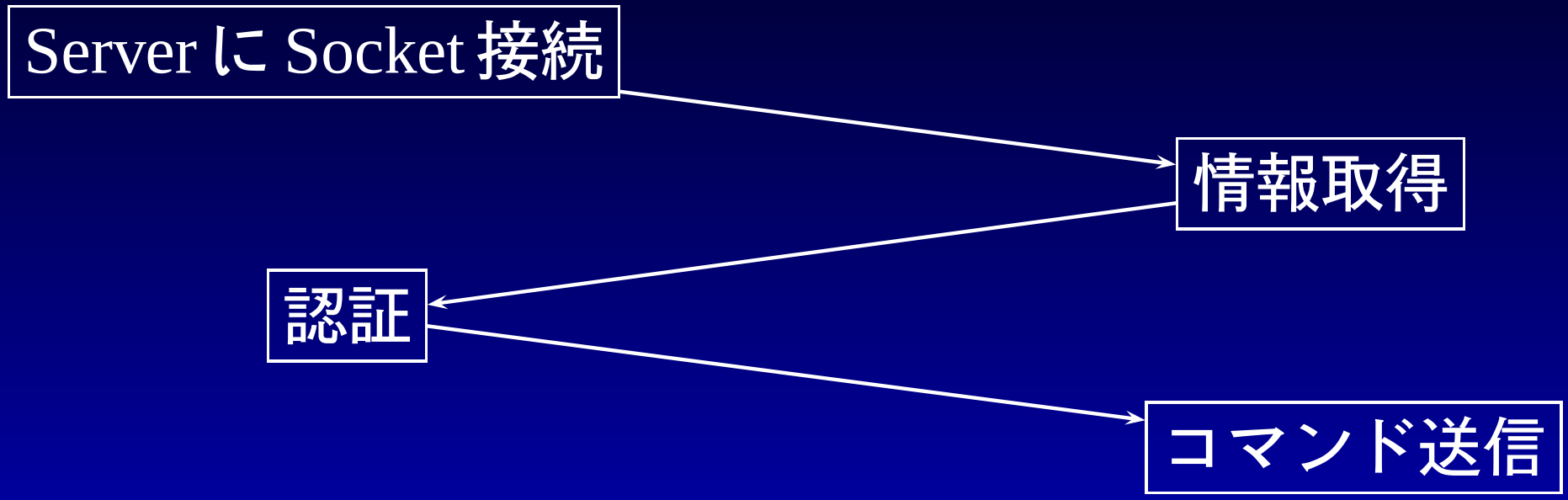
protocol version,
server version,
thread id,
salt を取得。

Clientから見た通信シーケンス



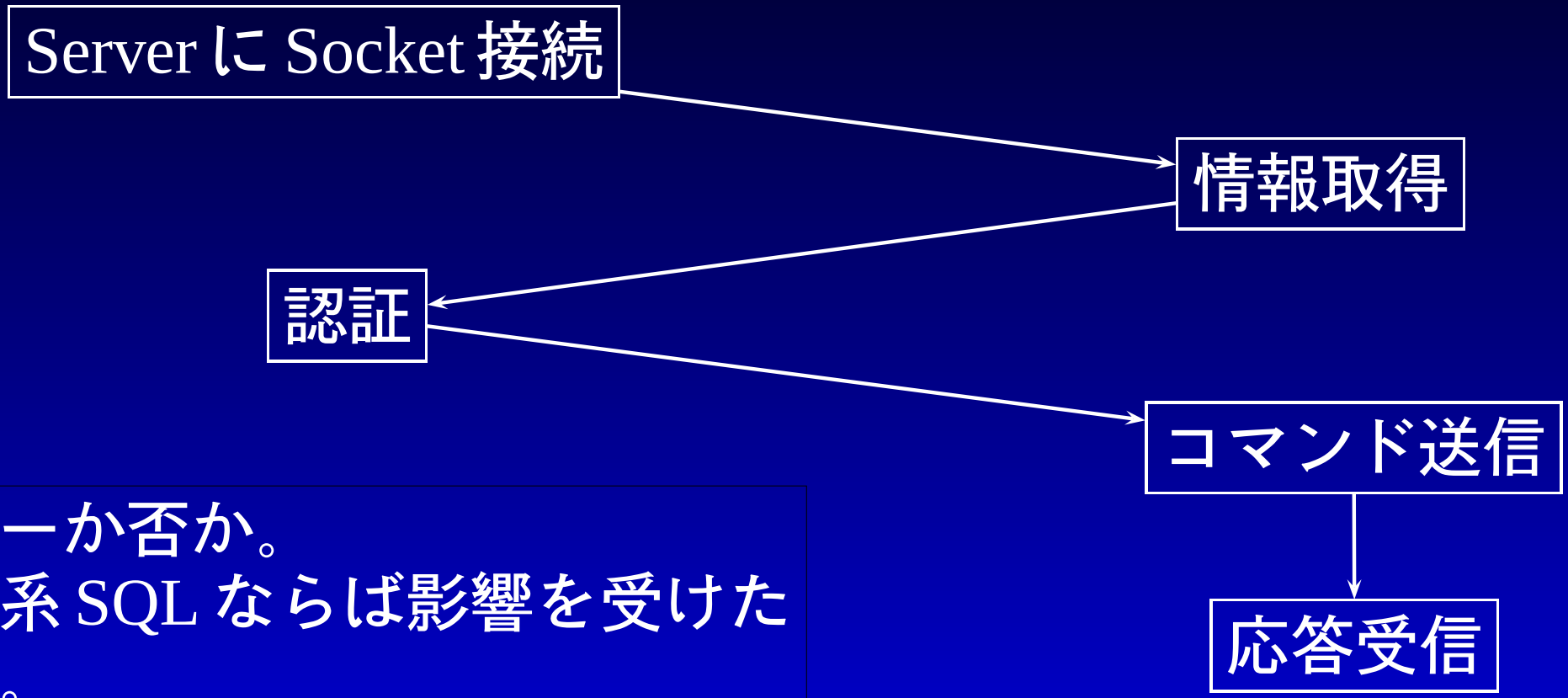
salt で password エンコードし送信。
(plain text ではない)

Clientから見た通信シーケンス



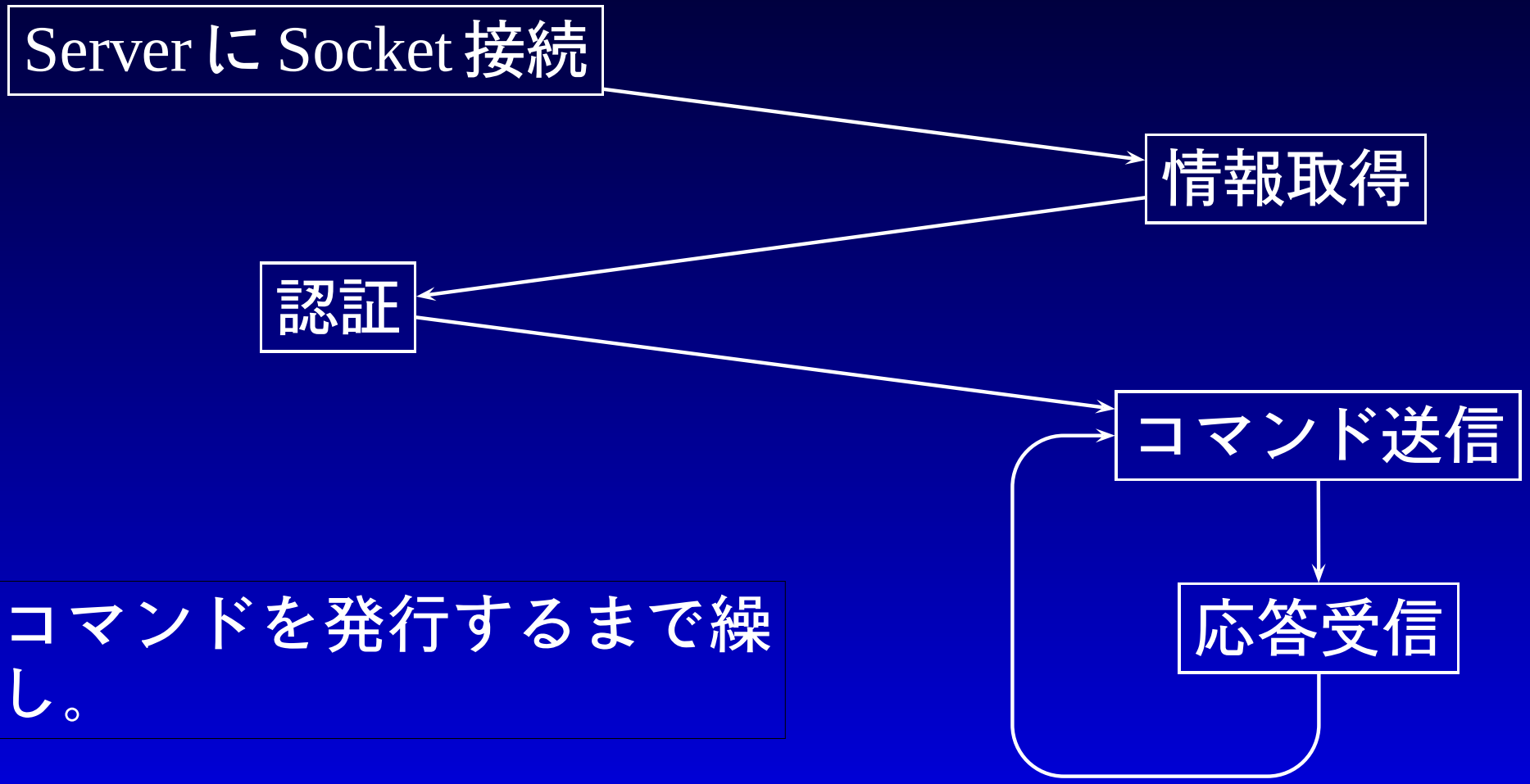
SQL や管理用コマンドを送信。
SQL の長さ (binary) + SQL

Clientから見た通信シーケンス



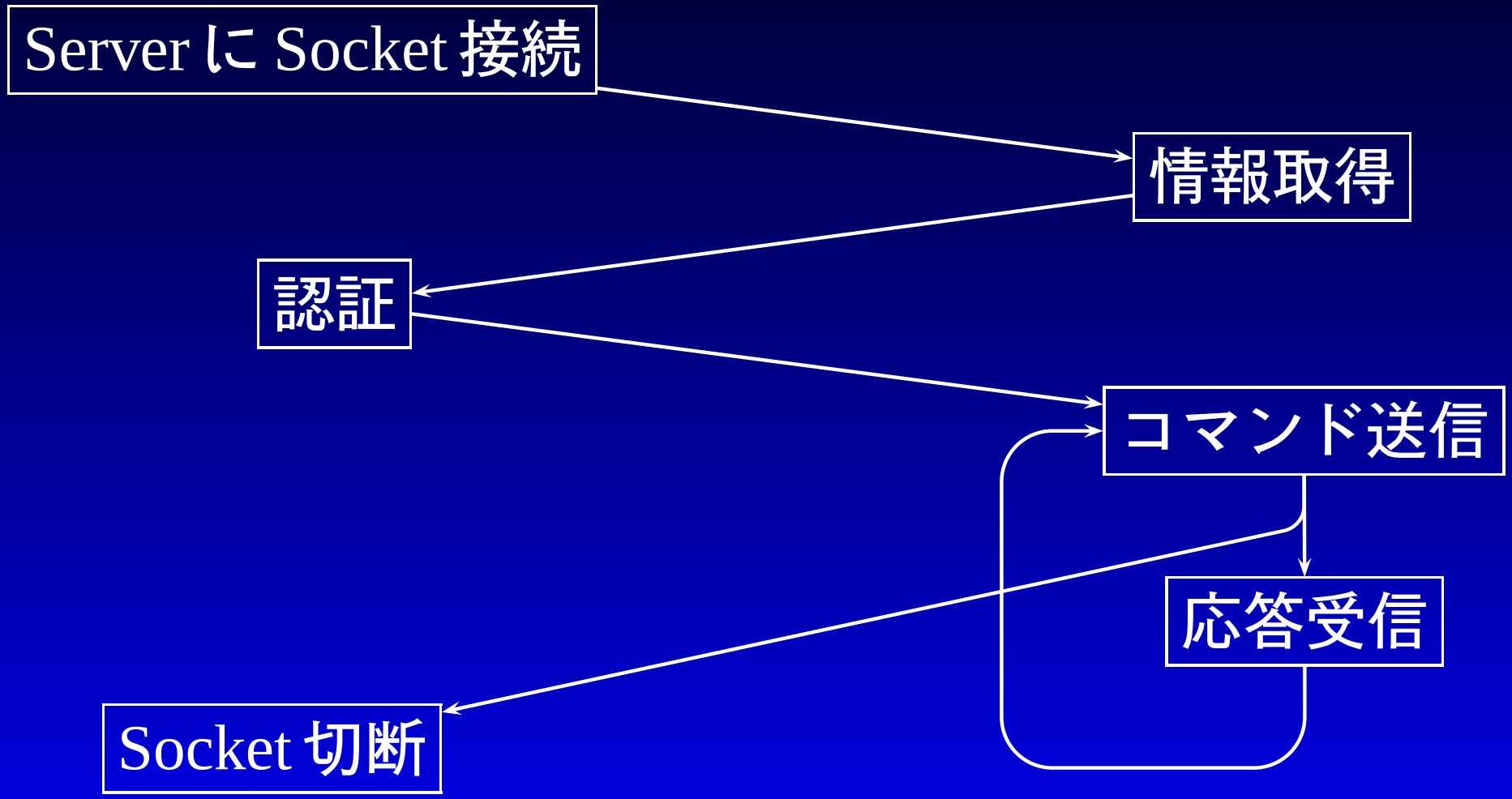
エラーか否か。
更新系 SQL ならば影響を受けた行数。
参照系 SQL ならカラム単位のデータ。

Clientから見た通信シーケンス



終了コマンドを発行するまで繰り返し。

Clientから見た通信シーケンス



Clientに要求される仕事

- UNIX or INET Socketで通信できること
- Binaryデータを操作できること

Clientに要求される仕事

- UNIX or INET Socketで通信できること
- Binaryデータを操作できること

つまり

- イマドキのOSとプログラミング言語の殆どがこの条件を満たす

送受信するパケットの構造

次の各処理におけるパケットの形式を説明します。

- ログインフェーズ
- クエリの発行
- 応答の受信

ログインフェーズ

Server に Socket を繋いだ後にまずやること。

- Server から接続情報を受信
- 認証情報を送信

Serverから接続情報を受信

```
28 00 00 00 0a 33 2e 32 33 2e 35 32
00 03 00 00 00 2b 21 60 44 6f 25 65
50 00 2c 00 08 02 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

Serverから接続情報を受信

```
28 00 00 00 0a 33 2e 32 33 2e 35 32
00 03 00 00 00 2b 21 60 44 6f 25 65
50 00 2c 00 08 02 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

パケットの長さ。

- Little-endian 32bit 整数 (unsigned long)

0x28000000 → 40byte

Serverから接続情報を受信

```
28 00 00 00 0a 33 2e 32 33 2e 35 32
00 03 00 00 00 2b 21 60 44 6f 25 65
50 00 2c 00 08 02 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

プロトコルバージョン。

- 8bit整数 (unsigned char)

0x0a → Version 10

Serverから接続情報を受信

```
28 00 00 00 0a 33 2e 32 33 2e 35 32
00 03 00 00 00 2b 21 60 44 6f 25 65
50 00 2c 00 08 02 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

サーババージョン。

- ASCII の文字列
- 末尾は"¥0"

0x33 0x2e 0x32 0x33 0x2e 0x35 0x32 0x00 → "3.23.52"

Serverから接続情報を受信

```
28 00 00 00 0a 33 2e 32 33 2e 35 32
00 03 00 00 00 2b 21 60 44 6f 25 65
50 00 2c 00 08 02 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

Thread ID

- Little-endian 32bit 整数 (unsigned long)

0x03000000 → 3

Serverから接続情報を受信

```
28 00 00 00 0a 33 2e 32 33 2e 35 32
00 03 00 00 00 2b 21 60 44 6f 25 65
50 00 2c 00 08 02 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

Passwordエンコード用 Salt

- 8Byteのバイナリ値(unsigned char[8])

Serverから接続情報を受信

```
28 00 00 00 0a 33 2e 32 33 2e 35 32
00 03 00 00 00 2b 21 60 44 6f 25 65
50 00 2c 00 08 02 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

あとは知らない。(´Д`;)いいんでつかっ

認証情報の送信

```
18 00 00 01 0d 24 00 00 00 72 6f 6f
74 00 5b 4e 57 41 57 58 4c 4f 00 6d
79 73 71 6c
```

認証情報の送信

```
18 00 00 01 0d 24 00 00 00 72 6f 6f
74 00 5b 4e 57 41 57 58 4c 4f 00 6d
79 73 71 6c
```

パケットの長さ。

- 8bit の整数 (unsigned char)

認証情報の送信

```
18 00 00 01 0d 24 00 00 00 72 6f 6f
74 00 5b 4e 57 41 57 58 4c 4f 00 6d
79 73 71 6c
```

ユーザ名。

- ASCII の文字列
- 末尾は"¥0"

0x72 0x6f 0x6f 0x74 0x00 → "root"

認証情報の送信

```
18 00 00 01 0d 24 00 00 00 72 6f 6f
74 00 5b 4e 57 41 57 58 4c 4f 00 6d
79 73 71 6c
```

Salt でエンコードした Password。

- ASCII の文字列
- 末尾は"¥0"

0x5b 0x4e 0x57 0x41 0x57 0x58 0x4c 0x4f → "[XWAWXLO"

認証情報の送信

```
18 00 00 01 0d 24 00 00 00 72 6f 6f
74 00 5b 4e 57 41 57 58 4c 4f 00 6d
79 73 71 6c
```

データベース名。

- ASCII の文字列

0x6d 0x79 0x73 0x71 0x6c → "mysql"

クエリの発行

単純に SQL 文を送信するだけ。

SQLの送信

```
1c 00 00 00 03 73 65 6c 65 63 74 20
68 6f 73 74 2c 20 75 73 65 72 20 66
72 6f 6d 20 75 73 65 72
```

SQLの送信

```
1c 00 00 00 03 73 65 6c 65 63 74 20
68 6f 73 74 2c 20 75 73 65 72 20 66
72 6f 6d 20 75 73 65 72
```

パケットの長さ。

- Little-endian 32bitの整数値(unsigned long)

0x1c 0x00 0x00 0x00 → 28byte

SQL の送信

```
1c 00 00 00 03 73 65 6c 65 63 74 20
68 6f 73 74 2c 20 75 73 65 72 20 66
72 6f 6d 20 75 73 65 72
```

コマンド番号。

- 8bit の整数値 (unsigned char)
- 0 から 20 までの整数でコマンドを表す。
  - QUIT = 1
  - QUERY = 3
  - CHANGE_USER = 17

0x03 → Query コマンド

SQLの送信

```
1c 00 00 00 03 73 65 6c 65 63 74 20
68 6f 73 74 2c 20 75 73 65 72 20 66
72 6f 6d 20 75 73 65 72
```

SQL。

- 文字列
- ¥, ¥0, ¥n, ¥r, ", ’, 0x1a は"¥"でエスケープ

→ "select host, user from user"

応答の受信

クエリの応答を受信する。

- 参照系 SQL - SELECT
- 更新系 SQL - UPDATE, INSERT

要求によって応答の形式が異なる。

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

カラム数。

0x02 $\rightarrow$ ２つのフィールドを持つ応答

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

長さ テーブル名 → "user"

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

長さ カラム名 → "host"

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

型情報 etc

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

長さ テーブル名 長さ カラム名 型情報

→ "user.user"

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

長さ データ, 長さ データ

→ "localhost" "oyama"

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

長さ データ, 長さ データ
長さ データ, 長さ データ

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

長さ データ, 長さ データ
長さ データ, 長さ データ
長さ データ, 長さ データ

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

9 "localhost", 5 "oyama"

13 "www.module.jp", 6 "nobody"

13 "www.module.jp", 5 "wheel"

レコードの受信

```
01 00 00 01 02 14 00 00 02 04 75 73
65 72 04 68 6f 73 74 03 3c 00 00 01
fe 03 83 40 00 14 00 00 03 04 75 73
65 72 04 75 73 65 72 03 10 00 00 01
fe 03 83 40 00 01 00 00 04 fe 10 00
00 05 09 6c 6f 63 61 6c 68 6f 73 74
05 6f 79 61 6d 61 15 00 00 06 0d 77
77 77 2e 6d 6f 64 75 6c 65 2e 6a 70
06 6e 6f 62 6f 64 79 14 00 00 07 0d
77 77 77 2e 6d 6f 64 75 6c 65 2e 6a
70 05 77 68 65 65 6c 01 00 00 08 fe
```

終端

更新件数の受信

```
2c 00 00 01 00 03 00 28 52 6f 77 73
20 6d 61 74 63 68 65 64 3a 20 33 20
20 43 68 61 6e 67 65 64 3a 20 33 20
20 57 61 72 6e 69 6e 67 73 3a 20 30
```

更新件数の受信

```
2c 00 00 01 00 03 00 28 52 6f 77 73
20 6d 61 74 63 68 65 64 3a 20 33 20
20 43 68 61 6e 67 65 64 3a 20 33 20
20 57 61 72 6e 69 6e 67 73 3a 20 30
```

影響を受けたレコード数

更新件数の受信

```
2c 00 00 01 00 03 00 28 52 6f 77 73
20 6d 61 74 63 68 65 64 3a 20 33 20
20 43 68 61 6e 67 65 64 3a 20 33 20
20 57 61 72 6e 69 6e 67 73 3a 20 30
```

Insert-ID

更新件数の受信

```
2c 00 00 01 00 03 00 28 52 6f 77 73
20 6d 61 74 63 68 65 64 3a 20 33 20
20 43 68 61 6e 67 65 64 3a 20 33 20
20 57 61 72 6e 69 6e 67 73 3a 20 30
```

メッセージの長さ

0x28 → 40byte

更新件数の受信

```
2c 00 00 01 00 03 00 28 52 6f 77 73
20 6d 61 74 63 68 65 64 3a 20 33 20
20 43 68 61 6e 67 65 64 3a 20 33 20
20 57 61 72 6e 69 6e 67 73 3a 20 30
```

応答メッセージ → "Rows matched: 3¥n Changed: 3¥n Warnings: 0"

MySQLプロトコルのまとめ

- データ長・コマンド番号以外のデータはASCIIが基本。
- binary値はLittle-endian。8bit, 16bit, 24bit, 32bitを使い分ける。
- 参照の応答にはレコード数などの情報は無し。→ client側で計算。
- シンプルでコンパクト。

Perlでどう実装する？

- UNIX or INET Socket を開く
- Little-endian の 8, 16, 24, 32bit 値を処理する
- Password のエンコード

解決すべきネタはこの程度。

Perlでどう実装する？

- UNIX or INET Socket を開く
- Little-endian の 8, 16, 24, 32bit 値を処理する
- Password のエンコード

解決すべきネタはこの程度。

→ 実装の検証は tcpdump で既存の client と比較・追従すれば良い。

UNIX or INET Socketを開く

旅の友だち `use IO::Socket;`

```
my $mysql = IO::Socket::UNIX->new(
    Type => SOCK_STREAM,
    Peer => '/tmp/mysql.sock',
);
```

でも

```
my $mysql = IO::Socket::INET->new(
    PeerAddr => $hostname,
    PeerPort => 3306,
    Proto => 'tcp',
);
```

でもお好きなように。

Binary 値のハンドリング

三種の神器 pack()/unpack()/substr();

- 8bit 値 (unsigned char) の変換

```
my $number = unpack 'C', $char8;
my $char8 = pack 'C', $number;
```

- Little-Endian な 16bit 値 (unsigned short) の変換

```
my $number = unpack 'v', $short16;
my $short16 = pack 'v', $number;
```

- Binary 文字列から任意の byte を切り出し

```
my $char8 = substr $raw_data, $position, 1;
my $short16 = substr $raw_data, $position, 2;
```

桁の小さい値なら簡単。

Little-endian 24bit, 32bit 値

- Little-endian な 24bit 値の変換
  んなテンプレートは無いので

```
my $int24 = substr $packet, $position, 3;
my $number = unpack('C', substr $int24, 0, 1)
  + (unpack('C', substr $int24, 1, 1) << 8)
  + (unpack('C', substr $int24, 2, 1) << 16);
```

- Little-endian な 32bit 値の変換

```
my $int32 = substr $packet, $position, 4;
my $number = unpack 'V', $int32;
```

Passwordのエンコード

最大の難関

- そもそもMySQLのプロトコルに関するドキュメントが無い(とおもう)
- アルゴリズムの名前がわからない
  → libmysql.cや出回っている２つのJDBCドライバのコードにもアルゴリズム名が書いていない。

Passwordのエンコード

最大の難関

- そもそもMySQLのプロトコルに関するドキュメントが無い(とおもう)
- アルゴリズムの名前がわからない
  → libmysql.cや出回っている２つのJDBCドライバのコードにもアルゴリズム名が書いていない。
  - じゃぁそのままC to PerlやJava to Perlの変換♪

Passwordのエンコード

最大の難関

- そもそもMySQLのプロトコルに関するドキュメントが無い(とおもう)
- アルゴリズムの名前がわからない
  → libmysql.cや出回っている２つのJDBCドライバのコードにもアルゴリズム名が書いていない。
  - じゃぁそのままC to PerlやJava to Perlの変換♪

しかしそこにはPerlの"闇"が潜んでいた...

Passwordエンコードの問題

32bit超の数値をグリグリ論理演算(xor, and etc..)。

Passwordエンコードの問題

32bit超の数値をグリグリ論理演算(xor, and etc..)。

- Perlで32bit以上の数値は論理演算できない。

Password エンコードの問題

32bit 超の数値をグリグリ論理演算 (xor, and etc..)。

- Perl で 32bit 以上の数値は論理演算できない。
  → OS の int 幅に依存。

Password エンコードの問題

32bit 超の数値をグリグリ論理演算 (xor, and etc..)。

- Perl で 32bit 以上の数値は論理演算できない。
  → OS の int 幅に依存。
- ならば `use Math::BigInt;`

Passwordエンコードの問題

32bit超の数値をグリグリ論理演算(xor, and etc..)。

- Perlで32bit以上の数値は論理演算できない。
  → OSのint幅に依存。
- ならば`use Math::BigInt;`
  → 遅すぎて認証が通らない環境があった。

Passwordエンコードの問題

32bit超の数値をグリグリ論理演算(xor, and etc..)。

- Perlで32bit以上の数値は論理演算できない。
  → OSのint幅に依存。
- ならば`use Math::BigInt;`
  → 遅すぎて認証が通らない環境があった。
- しかたないので32bit幅に丸めながら計算。

Passwordエンコードの問題

32bit超の数値をグリグリ論理演算(xor, and etc..)。

- Perlで32bit以上の数値は論理演算できない。
  → OSのint幅に依存。
- ならば`use Math::BigInt;`
  → 遅すぎて認証が通らない環境があった。
- しかたないので32bit幅に丸めながら計算。
  → 5.6.0以前のPerlは32bit幅超の割算にも支障がっ

Passwordエンコードの問題

32bit超の数値をグリグリ論理演算(xor, and etc..)。

- Perlで32bit以上の数値は論理演算できない。
  → OSのint幅に依存。
- ならば`use Math::BigInt;`
  → 遅すぎて認証が通らない環境があった。
- しかたないので32bit幅に丸めながら計算。
  → 5.6.0以前のPerlは32bit幅超の割算にも支障がっ

結局32bit幅にイチイチ丸めながら計算するが、5.6.0以前のPerlは"引算"で丸め処理。(ダサっ)

その成果

`Net::MySQL, DBD::mysqlPP`
100% Pure Perl の MySQL クライアントライブラリ。

- 2002 年 3 月 25 日頃調査と開発開始
- 即、私のエイプリルフールネタに採用 (逆です)
- 何故か利用者がボチボチいるようだ

今後の展望

今後の展望

一刻も早く共同開発者をゲットして引き継ぐ！(本気)

今後の展望

一刻も早く共同開発者をゲットして引き継ぐ！(本気)

というわけで共同開発者大募集中です。