MySQLハッキングの手引き

松信 嘉範 (MATSUNOBU Yoshinori)

日本MySQLユーザー会

http://opendatabaselife.blogspot.com

http://twitter.com/matsunobu

今日のテーマ

- MySQLのアーキテクチャ
- ソースコードの入手とビルド方法
- デバッグ方法
- プラグイン開発の方法
- 本体拡張の方法
- 本家へのコントリビュートの方法

自己紹介

- 本業は「MySQLコンサルタント」
  - パフォーマンスチューニング、HA等
  - たまにMySQL開発案件を行なう

- 書籍/連載
  - 「現場で使えるMySQL」 2006.3
  - 「Javaデータアクセス実践講座」 2008.2
  - Linux-DBに関する本 2009.9(予定)
  - 新連載「Real World MySQL」 DBマガジン、10月くらいから
  - そろそろ「RDBMS自作入門」のような本に挑戦しようかと。。

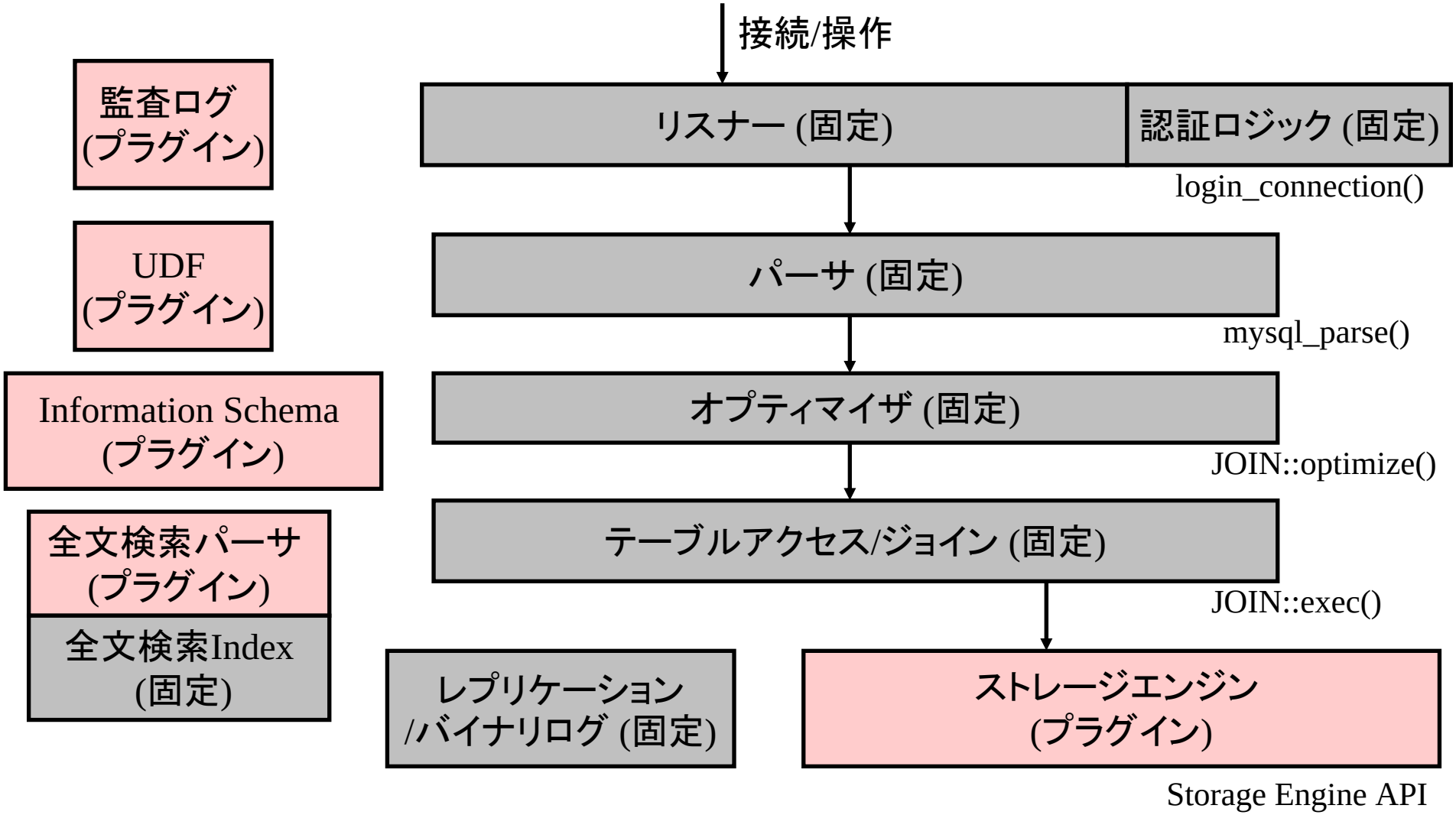
MySQLとは

- オープンソースのリレーショナルデータベース
- C/C++、一部アセンブラで実装

- 現在提供されているバージョン
  - 5.4 … パフォーマンス改善版MySQL、beta
  - 5.1 … 最新の安定版 (5.2/5.3は欠番)
  - 5.0 … 安定版
  - 4.1 … 安定版
  - 4.0 … 安定版 (サポート切れ)

- マイナーリリースが数ヶ月程度のサイクルで登場
  - ソースコード、およびプラットフォームごとのバイナリが提供

MySQLのアーキテクチャ



さまざまな特徴 (1)

- 外部ライブラリには極力依存しない作り
  - STLとかboostとかも使っていない
  - プラットフォーム依存性を減らすため
  - OS依存の命令は、my_xxxというラッパー関数で吸収
  - Drizzleでは方針が逆転している
    - STL、ProtocolBuffer、Gearman、etc..

- デバッグ用の機能
  - malloc/free、pthread_*などはラッパーを用意して、二重ロックや不正アクセス等を
    検知しやすくしている
  - 主要関数へのIN/OUT時にトレースを出すようにしている
  - Production用では無効化するように、マクロで制御している
  - MySQL5.4以降では、Dtraceプローブが埋め込まれている

さまざまな特徴 (2)

- エンディアンフリー
  - 基本的にリトルエンディアンに統一
    - 「1」の2バイト表現は0x01 00
  - 通信プロトコルもエンディアンフリー
- 関数ポインタ、サブクラスを多用し汎用性を上げる
  - オブジェクト指向の定石でもある
  - ストレージエンジンAPIなどプラグインを実現するために重要
  - SQL関数ごとにサブクラス、1レコードの処理ハンドラ用に関数ポインタ等
- 実行速度重視のビルド
  - 標準ビルドではRTTIを無効化
  - この影響で、実行時にPASSWORD()関数の場合にはこうして。。という特殊な処理が難しい

MySQLを拡張する

- MySQLハッキングのパターンは、大きく分けて2種類
  - MySQL本体に手を入れる
  - MySQLの「プラグイン」を開発する

- プラグインは共有ライブラリ (*.so, *.dll)
  - MySQL本体の改変が不要→標準バイナリにインストール可能
  - プラグインインターフェイスの範囲内で好きなように実装可能
  - 要件によっては、実装できない(本体の改変が必須な)場合がある
    - 例:ファイル内static変数には外部からアクセスできない
    - 例:監査ログインターフェイスはデバッグ目的では使えない
      (SQL文の実行後に呼ばれるため、実行時にクラッシュしても呼ばれない)

- MySQL本体を改変すれば何でもできる
  - 標準バイナリは当然ながら使えなくなる
  - 有用なものはMySQL本家にコントリビュートすることも可能

MySQLのソースコード入手方法

- MySQL本家サイトから入手
  - http://dev.mysql.com/downloads/mysql/5.1.html
  - tar.gz形式
  - マイナーリリースごとに公開

- MySQLソースコードツリーから入手
  - MySQLはソースコード管理に「bzr」を使用
  - Launchpad上でホストしている
    - https://code.launchpad.net/mysql-server
  - shell> bzr branch lp:mysql-server/5.4 mysql-5.4
  - 最新のものがほしい場合はこちらから

開発の流れ

- 改変したいバージョンのソースコードをダウンロード
- 初期ビルド→改変→デバッグビルド→テスト→改変…
  - cd /path/to/mysql-top
  - BUILD/autorun.sh
  - ./configure --多数のオプション
  - make
  - make install
- プラグイン開発の場合も、デバッグビルドしたMySQLを使う
  - プラグイン/MySQL本体ともにデバッグビルドで開発
- ビルド・デバッグ手段
  - Linux/その他：gcc, gdb, ddd, Eclipse/NetBeans, etc
  - Windows: Visual Studio

Configureオプション

- BUILD/以下にテンプレートがある
- バージョンによってconfigureオプションが違うので注意

- --prefix .. インストール先のディレクトリ
- --with-debug .. デバッグビルドか、通常ビルドか
- --with-comment .. 接続時に表示される文字列
- --with-plugins=
  - partitions … パーティション機能をサポートする
  - innobase … InnoDBストレージエンジンを入れる
  - archive … Archiveストレージエンジンを入れる
  - …
- --with-extra-charsets=complex .. 日本語等の対応
- --with-fast-mutexes
- --enable-thread-safe-client
- --enable-local-infile
- --with-pic
- ….

Visual Studio

- DOSプロンプトからwin¥configure.js を実行
  - 引数にWITH_PARTITION_STORAGE_ENGINE WITH_INNOBASE_STORAGE_ENGINE …などをつける
- DOSプロンプトからwin¥build-v8.batを実行
- Visual Studioを起動
- 「ビルド」ボタンを押す (sql_locale.ccなどは「UTF-8N (BOMなし)」で保存しなおす必要がある)
- mysqldプロジェクトを「スタートアップ・プロジェクト」に指定
- 右クリック→プロパティ→デバッグ→「コマンド引数」に起動オプションを指定
- mysqlデータベースをデータディレクトリに配置
- MySQLを起動

デモ(1) デバッグ / トレース

- MySQLに手を入れる予定が無くても、
  デバッグスキルを持つことには意味がある
  - クラッシュの原因分析のため
  - 動作原理を知るため
    - プロトコルの動作原理
    - クエリの実行の流れ

- Visual Studioから起動したMySQLサーバに対し、
  MySQLクライアントから接続を行い、
  どのように処理が流れていくかを紹介

プラグイン化の流れ

- 本体への機能改変やコントリビュートで起こる問題
  - そもそも本体の改変は難易度が高い
  - 投稿したパッチがいつまでもマージされない
  - 本体はQAの観点からもパッチの取り入れには保守的
- プラグインは、本体を改変せずに使うことができる
- MySQL本体でパッチが取り込まれるのを待つ必要が無い
- 本体側の改変が必要な場合は、バグレポートやWork Logが必要
- 全体的にはプラグイン化を推し進める流れ
- プラグイン化の流れをさらに推し進めたのがDrizzle

- MySQLでプラグインになっているもの
  - ユーザ定義関数 (4.0 -)
  - ストレージエンジン (5.1 -)
  - Information Schemaプラグイン (5.1 -)
  - 全文検索用パーサ (5.1 -)
  - 監査証跡 (5.4次 -)
  - セミ同期レプリケーション (5.4次 -)

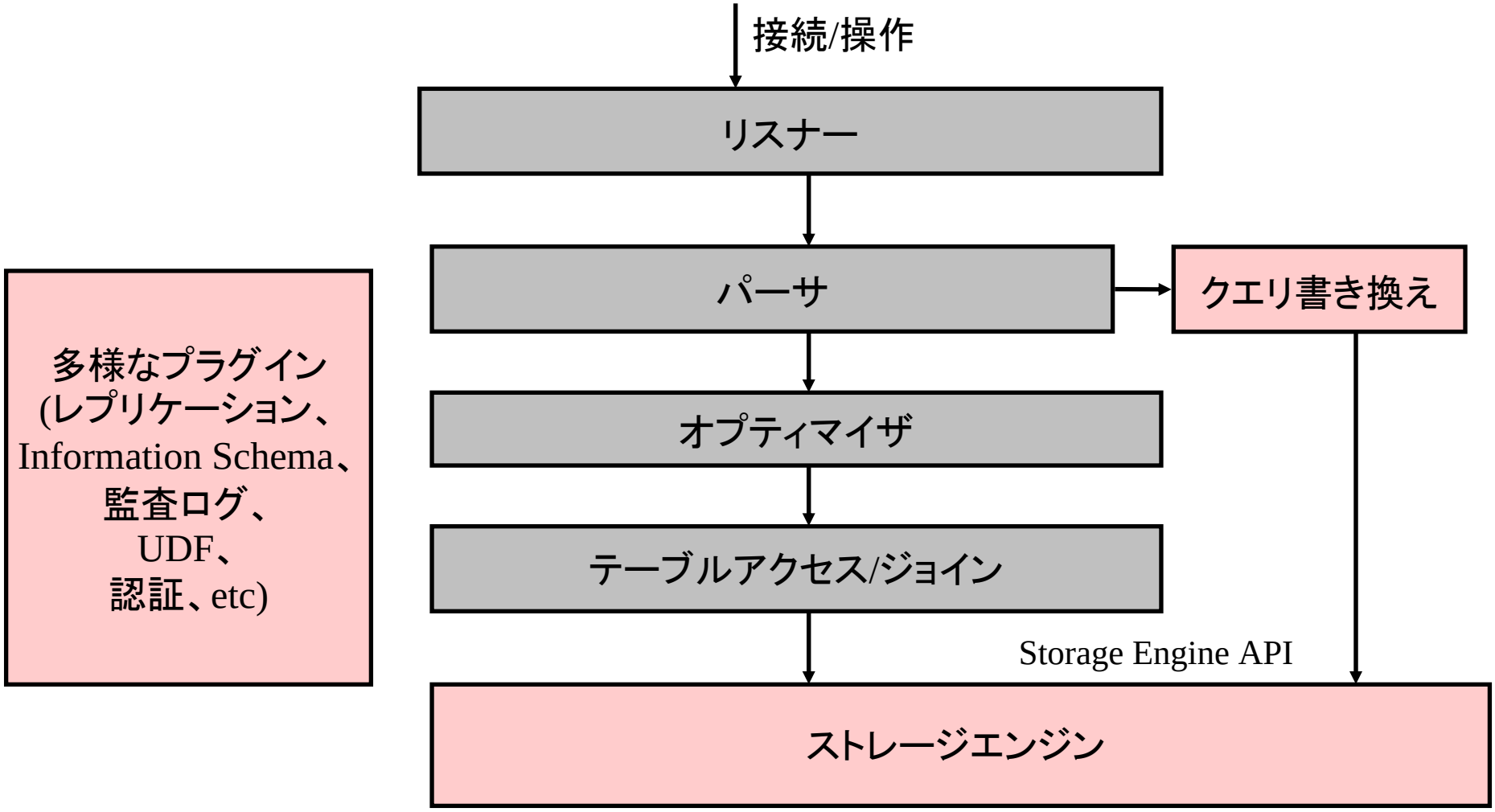
デモ (2) プラグイン開発

- Information Schemaプラグイン
  - cat /proc/meminfoの結果を返してみる
- UDF
  - ストレージエンジンAPIを呼んで、テーブルに直接アクセスしてみる
  - Key Value Storeを使うような感覚
  - 「友人100人の、最新の投稿メッセージIDを取得する」クエリを考えてみる
  - SELECT id FROM message WHERE user_id=? ORDER BY id DESC LIMIT 1;
  - これをuser_idを変えて100回繰り返す
  - (user_id IN (…)だとMySQLではフルインデックススキャンになってしまう)
  - ストアド化すると高速化が期待できる
  - UDFをストアドのかわりに使うという例

プラグイン開発の一般的な注意事項

- MySQL本体をクラッシュさせてしまう危険がある
  - プラグインを実行するプロセスはmysqld
  - プラグイン内部の不正メモリアクセスでも、mysqldがSIGSEGVで落ちる
  - 悪意あるユーザがバグつきプラグインをINSTALLしてクラッシュさせることも可能
  - 一般ユーザに権限付与しないこと
- セキュリティホールになりうるので権限管理に注意
  - mysqldをroot権限で動かさないこと
    - プラグインはOSコマンドを実行できる。実行者はmysqldの権限になる
- プラグインの中身によっては、バージョン依存性が高くなる
  - MySQL本体で使っているマクロの一部は、展開のされ方がデバッグ版と通常版、configureオプションなどによって変わる
  - そのマクロをプラグインから使う場合には当然依存性が出る

次世代のRDBMSアーキテクチャ?



MySQL本体に手を入れる

- 自分で使うだけなら、ソース改変→ビルド→テストで良い
- MySQL本家に反映してもらうには、
  「レビュー」のプロセスが必要
- 多くの場合、「互換性の確保」が大きな壁になる

- WorkLogに登録
  - http://forge.mysql.com/worklog/
  - 新機能の場合はこちらが良い

- またはバグレポートに投稿
  - 単なるバグの場合はこちらが良い

デモ(3) MySQL本体拡張

- レコードの最大サイズ(TEXT/BLOB除く)を64KBから拡張
- バイナリログの「追記」→「上書き」への変更
- インデックスへのコメント記述機能

- レコードサイズ拡張
  - CREATE TABLE t1 (c1 VARCHAR(50000), c2 VARCHAR(50000)); を作れるようにする
  - max_row_lengthは2バイトで.frm上で管理されている
  - .frmのフォーマット変更が必要
  - バージョン互換への配慮が必要

ソースコードレベルの議論

- 技術的な議論
  - internals@lists.mysql.com
- バグレポート
  - http://bugs.mysql.com
  - パッチの投稿も可能
- パッチの投稿・レビュー・議論
  - commits@lists.mysql.com
  - コミットメールをここに投稿する。レビューおよびその後の議論はすべてこれをCCに入れて行なう

WorkLogとは

- MySQLの新機能の開発計画を管理
- http://forge.mysql.com/worklog/

- 流れ
  - WorkLogに新規エントリを投稿
  - SunのMySQL開発者がレビュー担当になる
  - Sun Contributor Agreement (SCA)を締結
  - パッチをコミット
  - パッチをレビュー
  - レビュー結果を踏まえてパッチをコミット
  - レビュー完了
  - 本家にマージ

パッチのレビュー

- コーディング規約に準拠していること
- テストケースが用意されていること
  - mysql-testテストケース
- その他常識的なコードであること
  - スレッドセーフである
  - 効率的な実装になっている
  - 変数名が自然
  - コメントが読みやすい

コーディング規約 (抜粋)

- http://forge.mysql.com/wiki/MySQL_Internals_Coding_Guidelines

- インデント/スペース/行管理
  - インデントには半角スペース2個。タブは使用禁止
  - 改行は¥nを使用。CR+LF(¥r¥n)は禁止。
  - 1行は最大80バイト
  - 1つの関数内に、連続した2行以上の空白行を入れてはいけない
  - 関数と関数の区切りには2行の改行を入れる

- 変数の代入、条件分岐
  - x= 1; のように、変数名と演算子の間にスペースを入れない。演算子の後にはスペースを1個入れる。
  - ifと(の間には半角スペースを1個入れる
  - If { ではなく ifと{の間には改行を入れる
  - などなど多数

テスト

- mysql-testという単体テストツールを走らせる
- http://dev.mysql.com/doc/mysqltest/en/index.html
- 単体テストツール
  - mysql-test-run
  - perl mysql-test-run.pl
- ストレステストツール
  - mysql-stress-test.pl
- 回帰テスト用に使う
- 1つの機能追加にあたり、最低でも1個の妥当なテストケースを追加

WorkLogの例

- 監査プラグイン
  - http://forge.mysql.com/worklog/task.php?id=3771
- セミ同期レプリケーション
  - http://forge.mysql.com/worklog/task.php?id=1720
- Preallocating Binlog
  - http://forge.mysql.com/worklog/task.php?id=4925

バイナリログの追記→上書きへの変更パッチ (1)

- 耐障害性のため、コミット時に同期書き込みするニーズがある
- sync-binlog=1
- 大半のファイルシステムで、追記→同期書き込みは、上書き→同期書き込みに比べて圧倒的に遅い
  - 10,000+ fsync/s vs 3,000 fsync/sくらいの違いはある
  - ファイルサイズを増やすための領域確保のオーバーヘッドが大きい
- バイナリログは追記で、書き込みのたびにファイルサイズが増える
- 最初から上書きしておけば速くなるはず

バイナリログの追記→上書きへの変更パッチ (2)

- 技術的課題
  - 最初にサイズを割り当てるが、そのコストをゼロにしたい
    - 専用コマンドを用意して割り当てておく(手動)、posix_fallocate()使用(自動)
  - レプリケーション、mysqlbinlog、SHOW BINLOG EVENTS等既存のすべてのコマンドが正常動作する必要がある
    - バイナリログを転送しきったら、次のイベントが来るまで待つという動作
    - 転送しきったかどうかはEOFで判定
    - これは上書き確保した場合にはできない。ファイルサイズを内部的に管理しておく必要がある
    - ファイルサイズが都度更新されるので、スレッドセーフになるようにファイルサイズを管理する必要がある
    - 今は、「すでに存在しているバイナリログの最大番号+1」を新規作成して使用する
    - これでは手動で割り当てたバイナリログが使われない

バグレポートの投稿

- バグレポートの投稿
  - http://bugs.mysql.com/report.php
  - アカウント登録が必要

  - Synopsis：現象を1行程度で登録
  - Category
  - Severity
  - Version
  - OS
  - Description
  - How to repeat
  - パッチをここで投稿することもできる

バグレポートから本体マージの例

- 「UPPER()/LOWER()関数がシフトJIS環境で正しく動作しない」
- http://bugs.mysql.com/bug.php?id=44352
- 4月18日に立岡さんよりパッチ含めて投稿
- 5月4日に修正版パッチのレビューが開始
- 5月28日にMySQL5.1.36本体への反映

- 簡単なものでも数ヶ月かかるものもあるし、
  1年以上放置される場合もありますorz

参考文献

- 基本はオンライン上で英語(Web、メーリングリスト、IRC)
  - http://forge.mysql.com
  - internals@lists.mysql.com
  - #mysql-dev

- 書籍の情報はどれも古いので参考程度に
  - Expert MySQL
    - UDF、ストレージエンジンを自作する方法など
  - Understanding MySQL Internals
    (翻訳本：詳解MySQL、オライリー)
    - THDやTABLEなど主要なデータ構造
  - MySQL構築バイブル (毎日コミュニケーションズ)
    - 通信プロトコル、関数の自作など
  - 超極める MySQL (翔泳社)
    - ひろせさんのUDF解説記事

ありがとうございました

- Q & A