安全なSQLの呼び出し方

HASHコンサルティング株式会社
徳丸 浩
Twitter id: @ockeghem

アジェンダ

1. リテラルとSQLインジェクション
2. SQLの呼び出し方
3. データベースと連動したSQL文生成
4. DBMS製品の実態調査

付録A. 技術情報

1.リテラルとSQLインジェクション

SQL文の構成要素

```
SELECT a,b,c FROM atable WHERE name='YAMADA' and age>=20
```

SQL

SQL 文を構成する要素には、キーワード、演算子、識別子、リテラルなどがあります。

キーワード(予約語)	SELECT FROM WHERE AND
演算子など	= >= ,
識別子	a b c atable name age
リテラル	'YAMADA' 20

独立行政法人情報処理推進機構「安全なSQLの呼び出し方」より引用 http://www.ipa.go.jp/security/vuln/websecurity.html

リテラルとは

【数値リテラルの例】

```
20
-17
0
3.14159
6.0221415E+23
```

SQL

【文字列リテラルの例】

```
'情報処理推進機構'
'052312'
'O''Reilly'
```

SQL

【日時リテラルの例】

```
DATE '2009-11-04'
TIME '13:59:26'
```

SQL

独立行政法人情報処理推進機構「安全なSQLの呼び出し方」より引用 http://www.ipa.go.jp/security/vuln/websecurity.html

文字列リテラルのエスケープとSQLインジェクション

以下のSQL

```
SELECT * FROM employee WHERE name = '$name'
```

O'Reillyを検索したい場合、単に文字列連結でパラメータとして与えると以下のように
Reillyがリテラルをはみ出す

```
SELECT * FROM employee WHERE name = 'O'Reilly'
```

はみ出した部分はSQL文として意味をなさないのでエラーになるが、エラーにならないように
入力を調整することも可能。これがSQLインジェクション
$name = "';DELETE FROM employee--" とした場合、SQL文は以下となる

```
SELECT * FROM employee WHERE name = '';DELETE
FROM employee--'
```

リテラルをはみだすことを防ぐには、以下のようにシングルクォートを重ねる

```
SELECT * FROM employee WHERE name = 'O''Reilly'
```

数値リテラルのエスケープとSQLインジェクション

以下のSQLについて。$idは整数型

```
SELECT * FROM employee WHERE id = $id
```

$id="123abc" とすると、abcの部分が数値リテラルから「はみ出す」

```
SELECT * FROM employee WHERE id = 123abc
```

はみ出した部分はSQL文として意味をなさないのでエラーになるが、エラーにならないように入力を調整することも可能。これがSQLインジェクション
$id= "1;DELETE FROM employee" とした場合、SQL文は以下となる

```
SELECT * FROM employee WHERE id = 1;DELETE FROM employee
```

リテラルをはみだすことを防ぐには、数値リテラルであることを確実にする。バリデーション あるいは整数へのキャスト

```
SELECT * FROM employee WHERE name = 1
```

2. SQLの呼び出し方

SQLの呼び出し方

SQLに動的なパラメータを埋め込む方法には2種類ある
(1)文字列連結によるSQL文組み立て

```
$name = ...;
$sql = "SELECT * FROM employee WHERE name='" . $name . "'";
```

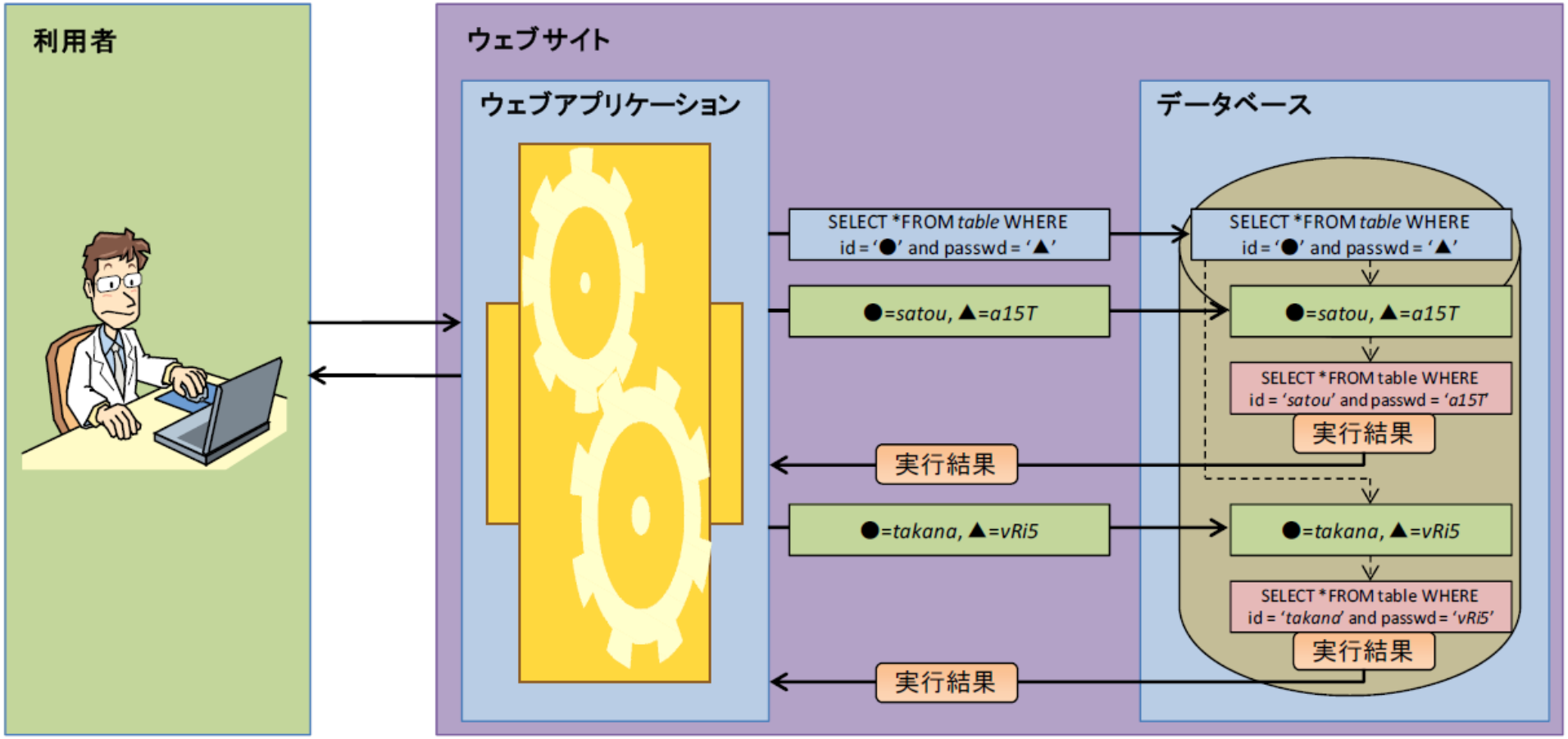
※ $nameをエスケープしていないのでSQLインジェクション脆弱性あり

(2)プレースホルダによるSQL文組み立て

```
PreparedStatement prep = onn.prepareStatement(
    "SELECT * FROM employee WHERE name=?");
prep.setString(1, "山田");
```

※ プレースホルダには2種類ある(静的・動的)

静的プレースホルダ

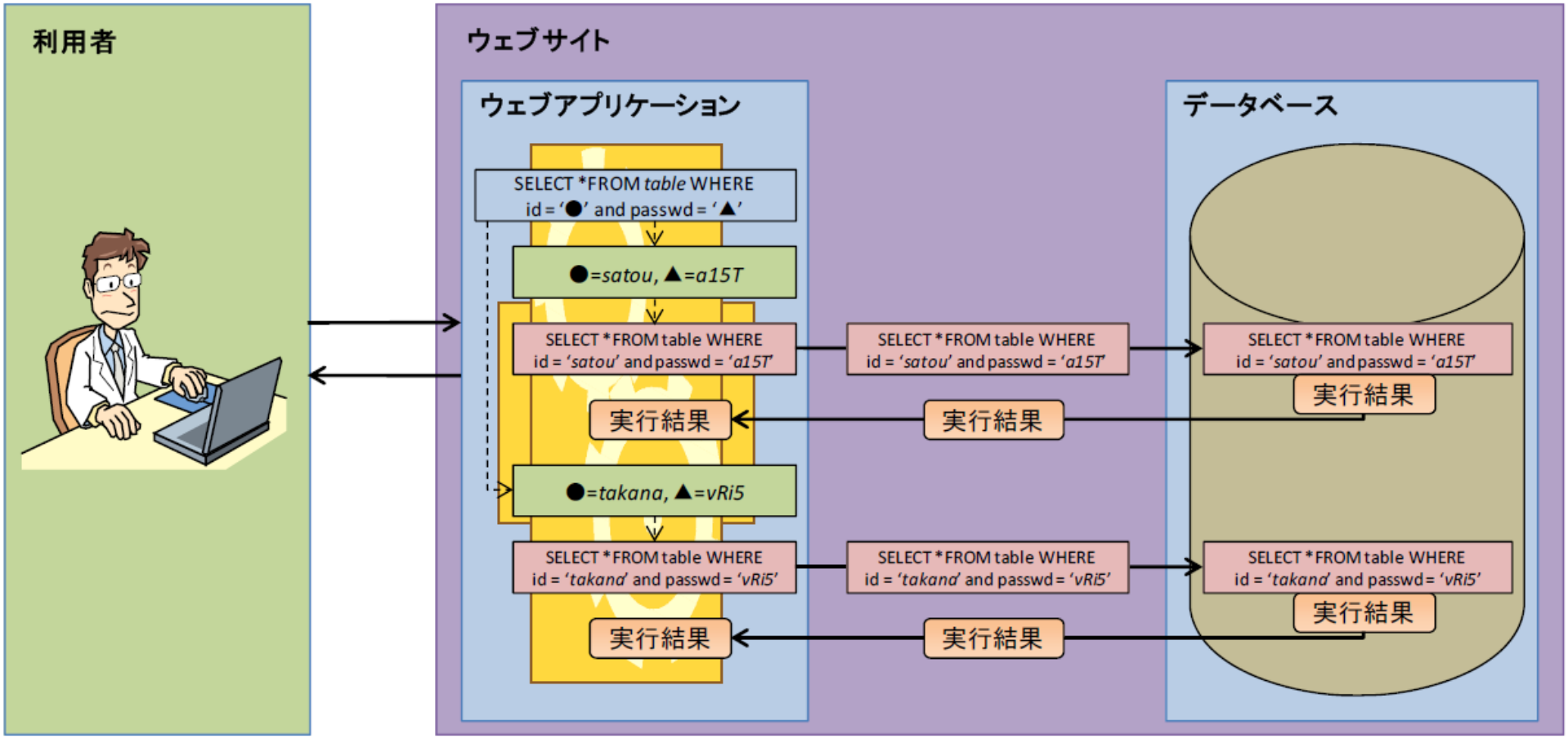


独立行政法人情報処理推進機構「安全なSQLの呼び出し方」より引用 http://www.ipa.go.jp/security/vuln/websecurity.html

- SQLとパラメータは別々にサーバーに送られる
- パラメータ抜きでSQLは構文解析される
- パラメータは後から割り当てられる

⇒ SQLインジェクション脆弱性の可能性が原理的になくなる

動的プレースホルダ

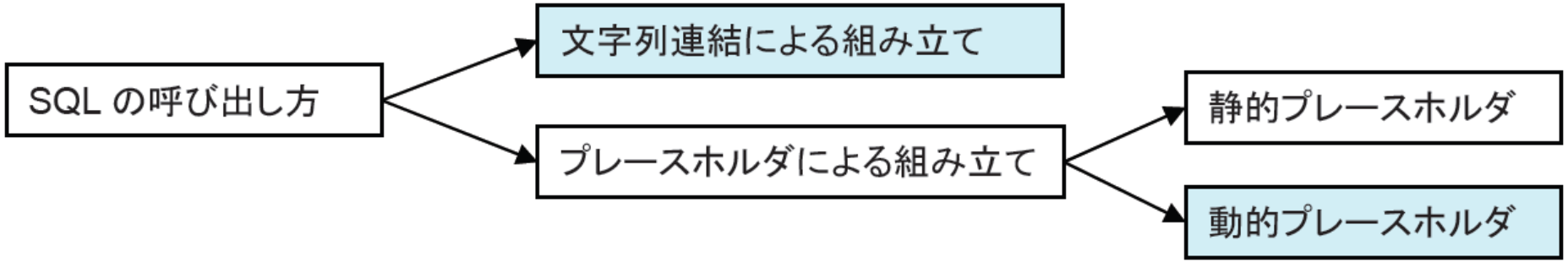


独立行政法人情報処理推進機構「安全なSQLの呼び出し方」より引用 http://www.ipa.go.jp/security/vuln/websecurity.html

- SQLとパラメータは呼び出し側で、エスケープ、連結される
- データベースサーバー側は、組み立てられた文字列をSQLとして実行するだけ

⇒ ライブラリにバグがなければ、SQLインジェクション脆弱性は発生しない

SQLの組み立て方とSQLインジェクションの可能性の関係



独立行政法人情報処理推進機構「安全なSQLの呼び出し方」より引用 http://www.ipa.go.jp/security/vuln/websecurity.html

- 文字列連結による組み立ては、アプリケーション開発者の無知や不注意によりSQLインジェクション脆弱性の可能性がある
- 動的プレースホルダは、ライブラリのバグによりSQLインジェクション脆弱性の可能性がある
- 静的プレースホルダは原理的にSQLインジェクション脆弱性の可能性がない

3. データベースと連動したSQL文生成

文字列連結によるSQL組み立てを安全に行うには

- 文字列連結によるSQL組み立て時のパラメータの要件
  - 文字列リテラルに対しては、エスケープすべき文字をエスケープすること
  - 数値リテラルに対しては、数値以外の文字を混入させないこと
- 意外に面倒
  - データベースによってエスケープすべきメタ文字が異なる
  - オプションによってもエスケープすべきメタ文字が異なる
- Perl、PHP等ではquoteメソッドが便利
  - Perl DBI
  - PHP　Pear::MDB2、PDO
- quoteメソッドはデータベースの種類や設定に応じたエスケープをしてくれる…はず
- 例外（バグ?、仕様?）もある

文字列リテラルのエスケープ

- どの文字をエスケープするのか?
  - SQL製品の文字列リテラルのルールに従う
  - ISO標準では、「'」→「''」
  - MySQLとPostgreSQLは「'」→「''」「¥」→「¥¥」
    - NO_BACKSLASH_ESCAPES=onの場合は、ISO標準と同じ方法になる
  - PostgreSQLの場合は、standard_conforming_stringsおよびbackslash_quoteの影響を受ける
    - standard_conforming_strings=onの場合は、ISO標準と同じ方法になる
    - PostgreSQL9.1以降で、standard_conforming_strings=onがデフォルトに
    - backslash_quoteの場合は、「'」→「¥'」というエスケープがエラーになる

	元の文字	エスケープ後
Oracle MS SQL IBM DB2	’	’’
MySQL PostgreSQL	’	’’ または ¥’ (’’を推奨)
	¥	¥¥

MySQLとPostgreSQLで「¥」のエスケープが必要な理由

SELECT * FROM XXX WHERE ID='$id'

$id として ¥'or 1=1# が入力されると

¥'or 1=1#
　↓エスケープ(「¥」のエスケープをしない場合)
¥''or 1=1#

元のSQLに適用すると、

SELECT * FROM XXX WHERE ID='¥'' or 1=1#'

すなわち、SQLの構文が改変された(「¥’」で「’」一文字と見なされる)

quoteメソッドの詳細

PHPのPear::MDB2におけるquoteの呼び出し

```
require_once ‘MDB2.php’;  //ライブラリのロード
// DBへの接続（PostgreSQLの場合）
$db = MDB2::connect('pgsql://dbuser:password@hostname/dbname?charset=utf8');

// 文字列型を指定して、文字列リテラルのクォート済み文字列を得る
（略）$db->quote($s, 'text')（略）

// 数値型を指定して、数値リテラルの文字列を得る
（略）$db->quote($n, 'decimal')（略）
```

データ	型指定	戻り値
abc	’text’	’abc’（PHPの文字列型の値、クォートを含む）
O’Reilly	’text’	O’’Reilly’（PHPの文字列型の値、クォートを含む）
−123	’decimal’	−123（PHPの文字列型の値）
123abc	’decimal’	123（PHPの文字列型の値）
−123	’integer’	−123（PHPの整数型の値）
123abc	’integer’	123（PHPの整数型の値）

独立行政法人情報処理推進機構「安全なSQLの呼び出し方」より引用 http://www.ipa.go.jp/security/vuln/websecurity.html

quoteメソッドの数値データの処理結果

Perl DBI/DBD、PHPのPDO, Pear::MDB2でquoteの処理内容を調査

1a¥' をINTEGER型を指定してquoteすると、どうなるか？

サンプルスクリプト:

```
DBI:   $dbh->quote("1a¥¥'", SQL_INTEGER)
PDO:   $dbh->quote("1a¥¥'", PDO::PARAM_INT)
MDB2:  $dbh->quote("1a¥¥'", 'integer')
```

モジュール名	結果	
DBI(DBD::mysql)	1a¥'	⇐ なにもしていない！（脆弱性）
DBI(DBD::PgPP)	'1a¥¥¥''	
PDO	'1a¥¥¥''	
MDB2	1(int型)	⇐ 正しい結果

- quoteメソッドに期待したが、現状SQLの仕様通り動作するのはMDB2のみ
- プレースホルダの利用を推奨

4. DBMS製品の実態調査

Java + Oracle の場合

項目	調査結果
プレースホルダの実装	静的プレースホルダのみ
動的プレースホルダのエスケープ処理	調査対象外（動的プレースホルダは提供されていない）
quoteメソッドの処理	調査対象外（quoteメソッドは提供されていない）
文字エンコーディングの扱い	DB接続にはUTF-8が使用される

Java + Oracle + ojdbc6.jarでは、常に静的プレースホルダが使用されるため、Javaの PreparedStatementインターフェースを使っている限り、注意点はありません。
Javaではquoteメソッドに該当するライブラリが提供されておらず、データベースエンジンの種類や設定に連動したエスケープ処理ができないため、文字列連結によるSQL文の組み立ては推奨されません。

PHP + PostgreSQL の場合

下記の点から、PEAR::MDB2について調査しました。
•MDB2はPostgreSQLの他、MySQL、Oracle等の複数のDBMSに対して、SQL呼び出しを抽象化したインターフェースを提供している
•Pear::DB、Pear::MDBなどの同種のモジュールは既に開発が終了しているのに対し、MDB2は開発が継続されている
•文字エンコーディングを考慮している
•静的プレースホルダを利用できる
•プレースホルダへの値のバインドとクォートの際に、データの型を考慮している

項目	調査結果
プレースホルダの実装	静的プレースホルダのみ
動的プレースホルダのエスケープ処理	調査対象外（動的プレースホルダは提供されていない）
quoteメソッドの処理（文字列リテラルの生成）	正しく処理される
quoteメソッドの処理（数値リテラルの生成)	正しく処理される
文字エンコーディングの扱い	DB接続時に文字エンコーディングを指定できる

PHP + MDB2 + PostgreSQLの組み合わせでは、常に静的プレースホルダが使用されるため、プレースホルダを使っている限り、注意点はありません。
プレースホルダの代わりにquoteメソッドを使うことも可能で、quoteメソッドは、文字列リテラル、数値リテラルともに、正しく生成します。

Perl + MySQL の場合(DBI + DBD::MySQL)

項目	調査結果
プレースホルダの実装	動的プレースホルダまたは静的プレースホルダ
動的プレースホルダのエスケープ処理	正しく処理される
quoteメソッドの処理（文字列リテラルの生成）	正しく処理される
quoteメソッドの処理（数値リテラルの生成)	正しく処理されない（入力をそのまま返す）
文字エンコーディングの扱い	DB接続時にUTF-8を明示的に指定できる

- MySQLで静的プレースホルダを使用する場合、mysql_server_prepare=1を指定すること
- エスケープ対象の文字はNO_BACKSLASH_ESCAPESを正しく反映する
- 数値に対してquoteメソッドは何もしない（脆弱性!）
- mysql_enable_utf8=1 によりUTF-8を明示できる

```
$dbh->quote("1 or 1=1", SQL_INTEGER); # → "1 or 1=1" を返す
```

Java + MySQLの場合（MySQL Connector/J）

項目	調査結果
プレースホルダの実装	動的プレースホルダまたは静的プレースホルダ
動的プレースホルダのエスケープ処理	正しく処理される（ただし、バージョンによっては付録A.3の問題がある）
quoteメソッドの処理	調査対象外（quoteメソッドは提供されていない）
文字エンコーディングの扱い	DB接続時に文字エンコーディングを指定できる

- 静的プレースホルダを使用するためにはuseServerPrepStmts=true を指定する
- 動的プレースホルダを使用する場合、NO_BACKSLASH_ESCAPESを反映したエスケープを行う
- DB接続時に、characterEncoding パラメータにより文字エンコーディングの指定が可能

付録A. 技術情報

【文字コードの問題1】5C問題によるSQLインジェクション

- 5C問題とは
  - Shift_JIS文字の2バイト目に0x5Cが来る文字に起因する問題
    ソ、表、能、欺、申、暴、十 … など出現頻度の高い文字が多い
  - 0x5CがASCIIではバックスラッシュであり、ISO-8859-1など1バイト文字と解釈された場合、日本語の1バイトがバックスラッシュとして取り扱われる
  - 一貫して1バイト文字として取り扱われれば脆弱性にならないが、1バイト文字として取り扱われる場合と、Shift_JISとして取り扱われる場合が混在すると脆弱性が発生する

ソースコード（要点のみ）

```
<?php
  header('Content-Type: text/html; charset=Shift_JIS');
  $key = @$_GET['name'];
  if (! mb_check_encoding($key, 'Shift_JIS')) {
   die('文字エンコーディングが不正です');
  }
  // MySQLに接続(PDO)
  $dbh = new PDO('mysql:host=localhost;dbname=books', 'phpcon', 'pass1');
  // Shift_JISを指定
  $dbh->query("SET NAMES sjis");
  // プレースホルダによるSQLインジェクション対策
  $sth = $dbh->prepare("SELECT * FROM books WHERE author=?");
  $sth->setFetchMode(PDO::FETCH_NUM);
  // バインドとクエリ実行
  $sth->execute(array($key));
?>
```

5C問題によるSQLインジェクションの説明

対策

- 文字エンコーディング指定のできるデータベース接続ライブラリを選定し、文字エンコーディングを正しく指定する

```
$dbh = new PDO('mysql:host=xxxx;dbname=xxxx;charset=cp932',
                    'user', 'pass', array(
  PDO::MYSQL_ATTR_READ_DEFAULT_FILE => '/etc/mysql/my.cnf',
     PDO::MYSQL_ATTR_READ_DEFAULT_GROUP => 'client', ));
    # http://gist.github.com/459499 より引用(by id:nihen)
```

- 静的プレースホルダを使うよう指定する、あるいはプログラミングする

```
$dbh->setAttribute(PDO::ATTR_EMULATE_PREPARES,
                                  false);
```

- 詳しくは「安全なSQLの呼び出し方」を参照
  http://www.ipa.go.jp/security/vuln/websecurity.html

【文字コードの問題2】U+00A5によるSQLインジェクション

```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection(
  "jdbc:mysql://localhost/books?user=phpcon&passw
  ord=pass1");
String sql = "SELECT * FROM books where
  author=?";
// プレースホルダ利用によるSQLインジェクション対策
PreparedStatement stmt =
  con.prepareStatement(sql);
// ? の場所に値を埋め込む(バインド)
stmt.setString(1, key);
ResultSet rs = stmt.executeQuery();  // クエリの実行
```

U+00A5によるSQLインジェクションの原理

【入力文字列（Unicode）】

コードポイント	00A5	0027	006F	0072	0020	0031	003D	0031	0023
文字	¥	’	o	r	SP	1	=	1	#

【エスケープ処理後の文字列（Unicode）】

コードポイント	00A5	005C	0027	006F	0072	0020	0031	003D	0031	0023
文字	¥	\	’	o	r	SP	1	=	1	#

【Shift_JIS に変換した文字列】

文字コード	5C	5C	27	6F	72	20	31	3D	31	23
文字	¥	¥	’	o	r	SP	1	=	1	#

【動的プレースホルダにバインドした SQL 文】

```
SELECT * FROM test WHERE name='¥¥' or 1=1#'
```

SQL

IPA:安全なSQLの呼び出し方（http://www.ipa.go.jp/security/vuln/websecurity.html）より引用

U+00A5によるSQLインジェクションの条件と対策

- 脆弱性が発生する条件
  - JDBCとしてMySQL Connector/J 5.1.7以前を使用
  - MySQLとの接続にShift_JISあるいはEUC-JPを使用
  - 静的プレースホルダを使わず、エスケープあるいは動的プレースホルダ（クライアントサイドのバインド機構）を利用している
- 対策（どれか一つで対策になるがすべて実施を推奨）
  - MySQL Connector/Jの最新版を利用する
  - MySQLとの接続に使用する文字エンコーディングとしてUnicode(UTF-8)を指定する
    （接続文字列にcharacterEncoding=utf8を指定する）
  - 静的プレースホルダを使用する
    （接続文字列にuseServerPrepStmts=trueを指定する）

SQLインジェクション対策

- 入力値
  - 文字エンコーディングの検証 and/or 文字エンコーディングの変換
  - 要件に従った入力値の検証（制御文字のチェックは必須）
- SQL呼び出し
  - ともかくプレースホルダを使うこと
  - 静的プレースホルダの利用が「原理的に」安全
  - 「安全なSQLの呼び出し方」をよく読む
- 文字コードの選定
  - アプリケーションを通してUnicodeを使う
    - HTTPメッセージはUTF-8
    - アプリケーションの内部はUTF-8かUTF-16
  - ケータイ向けサイトはHTTPメッセージの文字エンコーディングをShift_JISにするが、内部はUTF-8とする
    - EUC-JPという選択もあり得るが、使える言語が少ない
    - 円記号U-00A5 → バックスラッシュ(5C) の変換に注意
  - 尾骶骨テストのすすめ