node-mysql上手指南

node-mysql

A pure node.js JavaScript Client implementing the MySql protocol.

1 node-mysql安装

Install

npm install mysql

npm install mysqljs/mysql

2 mysql连接

mysql连接

1.连接

创建连接对象，需要传入连接数据库的一些连接参数，也就是`createConnection(option)`里的`option`，`option`是一个对象，以键值对的形式传入`createConnection()`方法里。基本的参数如下：

- `host` 主机名
- `user` 连接数据库的用户
- `password` 密码
- `database` 数据库名称

```
const mysql      = require('mysql');
const connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
  password : '123456',
  database : 'my_db'
});

connection.connect(function(err) {
  if (err) {
    console.error('连接错误: ' + err.stack);
    return;
  }

  console.log('连接ID ' + connection.threadId);
});
```

mysql连接

02

2.关闭

关闭一个连接使用`end()`方法，`end()`方法提供一个回调函数。

```
const mysql      = require('mysql');
const connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
  password : '123456',
  database : 'my_db'
});

connection.connect();

connection.query('SELECT 1 + 1 AS solution', function (error, results, fields) {
    if (error) throw error;
    console.log('The solution is: ', results[0].solution);
  });

connection.end();
```

3 连接池

连接池 Pooling connections

连接池是负责分配、管理和释放数据库连接，允许应用程序重复使用一个现有的数据库连接，而不是重新建立一个连接，释放空闲时间超过最大允许空闲时间的数据库连接以避免因为连接未释放而引起的数据库连接遗漏。创建连接池以下几个独有的选项：

- `acquireTimeout` 获取连接的毫秒（默认：10000）
- `waitForConnections` 没有连接或达到最大连接时连接的形为
- `connectionLimit` 单次可创建最大连接数（默认：10）
- `queueLimit` 连接池的最大请求数

```
const mysql = require('mysql');
const pool  = mysql.createPool({
  connectionLimit : 10,
  host            : 'example.org',
  user            : 'bob',
  password        : 'secret',
  database        : 'my_db'
});

pool.query('SELECT 1 + 1 AS solution', function (error, results, fields) {
  if (error) throw error;
  console.log('The solution is: ', results[0].solution);
});
```

连接池 Pooling connections

1.getConnection()方法连接可以共享一个连接，或管理多个连接

2.关闭连接池连接

```
const mysql = require('mysql');
const pool  = mysql.createPool({
  connectionLimit : 10,
  host            : 'example.org',
  user            : 'bob',
  password        : 'secret',
  database        : 'my_db'
});

pool.getConnection(function(err, connection) {
    if (err) {
        resolve( errMes(err) )
      } else {
        connection.query(sql, values, ( err, rows) => {

          if ( err ) {
            reject( errMes(err) )
          } else {
            resolve( rows )
          }
          connection.release()
        })
    }
})
```

4 连接池集群（PoolCluster）

连接池集群（PoolCluster）

数据库集群（Cluster）是利两台或多台数据库服务器，构成一个虚拟单一数据库逻辑映像，并像单数据库系统那样，向客户端提供透明的数据服务。MySQL同样支持建立数据库集群，利用`node-mysql`模块可以建立一个面向MySQL集群（MySQL Cluster）的连接。

连接池集群（PoolCluster）

02

1.连接池集群连接

- `PoolCluster` 使我们可以建立一个面向多台主机的连接，它由 `createPoolCluster()` 方法创建返回：

```
const poolCluster = mysql.createPoolCluster();

poolCluster.add(config); // anonymous group
poolCluster.add('MASTER', masterConfig);
poolCluster.add('SLAVE1', slave1Config);
poolCluster.add('SLAVE2', slave2Config);

poolCluster.remove('SLAVE2'); // By nodeId
poolCluster.remove('SLAVE*'); // By target group : SLAVE1-2

// 目标群组 : 所有(anonymous, MASTER, SLAVE1-2), Selector : round-robin(default)
poolCluster.getConnection(function (err, connection) {});

// 目标群组 : MASTER, Selector : round-robin
poolCluster.getConnection('MASTER', function (err, connection) {});

// Target Group : SLAVE1-2, Selector : order
// If can't connect to SLAVE1, return SLAVE2. (remove SLAVE1 in the cluster)
poolCluster.on('remove', function (nodeId) {
  console.log('REMOVED NODE : ' + nodeId); // nodeId = SLAVE1
});

poolCluster.getConnection('SLAVE*', 'ORDER', function (err, connection) {});

// of namespace : of(pattern, selector)
poolCluster.of('*').getConnection(function (err, connection) {});

var pool = poolCluster.of('SLAVE*', 'RANDOM');
pool.getConnection(function (err, connection) {});
pool.getConnection(function (err, connection) {});

// close all connections
poolCluster.end(function (err) {
  // all connections in the pool cluster have ended
})
```

连接池集群（PoolCluster）

2.连接池集群选项

- `canRetry` ：当为true时，PoolCluster会在连接失败时尝试重连（默认：true）
- `removeNodeErrorCount` ：连接失败时Node的errorCount计数会增加。当累积到这个值时移除PoolCluster这个节点（默认：5）
- `restoreNodeTimeout` ：连接失败后重试连接的毫秒数（默认：0）
- `defaultSelector` ：默认的选择器（selector）（默认：RR，RR依次选择、RANDOM随机选择、ORDER选择第一个可用节点）

5 执行SQL语句

执行SQL语句

mysqljs 对数据库的操作都是通过SQL语句实现的，通过SQL语句可以实现创建数据库、创建表、及对表中数据库的增/删/改/查等操作。

在node-mysql中，通过Connection或Pool实例的query()执行SQL语句，所执行的SQL语句可以是一个SELECT查询或是其它数据库操作。query()方法有三种形式。

执行SQL语句

第一种形式

`query(sqlString, callback)`

- `sqlString`－要执行的SQL语句
- `callback`－回调函数，其形式为`function (error, results, fields) {}`

```
connection.query('SELECT * FROM `books` WHERE `author` = "David"', function (error, results, fields) {
    // error 错误对象，在查询发生错误时存在
    // results 查询结果
    // fields 查询结果的字段信息
});
```

执行SQL语句

第二种形式

`query(sqlString, values, callback)`

- `sqlString`－要执行的SQL语句
- `values`－{Array}，要应用到查询占位符的值
- `callback`－回调函数，其形式为`function (error, results, fields) {}`

```
connection.query('SELECT * FROM `books` WHERE `author` = ?', ['David'], function (error, results, fields) {
    // error 错误对象，在查询发生错误时存在
    // results 查询结果
    // fields 查询结果的字段信息
});
```

执行SQL语句

第三种形式

`.query(options, callback)`

- `options`－{Object}，查询选项参数
- `callback`－回调函数，其形式为`function (error, results, fields) {}`

```
connection.query({
    sql: 'SELECT * FROM `books` WHERE `author` = ?',
    timeout: 40000, // 40s
    values: ['David']
  }, function (error, results, fields) {
    // error 错误对象，在查询发生错误时存在
    // results 查询结果
    // fields 查询结果的字段信息
  });
```

6 其他

查询编码与安全

为了防止SQL注入，可以传入参数进行编码。参数编码方法有：`mysql.escape()`/`connection.escape()`/`pool.escape()`，这三个方法可以在你需要的时候调用：

`escape()`方法编码规则如下：

- Numbers不进行转换
- Booleans转换为true/false
- Date对象转换为'YYYY-mm-dd HH:ii:ss'字符串
- Buffers转换为hex字符串，如X'0fa5'
- Strings进行安全转义
- Arrays转换为列表，如['a', ‘b']会转换为'a', 'b'
- 多维数组转换为组列表，如[['a', 'b'], ['c', 'd']]会转换为'a', 'b'), ('c', 'd')
- Objects会转换为key=value键值对的形式
- undefined/null会转换为NULL

查询编码与安全


```
var userId = 'some user provided value';
var sql    = 'SELECT * FROM users WHERE id = ' + connection.escape(userId);
connection.query(sql, function(err, results) {
  // ...
});
```

可以使用`?`做为查询参数占位符，这与查询值编码效果是一样的：

```
connection.query('SELECT * FROM users WHERE id = ?', [userId], function(err, results) {
  // ...
});
```

参考资料

- mysqljs： https://github.com/mysqljs/mysql
- 在Nodejs中使用MySQL数据库的最佳实践是什么？
- Sequelize： http://docs.sequelizejs.com/
- node-mysql中防止SQL注入
- node-mysql 模块介绍

02

OPEN ORIENTED

凹 凸 实 验 室

Object-Relational Mapping

关系数据库表结构映射到对象