

# Overview

ODBC (Open Database Connectivity) is a technology that allows one to connect to a whole variety of databases using a single API, the ODBC API.

## TODBCConnection

In Lazarus, you can find the `TODBCConnection` component on the SQLdb component tab. You can also use the `TODBCConnection` component in your code by adding `ODBCConn` to your uses clause.

## References:

- General info
- Libraries
- Field types
- Controls
- FAQ
- SQL how-to
- Working With TSQLQuery
- In-memory database applications

- Overview
- 0 - Database set-up
- 1 - Getting started
- 2 - Editing
- 3 - Queries
- 4 - Data modules
- [SQLdb Programming Reference](#)

Advantage - MySQL - MSSQL - Postgres -  
Interbase - Firebird - Oracle - ODBC - Pa  
radox - SQLite - dBASE - MS Access - Ze  
os



- executing queries and retrieving result sets
- most field types, including blobs
- query parameters (string and integer types)
- preparing queries

- UpdateIndexDefs (so you can use ApplyUpdates)

What is left to be implemented:

- proper transaction support; currently each connection corresponds to one transaction
- some field types
  - SQL\_TYPE\_UTC\* (these are mentioned in the ODBC docs, but seem not to be used in implementations)
  - SQL\_INTERVAL\_\* (what would be the corresponding TFieldType?)
  - SQL\_GUID (TGUIDField was not implemented, until recently)

## Why use ODBC?

FreePascal ships with components for connecting to several databases, such as MySQL, PostgreSQL, Firebird, Microsoft SQL Server (since 2.6.1), Oracle, etc.

For those databases missing from the list, like [MS Access](#) and perhaps Microsoft SQL Server, ODBC is an acceptable and well established solution. (VB.net/C# developers are recommended to use SQLClient rather than OLEDB or ODBC for the ultimate in performance, but here in Lazarus ODBC will work adequately and reliably)

The TODBCConnection component was developed originally to circumvent the strict MySQL license for applications that are not GPLed or do not obey MySQL AB's [FLOSS exception](#).

## Installing ODBC and ODBC Drivers

Before you can connect to your database using ODBC, you need to install

- an ODBC Driver Manager
- an ODBC driver specific to the DBMS you want to connect to

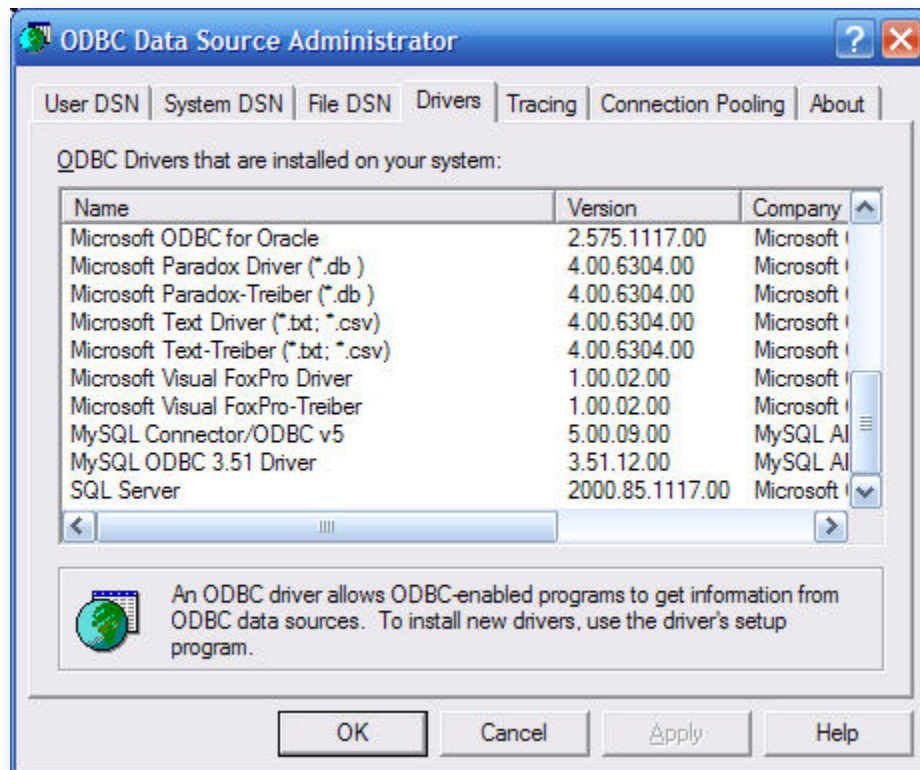
This section will give a brief overview of the steps involved. However, please consult the relevant documentation for a definitive reference.

### The ODBC Driver Manager

#### Windows

Windows has an ODBC Driver Manager built in, which allows DSNs to be set up, and other configuration. It is found in the *Control Panel*, in later Windows versions with categorised sections it was moved into the *Administrative Tools* area. Or you may simply click the Start button and enter *ODBC* into the Run box (in Windows 7 this appears as "*Search programs and files*").

You can, of course, create a desktop shortcut to the ODBC dialog if you find yourself using it frequently.



## Unices

Two popular ODBC Driver Managers for Unix-based platforms are [unixODBC](#) and [iODBC](#). [ODBCConn](#) is known to work with unixODBC; iODBC compatibility still has to be tested.

## Debian

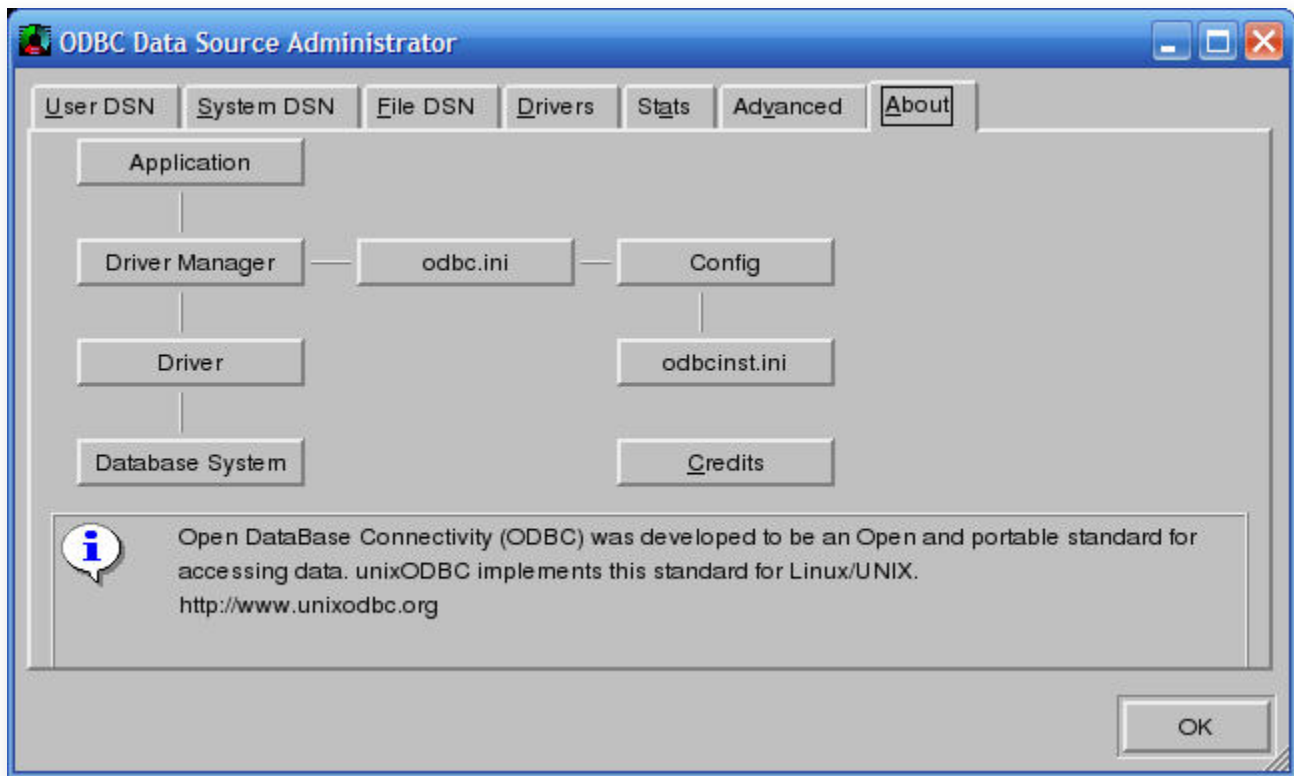
For Debian, you can install the *unixodbc* package:

```
aptitude install unixodbc
aptitude install unixodbc-bin # if you want some GUI tools
```

The [odbcsqlodyn](#) unit, and hence [odbcconn](#), will search for a file called [libodbc.so](#). It will *not* accept a file named like [libodbc.so.1](#) or [libodbc.so.1.0.0](#). Debian's *unixodbc* package does not create a symlink with the name [/usr/lib/libodbc.so](#); you must either

- create the link yourself: `ln -s libodbc.so.1 /usr/lib/libodbc.so`, or
- install the *unixodbc-dev* package, which does create the symlink.

If you installed the *unixodbc-bin* package, you can run the [ODBCConfig](#) program to configure ODBC drivers and DSNs.



## Ubuntu

For Ubuntu, follow the instruction for [Debian](#). Note: the *unixodbc-bin* package might not be available from the default package repository.

## ODBC Drivers

You can download latest ODBC drivers using the following links:

[Download SQL Server 32/64 bit ODBC driver](#) - latest version of SQL Server ODBC driver which supports Windows, Mac OS X, Linux both on 32 and 64 versions. Compatible with SQL Server 2014\2012\2008 R2\ 2008\2005 (including Express edition), SQL Server 2000 (including MSDE), SQL Server 7, SQL Server Compact 4.0\3.5\3.1. Works on all Lazarus versions.

[Download Oracle 32/64 bit ODBC driver](#) - latest version of Oracle ODBC driver which supports Windows, Mac OS X, Linux both on 32 and 64 versions. Compatible with Oracle servers: 12c, 11g, 10g, 9i, 8i, 8.0, including Oracle Express Edition 11g and 10g. Oracle Clients: 12c, 11g, 10g, 9i, 8i, 8.0. Works on all Lazarus versions.

[Download MySQL 32/64 bit ODBC driver](#) - latest version of MySQL ODBC driver which supports Windows, Mac OS X, Linux both on 32 and 64 versions. Compatible with MySQL servers: 6.0, 5.6, 5.5, 5.1, 5.0, 4.1, 4.0, and 3.23. MariaDB 5.x Works on all Lazarus versions.

[Download PostgreSQL 32/64 bit ODBC driver](#) - latest version of PostgreSQL ODBC driver which supports Windows, Mac OS X, Linux both on 32 and 64 versions. Compatible with PostgreSQL server v

ersions since 7.1 up to 9.4. Works on all Lazarus versions.

[Download SQLite 32/64 bit ODBC driver](#) - latest version of SQLite ODBC driver which supports Windows, Mac OS X, Linux both on 32 and 64 versions. Compatible with SQLite versions since 3.0 and higher. Works on all Lazarus versions.

[Download Firebird 32/64 bit ODBC driver](#) - latest version of Firebird ODBC driver which supports Windows, Mac OS X, Linux both on 32 and 64 versions. Compatible with Firebird versions 3, 2.x, 1.x. Works on all Lazarus versions.

[Download Interbase 32/64 bit ODBC driver](#) - latest version of Interbase ODBC driver which supports Windows, Mac OS X, Linux both on 32 and 64 versions. Compatible with all Interbase versions. Works on all Lazarus versions.

[Download SQL Azure 32/64 bit ODBC driver](#) - latest version of SQL Azure ODBC driver which supports Windows, Mac OS X, Linux both on 32 and 64 versions. Compatible with all SQL Azure versions. Works on all Lazarus versions.

## Connecting to an ODBC data source

The parameters for connecting to an ODBC data source are described in a *connection string*. This is a string of the form `NAME=VALUE;NAME=VALUE`.

`TODBCConnection` provides a wrapper around this connection string. Some of its properties are mapped to name-value pairs in the connection string, and custom parameters can be specified in the `Params` property (which is a `TStrings`).

Before going to the details of this wrapper, you must first have a basic understanding of how an ODBC data source is identified.

### ODBC connections via DSN

An ODBC driver manager provides alternative ways to make DSN shortcuts for a set of parameters:

- DSN (**D**ata**S**ource **N**ame): a system or user defined data source, identified by a (unique) name. DSNs can be configured using the ODBC Data Source Administrator or by manually editing the `odbc.ini` file (or registry).
- File DSN: a file which contains connection parameters. An ODBC Data Source Administrator usually allows you to create File DSNs from the GUI.

The parameters in a DSN or File DSN can always be combined with additional parameters in the connection string, for example to specify a password.

By setting up a DSN the connection details can be verified to work within the manager dialog, and then the named DSN is all that is needed to use the connection later. The connection details are therefore *decoupled* from your application, as only the DSN name is used in your software - the DSN acting as a go-between.

An advantage to using a named DSN is that you may easily switch between databases using the ODBC Manager, without changing your code. This is useful in commercial development to test for bugs using a number of sets of your clients' data.

## Without a DSN

You may also connect via ODBC without using a DSN, simply supplying all the details in the Connection String that you would otherwise set up in a DSN (driver,server,database,login details). This more direct method avoids the need to set up a DSN when installing your application on a new machine.

The ODBC specification defines a few parameters that can be used in a connection string:

- Two special parameters, `DSN` and `FILEDSN`, allow one to select a set of pre-defined parameters, as described above.
- The `DRIVER` specifies which ODBC driver to use. Obviously, this is a very important parameter.
- The `UID` and `PWD` parameters are used to supply a username and password.

All other parameters are driver dependent. Please refer to the documentation of the specific driver to learn more about available parameters and their names.

## TODBCConnection properties

The following table describes the mapping of `TODBCConnection` properties to ODBC connection string parameters:

Property	Type	Connection string parameter
Driver	string	<code>DRIVER</code>
Database Name	string	<code>DSN</code> , <i>not</i> to something like <code>DATABASENAME</code> , which is not part of the ODBC standard
FileDSN	string	<code>FILEDSN</code>



Passwor d	stri ng	PWD
UserNam e	stri ng	UID
HostNam e	stri ng	<i>none</i> ; there is no corresponding parameter in the ODBC standard
Params	TStr ings	<p>Use this to specify custom parameters. Each item must be of the form <code>NAM E=VALUE</code> .</p> <p>One important parameter that can be used is <code>AUTOCOMMIT</code> which determines if SQL statements are directly executed (without possibility of a rollback) (default setting, or set explicitly using <code>AUTOCOMMIT=1</code> ) or whether you need to manually call <code>StartTransaction</code> and <code>Commit</code>, <code>CommitRetaining</code> or <code>Rollback</code> (set using <code>AUTOCOMMIT=0</code> )</p>

Note that `TODBCConnection` will take care of escaping parameter values in the connection string (when required).

The `LoginPrompt` boolean property is not implemented yet. It would require finding the correct window handle, so a driver can show a GUI dialog to specify parameters. Note that this is not controlled by the connection string, but rather by the last parameter to the ODBC API function `SQLDriverConnect` .

## Examples

In this section, examples are given of connecting to certain DBMSs using their specific ODBC drivers.

### Connecting to Excel

Probably works on Windows only.

To avoid errors like: Could not start transaction! ODBC error details: LastReturnCode: SQL\_ERROR; Record 1: SqlState: HYC00; NativeError: 106; Message: [Microsoft][ODBC Excel Driver]Optional feature not implemented ;

... you'll have to enable autocommit, something like

```
ODBCConnection1.Params.Add('AUTOCOMMIT=1');
```

## Connecting to MySQL

For a reference of supported parameters by the MyODBC driver, see [\[1\]](#).

Additionally you may check this [mysql odbc connection](#) guide. It has a detail explanation of how to connect to MySQL using ODBC.

The driver name differs a bit depending on the platform and MyODBC version; examples are :

```
{ properties of connection object, i.e. conn: TODBConnection; }
conn.Driver := 'MySQL'; // (Unix)
conn.Driver := 'MySQL ODBC 3.51 Driver'; // (Windows)
conn.Driver := 'MySQL Connector/ODBC v5'; // (Windows)
// note: driver name doesn't need {} like it does in SQL Connection Strings elsewhere
```

Other parameters (when not using a DSN) :

```
conn.UserName := 'myUsername';
conn.Password := 'myPassword';
// conn.Params (TStrings) set using .add method :
conn.Params.Add('server=example.com');
conn.Params.Add('port=3306');
conn.Params.Add('database=myDatabase');
conn.Params.Add('charset=utf8');
```

## Connecting to MS Access

Please see [MS Access](#).

## Connecting to Microsoft SQL Server



**Note:** Newer versions than FPC 2.6.0 and Lazarus versions using those support a direct SQLDB connection to MS SQL Server; see [MSSQL](#)

Use this [mssql odbc driver connection](#) examples which include step by step tutorial and connection strings.

See <http://www.connectionstrings.com/sql-server> for more details on connection strings.

Microsoft SQL Server ODBC connection string example:

```
Server=<server name>; // or <server name>\<instance name>
Database=<database name>;
```

## Connecting to Oracle



Use this [odbc oracle connection](#) guide to perform easy and painless connection to Oracle.

Oracle ODBC connection string example:

```
Login Prompt=False;  
Data Source=ORCL;  
User ID=scott;  
Password=tiger
```

## Connecting to PosgreSQL

Use this [postgresql odbc connection guide](#) to perform easy and painless connection to PostgreSQL database.

PostgreSQL ODBC connection string example:

```
Login Prompt=False;  
Data Source=localhost;  
User ID=postgres;  
Password=postgres;  
Database=postgres;  
Schema=public
```

## Connecting to SQLite

Use this [sqlite odbc connection guide](#) to perform easy and painless connection to SQLite database.

SQLite ODBC connection string example:

```
Login Prompt=False;  
Database=c:\test.db3
```

## Connecting to Firebird

Use this [firebird odbc connection guide](#) to perform easy and painless connection to Firebird database.

Firebird ODBC connection string example:

```
Data Source=127.0.0.1;  
User ID=sysdba;  
Password=masterkey;  
Client Library=fbclient.dll;  
Database=c:\fbd.fdb
```

## Named DSN

On Windows, you can use something like this with an existing named DSN:

```
{ properties of connection object, i.e. conn: TODBCCConnection; }
conn.DatabaseName := '<Your_DSN_Name>';
{ Leave all other conn. properties empty }
```

## DSN-less

Without a DSN:

```
{ properties of connection object, i.e. conn: TODBCCConnection; }
conn.Driver := 'SQL Server';
// note: driver name doesn't need {} like it does in SQL Connection Strings elsewhere
{ Leave Hostname and DatabaseName properties empty }
{ set conn.Params (TStrings) : }
conn.Params.Add('Database=<yourdatabasename>');
conn.Params.Add('Server=.\SQLEXPRESS');
conn.Params.Add('Trusted_Connection=Yes'); // or 'Integrated Security=SSPI'
```

Note: In this example, we connect to the local machine (server=.), on the instance SQLEXPRESS. You can use a hostname instead of ., and you can omit the instance to connect to the default instance. See MS documentation for details.

Using a trusted connection (also known as integrated security or SSPI) means you login using your Windows user credentials. You can omit the Trusted\_Connection line, but then you need to specify user ID (Uid=...) and password (Pwd=...).

## DSN example

The example code below selects the rows from a MS SQL Server table called 'journal\_entries' and displays all the values of column 'journal\_entry' in a Memo control called Memo1.

```
// uses ODBCConn, sqldb;
procedure TForm1.Button1Click(Sender: TObject);
var
  S: String;
  conn : TODBCCConnection; // uses ODBCConn
  query: TSQLQuery;        // uses sqldb
  transaction: TSQLTransaction; // uses sqldb

begin
  conn := TODBCCConnection.Create(nil);
  query := TSQLQuery.Create(nil);
  transaction := TSQLTransaction.Create(nil);
  try
    try
      conn.DatabaseName := 'sqlserverdsn'; {replace this with your DSN, if you use any}
      conn.UserName:= 'sa'; //replace with your user name
      conn.Password:= 'thepassword'; //replace with your password
      // You can override the properties defined in the DSN, e.g.
      //conn.Params.Add('Database=some_other_db');
```

```

conn.Transaction := transaction;
query.DataBase := conn;
{ To avoid "could not retrieve primary key metadata":
- either use query.UsePrimaryKeyAsKey:=false
or (preferred)
- query.PacketRecords to -1
If you are using SQL Server, you could alternatively enable MARS in your SQL Native Client
See http://bugs.freepascal.org/view.php?id=13241 }
query.PacketRecords:=-1; //retrieve all data at once
query.SQL.Text := 'select journal_entry from journal_entries';
query.Open;
S := '';
while not query.EOF do
begin
    S := S + query.FieldByName('journal_entry').AsString + #13#10;
    query.Next;
end;
finally
    query.Free;
    conn.Free;
    transaction.Free;
end;
except
    on E: Exception do
        ShowMessage(E.message);
end;
Memo1.Text:= S;
end;

```

In fact it is possible to use conn.Params for all parts of the connection string, if we modify the above - this time showing the SQL Server version in a pop-up message :

```

conn.Params.Add('Driver=SQL Server');
conn.Params.Add('Server=OURBIGSERVER');
conn.Params.Add('Database=TestDataBase');
conn.Params.Add('Integrated Security=SSPI'); // use Windows logon credentials
// no other conn. properties set, apart from conn.Transaction := transaction;

query.DataBase := conn;
query.SQL.Text := 'select @@version';
query.Open;
showmessage(query.Fields[0].AsString); // Fields are numbered starting from 0

```

Functions are supported - e.g. 'SELECT \* FROM MyTestFunc()' - but there may be issues with *Stored Procedures* that return recordsets. For example, if 'EXEC MyStoredProcedure' returns data rows (verified in SQL Server Management Studio or with the *SqLcmd* DOS command: `C:\>sqlcmd -S MyServer -E -d MyDataBase -Q "exec MySP"`) then `query.open` will give an error: "Cannot open a non-select statement" (this may be fixed after FPC 2.6)

To call a Stored Procedure or to issue other SQL commands that return no rowsets (CREATE,INSERT,UPDATE,DELETE), use `query.ExecSQL` instead of `query.Open`

Stored Procedure parameters work, e.g. `'EXEC MySP @param1=23'`

The use of *Parameters* is recommended to avoid *SQL injection vulnerabilities*. In this case the SQL statement isn't built with values directly, but a named placeholder is used instead - and then the parameter is specified separately. Using this method it isn't possible to inject malicious SQL commands into variable fields in an attempt to tamper with the database; the server then knows specifically what part of the statement is a value and what is command. For example, instead of building a WHERE clause directly with the value 95 (`'... WHERE field2>95'`), instead use a named parameter like 'testval' -

```
query.SQL.Text := 'select * from mytable where field2>:testval';
query.Params.ParamByName('testval').AsString := '95';
query.open;
```

## Debugging ODBC & errors

### Problems with transactions

In Lazarus 1.2.4 (FPC 2.6.4), ODBC transaction behaviour has changed. This may cause previously working code to stop working.

For a solution, see [User\\_Changes\\_2.6.4#TODBCConnection\\_.28odbcconn.29\\_No\\_longer\\_autocommit](#)

### Error Messages

Each ODBC API call also returns a success code; if there is an error, more information can be retrieved using calls to `SQLGetDiagRec`. (For the sake of completeness: diagnostic records are also available when an API call returns `SQL_SUCCESS_WITH_INFO`.)

`TODBCConnection` checks the return code of each ODBC call and constructs an `EODBCException`. The `message` of this exception consists of:

- a message identifying what the `TODBCConnection` was doing when the error occurred
- the return code of the ODBC API call (e.g. `SQL_ERROR`)
- a number of diagnostic records (obtained using `SQLGetDiagRec`); each record consists of three fields:
  - a 5-character error code identifying the error

- a 'native error code'
- a message describing the error

### Function sequence error

If you get a 'Function sequence error' in the finalization section of the `ODBCConn` unit, then you probably did not properly clean up all your queries and connections.

### Rollback does not work

If you issue

```
TSQLTransaction.Rollback
```

commands but your SQL is committed regardless, your ODBC connection probably is in AUTOCOMMIT mode; see [the remarks about the AUTOCOMMIT connection parameter](#) on how to correct this.

## Tracing

Most ODBC managers have a tracing option in which all ODBC API calls are logged to a trace log. This can be very useful for debugging an ODBC application. The ODBC Data Source Administration GUI of both Windows and unixODBC have a tab where you can configure the tracing option.

Of course the trace log is mainly useful for developers that are familiar with the ODBC API, but it can help to identify the problem. Also, you can attach a trace log if you report a problem to the bug tracker.

## See also

- [MS Access](#) Using ODBC to access Microsoft Access databases



收藏 0



分享到微信



分享到QQ

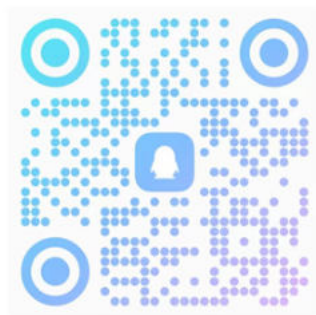


分享到微博

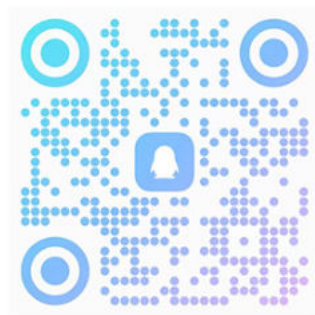
如果你对这篇内容有疑问，欢迎到本站[社区](#)发帖提问 参与讨论，获取更多帮助，或者扫码二维码加入 Web 技术交流群。



前端开发 交流群1  
413239701



前端开发 交流群2  
633411780



前端开发 交流群3  
300362254

**专注于前端开发相关技术，你可以在这里聊天灌水提问解惑。**

## 词条统计

浏览：71 次

字数：29354

最后编辑：7 年前

编辑次数：0 次



## 友情链接

[更多 >](#)

[文江博客](#)