微信公众号接口开发

使用 Node 连接数据库

安装 postgresql 数据库扩展

- npm i pg
- pg 支持 async/await 方式的调用。也支持回调模式。

使用示例

```
const dbcfg = require('./dbconfig');
const pg = require('pg');

var pdb = new pg.Client(dbcfg);
pdb.on('error', (err) => {
  console.log(err);
});

pdb.connect();

(async () => {
  let ret = await pdb.query('SELECT * FROM contents LIMIT 5 OFFSET 0');
  console.log(ret);
})();
```

数据库配置选项

```
{
    user : ‘[DB USER NAME]’,
    host : ‘127.0.0.1’,
    port : 5432,
    database : ‘[DATABASE NAME]’,
    password : ‘[PASSWORD]’
}
```

- 其中大写并用 [] 包含的项要替换成自己的值。

pg.Client

- pg.Client 返回一个连接实例，在规模不大的应用来说，可以满足正常的使用。

sql 参数

```
(async () => {
  let sql = "select * from contents where title ilike $1 or digest ilike $2";
  let ret = await pdb.query(sql, [
    '%node%', '%node%'
  ]);
  console.log(ret);
})();
```

返回值

- 在之前的示例中，仅仅是输出了返回值的信息。右侧是返回对象部分属性。
- 返回的对象，其中属性值 `rowCount` 表示更新或删除影响的数据条数，或者查询获取的数据条数。
- 查询成功后，在 `rows` 属性存储了数据对象。

```
Result {
  command: 'INSERT',
  rowCount: 1,
  oid: 0,
  rows: [],
  fields: [],
  _parsers: [],
  _types:
   TypeOverrides {
     _types:
      { getTypeParser: [Functi
        setTypeParser: [Functi
        arrayParser: [Object],
        builtins: [Object] },
     text: {},
     binary: {} },
```

连接池

- 在实际应用中，根据实际需要，选择连接池可以达到更好的性能，提高系统整体的业务处理能力。
- 连接池就是维护一个保存一些连接对象的实例，每次请求返回一个可用的连接对象，如果都不可用则会阻塞，可能导致超时。

创建连接池

```
const dbcfg = require('./dbconfig');
const pg = require('pg');

var dbpool = new pg.Pool(dbcfg);

(async () => {
  let sql = "select * from contents where title ilike $1";
  let ret = await dbpool.query(sql, [
    '%node%'
  ]);
  console.log(ret);
})();
```