1 babel

安装

1. 安装node
2. 淘宝镜像cnpm
3. cnpm node
4. cnpm init -y创建package.json文件，项目管理项目文件
5. npm install --save-dev @babel/core @babel/cli （babel官网）

   cnpm i 没写要装什么，会自动去package.json里找
6. package.json -- script -- 添加build启动脚本

   "build": "babel src -d lib"

   src 原文件目录

   lib 输出目录
7. 创建一个`babel.config.json`文件 （babel官网为准）

   ```
   {
     "presets": ["@babel/preset-env"]
   }
   ```
8. npm install @babel/preset-env --save-dev
9. npm run build

2 运行NodeJs程序：

1. d: d盘 盘符
2. cd目录名 进入目录
3. node xxx.js 运行
4. 补全
5. ↑ 重复上一个命令
6. ctrl+c 强制结束

3 系统模块

引用模块：require()

```
const assert=require('assert');
```

3.1 http模块

用法：

```
const http=require('http');

let server = http.createServer((req, res)=>{
    //有浏览器请求时执行的回调函数
    //request  请求  接收的数据（输入）
    //response 响应  发送的数据（输出）
    console.log(req.url);

    res.write("aaa");
    //必须end，否则服务器一直等
    res.end();
});

//监听
server.listen(8080);
```

结合fs：

```
const http=require('http');
const fs=require('fs');

let server=http.createServer((req, res)=>{
  fs.readFile(`www${req.url}`, (err, data)=>{
    if(err){
      res.write('404');     //?
    }else{
      res.write(data);
    }
    res.end();
  });
});
server.listen(8080);
```

3.1.1 数据交互

res.setHeader

res.writeHeader(404); 给机器看的，状态码

res.write('Not Found'); 给用户看的

3.1.1.1 Get数据

在"req.url"里面

```
const http=require('http');
const url=require('url');

let server=http.createServer((req, res)=>{
  let {pathname, query}=url.parse(req.url, true);

  console.log(pathname, query);

  res.end();
});
server.listen(8080);
```

3.1.2 POST数据

在body里，数据比较大，分好多次到达

```
//有一个段到达了,data是一个一个到达的，最终组装成一个完整的数据
req.on("data", data=>{});
//结束了
req.on("end", ()=>{});
```

```
const http = require("http");
const querystring = require("querystring");

let server = http.createServer((req, res)=>{
  let str = "";

  req.on("data", data=>{
    str+=data;
  });

  req.on("end", ()=>{
    let post = querystring.parse(str);

    console.log(str, post);
  });

  res.end();
});

server.listen(8080);
```

3.2 assert 断言

断言：判定某个条件必须为真（常用于函数里边的参数检查）

用法：

//条件必须为真

assert(条件, "一段话，条件不为真的时候提示");

```
const assert=require('assert');

function sum(a, b){
  assert(arguments.length==2, '必须传2个参数');
  assert(typeof a=='number', '第一个参数必须是数字');
  assert(typeof b=='number', '第二个参数必须是数字');

  return a+b;
}

console.log(sum(12, 5));
```

3.3 fs

读取文件 返回的是一个buffer

```
//读谁  回调函数
```

fs.readFile("1.txt", (err, data)=>{});

```
fs.readFile('1.txt', (err, data)=>{
  if(err){
    console.log('有错');
  }else{
    console.log(data.toString());
  }
});
```

写入文件

```
//往哪写  写啥  回调函数
```

fs.writeFile("3.txt", "内容", err=>{});

```
fs.writeFile('3.txt', 'erqwreqwere', err=>{
  if(err){
    console.log(err);
  }else{
    console.log('成功');
  }
});
```

注意：data.toString(); 仅限于确定真的是文字，例如html，css等，二进制数据不要转成字符串

又读又写

```
const fs=require('fs');

fs.readFile('ofo.png', (err, data)=>{
  fs.writeFile('ofo2.png', data, ()=>{});
});
```

3.3.1 读写流

POST表单enctype="multipart/form-data"

- 表单的三种POST：

  1.text/plain　　用的很少，纯文字
  2.application/x-www-form-urlencoded　　默认　　url编码方式，xxx=xxx&xxx=xx...
  3.multipart/form-data　　上传文件内容

- readFile先把所有数据全读到内存中，然后回调：
  1.极其占用内存
  2.资源利用极其不充分

- fs.writeFile 收到一部分就解析一部分　　极大节约内存
  - 流：
    读一点、发一点
  - 1.读取流　fs.createReadStream、req
    2.写入流　fs.createWriteStream、res
    3.读写流　压缩、加密

```
const fs = require("fs");

let r1 = fs.createReadStream("1.png");
let w1 = fs.createWriteStream("2.png");

//流是有方向的，读取方找写入方
r1.pipe(w1);

//所有的流都有on方法
r1.on("error", err=>{
  console.log("读取失败");
});

w1.on("finish", ()=>{
  console.log("写入成功");
});
```

3.3.2 zlib压缩

- 创建一个gzip压缩对象
  let gz=zlib.createGzip();

```
const zlib=require('zlib');
const fs=require('fs');

let rs=fs.createReadStream('jquery.js');
let ws=fs.createWriteStream('jquery.js.gz');

//创建一个gzip压缩对象
let gz=zlib.createGzip();

rs.pipe(gz).pipe(ws);

ws.on('finish', ()=>{
  console.log('完成');
});
```

- 结合服务器
  - res.setHeader('content-encoding', 'gzip');

    设置一个响应头，要不然读不出来，默认会下载

```
const http=require('http');
const fs=require('fs');
const zlib=require('zlib');

let server=http.createServer((req, res)=>{
  let rs=fs.createReadStream(`www${req.url}`);

  //rs.pipe(res);

  res.setHeader('content-encoding', 'gzip');

  let gz=zlib.createGzip();
  rs.pipe(gz).pipe(res);

  rs.on('error', err=>{
    res.writeHeader(404);
    res.write('Not Found');

    res.end();
  });
});
server.listen(8080);
```

3.4 os系统相关

```
const os=require('os');
console.log(os.cpus());
```

3.5 path路径

```
const path=require('path');

let str='/var/local/www/aaa/1.png';

//dirname   路径
//basename  文件名
//extname   扩展名
//path.resolve("文件名")把相对路径转成绝对路径，只解析路径，不会判断文件是否存在

console.log(path.extname(str));
```

3.6 Crypto签名—md5

```
const crypto=require('crypto');
              //以哪种方式
let obj=crypto.createHash('md5');

//obj.update('123456');

obj.update('123');
obj.update('4');
obj.update('56');
                //进制
console.log(obj.digest('hex'));

//双层md5
const crypto=require('crypto');

function md5(str){
  let obj=crypto.createHash('md5');
  obj.update(str);

  return obj.digest('hex');
}

console.log(md5(md5('123456')+'se32ssdfsd43'));
```

3.7 Events事件队列

events和函数调用的区别：

接触耦合（函数不能轻易改）

```
const Event=require('events').EventEmitter;

/*
let ev=new Event();
```

```
//1.监听(接收)
ev.on('msg', function (a, b, c){
  console.log('收到了msg事件：', a, b, c);
});

//2.派发(发送)
ev.emit('msg', 12, 5, 88);
*/

function msg(a, b, c){
  console.log('收到了msg事件：', a, b, c);
}

msg(12, 5, 88);
```

3.8 Query Strings

查询字符串，?号后边的就叫Query Strings

https://www.baidu.com/s?wd=md5%E8%A7%A3%E5%AF%86&rsv_spt=1&rsv_iqid=0xc69fdef90015a4d9&issp=1&f=8&rsv_bp=1&rsv_idx=2&ie=utf-8&tn=baiduhome_pg&rsv_enter=1&rsv_dl=tb&rsv_sug2=0&rsv_btype=i&inputT=3679&rsv_sug4=4767

解析 ? 号后边的部分

querystring.parse() 解析出一个json

```
const querystring=require('querystring');

let obj=querystring.parse('ie=utf-8&f=8&rsv_bp=0&rsv_idx=1&tn=baidu&wd=aa&rsv_pq=f80d982000063ffb&rsv_t=6498LAZdRZjq9v4v0hs88kZItnCjDpT6UNBKr%2FF83%2F%2Bg4eiPURW2eQl9Iwc&rqlang=cn&rsv_enter=1&rsv_sug2=0&inputT=10&rsv_sug4=10');

console.log(obj);
```

3.9 URL

解析整个URL

url.parse()

```
                          //把query也解析了
let obj = url.parse("url", true);
```

3.10 DNS

域名解析

```
const dns=require('dns');

dns.resolve('google.com', (err, res)=>{
  if(err){
    console.log('解析失败');
  }else{
    console.log(res);
  }
});
```

4 缓存

浏览器默认没有缓存，设置--preferences--network--关掉Disable cache(while DevTools is open)

5 数据库

- 认识数据库：
  1.库-文件夹：不能存数据，只能管理表
  2.表-文件：存数据

- 使用navicat：
  1.连接到服务器
  新建mysql连接

  2.打开连接
  双击

  3.新建库
  名字：尽量别用-
  字符集：utf8
  排序规则：utf8_general_ci
  4.新建表
  字段

  导入数据——必须先建库

- 字段类型：
  数字
  整数：tinyint(8位)、smallint、mediumint、int、bigint
  -128~127|0~255　　-21亿~21亿|0~43亿　18万万亿
  小数：float-单精度浮点数、double-双精度浮点数
  小数点后8位　　10^308

  字符串
  小：varchar　255
  大：text　　1G

  主键：
  1.性能高
  2.唯一

5.1 Nodejs操作数据库

1. cnpm i mysql -D

5.1.1 node中mysql写法：

let db=mysql.createConnection({配置});
let db=mysql.createPool({配置});

db.query(sql, (err, data)=>{});

```
const mysql=require('mysql');

//连接池
let db=mysql.createPool({host: 'localhost', user: 'root', password: '', port:
3309, database: '20180127'});

db.query(`INSERT INTO student_table (ID, name, gender, chinese, math, english)
VALUES(0, '小明', '男', 98, 5, 3);`, (err, data)=>{
  if(err){
    console.log('错了', err);
  }else{
    console.log(data);
  }
});
```

5.1.2 基础SQL语句

增删改查

1. 增　INSERT
   INSERT INTO 表 (字段列表) VALUES(值列表)

   INSERT INTO user_table (ID, name, gender, chinese, math, english) VALUES(0, 'blue', '男', 35, 18, 29);
2. 删　DELETE
   DELETE FROM 表 WHERE 条件

   DELETE FROM user_table WHERE ID=3;
3. 改　UPDATE
   UPDATE 表 SET 字段=值, 字段2=值2, ... WHERE 条件

   UPDATE user_table SET chinese=100 WHERE ID=2;
4. 查　SELECT
   SELECT 字段列表 FROM 表 WHERE 条件

   SELECT name, gender FROM user_table WHERE ID=2;

登录注册

```
const http=require('http');
const mysql=require('mysql');
```

```
const fs=require('fs');
const url=require('url');
const zlib=require('zlib');
const crypto=require('crypto');

const
_key='sadfslekrtuew5iutoselgdtjiypoydse4ufhs.edtyo;s8te4arfeliawkfhtsie5tlfia;se
fdshroiupeoutwyeli5gurse;ihf';

function md5(str){
  let obj=crypto.createHash('md5');

  obj.update(str);

  return obj.digest('hex');
}

function md5_2(str){
  return md5(md5(str)+_key);
}

let db=mysql.createPool({host: 'localhost', port: 3306, user: 'root', password:
'', database: '20211009'});

let server=http.createServer((req, res)=>{
  let {pathname, query}=url.parse(req.url, true);
  let {user, pass}=query;

  switch(pathname){
    //接口
    case '/reg':
      //校验
      if(!user){
        res.write('{"err": 1, "msg": "username can\'t be null"}');
        res.end();
      }else if(!pass){
        res.write('{"err": 1, "msg": "password can\'t be null"}');
        res.end();
      }else if(!/^\w{4,16}$/.test(user)){
        res.write('{"err": 1, "msg": "username is invaild"}');
        res.end();
      }else if(/['|"]/.test(pass)){
        res.write('{"err": 1, "msg": "password is invaild"}');
        res.end();
      }else{
        db.query(`SELECT * FROM user_table WHERE username='${user}'`, (err,
data)=>{
          if(err){
            res.write('{"err": 1, "msg": "database error"}');
            res.end();
          }else if(data.length>0){
            res.write('{"err": 1, "msg": "this username exsits"}');
            res.end();
          }else{
            db.query(`INSERT INTO user_table (ID,username,password)
VALUES(0,'${user}','${md5(pass)}')`, (err, data)=>{
              if(err){
                res.write('{"err": 1, "msg": "database error"}');
```

```
                res.end();
              }else{
                res.write('{"err": 0, "msg": "success"}');
                res.end();
              }
            });
          }
        });
      }
      break;
    case '/login':
      //校验
      if(!user){
        res.write('{"err": 1, "msg": "username can\'t be null"}');
        res.end();
      }else if(!pass){
        res.write('{"err": 1, "msg": "password can\'t be null"}');
        res.end();
      }else if(!/^\w{4,16}$/.test(user)){
        res.write('{"err": 1, "msg": "username is invaild"}');
        res.end();
      }else if(/['|"]/.test(pass)){
        res.write('{"err": 1, "msg": "password is invaild"}');
        res.end();
      }else{
        db.query(`SELECT * FROM user_table WHERE username='${user}'`, (err,
data)=>{
          if(err){
            res.write('{"err": 1, "msg": "database error"}');
            res.end();
          }else if(data.length==0){
            res.write('{"err": 1, "msg": "no this user"}');
            res.end();
          }else if(data[0].password!=md5(pass)){
            res.write('{"err": 1, "msg": "username or password is incorrect"}');
            res.end();
          }else{
            res.write('{"err": 0, "msg": "success"}');
            res.end();
          }
        });
      }
      break;
    default:
      //缓存      TODO
      //静态文件
      let rs=fs.createReadStream(`www${pathname}`);
      let gz=zlib.createGzip();

      res.setHeader('content-encoding', 'gzip');
      rs.pipe(gz).pipe(res);

      rs.on('error', err=>{
        res.writeHeader(404);
        res.write('Not Found');
        res.end();
      });
  }
```

```
});
server.listen(8080);
```

6 express

express是非破坏式的框架，会尽可能的保留原生的东西

1. cnpm i -y 初始化项目
2. cnpm i express -D 安装express

6.1 用法:

let server=express();

server.get('/xxx', ()=>{});
server.post('/xxx', ()=>{});
server.use('/xxx', ()=>{}); //通用——不限制方法

//不限制路径
server.get(()=>{});
server.post(()=>{});
server.use(()=>{});

```
const express=require('express');

let server=express();
server.listen(8080);

//get方法，'/a'是路径地址，他俩合起来判断了2个东西，1：用户以get方式要东西，2：要的是a这个路径东西
server.get('/a', ()=>{
  console.log('请求a');
});

server.post('/upload', ()=>{
  console.log('请求upload');
});
```

6.1.1 send()

res.send(any)

```
server.get("/a", (req, res)=>{
  //res.write("");返回的只能是字符串或者buffer
  res.send({a: 1, b: 2});//字符串、数组、json
  //res.end()不是强制性的
});
```

6.1.2 sendFile()

读取文件：可以选择

res.sendFile(绝对路径文件名);

```
res.sendFile(pathlib.resolve("a.txt"));
```

6.1.3 sendStatus()状态码

res.sendStatus(code);等价于writeHeader+write+end

```
res.sendStatus(404);
```

6.1.4 redirect()重定向

res.redirect('location');

6.1.5 next()

```
server.get('/a', (req, res, next)=>{
  console.log('bbbb');

  //走完第一个再走第二个，需要手动调用，不调用不走下一个
  next();
});

server.get('/a', (req, res, next)=>{
  console.log('aaaa');

  next();
});
```

6.2 中间件

server.use(中间件);

6.2.1 express.static(url) 静态文件

express.static("文件路径")

返回一个静态文件（压缩，缓存都解决了）不加选择的返回，只要它有就返回

```
server.use(express.static('www/'));
```

6.2.2 数据交互 GET、POST

- GET=>req.query

```
server.get('/a', (req, res)=>{
  console.log(req.query);
});
```

- 普通POST=>body-parser弃用了，express自身可以解析urlencoded了

  server.use(express.urlencoded({extended: false}));

  req.body

```
const express=require('express');

let server=express();
server.listen(8080);

server.use(express.urlencoded({extended: false}));

server.post('/upload', (req, res)=>{
  console.log(req.body);
});
```

- 文件POST=>multer

  cnpm i multer -D

  server.use(multer({dest: 'upload/'}).any())

  req.body
  req.files

```
const express=require('express');
const multer=require('multer');

let server=express();
server.listen(8080);
                    //存放的目录        什么类型的文件都接
server.use(multer({dest: 'upload/'}).any());

server.post('/upload', (req, res)=>{
  console.log(req.body);
  console.log(req.files);
});
```

6.2.3 自己写一个POST中间件

```
const express = require("express");
const body = require("./libs/my-body");

let server = express();
server.listen(8080);
```

```
server.use(body);

server.post("/upload", (req, res)=>{
  console.log(req.body);
});

//my-body.js
const querystring=require('querystring');

module.exports=(req, res, next)=>{
  let arr = [];
  req.on("data", data=>{
    arr.push(data);
  });
  req.on("end", ()=>{
    req.body = querystring.parse(Buffer.concat(arr).toString());

    next();
  });
}
```

6.3 cookie

6.3.1 cookie

cookie——存在浏览器里
 容量有限——4K
 不安全——用户、浏览器

1.防篡改
2.加密（cookie本身没法加密）

用法：

cnpm i cookie

接收cookie

```
const express=require('express');
const cookieParser = require("cookie-parser");

let server=express();
server.listen(8080);

server.use(cookieParser({}));

server.get("/upload", (req, res)=>{
  console.log(req.cookies);

  res.send("aaa");
});
```

发送cookie

```
const express=require('express');
const cookieParser=require('cookie-parser');

let server=express();
server.listen(8080);

server.use(cookieParser('dfdsgfjfyt6dr5setrgdxgxhgDGFDHY%7yt^%Tdfsdtryrt$%#$%$#$
#%RFgdfdfxghfEFGFDDF'));

server.get('/a', (req, res)=>{
  console.log(req.cookies);
    //返回带签名的内容
  console.log(req.signedCookies);
            //名字 值    发一个带签名的
  res.cookie('c', 33, {signed: true});

  res.send('aaa');
});
```

6.3.2 session

session——存在服务器
 容量不用担心
 安全——

在浏览器关闭的时候失效

cookie-session

session劫持
 sess_id拿走：
 1.session定期更换ID——有效期
 2.签名

 server.use(cookieSession({
  keys or secret
 }))

req.session

用法：

cnpm i cookie-session

```
const express=require('express');
const cookieSession=require('cookie-session');

let server=express();
server.listen(8080);

server.use(cookieSession({
  keys: ['afsfasfasdfsadf', 'xcvxchdftyuftuyfy', 'asdgsdrgse5t6tr5',
'fgdfghfty8654esdghfjhg'],
  //secret: 'xxx'
}));

server.get('/a', (req, res)=>{
  if(!req.session['count']){
    req.session.count=1;
  }else{
    req.session.count++;
  }

  res.send(`欢迎你第${req.session.count}次来访`);
  res.end();
});
```

6.4 路由

定义：根据地址(url)不同，调用不同的代码

用法：

1.创建
  let router=express.Router();

2.给路由填东西
 router.get('地址', ()=>{});
 router.post('地址', ()=>{});
  router.use('地址', ()=>{});

3.添加到父级
  server.use('路径', router);
  parentRouter.use('路径', router);

- 都写在主页面 前面的/一定要加

```
let articleRouter = express.Router();

articleRouter.get("/", (req, res)=>{
  res.send("文章首页1");
});

server.use("/article", articleRouter);
```

- routes目录

```
//主页面用法
server.use('/article', require('./routes/article'));
server.use('/user', require('./routes/user'));

server.use((req, res)=>{
  res.send('404你懂的');
});

//子页面要exports出去
module.exports=router;
```

7 服务端渲染

后端渲染(组装)：html生成出来
前端渲染(组装)：html生成出来
浏览器渲染：
输入url -> 加载 -> html,css,js,img,... -> 渲染(画)

8 模块化

在模块化系统下，没有真正的全局变量，所有语言(除JS之外)都有模块系统

1.前端
传统——requireJS-AMD、seaJS-CMD
CMD——公共模块定义
命名空间
AMD——异步模块定义
CMD+异步
现代——vue、angular、react
2.后端
Node模块系统

9 koa

异步更简单——async

本身带cookie

npm i koa koa-static koa-better-body koa-convert koa-router -D

koa-static 处理静态文件，没有压缩

koa-static-cache 有压缩

koa-better-body 既管普通POST也管文件POST

koa-convert 帮助中间件平滑的过渡到koa3的写法

9.1 koa和express的区别

express 非破坏式
koa 破坏式

express 不依赖router
koa 强依赖router

9.2 用法

1. 开启服务器

```
const koa=require('koa');
let server=new koa();
server.listen(8080);
```

2. (req, res, next)
   (ctx, next)

   ctx.request
      ctx.request.method
      ctx.request.url
      ctx.request.header/headers

   ctx.response
      ctx.response.status=xxx 状态码
      ctx.response.body=xxx 返回内容
         ctx.response.set('a', 12); 设置头

9.3 路由

koa强依赖router

const router=require('koa-router');

let r1=router();
server.use(r1.routes());

r1.get('xxx', async);
r1.post('xxx', async);
r1.use('xxx', async);
r1.put('xxx', async);
r1.delete('xxx', async);

```
const koa = require("koa");
const router = require("koa-router");

let server = new koa();
server.listen(8080);

let r1 = router();
server.use(r1.routes());

r1.get("/a", async (ctx, next)=>{
  console.log(ctx.request);
});
```

9.4 请求数据（better-body）

- GET　　ctx.request.query
  路由参数　ctx.params
- POST/文件
  server.use(convert(betterBody({
  uploadDir:　'上传路径',
  keepExtensions: bool（拓展名)
  })));
  ctx.request.fields　　数据+文件信息（普通数据）
  ctx.request.files　　文件信息(全)
  - better.body

```
const koa=require('koa');
const betterBody=require('koa-better-body');
const convert=require('koa-convert');
const pathlib=require('path');

let server=new koa();
server.listen(8080);

server.use(convert(betterBody({
  uploadDir:pathlib.resolve("./upload")
})));

server.use(async ctx=>{
  console.log(ctx.request.fields);
  console.log(ctx.request.files);
});
```

9.5 cookie

ctx.cookies.get(name)
ctx.cookies.set(name, val, options)
options:
  maxAge 毫秒单位时间
  expires Date对象
  path
  domain

```
const koa=require('koa');
const response = require('koa/lib/response');

let server=new koa();
server.listen(8080);

server.use(async ctx=>{
  console.log(ctx.cookies.get("a"));

  ctx.cookies.set("b", 12, {
    maxAge: 86400*1000
  });

  ctx.response.body="aaa";
});
```

9.6 session

cnpm i koa-session -D

server.use(session({}, server));
ctx.session

```
const koa=require('koa');
const session=require('koa-session');

let server=new koa();
server.listen(8080);

server.keys=require('./.keys');

server.use(session({}, server));

server.use(async ctx=>{
  if(ctx.session['count']){
    ctx.session['count']++;
  }else{
    ctx.session['count']=1;
  }

  ctx.response.body=`这是你第${ctx.session.count}次来访`;
});
```

9.7 ejs

cnpm i koa-ejs -D

ejs 非破坏式——保留HTML结构
1.输出
转义输出 <%= %>
非转义输出 <%- %>

.ejs文件

```
<div class="">
    我叫<%=name%>，我<%=age%>岁
</div>

<ul>
      <%- include(header_path) %>

      <% arr.forEach(item=>{ %>
      <li><%=item%></li>
      <% }) %>

      <%- include('component/footer.html') %>
</ul>
```

.js文件

```
const koa=require('koa');
const pathlib=require('path');
const ejs=require('koa-ejs');

let server=new koa();
server.listen(8080);

ejs(server, {
  root:     pathlib.resolve('template'),
  layout:   false,
  viewExt:  'ejs'//视图模板的拓展名
});

server.use(async ctx=>{
  await ctx.render('1', {
    name: 'blue',
    age: 18
  });

  console.log(ctx.render);
});
```

9.8 数据库

cnpm i mysql-pro -D

mysql-pro：
优点1.事务支持
优点2.防止注入

Transaction - 事务
要么都发生、要么都不发生

ACID：
A 原子性：要么都发生、要么都不发生
C 持久性：只要事务提交了，他的作用就是永久
I 隔离性：各个事务之间是独立
D 一致性：事务前后的状态是一致的

- 创建连接

```
const db=new Mysql({
  mysql: {
    host: 'localhost',
    port: 3309,
    user: 'root',
    password: '',
    database: '20180412'
  }
});
```

- 完整代码

  let data=await db.executeTransaction('SELECT * FRO user_table WHERE ID=? AND age=?', [id, 18]);

  ID=? AND age=?', [id, 18] 问号是占位符，真正的数据存在[ ]中

```
const koa=require('koa');
const router=require('koa-router');
const Mysql=require('mysql-pro');

const db = new Mysql({
  mysql: {
    host: "localhost",
    port: 3306,
    user: "root",
    password: "",
    database: "20211009"
  }
});

let server = new koa();
server.listen(8080);

let r1 = router();
server.use(r1.routes());

r1.get('/user', async ctx=>{
  let id=ctx.query.id;

  try{
    await db.startTransaction();
```

```
    let data=await db.executeTransaction('SELECT * FRO user_table WHERE ID=? AND age=?', [id, 18]);
    await db.stopTransaction();

    ctx.response.body=data;
  }catch(e){
    ctx.response.body='数据库正在维护中，请稍候重试';
  }
});
```

9.9 知乎

1. 建库、建表

2. 数据导进去
   INSERT INTO topic_table VALUES(1, 'html5'),(2, '移动端开发'),(3, 'javascript')
3. 写服务器