Promise

1.2 Promise 的基本概念

① Promise 是一个构造函数

- 我们可以创建 Promise 的实例 const p = new Promise()
- new 出来的 Promise 实例对象，代表一个异步操作

② Promise.prototype 上包含一个 .then() 方法

- 每一次 new Promise() 构造函数得到的实例对象，
- 都可以通过原型链的方式访问到 .then() 方法，例如 p.then()

③ .then() 方法用来预先指定成功和失败的回调函数

- p.then(成功的回调函数，失败的回调函数)
- p.then(result => {}, error => {})
- 调用 .then() 方法时，成功的回调函数是必选的、失败的回调函数是可选的

1. 终端中的快捷键

在 Windows 的 powershell 或 cmd 终端中，我们可以通过如下快捷键，来提高终端的操作效率：

① 使用 ↑ 键，可以快速定位到上一次执行的命令

② 使用 tab 键，能够快速补全路径

③ 使用 esc 键，能够快速清空当前已输入的命令

④ 输入 cls 命令，可以清空终端

param

get("hello:id")
此时id作为对象名
{id:xxx}
就能获取网址中的内容
通过 req.param来获取值

注意：req.query是获取字符串的参数，常在get使用

post

获取post的值需要 req.body 来获取参数
默认情况下 express不会自动解析请求体，需要通过中间件来增加功能

`app.use(express.urlencoded())`

遍历用户名

for(user of USERs){
conse.log(user.username)
}

模板引擎.

---在html静态网页中使用变量

1.存在好多模板引擎

https://expressjs.com/en/resources/template-engines.html

- Pug: Haml-inspired template engine (formerly Jade).
- Haml.js: Haml implementation.
- EJS: Embedded JavaScript template engine.
- hbs: Adapter for Handlebars.js, an extension of Mustache.js template engine.
- Squirrelly: Blazing-fast template engine that supports partials, helpers, custom tags, filters, and caching. Not white-space sensitive, works with any language.
- Eta: Super-fast lightweight embedded JS template engine. Supports custom delimiters, async, whitespace control, partials, caching, plugins.
- combyne.js: A template engine that hopefully works the way you’ d expect.
- Nunjucks: Inspired by jinja/twig.
- marko: A fast and lightweight HTML-based templating engine that compiles templates to CommonJS modules and supports streaming, async rendering and custom tags. (Renders directly to the HTTP response stream).
- whiskers: Small, fast, mustachioed.
- Blade: HTML Template Compiler, inspired by Jade & Haml.
- Haml-Coffee: Haml templates where you can write inline CoffeeScript.
- express-hbs: Handlebars with layouts, partials and blocks for express 3 from Barc.
- express-handlebars: A Handlebars view engine for Express which doesn’ t suck.
- express-views-dom: A DOM view engine for Express.
- rivets-server: Render Rivets.js templates on the server.
- LiquidJS: A simple, expressive and safe template engine.
- express-tl: A template-literal engine implementation for Express.
- Twing: First-class Twig engine for Node.js.
- Sprightly: A very light-weight JS template engine (45 lines of code), that consists of all the bare-bones features that you want to see in a template engine.
- html-express-js: A small template engine for those that want to just serve static or dynamic HTML pages using native JavaScript.

2.各具特色 **推荐使用 EJS** mde/ejs: Embedded JavaScript templates -- http://ejs.co (github.com)

```
$ npm install ejs
配置express模板引擎为ejs  app.set("view engine","ejs")
app.set("views",path.resolve(__dirname,"views"))//views是目录名
res.render("name",{}) //name是模板名字 在get/post请求路由中，请求的时候渲染过去
<%=name %>  //name是属性名
<%-name %>  //这样不会被转义原样输出
```

```
7  <% %>//可以直接写js代码逻辑
```

```
<% if(name === "孙悟空") {%>
    <h2>大师兄来了</h2>
<%} else{%>
    <h2>二师兄来了</h2>
<%}%>
```

同时 还可以分开写，用js来输出不同结果

重定向

res.redirect(" ")

避免数据提交后使用重新渲染，这样会导致数据重复提交

将新的数据写入json文件

Router

router是express中一个对象

```
const router = express.Router();
router.get("/hello",(req,res)=>{
   res.send("router hello")
})
app.use(router)
```

好处是 router和app不是同一个对象

所以能把路由和服务器主文件分开

路由里面也有next

这样把文件写入功能放到中间件（前面的不执行才执行中间件）

加上next()这样就能执行中间件了，类似于404中间件处理

cookie简介

cookie是http协议中来解决无状态问题的技术

cookie本质就是一个头

服务器以响应头的形式将cookie发送给客户端

这样服务器就可以根据cookie识别出客户端

cookie是服务器创建的，（带纸条的小饼干）。

cookie可以来实现登录功能

1.先请求cookie

```
app.get("get-cookie", (res, req) => {
  res.cookie("username","admin");
});
```

2.再次发送请求时候，响应头内会携带cookie

3.默认express不会自动解析cookie，需要安装中间件 cookie-parser 来解析

a.安装cookie-parser

b.引入cookie-parser

c.设置为中间件

e.req.cookie能正常读取了

cookie的不足

- cookie由服务器创建，浏览器存缓

每次浏览器访问需要将cookie发回

导致我们不能再cookie存储过多信息

并且cookie是存在客户端容易被篡改

-注意：

我们在cookie时不要存敏感数据

-所以为了cookie不足，我们吧把用户的数据存在一个服务器中

每个用户有一个对应id，每次只需要cookie把id发送给浏览器

浏览器只需要每次访问发回id，边可以读取服务器中的数据

这个技术称为 session（会话）

session

session是服务器的一个对象，这个对象才存储用户数据

每个session对象都有唯一id，id会通过cookie形式发送、
客户端每次只发送带有id的cookie
在express中可以通过安装 express-session组件来实现
1.安装 yarn add express-session
2.引入 const session = require("express-session");
3.设为中间件 app.use(session({
secret:"jiamiyanzhi" //加密盐值随便设置
}));

注意，使用session时候可以不引入cookie，不依赖cookie