REST全称是Representational State Transfer，中文意思是表述性状态转移

列出了GET，DELETE，PUT和POST的典型用法

接口的开发：接口请求方式　GET/POST/PUT/DELETE

编写接口：

　res.json(obj)　等效于　res.send(users)

　res.json(obj) 作用：

　　0.设置状态码200

　　1.设置响应头Content-Type: application/json

　　2.返回序列化数据JSON.stringify(obj)

案例：☑

```
const users = [
    {name: 'tom', age: 20}
]
router.get('/', function (req, res, next) {
    // 0.设置状态码200
    // 1.设置响应头Content-Type: application/json
    // 2.返回序列化数据JSON.stringify(obj)
    res.json(users); // 等效于res.send(users)
});
// 新增
router.post('/', function (req, res) {
    try {
        users.push(req.body);
        res.json({success: true, users})
    } catch (e) {
        res.json({success: false})
    }
```

```
})
// 更新
router.put('/', (req, res) => {
    try {
        const index = users.findIndex(u => u.name == req.body.name)
        users[index] = req.body;
        res.json({success: true, users})
    } catch (e) {
        res.json({success: false})
    }
})
// 删除
router.delete('/:name', (req, res) => {
    try {
        console.log(req.params.name);
        const index = users.findIndex(u => u.name == req.params.name)
    } catch (e) {
        res.json({success: false})
    }
})
```

跨域　浏览器从一个域名的网页去请求另一个域名的资源时，域名、端口、协议任一不同，都是跨域

浏览器同源策略（协议、主机名、端口）引起的前端接口调用问题

解决方案：

1、jsonp(Json With Padding)，前端+后端，绕过跨域

JSON返回的是一串数据、JSONP返回的是脚本代码(包含一个函数调用)

JSONP 只支持get请求、不支持post请求

原理：前端动态添加script到当前页面，其src指向接口URL，服务器返回一个函数执行语句，

该函数名有callback参数决定

实现：

前端：指定请求方式为jsonp, jquery中 {dataType:true}

```
<script src="/javascripts/jquery.js"></script>
<script>
        // jsonp
      $.ajax({
          url: 'http://127.0.0.1:8080/users/jsonp',
          dataType: 'jsonp',     //前端使用JSONP方式发送请求
          data: {name: 'tom'},
          success: function (users) {
              console.log(users);
          }
      })
</script>
```

后端：res.jsonp(data)

```
// jsonp
router.get('/jsonp', (req, res) => {
    // 参数通过查询参或者url参数获取
    res.jsonp(users);
})
```

res.jsonp(users)的底层实现：

// 底层实现原理，callback是回调函数名称

// res.send(`${req.query.callback}(${JSON.stringify(users)})`) //es6

// res.send(req.query.callback+'('+JSON.stringify(users)+')') //es5

2、PROXY 代理，后端方案

原理：通过同源服务器发送请求至接口服务器，再由同源服务器转发结果给前端，从而规避跨域

webpack-dev-server

proxy

3、CORS（Cross），后端方案

原理：

CORS是W3C规范，真正意义上解决跨域问题。他需要服务器对请求检查，对响应头做特殊处理，从而解决跨域

3.1、对于简单请求，添加：res.set('Access-Control-Allow-Origin', 'http://localhost:8080');

简单请求：方法是GET、POST、HEADER，同时对于post请求，不能有自定义请求头，同时编码方式必须是

application/x-www-urlencoded, multipart/form-data,text/plain之一

案例：

前端

```
// cors  get
$.ajax({
    url: 'http://127.0.0.1:8080/users/cors',
    success: function (users) {
        console.log(users);
    }
})
```

后端

```
router.get('/cors', (req, res) => {
    // 添加响应头Access-Control-Allow-Origin
    // 并指定请求的主机名，端口，协议，符合则允许请求
    res.set('Access-Control-Allow-Origin', 'http://localhost:8080');
    res.json(users);
})
```

3.2、对于预检测preflight请求，由于请求头有指定时，需要对请求做预处理

1）预检测路由 router.options()

2）响应头添加res.set('Access-Control-Allow-Headers', 'Content-Type');

3）如果请求方法是PUT或DELETE添加res.set('Access-Control-Allow-Methods',

'GET,POST,PUT');

案例：

前端

```
// post
$.ajax({
    url: 'http://127.0.0.1:8080/users/cors',
    method: 'post', // 默认编码urlencoded
    headers: {'Content-Type': 'application/json'},
    data: JSON.stringify({ name:'jerry', age:20 }),
    success: function (users) {
        console.log(users);
    }
})
```

请求指定了请求头为JSON

后端 ---------->要先进行预处理，然后再根据预处理结果判断请求是否跨域，决定是否响应

```
// 预检测
router.options('/cors', (req, res) => {
    res.set('Access-Control-Allow-Origin', 'http://localhost:8080'); //请求域
    res.set('Access-Control-Allow-Headers', 'Content-Type');  //请求头数据类型
    res.sendStatus(204);
})


router.post('/cors', (req, res) => {
    // 添加响应头Access-Control-Allow-Origin
    res.set('Access-Control-Allow-Origin', 'http://localhost:8080');
    res.json(users);
})
```

真正处理响应请求

对不同请求方法的预处理（DELETE，PUT）

前端

```
// put
$.ajax({
    url: 'http://127.0.0.1:8080/users/cors',
    method: 'put',
    headers: {'Content-Type': 'application/json'},
    data: JSON.stringify({name: 'tom', age: 21}),
    success: function (users) {
        console.log(users);
    }
})
```

后端

```
// 预检测
router.options('/cors', (req, res) => {
    res.set('Access-Control-Allow-Origin', 'http://localhost:8080'); //请求域
    res.set('Access-Control-Allow-Headers', 'Content-Type');  //请求头数据类型
    res.set('Access-Control-Allow-Methods', 'GET,POST,PUT');  //请求方法
    res.sendStatus(204);
})


router.put('/cors', (req, res) => {
    res.set('Access-Control-Allow-Origin', 'http://localhost:8080');
    const index = users.findIndex(u => u.name == req.body.name)
    if (index != -1) {
        users[index] = req.body;
    }
    res.json(users);
})
```

对于证书类型Credentials：请求中携带cookie的请求，需要额外处理

前端

```
// put
$.ajax({
    url: 'http://127.0.0.1:8080/users/cors',
    method: 'put',
    headers: {'Content-Type': 'application/json'},
    data: JSON.stringify({name: 'tom', age: 21}),
    xhrFields: {withCredentials: true}, // 请求携带cookie
    success: function (users) {
        console.log(users);
    }
})
```

后端

```
// 预检测
router.options('/cors', (req, res) => {
    res.set('Access-Control-Allow-Origin', 'http://localhost:8080'); //请求域
    res.set('Access-Control-Allow-Headers', 'Content-Type');  //请求头数据类型
    res.set('Access-Control-Allow-Methods', 'GET,POST,PUT');  //请求方法
    res.set('Access-Control-Allow-Credentials', 'true');  //请求中携带cookie
    res.sendStatus(204);
})


router.put('/cors', (req, res) => {
    res.set('Access-Control-Allow-Origin', 'http://localhost:8080');
    res.set('Access-Control-Allow-Credentials', 'true');
    const index = users.findIndex(u => u.name == req.body.name)
    if (index != -1) {
        users[index] = req.body;
    }
    res.json(users);
})
```

4、后端通过加载cors模块全局处理跨域问题

4.1、安装　npm i cors -S

4.2、在app.js中配置

引入： const cors = require('cors')

配置：应用中间件（如果不涉及允许请求携带cookie信息那么，可以这样配置：**app.use(cors());**）

app.use(cors()); 门户大开，相当于Access-Control-Allow-Origin：*

```
app.use(cors({
    origin: 'http://localhost:8080',  //允许的请求域
    credentials: true          //允许请求携带cookie的请求
}));
```

4.3、使用只需要改变后端即可

所有响应正常处理，不用本配预处理，不用配响应头

如前端

前端请求

```
// put
$.ajax({
    url: 'http://127.0.0.1:8080/users/cors',
    method: 'put',
    headers: {'Content-Type': 'application/json'},
    data: JSON.stringify({name: 'tom', age: 21}),
    success: function (users) {
        console.log(users);
    }
})
```

后端引入cors模块的响应处理：

```
router.put('/cors', (req, res) => {
    const index = users.findIndex(u => u.name == req.body.name)
    if (index != -1) {
        users[index] = req.body;
    }
    res.json(users);
})
```

其它配置:

```
//白名单
var whitelist = ['http://example1.com', 'http://example2.com']
app.use(cors({
  origin: function (origin, callback) {
    if (whitelist.indexOf(origin) !== -1) {
      callback(null, true)
    } else {
      callback(new Error('Not allowed by CORS'))
    }
  }
}));

//异步配置
var whitelist = ['http://example1.com', 'http://example2.com']
var corsOptionsDelegate = function (req, callback) {
  var corsOptions;
  if (whitelist.indexOf(req.header('Origin')) !== -1) {
    corsOptions = { origin: true } // 生成配置项
  } else {
    corsOptions = { origin: false }
  }
  callback(null, corsOptions) // 将选项作为参数2
}
```