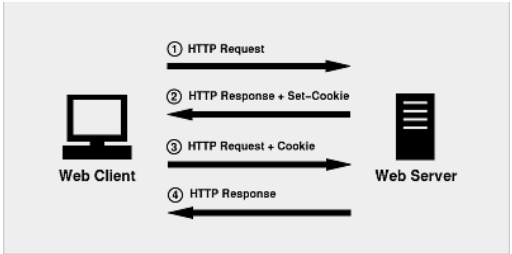
10 JSON Web Token实现用户验证

更新时间：2019-08-07 10:35:09

> 今天应做的事没有做，明天再早也是耽误了。
>
> ——裴斯泰洛齐

每一个web应用，尤其是牵扯到用户相关模块的应用，都离不开用户验证（让服务器知道你是谁）。传统的流程一般是基于session和cookie，步骤如下所示：

1. 输入用户名和密码登录。
2. 服务器验证通过后，在当前对话（session）里面保存相关数据，比如用户角色权限，登录时间等等。
3. 服务器向用户返回一个 sessionid，写入用户的 cookie。
4. 用户随后的每一次请求，都会通过 cookie，将 sessionid 带到服务器。
5. 服务器收到 sessionid，找到前期保存的数据，校验是否合法，由此得知用户的身份。



理解了基本的验证流程，我们就可能发现其中的一些问题。

session和cookie验证的问题：

1. cookie+session机制，session一般存于服务器内存中，如果在单机服务下还好，但是如果是分布式或服务器集

群，就要求 session 数据共享，每台服务器都能够读取 session，这样就会产生负载均衡问题。例如你在一台服务器上验证通过，但是如果后续此用户被导到另一台服务器，就无法识别验证身份了。

2. cookie+session 机制，session写入在cookie来储存，这样就有一定的CSRF安全风险。

JSON Web Token：

JSON Web Token简称JWT验证机制在一定程度上解决的上面的2个问题，它的流程是：

1. 用户登录，成功后服务器存储token并返回token给客户端。
2. 客户端收到数据后保存在客户端。
3. 客户端再次访问服务器，取出token放入http请求的headers中。
4. 服务器端采用拦截器校验，针对需要token的接口，校验成功则返回请求数据，校验失败则返回错误码。

由此可知，这里的token一般是加密的一串字符串，服务器存储token一般在数据库进行，而客户端请求时将token放入headers中，也可以防止CSRF的发生。接下来我们就在项目中使用这种方案。

安装jsonwebtoken：

jsonwebtoken是一个采用Node.js开发的JSON Web Token方案的实现，首先需要安装它：

```
npm install jsonwebtoken --save
```

安装完成功后，我们就需要将上面的流程用代码来实现，为了监听每一个API接口，我们可以借助Express框架的拦截器机制。

在后端项目的根目录中，wecircleServer的app.js里添加拦截器，代码如下：

```
//app.use是Express拦截器的方法
app.use(function(req, res, next) {

  // 拿取token 数据 按照自己传递方式写
  var token = req.headers['wec-access-token']||'xx';
  // 检查token是否有效（过期和非法）
  var user = tokenUtil.checkToken({token});
  if (user) {
    //将当前用户的信息挂在req对象上，方便后面的路由方法使用
    req.user = user;

    // 续期
    tokenUtil.setToken({user,res});

    next(); //继续下一步路由
  } else {
    //需要登录态域名白名单
    if (config.tokenApi.join(',').indexOf(req.path) < 0) {
      next();
      return;
    }
    res.json({ code: 1000, message: '无效的token.' });
  }
});
```

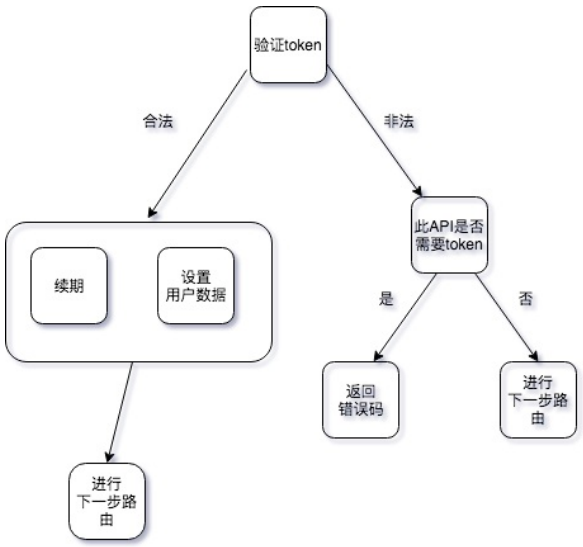
1. 在Express中，使用`app.use()`可以生成一个拦截器，优先于我们路由里的 post 和 get 执行。
2. 同时可以获取到一个`next`的参数，在拦截器的代码中使用`next()`，意思就是这个拦截器逻辑结束了，可以进行后续的其他处理，比如get或者post的逻辑，而同时我们可以在拦截器里设置一些数据，挂在到request上，方便后

续使用。

上面代码中用到了token.js，在后端项目的utils文件夹下新增token.js：

```
var jwt = require('jsonwebtoken');
var tokenTime = 1000 * 60 * 60 * 24;// 授权时效24小时

module.exports = {
 /*
 * 设置token
 */

 setToken(data){
  //将当前用户的信息通过token加密，并设置失效时间，得到token加密字符串
  //myjwttest标识token的key，固定字符串即可。
  var token = jwt.sign({user:data.user}, 'myjwttest', {
    expiresIn : tokenTime
  });
  //将token加密的字符串通过setCookie的方式传给客户端
  data.res.cookie('token', token, {
     maxAge: tokenTime,
  });
 },
 /*
 * 校验token
 */
 checkToken(data){
  var user = null;
  try {
   //如果根据token查到了用户信息，表示校验通过
   var decoded = jwt.verify(data.token, 'myjwttest');
   user = decoded.user;

  }catch(e){

  }

  return user
 }
};
```

这里的流程是：



1. 用户请求接口时，检查是否有token。

2. 有token就将当前用户的信息挂在req对象上，方便后面的路由方法使用。
3. 同时这里有一个续期的逻辑，因为如果我们强制将token设置成一个死的时间，那么无论多久，当时间过了之后，用户的token必定会过期，所以这个是不完美的。
4. 我们将用户的token通过cookie的方式存放在客户端，这样每次请求时，我们将token从cookie里获取到放入http的headers即可。
5. 如果你想把token存储在客户端localStorage里，也是可以的，但是并不推荐这样做，原因是localStorage在某些场景下是不可用的，例如如果浏览器是隐私模式时，而token这种数据又是及其重要的，所以建议放在cookie里最好，同时过期的逻辑可以直接由cookie控制。

在前端的service.js里，将token塞进去：

在前端项目的utils文件夹下，server.js中添加我们的token逻辑，代码如下：

```
// 封装post请求
function post (url, data = {}) {
  // 默认配置
  let sendObject = {
    url: url,
    method: 'post',
    headers: {
      'Content-Type': 'application/json;charset=UTF-8',
      'wec-access-token': getCookie('token')
    },
    data: data
  }
  return service(sendObject)
}
```

小节：

本章节主要讲解了常用的web应用的用户状态保存和校验的方案，并在代码里使用jsonwebtoken来对我们的项目进行登录校验的完善。
相关知识点：

1. session和cookie验证机制在分布式处理和CSRF上有一定的局限性，jsonwebtoken可以帮助我们解决这个问题。
2. 在Express中，使用拦截器对每个接口请求添加校验逻辑。

本章节完整源代码地址：Github

}