cookie的工作原理

一、实验介绍

1.1 实验内容

对初学者来说，cookie 与 session 的工作机制是较难理解的知识点。express 框架提供了 session 中间件 express-session，实现了session 机制。本课学习开发一个接口与 express-session 相同的中间件 exSession，从而透彻理解 session 的工作机制。

本实验首先介绍 cookie 的相关理论知识，然后通过具体的例子，观察浏览器的发出的 http 头部信息和接收的 http 头部信息，从而理解 cookie 的工作原理。

1.2 实验知识点

- 理解 http 协议头部中关于 cookie 的设置
- 使用 node.js 的原生接口访问 cookie
- 使用 express 框架的 cookie-parser 中间件访问 cookie

1.3 实验环境

- linux 终端环境，版本 Ubuntu 14.04
- node.js，版本0.10
- express 框架，版本4.0
- express 框架的中间件

1.4 代码获取

可通过以下命令下载源码参考对比进行学习

```
$ wget http://labfile.oss.aliyuncs.com/courses/901/node-session-examples.tgz
```

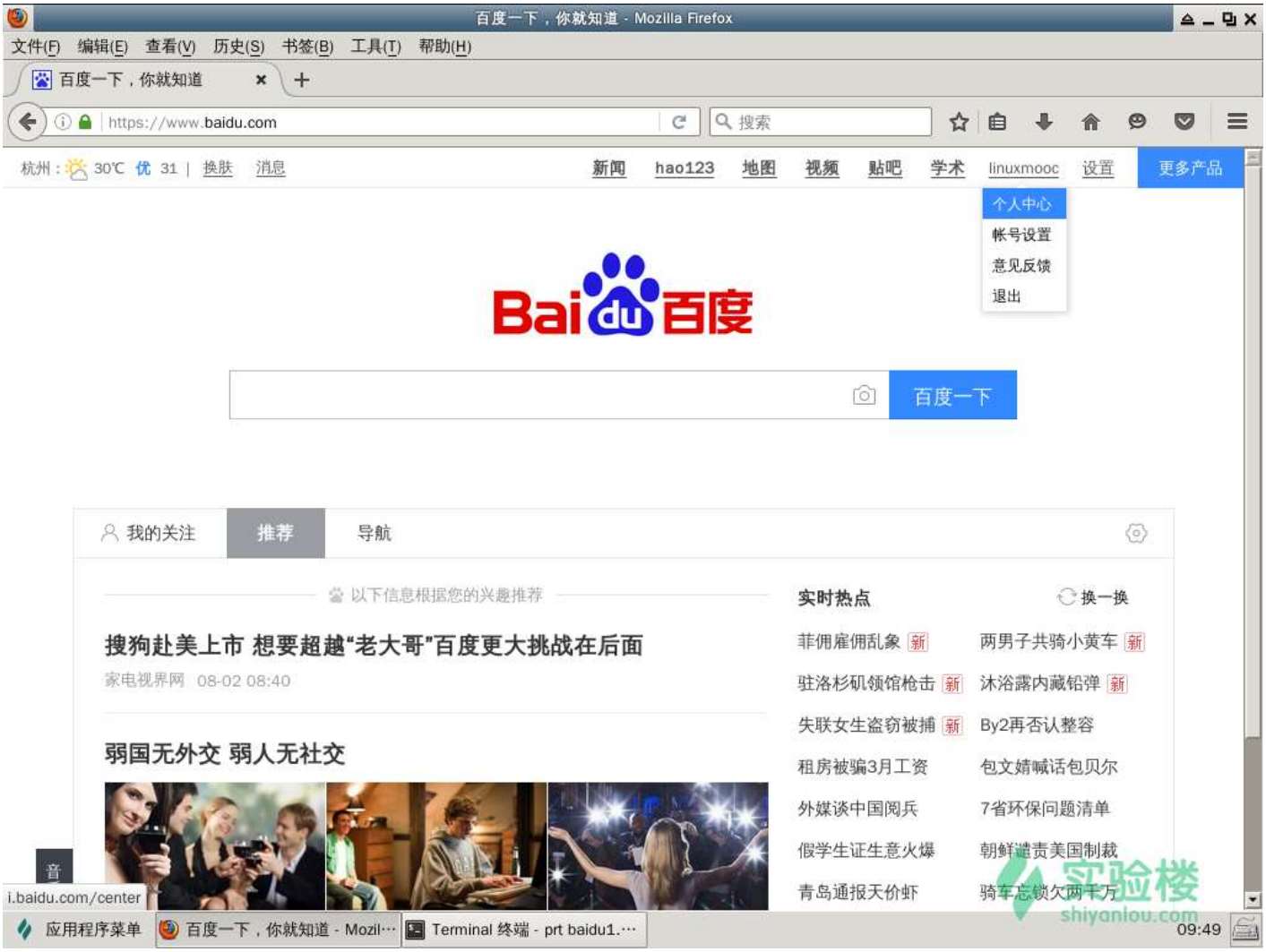
二、cookie 的工作原理

2.1 cookie 简介

Cookie 技术产生源于 HTTP 协议在互联网上的急速发展。随着互联网的快速发展，人们需要更复杂的互联网交互活动，就必须同服务器保持活动状态。为了适应用户的需求，技术上推出了各种保持Web浏览状态的手段，其中就包括了Cookie技术。

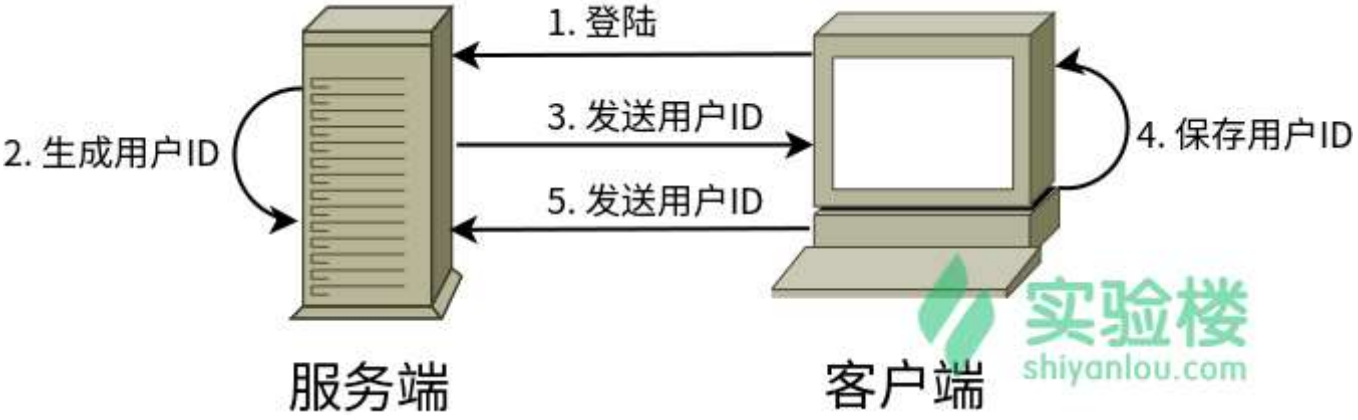
举个例子，用户 linuxmooc 在没有登陆 baidu 时，baidu 首页的界面如下图所示：

用户 linuxmooc 登陆 baidu 页面后，可以看到页面的右上角显示了他的名称 linuxmooc，如下图所示：



用户 linuxmooc 在没有登陆 baidu 时，访问的网页 url 是http://www.baidu.com；在登陆 baidu 后，访问的网页 url 仍然是http://www.baidu.com，但是这两次访问呈现的结果不相同，登陆前没有显示用户名，登陆后显示了用户名。问题是：服务端是如何区分这两次请求的？

服务端采用这样的机制区分这两次请求的，如下图所示：



1. 用户 linuxmooc 登陆 baidu
2. baidu 服务端会生成一个用户 ID
3. 然后，服务端将这个用户 ID 发送给浏览器
4. 浏览器收到这个用户 ID 后，会将这个用户ID保存在用户本地终端
5. 浏览器再次访问 baidu 站点时，浏览器会将保存在本地的用户ID再次发给 baidu 服务端

服务端收到浏览器发送的用户 ID 后，就知道此次请求来自于一个已经登陆的用户。在以上的交互过程中，保存在客户端的用户 ID 就被称为 cookie

2.2 cookie 定义

网站常常需要记录访问者的一些一些基本信息，例如如身份识别号码、密码、用户在 Web 站点购物的方式或用户访问该站点的次数。 网站为了辨别用户身份、进行 session 跟踪需要把数据储存在用户本地终端上，这些数据被称为 cookie。

> 资料来源 cookie 百度百科

2.3 服务端发送 cookie

通过设置 HTTP 的 Set-Cookie 消息头，Web 服务器可以指定存储一个 cookie。Set-Cookie 消息的格式如下面所示，括号中的部分都是可选的：

Set-Cookie:value [;expires=date][;domain=domain][;path=path][;secure]

消息头的第一部分，value 部分，通常是一个 name=value 格式的字符串。服务端向客户端发送的 HTTP 响应中设置 HTTP 的 Set-Cookie 消息头，一个具体的例子如下：

```
Connection:keep-alive
Content-Type:text/plain
Date:Fri, 14 Jul 2017 10:49:23 GMT
Set-Cookie:user=ZhangSan
Transfer-Encoding:chunked
```

在这个例子中，服务端向客户端发送的 Http 消息头中，设置了 'Set-Cookie:user=ZhangSan'，客户端浏览器将接受到字符串 'user=ZhangSan' 作为 cookie。

2.4 客户端发送 cookie

当一个 cookie 存在，并且条件允许的话，该 cookie 会在接下来的每个请求中被发送至服务器。cookie 的值被存储在名为 Cookie 的 HTTP 消息头中，并且只包含了 cookie 的值，其它的选项全部被去除。

客户端向服务端发送的 HTTP 请求中设置 Cookie 消息头，一个具体的例子如下：

```
Connection:keep-alive
Cookie:user=ZhangSan
Host:localhost:8080
User-Agent:Mozilla/5.0 AppleWebKit/537.36 Chrome Safari
```

在这个例子中，客户端向服务端发送的 Http 消息头中，设置了 'Cookie:user=ZhangSan'，服务端接受到字符串 'user=ZhangSan' 作为 cookie，从而确认此次请求对应的用户。

三 实验步骤

3.1 通过 node.js 的原生接口访问 cookie

3.1.1 功能说明

编写一个服务程序 ex0，该服务程序提供三个 url

- `localhost:8080/set` :设置 Set-Cookie 消息，在客户端中创建和保存 cookie
- `localhost:8080/show` :将客户端发送的 cookie 输出打印
- `localhost:8080/del` :设置 Set-Cookie 消息，将 cookie 从客户端中删除 为了演示最底层是如何访问 cookie，这个服务程序仅仅使用 node.js 的原生接口去访问 cookie 。

3.1.2 项目文件结构

```
ex0/
└── main.js
```

项目文件位于 ex0 目录下，只有一个源文件 main.js。

3.1.3 初始化项目

```
$ mkdir ex0
$ cd ex0
```

3.1.4 创建 main.js 的框架

```
var http = require('http');
http.createServer(function(req, res) {
    var url = req.url;

    if (url.startsWith('/set')) {
        // 设置Http的Set-Cookie消息头，将Cookie保存到客户端
    }

    if (url.startsWith('/show')) {
        // 将客户端发送的Cookie打印输出
    }

    if (url.startsWith('/del')) {
        // 设置Http的Set-Cookie消息头，将Cookie从客户端删除
    }
}).listen(8080);
```

3.1.5 实现 set 路由，在 main.js 中添加如下代码

```
    if (url.startsWith('/set')) {
        res.writeHead(200, {
            'Content-Type': 'text/plain',
            'Set-Cookie': 'user=ZhangSan'
```

```
        });
        res.write('Set-Cookie');
        res.end();
    }
```

- 在 res.writeHead 方法中
  - 设置状态码为200
  - 设置消息头 'Set-Cookie' 为 'user=ZhangSan'
  - 客户端收到 Set-Cookie 消息后，会将 'user=ZhangSan' 作为 cookie 保存在本地

3.1.6 实现 show 路由，在 main.js 中添加如下代码

```
    if (url.startsWith('/show')) {
        var cookie = req.headers.cookie;
        res.writeHead(200, {
            'Content-Type': 'text/plain'
        });
        res.write('Show-Cookie: ' + cookie);
        res.end();
    }
```

- 当客户端将 cookie 发送给服务端时，cookie 被 node.js 保存在 req.headers.cookie 字段中
- 服务端将客户端发送的 cookie 通过 res.write 方法回显给客户端

3.1.7 实现 del 路由，在 main.js 中添加如下代码

```
    if (url.startsWith('/del')) {
        res.writeHead(200, {
            'Content-Type': 'text/plain',
            'Set-Cookie': 'user=; expires=Thu, 01 Jan 1970 00:00:00 GMT'
        });
        res.write('Del-Cookie');
        res.end();
    }
```

- 在 res.writeHead 方法中
  - 设置状态码为200
  - 设置消息头 'Set-Cookie' 为 'user=; expires=Thu, 01 Jan 1970 00:00:00 GMT'
- 设置 cookie 的过期时间为1970年1月1日(Thu, 01 Jan 1970 00:00:00 GMT)
  - 因为当前时间肯定超过了是1970年1月1日，表示 cookie 已经过期
  - 所以浏览器会将 cookie('user=ZhangSan')删除

3.1.8 测试程序 ex0

- 启动程序 node main.js
- 在浏览器中第一次输入http://localhost:8080/show
  - 浏览器中显示结果为 Show-Cookie: undefined
  - 初始时刻，浏览器没有保存对应 localhost 的 cookie，所以结果为 undefined

- 在浏览器中输入http://localhost:8080/set
  - 浏览器中显示结果为 Set-Cookie
  - 服务器已经设置 Http 的 Set-Cookie 消息头，设置 cookie 为 'user=ZhangSan'
  - 客户端收到 cookie 后，会将该 cookie 保存到本地
- 在浏览器中第二次输入http://localhost:8080/show
  - 浏览器中显示结果为 Show-Cookie: user=ZhangSan
  - 客户端将已经保存在本地的 cookie('user=ZhangSan') 发送给服务端
  - 服务端将收到的 cookie 回显给客户端
- 在浏览器中输入http://localhost:8080/del
  - 浏览器中显示结果为 Del-Cookie
  - 服务器已经设置 Http 的 Set-Cookie 消息头，设置 cookie 为超时
  - 客户端收到 Http 响应后，会将超时的 cookie 删除
- 在浏览器中第三次输入http://localhost:8080/show
  - 浏览器中显示结果为 Show-Cookie: undefined
  - 浏览器已经删除存对应 localhost 的 cookie，所以结果为 undefined

3.2 通过 cookie-parser 中间件访问 cookie

3.2.1 功能说明

编写一个服务程序 ex1，提供三个 url

- `localhost:8080/set` :设置 Set-Cookie 消息，在客户端中创建和保存 cookie
- `localhost:8080/show` :将客户端发送的 cookie 输出打印
- `localhost:8080/del` :设置 Set-Cookie 消息，将 cookie 从客户端中删除

该服务程序 ex1 的功能与 ex0 相同，ex1 使用 express 框架的 cookie-parser 中间件去访问 cookie。cookie-parser 能够分析 cookie 字符串，提取对应的 name 和 value，从而简化对 cookie 的访问。

3.2.2 项目文件结构

```
ex1
├── main.js
└── package.json
```

项目文件位于 ex1 目录下，源文件 main.js 提供程序的主要功能。

3.2.3 初始化项目，创建项目目录

```
$ mkdir ex1
$ cd ex1
```

3.2.4 安装所需的 node.js 模块：express 框架和 cookie-parser 中间件

```
$ npm init
$ npm install --save express
$ npm install --save cookie-parser
```

3.2.5 创建 main.js 的框架

```
var express = require('express');
var cookieParser = require('cookie-parser');
var app = express();
app.use(cookieParser());

app.get('/show', function(req, res) {
    // 将客户端发送的Cookie打印输出
});

app.get('/set', function(req, res) {
    // 设置Http的Set-Cookie消息头，将Cookie保存到客户端
});

app.get('/del', function(req, res) {
    // 设置Http的Set-Cookie消息头，将Cookie从客户端删除
});

app.listen(8080);
```

3.2.6 实现 set 路由，在 main.js 中添加如下代码

```
app.get('/set', function(req, res) {
    res.cookie('user', 'ZhangSan');
    res.send('<h1>Set-Cookie</h1>');
});
```

- 在 res.cookie 方法中，设置消息头 'Set-Cookie' 为 'user=ZhangSan'

3.2.7 实现 show 路由，在 main.js 中添加如下代码

```
app.get('/show', function(req, res) {
    var cookies = req.cookies;
    var text = JSON.stringify(cookies);
    res.send('<h1>Show-Cookie: ' + text + '</h1>');
});
```

- 当客户端将 cookie 发送给服务端时
- 经过 cookie-parser 中间的处理，cookie 被保存在 req.cookies 字段中
- 服务端将客户端发送的 cookie 通过 res.send 方法回显给客户端

3.2.8 实现 del 路由，在 main.js 中添加如下代码

```
app.get('/del', function(req, res) {
    res.clearCookie('user');
    res.send('<h1>Del-Cookie</h1>');
});
```

- 在 res.clearCookie 方法中，设置消息头 'Set-Cookie' 为 'user=; expires=Thu, 01 Jan 1970 00:00:00 GMT'
- 设置 cookie 的过期时间为1970年1月1日(Thu, 01 Jan 1970 00:00:00 GMT)
  - 因为当前时间肯定超过了是1970年1月1日，表示 cookie 已经过期
  - 所以浏览器会将 cookie('user=ZhangSan')删除

3.2.9 测试程序 ex1

- 启动程序 `node main.js`
- 在浏览器中第一次输入http://localhost:8080/show
  - 浏览器中显示结果为 Show-Cookie: {}
  - 初始时刻，浏览器没有保存对应 localhost 的 cookie，所以结果为 {}
- 在浏览器中输入http://localhost:8080/set
  - 浏览器中显示结果为 Set-Cookie
  - 服务器已经设置 Http 的 Set-Cookie 消息头，设置 cookie 为 'user=ZhangSan'
  - 客户端收到 cookie 后，会将该 cookie 保存到本地
- 在浏览器中第二次输入http://localhost:8080/show
  - 浏览器中显示结果为 Show-Cookie: user=ZhangSan
  - 客户端将已经保存在本地的 cookie('user=ZhangSan') 发送给服务端
  - 服务端将收到的 cookie 回显给客户端
- 在浏览器中输入http://localhost:8080/del
  - 浏览器中显示结果为 Del-Cookie
  - 服务器已经设置 Http 的 Set-Cookie 消息头，设置 cookie 为超时
  - 客户端收到 Http 响应后，会将超时的 cookie 删除
- 在浏览器中第三次输入http://localhost:8080/show
  - 浏览器中显示结果为 Show-Cookie: {}
  - 浏览器已经删除存对应 localhost 的 cookie，所以结果为 {}