9 Cookie-файлы и сеансы

В этой главе вы узнаете, как пользоваться cookie-файлами и сеансами, чтобы улучшить взаимодействие с пользователями путем запоминания их предпочтений при переходе с одной страницы на другую или между сеансами браузера.

HTTP — протокол *без сохранения состояния*. Это означает, что, когда вы загружаете страницу в своем браузере и затем переходите на другую страницу того же сайта, ни у сервера, ни у браузера нет никакой внутренней возможности знать, что это тот же браузер посещает тот же сайт. Другими словами, Интернет работает таким образом, что *каждый HTTP-запрос содержит всю информацию, необходимую серверу для удовлетворения этого запроса*.

Однако есть проблема: если бы на этом все заканчивалось, мы никогда бы не смогли войти ни на один сайт. Потоковое видео не работало бы. Сайты забывали бы ваши настройки при переходе с одной страницы на другую. Так что обязан существовать способ формирования состояния поверх HTTP, и тут в кадре появляются cookie-файлы и сеансы.

Cookie-файлы, к сожалению, заработали дурную славу из-за тех неблаговидных целей, для которых использовались. Это печально, поскольку cookie-файлы на самом деле очень важны для функционирования современного Интернета (хотя HTML5 предоставил некоторые новые возможности, например локальные хранилища, которые можно использовать для тех же целей).

Идея cookie-файла проста: сервер отправляет фрагмент информации, который браузер хранит на протяжении настраиваемого промежутка времени. Содержание этого фрагмента информации полностью зависит от сервера. Часто это просто уникальный идентификационный номер (ID), соответствующий конкретному браузеру, так что поддерживается определенная имитация сохранения состояния.

Есть несколько важных фактов о cookie-файлах, которые вы должны знать.

- *Cookie-файлы не тайна для пользователя*. Все cookie-файлы, отправляемые сервером клиенту, доступны последнему для просмотра. Нет причин, по которым вы не могли бы отправить что-либо в зашифрованном виде для защиты содержимого, но необходимость в этом возникает редко (по крайней мере если вы не занимаетесь чем-то неблаговидным!). Подписанные cookie-файлы, о которых

мы немного поговорим далее, могут обфусцировать (то есть затруднить понимание) содержимое cookie-файла, но это не даст никакой криптографической защиты от любопытных глаз.

- *Пользователь может удалить или запретить cookie-файлы.* Пользователи полностью контролируют cookie-файлы, а браузеры позволяют удалять их скопом или по отдельности. Но, если вы не замышляете что-то нехорошее, у вас нет причин делать это, однако такая возможность удобна при тестировании. Пользователи также могут запретить cookie-файлы, что создает больше проблем, поскольку лишь простейшие веб-приложения могут обходиться без cookie-файлов.
- *Стандартные cookie-файлы могут быть подделаны.* Всякий раз, когда браузер выполняет запрос к вашему серверу со связанным cookie-файлом и вы слепо доверяете содержимому этого cookie-файла, вы превращаетесь в мишень для атаки. Верхом безрассудства было бы, например, выполнять содержащийся в cookie-файле код. Используйте подписанные cookie-файлы, чтобы наверняка знать, что они не подделаны.
- *Cookie-файлы могут использоваться для атак.* В последние годы появилась категория атак, называемых межсайтовым скриптингом (cross-site scripting, XSS). Один из методов XSS включает зловредный JavaScript, меняющий содержимое cookie-файлов. Это еще одна причина не доверять содержимому возвращаемых вашему серверу cookie-файлов. В этой ситуации помогает использование подписанных cookie-файлов (вмешательство будет заметно в подписанном cookie-файле, и неважно, кто его изменил: пользователь или зловредный JavaScript), кроме того, есть настройка, указывающая, что cookie-файлы должен изменять только сервер. Такие cookie-файлы, возможно, годятся не для всех случаев, но они, безусловно, безопаснее.
- *Пользователи заметят, если вы станете злоупотреблять cookie-файлами.* Пользователей будет раздражать, если вы станете устанавливать множество cookie-файлов на их компьютеры или хранить там массу данных. Лучше этого избегать. Старайтесь сводить использование cookie-файлов к минимуму.
- *Сеансы предпочтительнее cookie-файлов.* В большинстве случаев для сохранения состояния вы можете использовать *сеансы*. Это не только разумно, но и удобно, так как вам не нужно беспокоиться о возможном неправильном использовании хранилищ ваших пользователей, а также безопасно. Конечно, сеансы используют cookie-файлы, но в этом случае всю грязную работу за вас выполнит Express.

Cookie-файлы не магия: когда сервер хочет сохранить на клиенте cookie-файл, он отправляет заголовок Set-Cookie, содержащий пары «имя/значение»; а когда клиент отправляет запрос серверу, от которого он получил cookie-файлы, он отправляет многочисленные заголовки запроса Cookie, содержащие значения cookie-файлов.

Внешнее хранение данных доступа

Для безопасности использования cookie-файлов необходим так называемый *секрет cookie*. Секрет cookie-файла представляет собой строку, известную серверу и используемую для шифрования защищенных cookie-файлов перед их отправкой клиенту. Это не пароль, который необходимо помнить, так что он может быть просто случайной строкой. Для генерации секрета cookie я обычно использую случайное число или генератор случайных паролей (http://bit.ly/2QcjuDb), на создание которого вдохновил комикс xkcd.

Внешнее хранение данных доступа к сторонним ресурсам, таких как секрет cookie-файла, пароли к базам данных и токены для доступа к API (Twitter, Facebook и т. п.), является распространенной практикой. Это не только облегчает сопровождение (путем упрощения нахождения и обновления данных доступа), но и позволяет исключить файл с данными доступа из системы контроля версий, что особенно критично для репозиториев с открытым исходных кодом, размещаемых в GitHub или других общедоступных репозиториях исходного кода.

Для этого мы будем хранить данные доступа в файле JSON. Создайте файл `.credentials.development.json`:

```
{
  "cookieSecret": "...здесь находится ваш секрет cookie-файла"
}
```

Это файл данных доступа, которым мы будем пользоваться в процессе разработки. Таким образом, вы можете использовать разные данные доступа для разных окружений: окружения конечных пользователей (production), тестирования (test) или других, что очень удобно.

Чтобы упростить управление зависимостями при росте нашего приложения, добавим слой абстракций над файлом данных доступа. Наша версия будет очень простой. Создайте файл `config.js`:

```
const env = process.env.NODE_ENV || 'development'
const credentials = require(`./.credentials.${env}`)
module.exports = { credentials }
```

Теперь, чтобы случайно не добавить этот файл в наш репозиторий, внесите `credentials.js` в ваш файл `.gitignore`:

```
const { credentials } = require('./config')
```

В дальнейшем мы будем использовать этот же файл для хранения других данных доступа, но пока нам достаточно лишь нашего секрета cookie-файла.

Если вы используете прилагаемый к книге репозиторий, вам придется создать собственный файл данных доступа, так как он не включен в репозиторий.

Cookie-файлы в Express

Прежде чем устанавливать в своем приложении cookie-файлы и обращаться к ним, вам необходимо подключить промежуточное ПО `cookie-parser`. Сначала выполните `npm install cookie-parser`, затем (`ch09/meadowlark.js` в прилагаемом репозитории):

```
const cookieParser = require('cookie-parser')
app.use(cookieParser(credentials.cookieSecret))
```

Как только вы это сделали, можете устанавливать cookie-файлы или подписанные cookie-файлы везде, где у вас есть доступ к объекту ответа:

```
res.cookie('monster', 'нам-нам')
res.cookie('signed_monster', 'нам-нам', { signed: true })
```

Подписанные cookie-файлы имеют приоритет перед неподписанными. Если вы назовете ваш подписанный cookie-файл signed_monster, у вас не может быть неподписанного cookie-файла с таким же названием (он вернется как undefined).

Чтобы извлечь значение cookie-файла (если оно есть), отправленного с клиента, просто обратитесь к свойствам `cookie` или `signedCookie` объекта запроса:

```
const monster = req.cookies.monster
const signedMonster = req.signedCookies.signed_monster
```

Вы можете использовать в качестве имени cookie любую строку, какую пожелаете. Например, мы могли применить 'signed monster' вместо 'signed_monster', но тогда пришлось бы использовать скобочную нотацию для извлечения cookie-файла: req.signedCookies['signed monster']. По этой причине я рекомендую применять имена cookie-файлов без специальных символов.

Для удаления cookie-файла используйте `req.clearCookie`:

```
res.clearCookie('monster')
```

При установке cookie-файла вы можете указать следующие опции:

- `domain` — управляет доменами, с которыми связан cookie-файл, что позволяет привязывать cookie-файлы к конкретным поддоменам. Обратите внимание, что вы не можете установить cookie-файл для домена, отличного от того, на котором работает ваш сервер: он просто не будет выполнять какие-либо действия;
- `path` — управляет путем, на который распространяется действие данного cookie-файла. Обратите внимание, что в путях предполагается неявный символ

подстановки (wildcard) в конце: если вы используете путь `/` (по умолчанию), он будет распространяться на все страницы вашего сайта. Если используете путь `/foo`, он будет распространяться на пути `/foo`, `/foo/bar` и т. д.;

- `maxAge` — определяет, сколько времени (в миллисекундах) клиент должен хранить cookie-файл до его удаления. Если вы опустите эту опцию, cookie-файл будет удален при закрытии браузера (можете также указать дату окончания срока действия cookie-файла с помощью опции `expires`, но синтаксис при этом удручающий. Я рекомендую использовать `maxAge`);
- `secure` — указывает, что данный cookie-файл будет отправляться только через защищенное (HTTPS) соединение;
- `httpOnly` — установка этому параметру значения `true` указывает, что cookie-файл будет изменяться только сервером. То есть JavaScript на стороне клиента не может его изменять. Это помогает предотвращать XSS-атаки;
- `signed` — установите значение `true`, чтобы подписать данный cookie-файл, делая его доступным в `res.signedCookies` вместо `res.cookies`. Поддельные подписанные cookie-файлы будут отклонены сервером, а значение cookie-файла возвращено к первоначальному значению.

Просмотр cookie-файлов

Вероятно, в рамках тестирования вам понадобится способ просматривать cookie-файлы в вашей системе. У большинства браузеров имеется возможность просматривать cookie-файлы и хранимые ими значения. Например, в Chrome откройте инструменты разработчика и выберите вкладку Application (Приложение). В дереве слева вы увидите пункт Cookies. Разверните его — и увидите в списке сайт, который просматриваете в текущий момент. Нажмите на него — перед вами появятся все связанные с этим сайтом cookie-файлы. Вы можете также щелкнуть правой кнопкой мыши на домене для очистки всех cookie-файлов или на отдельном cookie-файле для его удаления.

Сеансы

Сеансы — всего лишь более удобный инструмент для сохранения состояния. Для реализации сеансов необходимо *что-нибудь* хранить на клиенте, в противном случае сервер не сможет распознать, что следующий запрос выполняет тот же клиент. Обычный способ для этого — cookie-файл, содержащий уникальный идентификатор. Сервер будет использовать этот идентификатор для извлечения информации о соответствующем сеансе.

Cookie-файлы — не единственный способ достижения этой цели. Во время пика паники по поводу cookie-файлов, когда процветало злоупотребление ими, многие пользователи просто отключали cookie-файлы. Тогда были изобретены другие методы сохранения состояния, такие как добавление к URL сеансовой информации. Эти методики оказались сложными, неэффективными, выглядели неряшливо, и лучше пусть они остаются в прошлом. HTML5 предоставляет другую возможность для сеансов — *локальное хранилище*, которое имеет преимущество перед cookie-файлами, если вам нужно хранить большое количество данных. За более подробной информацией обращайтесь к документации MDN для `Window.localStorage` (https://mzl.la/2CDrGo4).

Вообще, существует два способа реализации сеансов: хранить все в cookie-файле или хранить в cookie-файле только уникальный идентификатор, а все остальное — на сервере. Первый способ называется «сеансы на основе cookie-файлов» и является не самым удачным вариантом использования cookie. Как бы то ни было, он означает хранение всего вносимого вами в сеанс в браузере клиента, а такой подход я никак не могу рекомендовать. Я посоветовал бы этот подход, только если вы собираетесь хранить лишь небольшой фрагмент информации, не возражаете против доступа к нему пользователя и уверены, что с течением времени такая система не выйдет из-под контроля. Если вы хотите использовать этот подход, взгляните на промежуточное ПО `cookie-session` (http://bit.ly/2qNv9h6).

Хранилища в памяти

Если вы склоняетесь к хранению сеансовой информации на сервере, что я и советую делать, вам потребуется место для хранения. Простейший вариант — сеансы в памяти. Их легко настраивать, однако у них есть колоссальный недостаток: при перезагрузке сервера (а во время работы с данной книгой вы будете делать это многократно!) ваша сеансовая информация пропадет. Хуже того, при масштабировании до нескольких серверов (см. главу 12) обслуживать запрос каждый раз может другой сервер: сеансовая информация иногда будет в наличии, а иногда — нет. Очевидно, что это совершенно неприемлемо при реальной эксплуатации, однако вполне достаточно для нужд разработки и тестирования. Мы узнаем, как организовать постоянное хранение сеансовой информации, в главе 13.

Сначала установим `express-session` (`npm install express-session`), затем, после подключения парсера cookie-файлов, подключим `express-session` (`ch09/meadowalrk.js` в репозитории, прилагаемом к книге):

```
const expressSession = require('express-session')
// Убедитесь, что вы подключили
// промежуточное ПО cookie
// до промежуточного ПО session!
app.use(expressSession({
```

```
  resave: false,
  saveUninitialized: false,
  secret: credentials.cookieSecret,
}))
```

Промежуточное ПО `express-session` принимает конфигурационный объект со следующими опциями.

- `resave` — заставляет сеанс заново сохраняться в хранилище, даже если запрос не менялся. Предпочтительнее устанавливать этот параметр в `false` (для получения дополнительной информации см. документацию по `express-session`).
- `saveUninitialized` — установка этого параметра в `true` приводит к сохранению новых (неинициализированных) сеансов в хранилище, даже если они не менялись. Предпочтительнее (и даже необходимо, если вам требуется получить разрешение пользователя перед установкой cookie-файла) устанавливать этот параметр в `false` (см. документацию по `express-session` для получения дополнительной информации).
- `secret` — ключ (или ключи), используемый для подписания cookie-файла идентификатора сеанса. Может быть тем же ключом, что и применяемый для `cookie-parser`.
- `key` — имя cookie-файла, в котором будет храниться уникальный идентификатор сеанса. По умолчанию `connect.sid`.
- `store` — экземпляр сеансового хранилища. По умолчанию это экземпляр `MemoryStore`, что вполне подходит для наших текущих целей. В главе 13 мы рассмотрим, как использовать в качестве хранилища базу данных.
- `cookie` — настройки для cookie-файла сеанса (`path`, `domain`, `secure` и т. д.). Применяются стандартные значения по умолчанию для cookie-файлов.

Использование сеансов

Как только вы настроите сеансы, их использование станет элементарным: просто воспользуйтесь свойствами переменной `session` объекта запроса:

```
req.session.userName = 'Anonymous'
const colorScheme = req.session.colorScheme || 'dark'
```

Обратите внимание, что при работе с сеансами нам не нужно использовать объект запроса для чтения значения и объект ответа для установки значения — все это выполняется через объект запроса (у объекта ответа нет свойства `session`). Чтобы удалить сеанс, можно использовать оператор JavaScript `delete`:

```
req.session.userName = null // Этот оператор устанавливает
                            // 'userName' в значение null, но не удаляет.

delete req.session.colorScheme // А этот удаляет 'colorScheme'.
```

Использование сеансов для реализации уведомлений

Уведомления — просто средство обеспечения обратной связи с пользователями таким способом, который не мешал бы их навигации. Простейший метод реализации уведомлений — использование сеансов (можно также использовать строку запроса, но при этом возникнут более уродливые URL, а уведомления начнут попадать в закладки браузера, чего, вероятно, вам не хотелось бы). Сначала настроим HTML. Мы будем использовать уведомления (alert) Bootstrap, так что убедитесь, что Bootstrap у вас подключен (см. документацию по основам Bootstrap: http://bit.ly/36YxeYf). Вы можете подключить файлы Bootstrap CSS и JavaScript в вашем основном шаблоне, пример этого есть в репозитории, прилагаемом к книге. В файле шаблона, где-нибудь в заметном месте (обычно непосредственно после шапки сайта), поместите следующее:

```
{{#if flash}}
  <div class="alert alert-dismissible alert-{{flash.type}}">
    <button type="button" class="close"
      data-dismiss="alert" aria-hidden="true">&times;<button>
    <strong>{{flash.intro}}</strong> {{{flash.message}}}
  </div>
{{/if}}
```

Обратите внимание, что мы используем три фигурные скобки для `flash.message` — это позволит использовать простой HTML в наших сообщениях (возможно, нам захочется выделить какие-то слова или включить гиперссылки). Теперь подключим необходимое промежуточное ПО для добавления в контекст объекта `flash`, если таковой имеется в сеансе. Сразу после однократного отображения уведомления желательно удалить его из сеанса, чтобы оно не отображалось при следующем запросе. Мы создадим промежуточное ПО, которое будет проверять сеанс на наличие уведомлений и, если они есть, передавать их в объект `res.locals`, делая доступными для представлений. Мы поместим это промежуточное ПО в файл `lib/middleware/flash.js`:

```
module.exports = (req, res, next) => {
  // Если имеется уведомление,
  // переместим его в контекст, а затем удалим.
  res.locals.flash = req.session.flash
  delete req.session.flash
  next()
})
```

В файле `meadowalrk.js` до каких-либо маршрутов представления подключим промежуточное ПО уведомлений:

```
const flashMiddleware = require('./lib/middleware/flash')
app.use(flashMiddleware)
```

Теперь посмотрим, как использовать уведомления в реальности. Представьте, что мы подписываем пользователей на новостную рассылку и хотим перенаправить их в архив рассылки после того, как они подпишутся. Обработчик форм при этом мог бы выглядеть примерно так:

```
// Немного измененная версия официального регулярного выражения
// W3C HTML5 для электронной почты:
// https://html.spec.whatwg.org/multipage/forms.htmlvalid-e-mail-address.
const VALID_EMAIL_REGEX = new RegExp('^[a-zA-Z0-9.!#$%&\'*+\/=?^_`{|}~-]+@' +
  '[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?' +
  '(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)+$');

app.post('/newsletter', function(req, res){
  const name = req.body.name || '', email = req.body.email || ''
  // Проверка вводимых данных
  if(VALID_EMAIL_REGEX.test(email)) {
    req.session.flash = {
      type: 'danger',
      intro: 'Ошибка проверки!',
      message: 'Введенный вами адрес электронной почты
      некорректен.',
    }
    return res.redirect(303, '/newsletter/archive');
  }
  // NewsletterSignup – пример объекта, который вы могли бы
  // создать; поскольку все реализации различаются, оставляю
  // написание этих зависящих от конкретного проекта интерфейсов
  // на ваше усмотрение. Это просто демонстрация того, как типичная
  // реализация на основе Express может выглядеть в вашем проекте.
  new NewsletterSignup({ name, email }).save((err) => {
    if(err) {
      req.session.flash = {
        type: 'danger',
        intro: 'Ошибка базы данных!',
        message: 'Произошла ошибка базы данных.
        Пожалуйста, попробуйте позже',
      }
      return res.redirect(303, '/newsletter/archive')
    }
    req.session.flash = {
      type: 'success',
      intro: 'Спасибо!',
      message: 'Вы были подписаны на информационный бюллетень.',
    };
    return res.redirect(303, '/newsletter/archive')
  })
})
```

Обратите внимание, что мы отличаем ошибки проверки вводимых данных от ошибки базы данных. Запомните: если мы выполняем проверку вводимых данных

в клиентской части, ее также следует выполнять в серверной части, поскольку пользователи со злыми намерениями проверку в клиентской части могут обойти.

Здорово иметь реализованный механизм уведомлений на сайте, даже если он не подходит для всех случаев (например, уведомления не всегда подходят для мастеров с несколькими формами или потоками подсчета стоимости в корзине для виртуальных покупок). Кроме того, их очень удобно использовать во время разработки в качестве способа обеспечения обратной связи, даже если потом вы замените их на другой механизм. Добавление поддержки для уведомлений — то, что я делаю при настройке сайта в первую очередь, и мы будем использовать этот метод на протяжении всей книги.

Поскольку уведомление перемещается из сеанса в res.locals.flash в промежуточном ПО, вам придется выполнять перенаправление, чтобы оно отобразилось. Если вы хотите отображать уведомление без перенаправления, устанавливайте значение res.locals.flash вместо req.session.flash.

В примере в этой главе применяется браузерная отправка форм с перенаправлением, так как управление интерфейсом пользователя с помощью сеансов обычно не используется в приложениях с отправкой форм через Ajax. При этом событии (отправке формы) желательно указывать любые ошибки в объекте формата JSON, который возвращает обработчик формы, а клиентский код будет модифицировать DOM на сервере, чтобы сообщения об ошибках отображались динамически. Это не значит, что для приложений с рендерингом на стороне клиента сеансы бесполезны, но они редко используются для этих целей.

Для чего использовать сеансы

Сеансы удобны, когда вам необходимо сохранить предпочтения пользователя, относящиеся к нескольким страницам. Чаще всего сеансы используются для предоставления информации об аутентификации пользователя: вы входите в систему — и создается сеанс. После этого вам не нужно выполнять вход в систему всякий раз, когда вы перезагружаете страницу. Сеансы могут быть полезны даже без учетных записей пользователя. Для сайтов вполне обычно запоминать, какая сортировка вам нравится или какой формат представления даты вы предпочитаете, — и все это без необходимости входить в систему.

Несмотря на то что я советую отдавать предпочтение сеансам, а не cookie-файлам, важно понимать, как работают последние (в частности, потому, что они делают возможным функционирование сеансов). Это поможет вам в вопросах диагностики, а также облегчит процесс понимания механизмов безопасности и защиты персональной информации вашим приложением.

Резюме

Понимание cookie-файлов и сеансов помогает лучше разобраться в том, как веб-приложения поддерживают видимость сохранения состояния, когда базовый протокол HTTP этого не подразумевает. Мы познакомились с методиками работы с cookie-файлами и сеансами, позволяющими контролировать взаимодействие пользователя с вашим сайтом.

Кроме того, мы писали промежуточное ПО, не вдаваясь при этом в подробные пояснения. В следующей главе мы вплотную займемся промежуточным ПО и изучим все, что к нему относится!