

Работа с сетью

Высокий уровень

Юрий Литвинов
yurii.litvinov@gmail.com

11.10.2019г

Протоколы прикладного уровня

- ▶ Если вы пользуетесь “голыми” сокетами под .NET, скорее всего, вы делаете что-то не так
- ▶ TCP и UDP обеспечивают транспорт, вся полезная работа делается протоколами прикладного уровня
- ▶ Их тысячи:
 - ▶ DNS
 - ▶ Электронная почта: SMTP, IMAP, POP3
 - ▶ Различные виды удалённого вызова: WCF, Apache Thrift, gRPC, ...
 - ▶ WWW: HTTP, HTTPS
 - ▶ Стриминг: RTP, RTCP
 - ▶ P2P и доставка контента: BitTorrent
 - ▶ ...

Как работает браузер

1. Определяет URL, указывающий на желаемую страницу
2. Выполняет DNS-запрос на доменное имя из URL, узнаёт IP
3. Устанавливает TCP-соединение с портом 80 целевой машины
4. Отправляет HTTP-запрос на получение файла
5. Получает ответ от сервера с HTML-страницей
6. Если страница содержит URL, необходимые для её отображения, браузер повторяет процесс
 - ▶ Картинки, скрипты, стили и т.д.
7. Браузер отображает страницу, отдаёт скрипты на интерпретацию, при необходимости запускает плагины
8. Если новых запросов некоторое время не поступает, браузер разрывает соединение с сервером

Протокол HTTP

- ▶ Простой текстовый протокол поверх TCP
- ▶ Запрос-ответ
- ▶ Вид запроса:
 - <Метод> параметр <версия протокола>
 - <Заголовки>
 - <Тело запроса>
 - <Пустая строка>

Пример

Слегка сокращённый

GET http://se.math.spbu.ru/SE HTTP/1.1

Host: se.math.spbu.ru

Connection: keep-alive

Upgrade-Insecure-Requests: 1

User-Agent: Chrome/68.0.3440.106

Accept: text/html,application/xhtml+xml

Referer: http://se.math.spbu.ru/

Accept-Encoding: gzip, deflate

Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7

Ответ

Очень сокращённый

HTTP/1.1 200 OK

Server: Zope/(Zope 2.10.6-final, python 2.4.6, linux2) ZServer/1.1 Plone/3.1.3

Date: Sat, 01 Sep 2018 12:57:32 GMT

Content-Length: 29068

Content-Language: ru

Expires: Sat, 1 Jan 2000 00:00:00 GMT

Content-Type: text/html; charset=utf-8

```
<!DOCTYPE html ...>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ru"
```

```
  lang="ru">
```

```
  <head>
```

```
  </head>
```

```
  <body>
```

```
  </body>
```

```
</html>
```

Виды запросов

Методы

- ▶ **GET** — получить страницу
- ▶ **HEAD** — получить только заголовок
- ▶ **PUT** — залить новую страницу на сервер
- ▶ **POST** — добавить что-нибудь к странице
- ▶ **DELETE** — удалить страницу
- ▶ **TRACE** — отправить запрос обратно (для отладки)
- ▶ **CONNECT** — подключиться через прокси
- ▶ **OPTIONS** — узнать, что можно использовать

Коды ответов

Код	Значение	Примеры
1xx	Информация	100 — сервер согласен обрабатывать запросы
2xx	Успех	200 — запрос успешно обработан, 204 — ответ пустой
3xx	Перенаправление	301 — страница перемещена, 304 — возьмите из своего кэша
4xx	Ошибка клиента	404 — страница не найдена, 403 — нет прав
5xx	Ошибка сервера	500 — сервер упал, 503 — попробуйте позже

Как это работает в .NET

HttpClient

class Program

```
{  
    private static async Task Main(string[] args)  
    {  
        var httpClient = new HttpClient();  
  
        var response = await httpClient.GetAsync("http://hwproj.me/");  
        if (response.IsSuccessStatusCode)  
        {  
            var content = await response.Content.ReadAsStringAsync();  
            Console.WriteLine(content);  
        }  
    }  
}
```

Exponential backoff

```
static async Task<string> DownloadStringWithRetries(string uri)
{
    using (var client = new HttpClient()) {
        var delay = TimeSpan.FromSeconds(1);
        for (int i = 0; i != 3; ++i) {
            try {
                return await client.GetStringAsync(uri);
            } catch {
            }
            await Task.Delay(delay);
            delay *= 2;
        }

        // Попробуем последний раз, дав ошибке распространиться
        return await client.GetStringAsync(uri);
    }
}
```

Зачем всё это?

- ▶ Писать свой браузер можно, но не нужно
- ▶ HTTP — основа для протоколов общения “приложение-приложение”
- ▶ Веб-сервисы — страницы в интернете, предназначенные не для браузеров, а для других приложений
 - ▶ Интерфейс для облачных сервисов типа GMail, Google Drive, ВКонтакте, Twitter и т.д.
 - ▶ Браузерные клиенты часто пользуются тем же API, что доступен сторонним разработчикам
 - ▶ Публичные API для приложений (например, бэкапы смартфонов)
 - ▶ Распределённые приложения в одной корпоративной сети
 - ▶ Сервер базы данных, сервер бизнес-логики, сервера вспомогательных служб со своими API

Как оно работает

- ▶ Удалённый вызов — клиент посылает HTTP-запрос с именем метода и параметрами, сервер исполняет запрос и отправляет ответ обратно
- ▶ Требуется сериализация: XML, JSON, protobuf, ...
- ▶ Требуется механизм общения сервера и клиента:
 - ▶ SOAP — старый, громоздкий, но до сих пор очень популярный протокол, использует XML
 - ▶ WCF — библиотека для разработки веб-сервисов под .NET, несколько устарела, но до сих пор очень популярна, может использовать SOAP
 - ▶ REST — легковесный протокол общения, очень-очень популярен
- ▶ Машиночитаемое описание возможностей веб-сервиса

Representational State Transfer

REST

- ▶ “Легковесный” интерфейс для веб-сервисов, построенный на HTTP-запросах
- ▶ Запрос и его параметры передаются в основном через URL
 - ▶ Иногда используется тело HTTP-запроса с JSON или бинарными данными
- ▶ Сервер не хранит состояние сессии, запросы всё таскают с собой
 - ▶ Удобно, запрос может прийти откуда угодно и когда угодно, лишь бы он был правильный

Интерфейс сервиса

- ▶ Коллекции
 - ▶ `http://api.example.com/resources/`
- ▶ Элементы
 - ▶ `http://api.example.com/resources/item/17`
- ▶ HTTP-методы
 - ▶ GET
 - ▶ PUT
 - ▶ POST
 - ▶ DELETE
- ▶ Передача параметров прямо в URL
 - ▶ `http://api.example.com/resources?user=me&access_token=ASFQF`

Пример, Google Drive REST API

- ▶ GET <https://www.googleapis.com/drive/v2/files> — список всех файлов
- ▶ GET <https://www.googleapis.com/drive/v2/files/fileId> — метаданные файла по его Id
- ▶ POST <https://www.googleapis.com/upload/drive/v2/files> — загрузить новый файл
- ▶ PUT <https://www.googleapis.com/upload/drive/v2/files/fileId> — обновить файл
- ▶ DELETE <https://www.googleapis.com/drive/v2/files/fileId> — удалить файл

REST и HttpClient

Сырой REST

```
private static async Task Main(string[] args)
{
    var httpClient = new System.Net.Http.HttpClient();

    var request = "https://api.exchangeratesapi.io/latest?base=USD&symbols=RUB";
    var response = await httpClient.GetAsync(request);
    if (response.IsSuccessStatusCode)
    {
        var content = await response.Content.ReadAsStringAsync();
        var data = JsonConvert.DeserializeObject<JObject>(content);
        var baseCurrency = data["base"];
        var ruble = data.Value<JToken>("rates").Values<JProperty>().First();
        Console.WriteLine($"1 {baseCurrency} = {ruble.Value} {ruble.Name}");
    }
}
```


REST и клиентские библиотеки

Google Drive API

class Program

```
{
    private static readonly string[] Scopes = { DriveService.Scope.DriveReadonly };
    private const string ApplicationName = "GDriveDemo";

    static async Task Main(string[] args) {
        UserCredential credential;

        using (var stream =
            new FileStream("credentials.json", FileMode.Open, FileAccess.Read)) {
            var credPath = "token.json";
            credential = await GoogleWebAuthorizationBroker.AuthorizeAsync(
                GoogleClientSecrets.Load(stream).Secrets,
                Scopes,
                "user",
                CancellationToken.None,
                new FileDataStore(credPath, true));
            Console.WriteLine("Credential file saved to: " + credPath);
        }
        ...
    }
}
```

Google Drive API (2)

```
class Program
```

```
{
```

```
...
```

```
static async Task Main(string[] args) {
```

```
    UserCredential credential;
```

```
...
```

```
var service = new DriveService(new BaseClientService.Initializer() {
```

```
    HttpClientInitializer = credential,
```

```
    ApplicationName = ApplicationName,
```

```
});
```

```
FilesResource.ListRequest listRequest = service.Files.List();
```

```
listRequest.PageSize = 10;
```

```
listRequest.Fields = "nextPageToken, files(id, name)";
```

```
...
```

```
}
```

```
}
```

Google Drive API (3)

```
class Program
```

```
{
```

```
....
```

```
static async Task Main(string[] args) {
```

```
    FilesResource.ListRequest listRequest = service.Files.List();
```

```
    ...
```

```
    IList<Google.Apis.Drive.v3.Data.File> files = listRequest.Execute().Files;
```

```
    Console.WriteLine("Files:");
```

```
    if (files != null && files.Count > 0) {
```

```
        foreach (var file in files) {
```

```
            Console.WriteLine($"{file.Name} ({file.Id})");
```

```
        }
```

```
    } else {
```

```
        Console.WriteLine("No files found.");
```

```
    }
```

```
    Console.Read();
```

```
}
```

```
}
```

Как это отлаживать

Fiddler

The screenshot displays the Fiddler Web Debugger interface. The top menu includes File, Edit, Rules, Tools, View, and Help. Below the menu is a toolbar with various icons for WinConfig, Replay, Go, Stream, Decode, and other functions. The main window is divided into two panes. The left pane shows a list of intercepted sessions with columns for #, Result, Protocol, Host, URL, Body, Caching, and Content-Type. The right pane shows the details of the selected session (session 2), including the raw HTTP request and response.

#	Result	Protocol	Host	URL	Body	Caching	Content-Type
1	200	HTTP	se.math.spbu.ru	/	217		text/html; charset=utf-8
2	200	HTTP	se.math.spbu.ru	/SE	29 068	Expires...	text/html; charset=utf-8
3	200	HTTP	clients4.google.com	/443	0		
4	304	HTTP	se.math.spbu.ru	/SE/newsitem_icon.gif	0		
5	304	HTTP	se.math.spbu.ru	/SE/spinner.gif	0		
6	304	HTTP	se.math.spbu.ru	/SE/bullet.gif	0		
7	304	HTTP	se.math.spbu.ru	/SE/link_icon.gif	0		
8	304	HTTP	se.math.spbu.ru	/SE/input_background.gif	0		
9	304	HTTP	se.math.spbu.ru	/SE/search_icon.gif	0		
10	200	HTTP	www.google-analytics.co...		0		
11	200	HTTP	Tunnel to	29.client-channel.google...	0		

The right pane shows the details of the selected session (session 2). The raw HTTP request is displayed as follows:

```
GET http://se.math.spbu.ru/SE HTTP/1.1
Host: se.math.spbu.ru
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3496.102 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: ru-RU;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: __ym_uid=14964115231040825938; __utmz=108296320.1508163081.17
```

The raw HTTP response is displayed as follows:

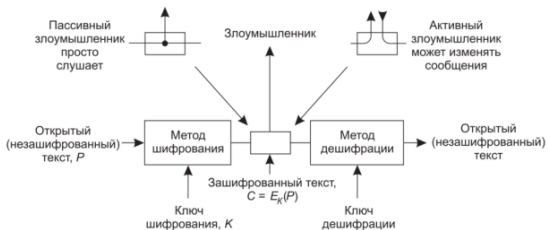
```
HTTP/1.1 200 OK
Server: Zope/(Zope 2.10.6-final, python 2.4.6, linux2) ZServer/1.0
Date: Mon, 03 Sep 2018 11:28:04 GMT
Content-Length: 29068
Content-Language: ru
Expires: Sat, 1 Jan 2000 00:00:00 GMT
Content-Type: text/html; charset=utf-8

<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ru" lang="ru">
```

Сетевая безопасность

- ▶ Почти все сервисы требуют авторизации и обеспечения безопасности
- ▶ Аутентификация — установление личности (точнее, идентичности) участника взаимодействия
 - ▶ Обычно взаимна
- ▶ Авторизация — установление прав на выполнение операции
- ▶ Шифрование — обеспечение конфиденциальности передаваемой информации
- ▶ Также важны:
 - ▶ Целостность — злоумышленник ничего не поменял
 - ▶ Актуальность — злоумышленник не проиграл старое сообщение

Шифрование



© Э. Таненбаум

- ▶ Алгоритм шифрования считается известным, секретен только ключ
- ▶ Усложнение алгоритма шифрования не всегда повышает криптостойкость

Шифрование с симметричным ключом

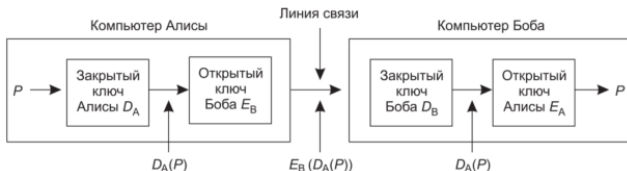
- ▶ Шифры:
 - ▶ Data Encryption Standard (DES, Triple DES)
 - ▶ Advanced Encryption Standard (AES)
- ▶ Режимы шифрования:
 - ▶ Electronic Code Book — быстро, надёжно, но не криптостойко
 - ▶ Cipher Block Chaining — более криптостоек, не устойчив к ошибкам передачи
 - ▶ Initialization Vector (IV)
 - ▶ Stream Cipher Mode — устойчив к ошибкам передачи, довольно быстр
 - ▶ Тоже есть IV, вычисляется keystream (можно заранее)
 - ▶ Counter Mode — для произвольного доступа к зашифрованным блокам

Шифрование с открытым ключом

Или почему нельзя отдать ключи от Telegram

- ▶ Алгоритм делится на две части, D и E , так, что $D(E(P)) = P$
- ▶ D очень сложно получить по E
 - ▶ Например, найти простые сомножители огромного числа или дискретный логарифм по заданному модулю
- ▶ D (ключ от D) держится в секрете, E выкладывается в открытый доступ
- ▶ Если Боб хочет послать Алисе сообщение, он берёт её открытый ключ E_A , шифрует им сообщение P и отправляет Алисе
- ▶ Алиса дешифрует сообщение, вычисляя $D_A(E_A(P))$
- ▶ У каждого пользователя своя пара ключей
- ▶ Алгоритмы: RSA, ElGamal, эллиптические шифры

Цифровые подписи



© Э. Таненбаум

- ▶ Надо, чтобы $D(E(P)) = P$ (это так для большинства криптосхем)
- ▶ Шифровать всё сообщение слишком медленно
- ▶ Message Digest-ы — хорошие хеши сообщений
 - ▶ MD5, SHA-1
 - ▶ MD5 уязвим для “атаки дней рождения”
- ▶ Подписывается только хеш, это почти так же криптостойко, но в сотни раз быстрее

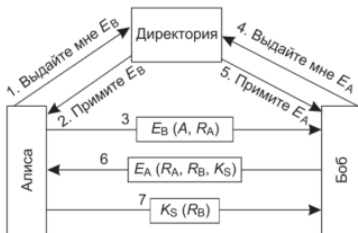
Сертификаты



© Э. Таненбаум

- ▶ Сертификат — сообщение, подтверждающее идентичность ключа, подписанное Certificate Authority (стандарт X.509)
- ▶ Цепочка сертификатов — СА верхнего уровня подписывает сертификаты СА уровнем ниже, чтобы они могли подписывать сертификаты пользователей
- ▶ Корневые сертификаты — сертификаты, которым принято доверять
- ▶ Самоподписанные сертификаты — не доверенные, используются для отладки

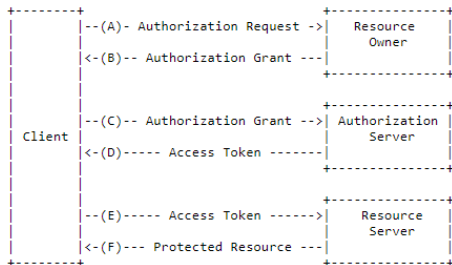
Аутентификация с открытым ключом



© Э. Таненбаум

- ▶ R_A — **nonce** (number used once), для предотвращения атаки повтором
- ▶ K_S — ключ сеанса для симметричного шифра

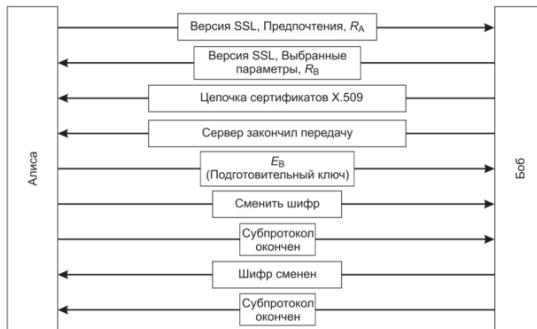
OAuth 2



© RFC 6749

- ▶ Позволяет разрешить пользование ресурсом, не раскрывая хозяину ресурса логин и пароль пользователя
 - ▶ Логин по аккаунту в Google или аккаунту в VK

HTTPS



© Э. Таненбаум

- ▶ SSL (Secure Sockets Layer)
- ▶ HTTPS — HTTP через SSL