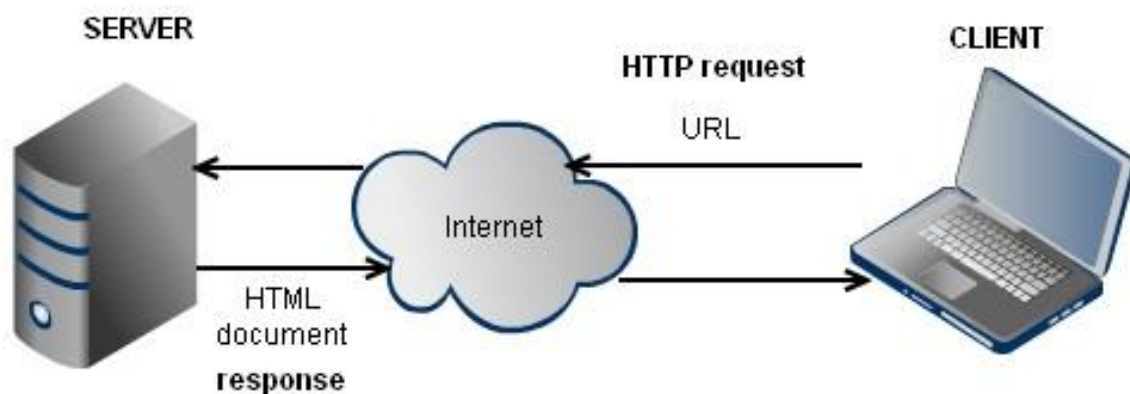


**Клиент-серверная
архитектура.
HTTP запросы.
REST**

Клиент-серверная архитектура

Клиент-серверная архитектура

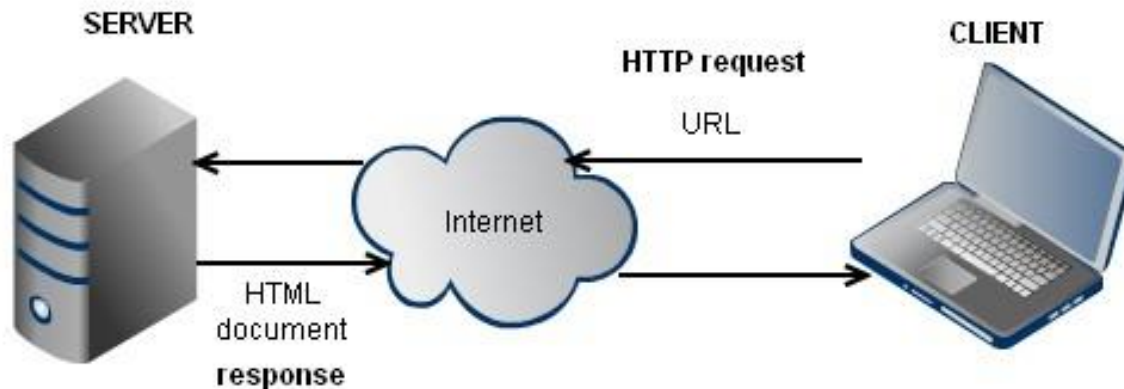
- Многие приложения строятся по этому принципу
- Есть две взаимодействующие стороны – **клиент** и **сервер**, которые взаимодействуют друг с другом по сети при помощи **протокола**, например, HTTP
- Сообщения протокола HTTP представляют из себя текст определенного формата. По сути, клиент и сервер обмениваются текстом



Клиентской программой является браузер

Клиент-серверная архитектура

- У сервера может быть много клиентов
- Сервер пассивен – он никогда не инициирует взаимодействие с клиентом
- Инициатором всегда выступает клиент – клиент что-то просит, а сервер ему отвечает

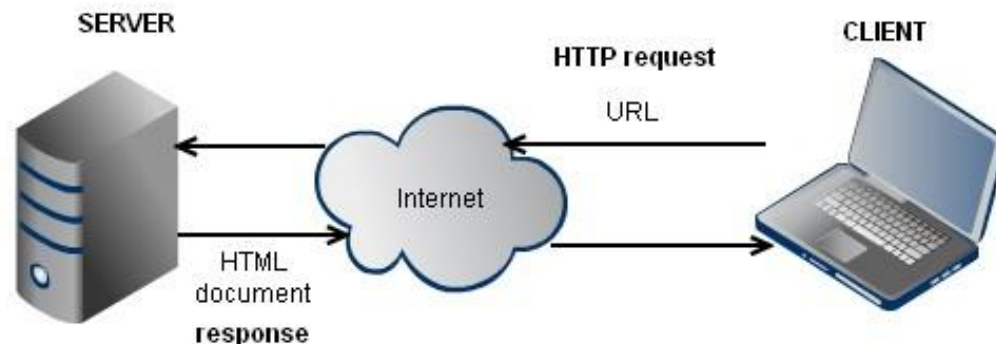


Как в целом все работает?

- На компьютере-сервере устанавливают **серверную часть** приложения
- Серверная часть может быть написана практически на любом языке – Java, C#, PHP, Python, Ruby и т.д.
- Этот код запускается в рамках **веб-сервера** – специальной программы, которая умеет принимать HTTP запросы и отдавать HTTP-ответы

Как в целом все работает?

- Серверная часть умеет «слушать» запросы от клиентов. То есть когда клиент запрашивает какой-то адрес на сервере, то на сервере запускается одна из функций серверной части
- Эта функция получает параметры, которые послал клиент, затем выполняет свой код, который формирует HTTP ответ. По сути – это просто текст. Этот текст, например, может содержать текст HTML страницы
- Браузер получает ответ и отрисовывает переданную страницу



Пример HTTP запроса

- **Пример запроса от браузера:**
- GET / HTTP/1.1 // метод, адрес и версия HTTP
Host: ya.ru // заголовки – пары ключ-значение
Connection: keep-alive
// тело сообщения (тут его нет)
// если тело есть, то оно отделяется одной пустой строкой
- **Метод** – это команда протокола HTTP. Есть методы GET, POST, PUT, DELETE и другие
- **Адрес** – идет относительно host
- <https://habrahabr.ru/post/215117/>

Пример HTTP ответа

- **Пример запроса от браузера:**
- HTTP/1.1 200 OK // версия HTTP, код ответа
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Cache-Control: no-cache,no-store,max-age=0,must-revalidate
Content-Length: 11369

<!DOCTYPE html><html>...</html> // тело – текст страницы

Пример

- Пример – показать в Chrome запрос ya.ru и ответ
- Пример – еще посмотреть что происходит при наборе текста в поле ввода на этом сайте, и при нажатии на кнопку

HTTP статусы, говорящие об ошибках

- Это почти все статусы вида 4xx, 5xx, рассмотрим некоторые:
- **400 Bad Request** – неверный запрос (код на стороне клиента посылает данные не в том формате, что ожидает сервер)
- **404 Not Found** – обращение по несуществующему адресу
- **500 Internal Server Error** – внутренняя ошибка сервера (какой-то конкретный вызов упал)
- **503 Service Unavailable** – сервис недоступен (обычно когда веб-сервер лежит целиком)
- **405 Method Not Allowed** – запрос не тем HTTP методом (например, указали GET вместо POST)

HTTP статусы авторизации

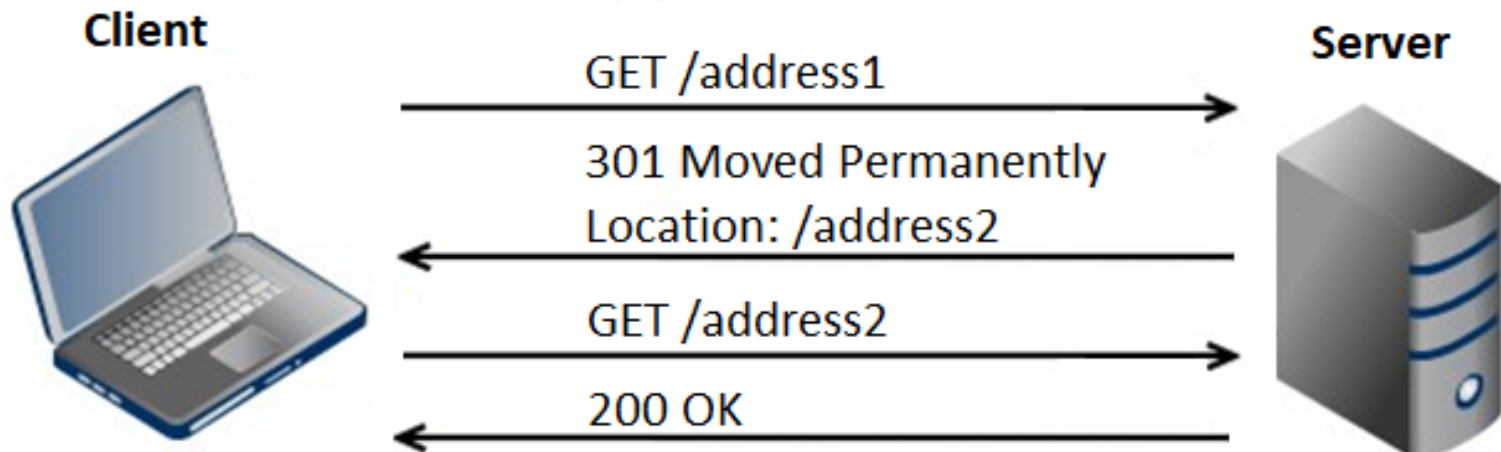
- **401 Unauthorized** – не авторизован
- Текущий пользователь не авторизован (не залогинился), и поэтому не получит доступ к ресурсу
- При некоторых схемах аутентификации браузер, получив статус 401, может попросить ввести логин и пароль
- **403 Forbidden** – запрещено
- Пользователь залогинен, но у него нет прав на обращение к ресурсу

Редиректы

- **Редирект** – это перенаправление клиента сервером на другой адрес
- Виды редиректов:
 - **301 Moved Permanently** – перемещено навсегда
 - Это постоянный редирект
 - **302 Moved Temporarily** – перемещено временно

Редиректы

- Клиент делает запрос на сервер по адресу /address1
- Сервер понимает, что ресурс по этому адресу был перемещен, надо перенаправить клиента на /address2
- Для этого сервер посылает код ответа **301** или **302**, и отправляет заголовок **Location**, в котором указан новый адрес ресурса
- Клиент получает ответ. Браузер обычно автоматически переходит по адресу из заголовка **Location**



Методы HTTP запросов

- **Первая строка запроса:** GET / HTTP/1.1
- В ее начале идет **имя метода** (тип операции в HTTP)
- Самые распространенные методы:

HTTP метод	Смысл	Пример (список контактов)
GET	Получить ресурс	Получить контакт
POST	Создать ресурс	Добавить контакт
PUT	Обновить ресурс	Обновить контакт
DELETE	Удалить ресурс	Удалить контакт

Методы HTTP запросов

- PUT и DELETE почти никто не использует, вместо него используют POST
- **Итого:**

HTTP метод	Смысл	Пример (список контактов)
GET	Получить ресурс	Получить контакт
POST	Любые действия с ресурсом, которые могут его изменить	Добавить контакт Удалить контакт Обновить контакт

Пример

- Пример – показать запросы с ошибочными статусами
- 404: <https://partner.s7.ru/something>
- 401: <https://partner-services.s7.ru/> и не вводить логин
- [https://partner.s7.ru/ layouts/S7/S7NewsService.asmx/GetNewsByID](https://partner.s7.ru/layouts/S7/S7NewsService.asmx/GetNewsByID)
- Метод POST, пример тела: {itemID: "1989"}
- 500: не передать itemID

Кэш и кэширование

- **Кэш** – промежуточный буфер с быстрым доступом для хранения данных
- Использование кэша называется **кэшированием**
- Используется для оптимизации
- Смысл такой – есть некоторая долгая операция
- Например, вычисление суммы чисел от 1 до 100
- Понятно, что сколько раз эту сумму ни вычисляй, результат будет одним и тем же
- Поэтому можно просто 1 раз вычислить эту сумму при первом обращении, запомнить результат, а потом всегда выдавать запомненный результат

Кэш и кэширование

- Конечно, этот пример про сумму чисел очень простой
- В самом деле кэширование применяют для более тяжелых операций
- Например, браузер может кэшировать HTML страницы, скрипты и CSS стили
- Ведь они меняются редко, поэтому нет смысла загружать их с сайта заново каждый раз
- Браузеры кэшируют GET-запросы, а POST - никогда

Правильное использование GET и POST

- GET должен использоваться только для немодифицирующих операций
- Иначе могут возникать проблемы – браузер может **кэшировать** GET запросы, и не выполнять их повторно
- То есть, если делать удаление ресурса через GET, то браузер иногда может даже не отправить запрос
- POST должен использоваться для модифицирующих операций

Cookies

- **Cookie (куки)** – небольшой фрагмент данных, который может храниться на стороне клиента, и который прикрепляется к каждому запросу
- Куки чаще всего применяются для:
 - **Аутентификации** - «залогиненность»
 - **Хранения сессии** - данных, связанных с текущим пользователем в этом сеансе. Например, корзина в интернет-магазине
 - **Хранения персональных настроек** – выбор конкретного языка на сайте, или расположения элементов, или темы оформления
- <https://ru.wikipedia.org/wiki/Cookie>

Cookies

- В инструментах Chrome есть раздел **Application -> Cookies**, там их можно смотреть/менять/удалять/добавлять
<http://joxi.ru/v29JeMliGEj5vA>
- Cookie – это пары ключ-значение (**Name** и **Value**)
- Cookie привязываются к определенному адресу (**Path**), либо к домену в целом (**Domain**)
- Если запрос относится к этому адресу, то клиент будет отправлять cookie, относящиеся к этому адресу
- У cookie есть срок истекания (**Expires**) – когда наступит указанное время, то cookie считается истекшей, и автоматически удаляется браузером

Аутентификация

- Для сервера каждый запрос никак не связан с предыдущими запросами. Т.е. он вас не помнит
- Чтобы все-таки сервер мог вас помнить, и понимать что вы это вы, при авторизации он выдает вам cookie авторизации
- При этом сервер запоминает, что пользователь с таким-то логином сейчас пользуется такой-то cookie
- Эта cookie сохраняется на клиенте и будет отправляться при каждом запросе на сервер
- Сервер будет брать эту cookie, и по ней понимать, что вы залогинены под тем-то логином

Создание и отправка Cookie

- Если сервер хочет установить клиенту cookie, он шлет заголовок **Set-Cookie**
 - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>
- Клиент при каждом запросе будет отправлять заголовок **Cookie**, в котором будут перечислены все cookie
 - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cookie>
- Браузер через код на JavaScript может сам на своей стороне создавать cookie, которые будут отправляться на сервер

Понятие API

- **API (Application Programming Interface)** – набор функций, предоставляемых приложением
- Например, есть какой-то сервер
- У него по некоторым адресам доступны некоторые команды, это и есть API:
 - <https://api.vk.com/method/database.getCountries>
 - <https://restcountries.eu/rest/v2/region/europe>

REST

- **REST** – архитектура для клиент-серверного взаимодействия, основанная на протоколе HTTP, которая характеризуется следующими признаками:
 - **Отсутствие состояния** – для сервера каждый запрос клиента никак не связан с предыдущим. Сервер не запоминает состояние между запросами
 - **Ориентированность на ресурсы** – API пишется в терминах ресурсов, а не команд

REST - материалы

- Что почитать:
- <https://ru.wikipedia.org/wiki/REST>
- <https://habrahabr.ru/post/38730/>

Клиент-сервер

- Эта архитектура применима не только к сайтам, но и к мобильным и десктопным приложениям
- Мобильное или десктопное приложение может получать какие-то данные с сервера и отображать их
- И может посылать данные на сервер
- Обычно там также используется протокол HTTP
- Чаще всего данные передают в форматах **XML** или **JSON**

Передача параметров в GET запросах

- В GET запросах общепринято передавать параметры в самом URL (адресе)
- Тело запроса при этом оставляют пустым
- http://auto.drom.ru/toyota/camry/?minprice=50000&minyear=2016&mv=1.0&go_search=2
- Часть URL, начинающаяся с символа ?, называется **query string (строка запроса)**
- По идее там может быть любой текст, но общепринято передавать там параметры в виде **параметр1=значение1&параметр2=значение2** и т.д.
- То есть параметры разделяют символом & (амперсанд)

Передача параметров в GET запросах

- http://auto.drom.ru/toyota/camry/?minprice=50000&minyear=2016&mv=1.0&go_search=2
- Сервер вытаскивает из url этот query string, разбирает параметры
- В данном случае получается:
 - minprice = 50000
 - minyear = 2016
 - mv = 1.0
 - go_search = 2

Передача параметров в POST запросах

- В POST запросах общепринято передавать параметры в теле запроса
- Чаще всего используют формат JSON
- В данном примере передается объект с полем `newsItemId` равным строке 2171
- `POST /_layouts/S7/S7NewsService.asmx/GetNewsFiles`
`HTTP/1.1`
`Host: partner.s7.ru`
`Content-Length: 21`
`Origin: https://partner.s7.ru`
`Content-Type: application/json`

`{"newsItemId":"2171"}`

Передача параметров в адресе

- Иногда параметры запроса передают в самом адресе запроса:
- <https://restcountries.eu/rest/v2/region/europe>
- Здесь часть europe не является фиксированной, это название региона
- Другие доступные регионы: africa, americas, asia, oceania
- Сервер, если адреса прописаны соответствующим образом, умеет вытащить данные из самого URL и использовать их

**Выполнение
запросов**

Выполнение HTTP-запросов

- Допустим, мы хотим выполнить REST или RPC запрос
- В этом могут помочь так называемые REST-клиенты:
 - **Advanced REST Client** – расширение для Chrome
 - **Postman** – расширение для Chrome
- Они позволяют легко сформировать и выполнить запрос с указанными параметрами

Advanced REST клиент

1. Введите адрес, по которому сделать запрос
2. Выберите метод – GET/POST/PUT/DELETE
3. Задайте тело сообщения, если нужно (чаще всего в формате JSON)
4. Если это JSON запрос, обязательно укажите заголовок Content-type: application/json
 - Иначе сервер может не понять, что мы хотим передать JSON, и может выдать ошибку
 - Есть краткий способ выбрать content-type, т.к. это частая операция

Advanced REST клиент

Request

☒ Use XHR



> https://partner.s7.ru/_layouts/S7/S7NewsService.aspx/GetNewsByID

URL

☐ GET ☒ POST ☐ PUT ☐ DELETE ☐ PATCH

Other methods

application/json

Raw headers

Headers form

Headers sets

Variables

Content-Type

application/json

заголовки

content type можно
выбрать и тут, заголовок
добавится сам

ADD HEADER



30 bytes

Raw payload

Data form

Files

{itemID: 1989}

тело запроса

SEND

200 OK

324.00 ms

DETAILS

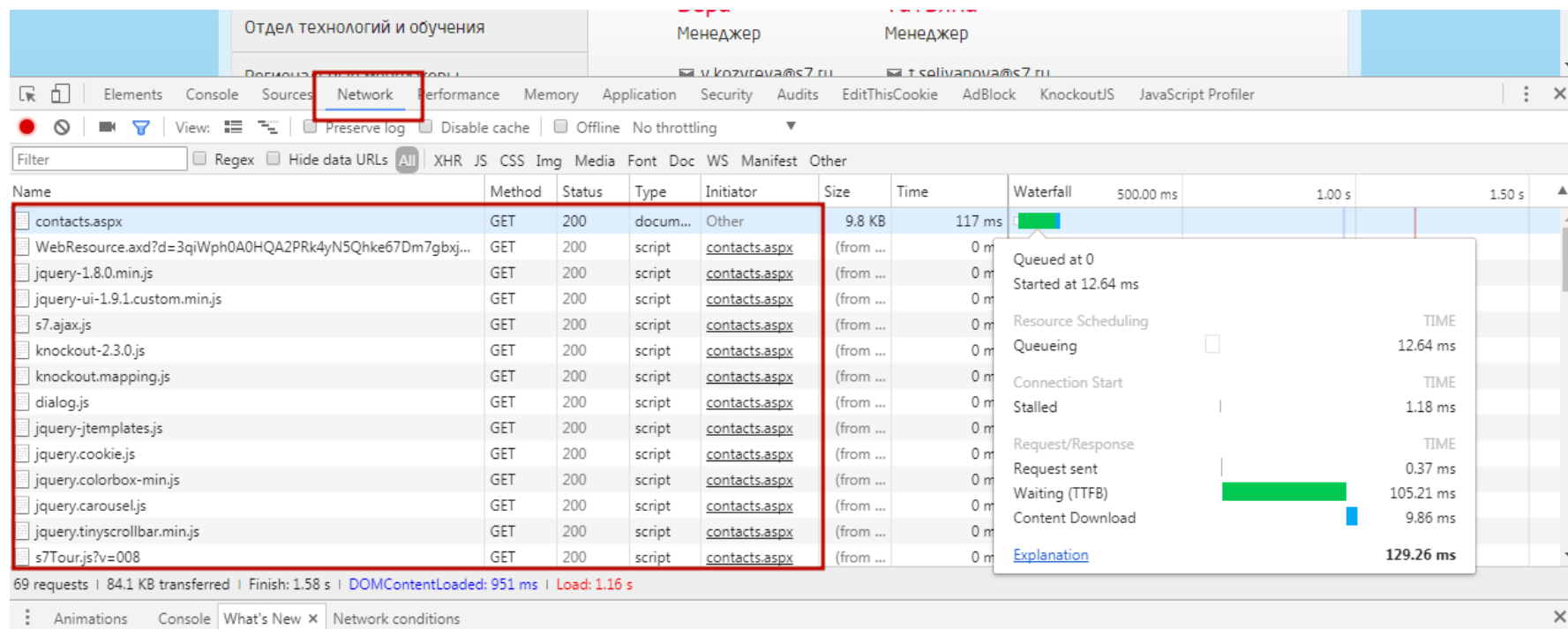
Практика

- Установите Advanced REST Client в Chrome
- Выполнить следующий запрос:
- URL: <https://partner.s7.ru/layouts/S7/S7DesignService.asmx/GetContacts>
- Метод: POST
- Не забудьте Content-type: application/json
- В теле запроса надо передать JSON, у которого есть следующие поля:
 - pageIndex, передайте 0
 - groupName, передайте строку Руководители

**Перехват
запросов**

Перехват запросов браузера

- Откройте Chrome Developer Tools – F12
- Выберите вкладку Network
- В таблице будут отображены запросы для данной страницы, выполненные с момента открытия Developer Tools



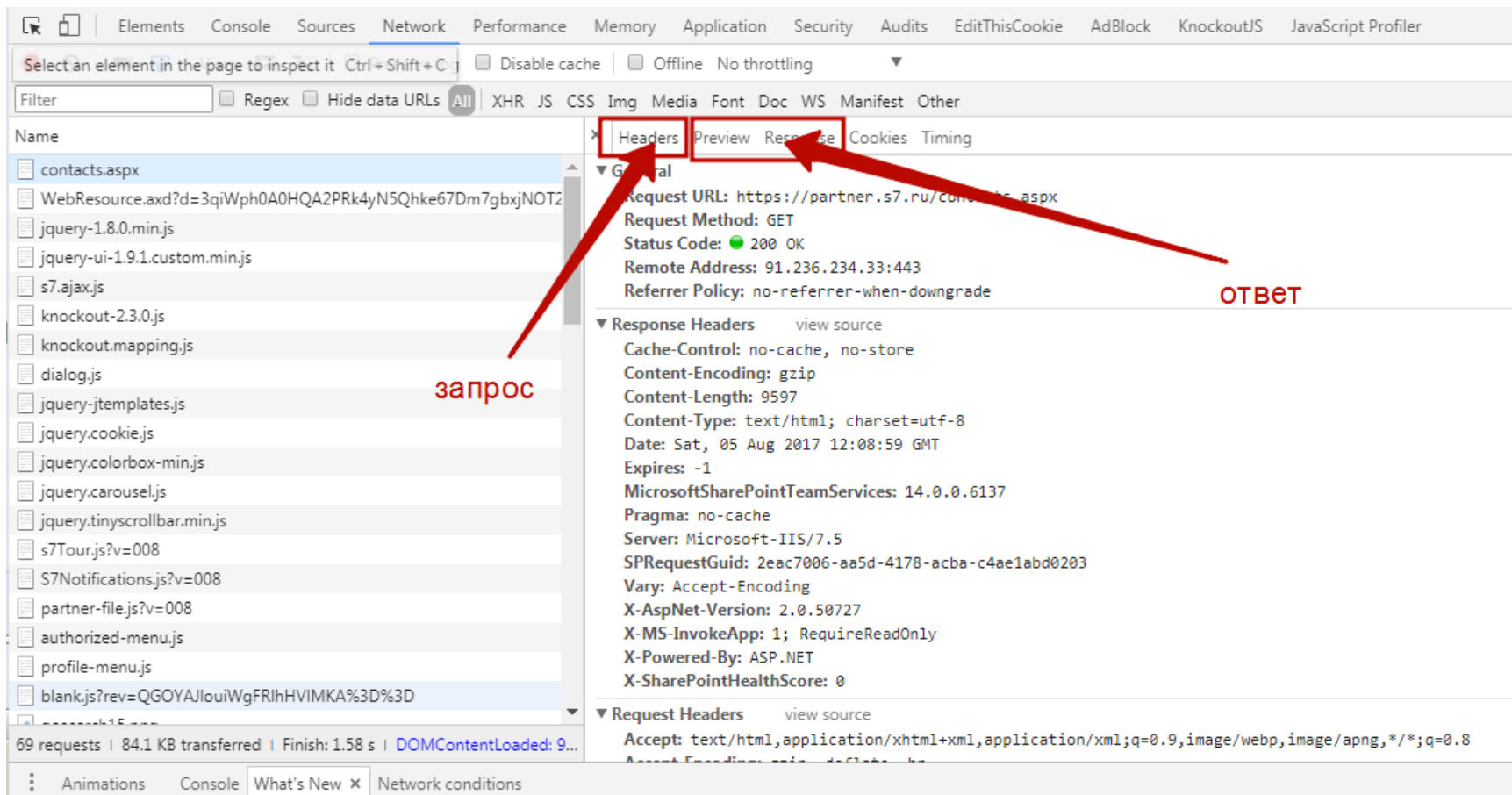
The screenshot shows the Chrome Developer Tools interface with the Network tab selected. A red box highlights the 'Network' tab in the top bar. Below it, a table lists network requests. The first request, 'contacts.aspx', is highlighted with a red box. A detailed view of this request is shown on the right, including a waterfall chart and a timeline of events.

Name	Method	Status	Type	Initiator	Size	Time
contacts.aspx	GET	200	docum...	Other	9.8 KB	117 ms
WebResource.axd?d=3qiWph0A0HQA2PRk4yN5Qhke67Dm7gbxj...	GET	200	script	contacts.aspx	(from ...)	0 m
jquery-1.8.0.min.js	GET	200	script	contacts.aspx	(from ...)	0 m
jquery-ui-1.9.1.custom.min.js	GET	200	script	contacts.aspx	(from ...)	0 m
s7.ajax.js	GET	200	script	contacts.aspx	(from ...)	0 m
knockout-2.3.0.js	GET	200	script	contacts.aspx	(from ...)	0 m
knockout.mapping.js	GET	200	script	contacts.aspx	(from ...)	0 m
dialog.js	GET	200	script	contacts.aspx	(from ...)	0 m
jquery-jtemplates.js	GET	200	script	contacts.aspx	(from ...)	0 m
jquery.cookie.js	GET	200	script	contacts.aspx	(from ...)	0 m
jquery.colorbox-min.js	GET	200	script	contacts.aspx	(from ...)	0 m
jquery.carousel.js	GET	200	script	contacts.aspx	(from ...)	0 m
jquery.tinyscrollbar.min.js	GET	200	script	contacts.aspx	(from ...)	0 m
s7Tour.js?v=008	GET	200	script	contacts.aspx	(from ...)	0 m

69 requests | 84.1 KB transferred | Finish: 1.58 s | DOMContentLoaded: 951 ms | Load: 1.16 s

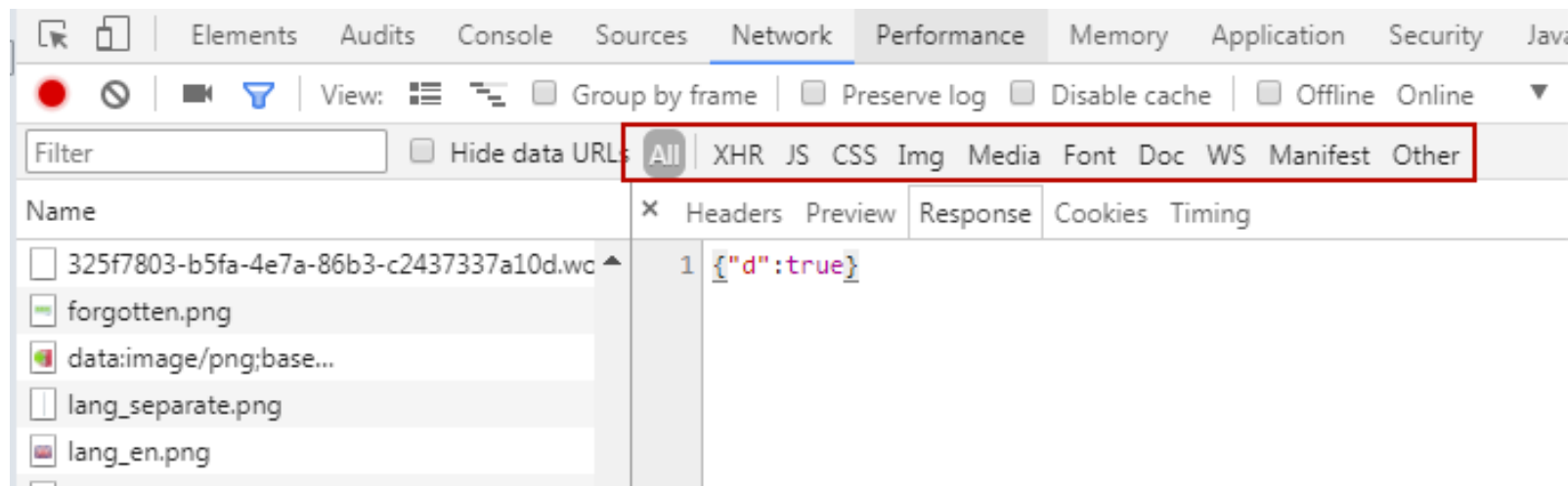
Перехват запросов браузера

- Если выбрать конкретный запрос, то можно посмотреть что было отправлено, и какой ответ пришел



Фильтр запросов

- Есть возможность фильтровать запросы по типу



- **All** – все запросы
- **XHR** – только AJAX запросы (запросы из JS кода на сервер без перезагрузки страницы). Обычно передается JSON
- **JS** – скрипты, **CSS** – стили, **Img** – картинки
- **Doc** – документ (загрузка самой страницы)

Preserve log

- По умолчанию при перезагрузке страницы список запросов сбрасывается
- Но это не всегда удобно – с некоторых страниц есть **редиректы** – перенаправления на другие адреса
- И так мы не можем их увидеть
- <https://partner.s7.ru/agenthome.aspx>
- Чтобы список выполненных запросов не очищался, поставьте галочку Preserve log

