Прикладное программирование

Лекция №10.

СЕТЕВЫЕ ТЕХНОЛОГИИ

- Создание сетевых приложений в Java
- Как потерять ~17 млн долларов США
- Основы компьютерных сетей
- Как устроен HTTP-протокол
- IP-адреса в Java-приложениях
- Клиент-серверные приложения

Улыбнитесь! Сейчас вылетит …

Создание сетевых приложений в Java

Встроенная поддержка сетевых взаимодействий является одной из базовых черт языка **Java**.

Разработчику предлагается широкий спектр возможностей для создания сетевых приложений. Часть из них доступна только в Enterprise-версии платформы **Java (Java EE)**, в то время как некоторые являются встроенными в стандартную версию **Java (Java SE)**.

Подробнее с возможностями Java EE мы познакомимся в следующем семетре.

Java SE VS Java EE

Java SE:

- Взаимодействие посредством сокетов (низкоуровневое);
- Взаимодействие посредством классов **URL** и **URLConnection**;
- Применение технологии удаленного вызова методов **RMI** **(Remote Method Invocation)**.

Java EE:

- Взаимодействие на основе **протокола HTTP**;
- Распределенное взаимодействие на основе технологии компонентов **EJB** **(Enterprise Java Beans)**.

Низкоуровневое сетевое взаимодействие в Java-программах

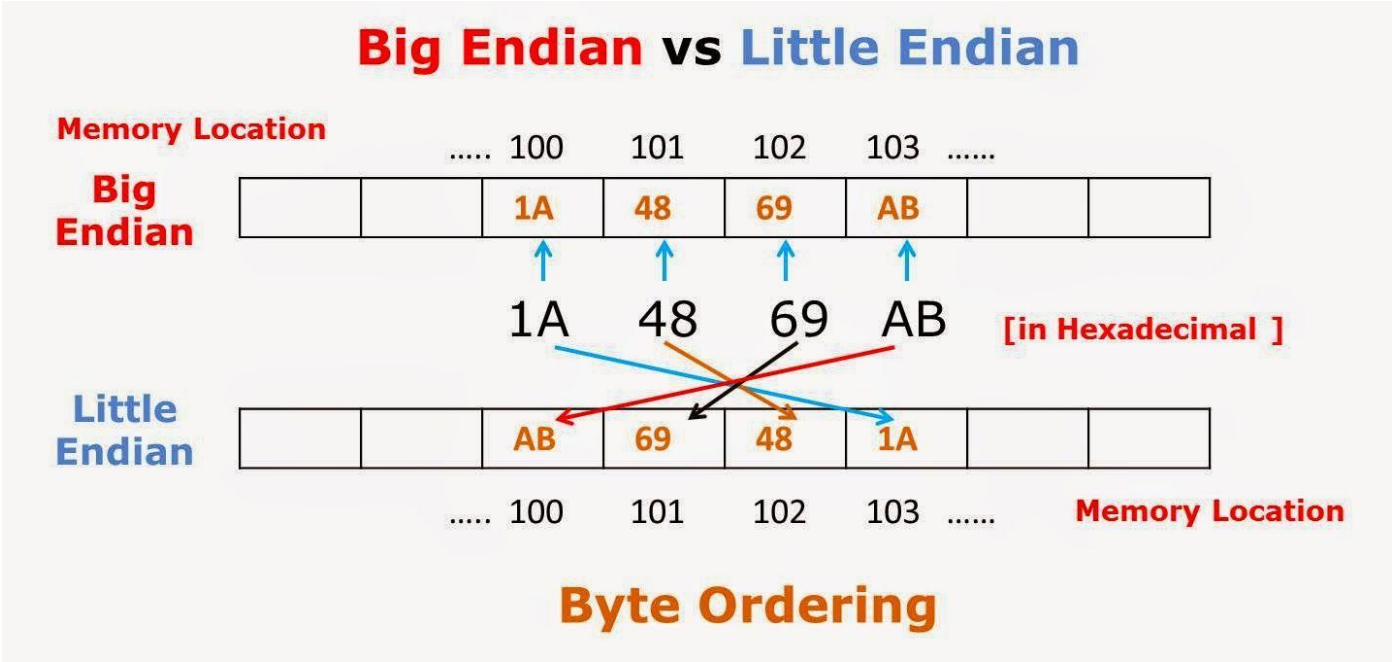
Низкоуровневая работа с удаленными ресурсами очень похожа на работу с локальными файлами.

Используемая модель **сетевого ввода-вывода данных** основана на **объектах потоков**, что позволяет применять для отправки и получения данных **те же самые методы**, что и для работы с другими потоками.

Специфика сетевого программирования настолько, насколько это возможно, сокрыта внутри реализации **JVM**.

Пример особенностей сетевого взаимодействия

Важным вопросом при разработке приложений, участвующих в сетевой коммуникации, является преобразование форматов данных при их передаче по сети. На различных аппаратных платформах используются отличающиеся способы хранения многобайтовых переменных в памяти:



Информация в виде последовательности байтов

В современной вычислительной технике и цифровых системах связи информация представлена **в виде последовательности байтов.**

Если число не может быть представлено одним байтом, имеет важное значение, **в каком порядке байты записываются в памяти компьютера** либо **передаются по линиям связи**.

Часто выбор порядка записи байтов **произволен** и определяется только **соглашениями** либо **стандартами**.

Информация в виде последовательности байтов

- **Порядок от старшего к младшему (A_n…A_0)** является стандартным для **протоколов TCP/IP** и используется в заголовках пакетов данных и во многих протоколах более высокого уровня, разработанных для использования поверх **TCP/IP**.

  Поэтому **порядок байтов от старшего к младшему** часто называют **«сетевым порядком байтов».**

  При таком порядке байтов удобно проводить **сравнение строк**, а также он применяется во многих форматах файлов (**PNG**, **FLV**, **EBML**, **JPEG**).

Информация в виде последовательности байтов

- **Порядок от младшего к старшему (A0…An)** – порядок записи был принят в памяти персональных компьютеров с процессорами архитектуры x86 – «Интеловский порядок байтов».

Современные процессоры позволяют работать с:

- **Одно-**,
- **Двух-**,
- **Четырёх-**,
- **Восьми-**

байтовыми операндами.

При таком порядке байтов очень удобно то, что при увеличении размера («байтовости») операнда его значение **не изменится**.

Пример особенностей сетевого взаимодействия

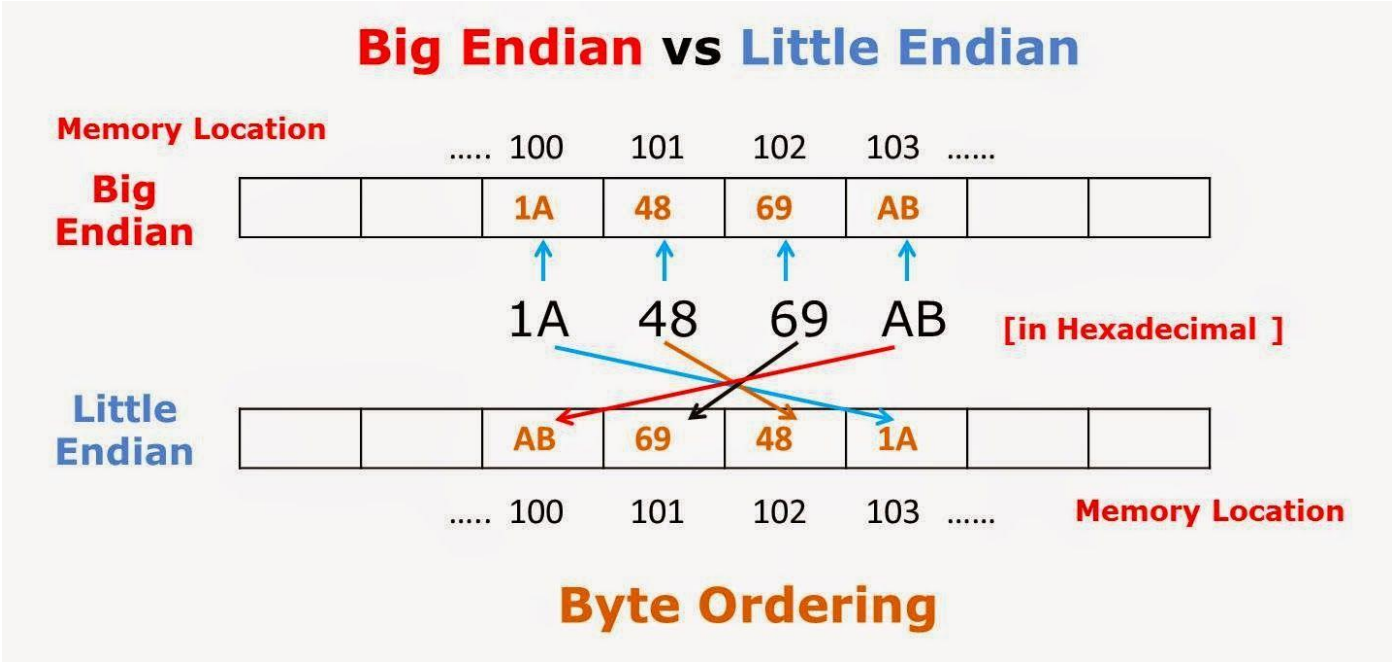
Решим задачу:

- Вы представители известной IT-компании.
- Вы занимаетесь разработкой программы для осуществления банковских переводов.
- Компания имеет счета в обоих банках и хочет переместить огромную сумму с одного счета на другой.
- Договоримся, что размер суммы передается по сети между программами, работающими на серверах различных банков.

Пример особенностей сетевого взаимодействия

Условие задачи:

Перевести **$16777216** (в шестнадцатеричном виде - **0x01 00 00 00**).



Пример особенностей сетевого взаимодействия

БАНК 1 (Little-endian)
Начальный баланс: $16777216. Уменьшить баланс на $16777216. **Конечный баланс:** $0. **Отправить по сети байты:** 00, 00, 00, 01 (число 0x01000000).

Пример особенностей сетевого взаимодействия

БАНК 1 (Little-endian)	**БАНК 2 (Big-endian)**
Начальный баланс: $16777216. Уменьшить баланс на $16777216. **Конечный баланс:** $0. **Отправить по сети байты:** 00, 00, 00, 01 (число 0x01000000).	**Начальный баланс:** $0. **Получить по сети байты:** 00, 00, 00, 01 (число 0x00000001). Увеличить баланс на $1. **Конечный баланс:** $1.

Wake the **** up , Students.
We have a technology to learn!

Сетевая модель OSI

Сетевая модель **OSI (Open Systems Interconnection Basic Reference)** — модель сетевых протоколов **OSI/ISO**.

Данная модель представляет собой международный стандарт для проектирования сетевых коммуникаций и определяет различные уровни взаимодействия систем.

Сетевая модель OSI

Модель OSI			
Верхние уровни	**Прикладной** (Application)	**Данные**	Доступ к сетевым службам
	Представления (Presentation)	**Данные**	Представление и шифрование данных
	Сеансовый (Session)	**Данные**	Управление сеансом связи
Нижние уровни	**Транспортный** (Transport)	**Блоки/Сегменты**	Прямая связь между конечными пунктами и надёжность
	Сетевой (Network)	**Пакеты**	Определение маршрута и логическая адресация
	Канальный (Data Link)	**Биты/Кадры**	Физическая адресация
	Физический (Physical)	**Биты**	Работа со средой передачи, сигналами и двоичными данными

Сетевая модель OSI (Верхние уровни)

- **Прикладной уровень (Application layer) –** высокоуровневые функции сетевого взаимодействия:
  - передача файлов,
  - отправка сообщений по электронной почте.

Сетевая модель OSI (Верхние уровни)

- **Уровень представления (Presentation layer) –** обеспечивает гарантию того, что передаваемая информация будет понятна прикладному уровню в другой системе.

  При необходимости выполняет преобразование форматов данных в некоторый общий формат представления, а на приеме, соответственно, выполняет обратное преобразование.

  Может выполняться **шифрование** и **дешифрование данных**.

  Пример: протокол SSL (Secure Socket Layer).

Сетевая модель OSI (Верхние уровни)

- **Сессионный (сеансовый) уровень (Session layer) –** позволяет двум программам поддерживать **сессию/сеанс** (продолжительное взаимодействие по сети).

  Управляет установлением сеанса, обменом информацией и завершением сеанса, а также отвечает за идентификацию и обеспечивает работу служб безопасности с целью упорядочивания доступа.

Сетевая модель OSI (Нижние уровни)

- **Транспортный уровень (Transport layer) –** передача данных между двумя программами, функционирующими на разных компьютерах без потерь и дублирования информации.

Сетевая модель OSI (Нижние уровни)

- **Сетевой уровень (Network layer) –** обеспечивает доставку данных между компьютерами сети, представляющей собой объединение различных физических сетей.

  Данный уровень предполагает наличие средств логической адресации, позволяющих однозначно идентифицировать компьютер в объединенной сети.

  Одной из главных функций, выполняемых средствами данного уровня, является **целенаправленная передача данных конкретному получателю.**

Сетевая модель OSI (Нижние уровни)

- **Канальный уровень (Data Link layer) –** организует передачу данных между абонентами через физический уровень, поэтому на данном уровне предусмотрены **средства адресации**, которые позволяют *однозначно идентифицировать отправителя и получателя* во всем множестве абонентов, подключенных к общей линии связи.

Также упорядочивает передачи с целью параллельного использования одной линии связи несколькими парами абонентов и обеспечивает проверку ошибок, которые могут возникать при передаче данных физическим уровнем.

Сетевая модель OSI (Нижние уровни)

- **Физический уровень (Physical layer) –** определяет способ физического соединения компьютеров в сети.

  Функциями средств, относящихся к данному уровню, являются передача сигналов и побитовое преобразование цифровых данных в сигналы, передаваемые по физической среде (например, по кабелю).

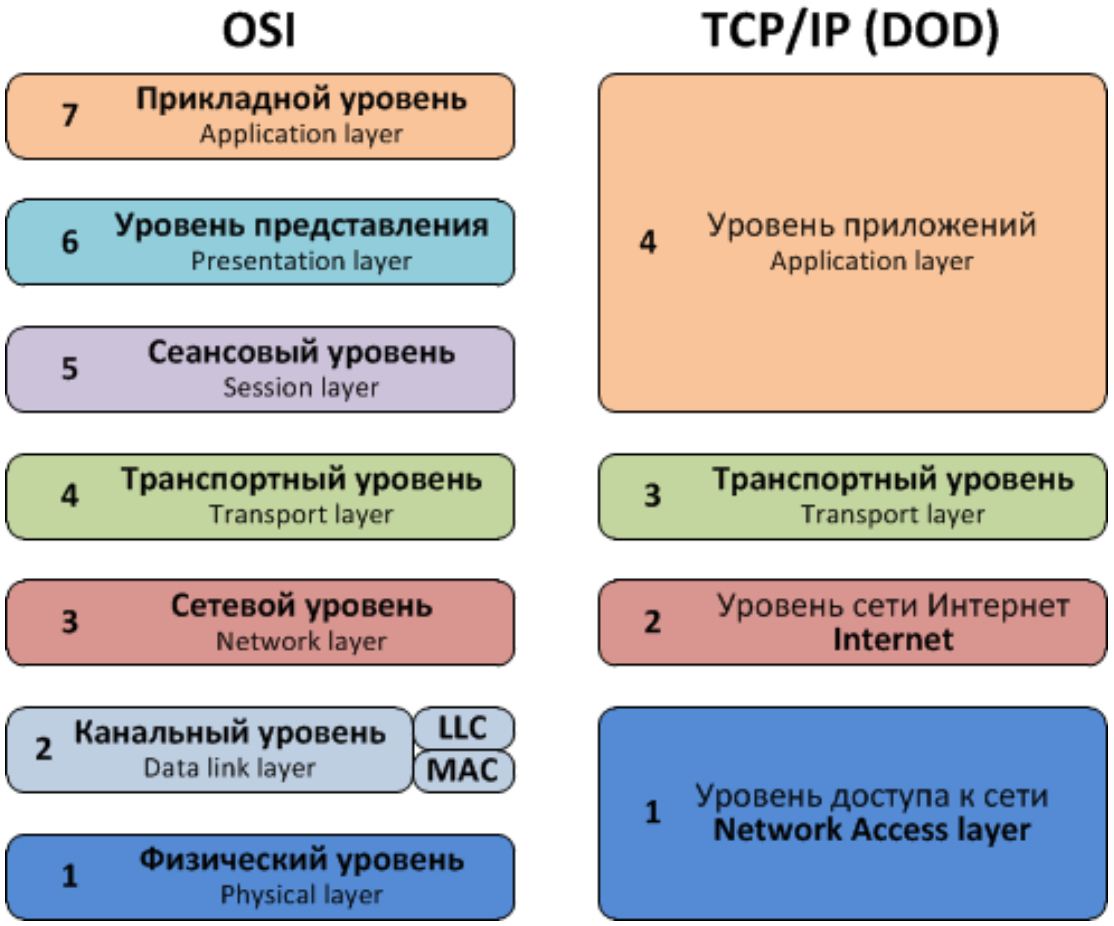
Сетевая модель DOD (Модель TCP/IP)

TCP/IP — Transmission Control Protocol / Internet Protocol – это сетевая модель передачи данных, представленных в цифровом виде, которая описывает способ передачи данных от источника информации к получателю.

Также известна как модель **DOD (Department of Defense).**

Представляет собой набор интернет-протоколов, которые обеспечивают сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься.

Сетевая модель DOD (Модель TCP/IP)



Модель предполагает прохождение информации **через четыре уровня абстракции**, каждый из которых описывается **протоколом передачи данных**:

- **Прикладной уровень**
- **Транспортный уровень**
- **Уровень сети интернет**
- **Уровень доступа к сети**

Особенности протокола TCP

TCP — ориентированный на соединение протокол, что означает необходимость **«рукопожатия»** для установки соединения между двумя хостами. Как только соединение установлено, пользователи могут отправлять данные в обоих направлениях.

- **Надёжность** — TCP управляет подтверждением, повторной передачей и тайм-аутом сообщений. Производятся многочисленные попытки доставить сообщение. Если оно потеряется на пути, сервер вновь запросит потерянную часть.
- **Упорядоченность** — если два сообщения последовательно отправлены, первое сообщение достигнет приложения-получателя первым. Если участки данных прибывают в неверном порядке, TCP отправляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению.

Особенности протокола TCP

TCP — ориентированный на соединение протокол, что означает необходимость **«рукопожатия»** для установки соединения между двумя хостами. Как только соединение установлено, пользователи могут отправлять данные в обоих направлениях.

- **Тяжеловесность** — TCP необходимо три пакета для установки сокет-соединения перед тем, как отправить данные. TCP следит за надёжностью и перегрузками.
- **Потоковость** — данные читаются как поток байтов, не передается никаких особых обозначений для границ сообщения или сегментов.

User Datagram Protocol

UDP — более простой, основанный на сообщениях протокол без установления соединения.

Протоколы такого типа **не устанавливают выделенного соединения между двумя хостами**. Связь достигается путём передачи информации в одном направлении от источника к получателю без проверки готовности или состояния получателя.

Особенности протокола UDP

Применяется в приложениях **для голосовой связи** и **потокового видео**, а также в **компьютерных играх**, т.к. допускается **потеря некоторых пакетов**.

- **Ненадёжность** — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути.
- **Неупорядоченность** — если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан.
- **Легковесность** — нет упорядочивания сообщений, отслеживания соединений и т.д.

Особенности протокола UDP

Применяется в приложениях **для голосовой связи** и **потокового видео**, а также в **компьютерных играх**, т.к. допускается **потеря некоторых пакетов**.

- **Датаграммы** — пакеты посылаются по отдельности и проверяются на целостность только, если они прибыли. После получения операция чтения на сокете-получателе выдаст сообщение таким, каким оно было изначально послано.
- **Нет контроля перегрузок** — UDP сам по себе не избегает перегрузок. Для приложений с большой пропускной способностью возможно вызвать коллапс перегрузок.

Стек протоколов TCP/IP:

На стеке протоколов **TCP/IP** построено всё взаимодействие пользователей в **IP-сетях**. Стек является **независимым** от физической среды передачи данных, благодаря чему обеспечивается полностью прозрачное взаимодействие между **проводными** и **беспроводными сетями**.

Основные протоколы модели TCP/IP прикладного уровня:

- **HTTP** — Hypertext Transfer Protocol (www);
- **HTTPS** — Hypertext Transfer Protocol *Secure* (www);
- **NNTP** — Network News Transfer Protocol (группы новостей);
- **SMTP** — Simple Mail Transfer Protocol (посылка почты);
- **POP3** — Post Office Protocol (чтение почты с сервера);
- **FTP** — File Transfer Protocol (протокол передачи файлов).

HTTP

HTTP (Hypertext Transfer Protocol) – это протокол передачи данных между распределёнными системами.

Основой HTTP является **технология «клиент-сервер»**:

- **Потребители (клиенты)** – инициируют соединение и посылают запрос;
- **Поставщики (серверы)** – ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

Характеристики HTTP-протокола

- **HTTP протокол** работает по технологии **клиент-сервер**.
- Для передачи данных по протоколу HTTP используется порт **80 TCP** или **8080**.
- Спецификация протокола **RFC 2616**.
- Для идентификации ресурса HTTP протокол использует **URI**.
- **HTTP протокол** не имеет промежуточных состояний между запросом и ответом.
- **HTTP протокол** синхронный, но он позволяет отправлять клиенту несколько запросов подряд, не дожидаясь ответа сервера, при условии, что сервер даст ответы на запросы в том порядке, в котором они приходили.

How to create HTTPS?

Характеристики HTTPS-протокола

HTTPS (***HyperText Transfer Protocol Secure*** — безопасный протокол передачи гипертекста) — это расширение протокола HTTP, поддерживающее шифрование посредством криптографических протоколов **SSL** и **TLS**.

Чем отличаются HTTP от HTTPS

- HTTPS не является отдельным протоколом передачи данных, а представляет собой *расширение протокола HTTP* с надстройкой шифрования;
- Передаваемые по протоколу HTTP данные **не защищены**, HTTPS обеспечивает конфиденциальность информации путем ее шифрования;
- **HTTP** использует порт **80**, **HTTPS** — порт **443**.

URL

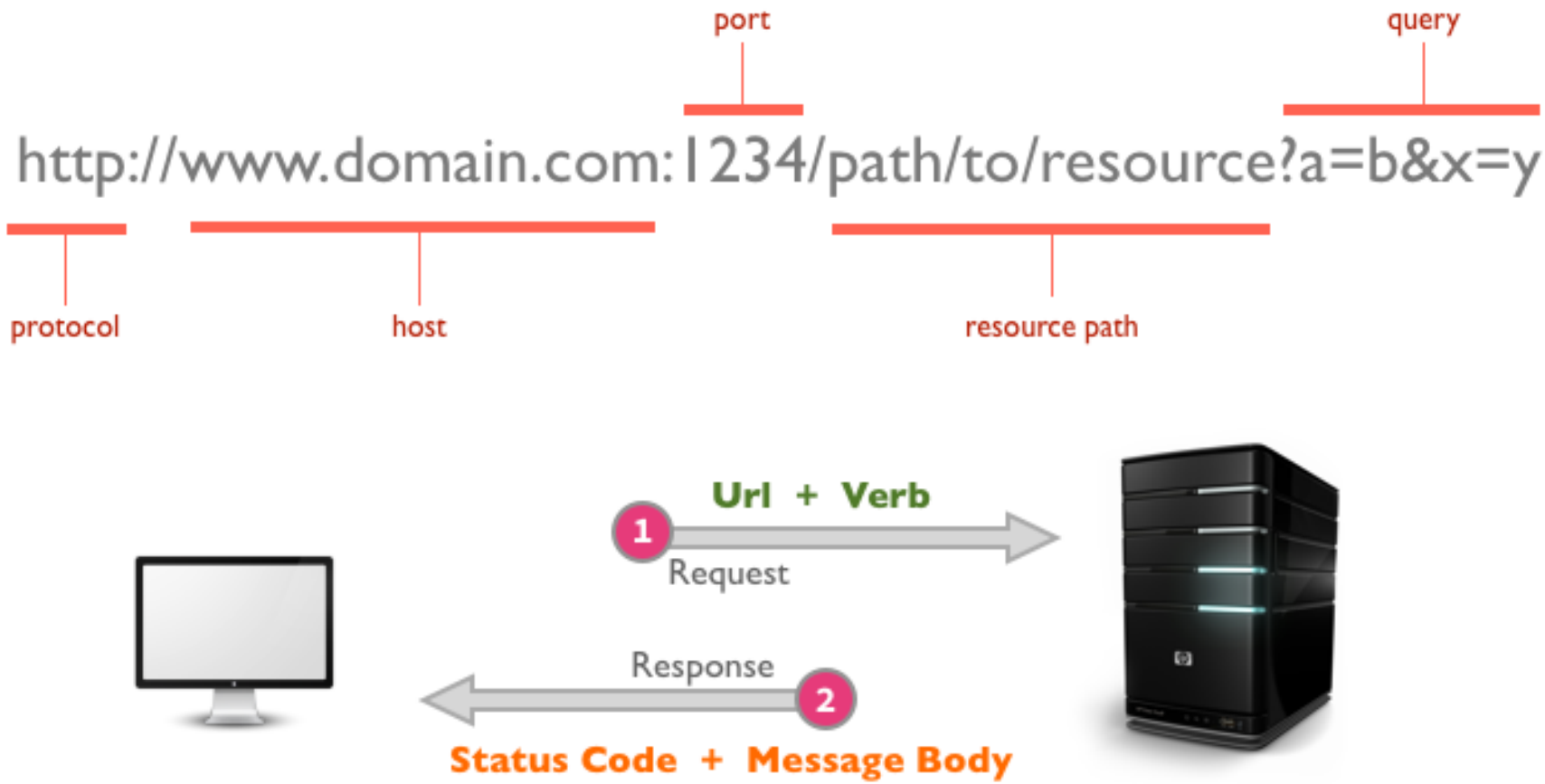
URI — унифицированный (единообразный) идентификатор ресурса.

URL — это **URI**, который, кроме **идентификации ресурса**, предоставляет ещё и информацию **о местонахождении этого ресурса**.

С помощью **URL** определяется **Host**, с которым ведется общение.

***HOST** – устройство, которое предоставляет услуги **формата «клиент-сервер»** **в режиме сервера** по интерфейсам и уникально определённое на них.

URL


port
query
http://www.domain.com:1234/path/to/resource?a=b&x=y
protocol
host
resource path
Url + Verb
1
Request
Response
2
Status Code + Message Body

Методы в HTTP протоколе

Методы определяют способ обращения клиента к серверу и сообщают, какую информацию хочет получить или передать клиент серверу.

В целях безопасности **HTTP-сервер** можно настроить так, чтобы он выполнял или игнорировал некоторые методы.

Методы в HTTP протоколе

- **GET**: **получить доступ** к существующему ресурсу. В URL перечислена вся необходимая информация, чтобы сервер смог найти и вернуть в качестве ответа искомый ресурс.
- **POST**: используется **для создания нового ресурса** (содержит в себе всю нужную информацию для создания нового ресурса).
- **PUT**: **обновить** текущий ресурс (содержит обновляемые данные).
- **DELETE**: служит для **удаления** существующего ресурса.

Методы в HTTP протоколе

- **HEAD**: аналогичен **GET**. При данном запросе **не передаётся сообщение**. Сервер получает только **заголовки**. Часто используется, чтобы определить, был ли изменён ресурс.
- **TRACE**: во время передачи запрос проходит через множество **точек доступа** и **прокси серверов**, каждый из которых вносит свою информацию: **IP**, **DNS**. С помощью данного метода, можно увидеть **всю промежуточную информацию**.
- **OPTIONS**: используется для **определения возможностей сервера**, его **параметров** и **конфигурации** для конкретного ресурса.

Коды состояния

В ответ на запрос, сервер отправляет ответ, который содержит **код состояния**, который сообщает, что **произошло с запросом**, как **сервер понял этот запрос** и что **нужно сделать клиенту**, чтобы сервер мог **корректно выполнить запрос**.

Сообщения:

- **1xx: информационные** - запрос получен, но еще не обработан.
- **2xx: успешные** - успешно принял и обработал сообщение клиента.
- **3xx: перенаправление** - нужны дополнительные действия, чтобы завершить процесс, часто перенаправление клиента на другой адрес.
- **4xx: ошибка клиента** - произошла ошибка со стороны клиента.
- **5xx: серверная ошибка** - произошла ошибка на стороне сервера.

1xx: Информационные сообщения

- **100 Continue:** клиент ещё отправляет оставшуюся часть запроса.

*Набор этих кодов был введён в **HTTP/1.1**.*

*Клиенты, работающие с **HTTP/1.0** игнорируют данные заголовки.*

2xx: Сообщения об успехе

При **GET**-запросе, сервер отправляет ответ **в теле сообщения**.

Если клиент получил код из серии **2xx**, то запрос прошел успешно.

- **200 OK**: все в порядке.
- **201 Created**: в результате успешного выполнения запроса был создан новый ресурс.
- **202 Accepted**: запрос принят, но может не содержать ресурс в ответе. *Полезно для асинхронных запросов на стороне сервера. Сервер определяет, отправить ресурс или нет.*
- **204 No Content**: в теле ответа нет сообщения.
- **205 Reset Content**: указание серверу о сбросе представления документа.
- **206 Partial Content**: ответ содержит только часть контента.

3xx: Перенаправление / Переадресация

Сообщение клиенту о необходимости совершить ещё одно действие либо уведомление о перенаправлении клиента на другой адрес.

- **301 Moved Permanently**: ресурс **теперь** можно найти по другому URL адресу.
- **303 See Other**: ресурс **временно** можно найти по другому URL адресу. *Заголовок Location содержит временный URL.*
- **304 Not Modified**: сервер определяет, что ресурс **не был изменён**, и клиенту нужно задействовать **закэшированную** версию ответа.

4xx: Клиентские ошибки

Данный класс сообщений используется сервером, если он решил, что запрос был отправлен с ошибкой.

- **400 Bad Request**: вопрос был сформирован неверно.
- **401 Unauthorized**: необходима аутентификация. *Информация передаётся через заголовок Authorization.*
- **403 Forbidden**: сервер не открыл доступ к ресурсу.
- **404 Not Found:** ресурс не найден на сервере.
- **405 Method Not Allowed**: неверный HTTP метод был задействован для того, чтобы получить доступ к ресурсу.
- **409 Conflict**: сервер не может до конца обработать запрос, т.к. пытается изменить более новую версию ресурса. *Часто происходит при **PUT запросах**.*

5xx: Ошибки сервера

Ряд кодов, которые используются для определения ошибки сервера при обработке запроса.

- **500 Internal Server Error:** внутренняя ошибка сервера.
- **501 Not Implemented**: сервер не поддерживает запрашиваемый функционал.
- **503 Service Unavailable**: на сервере произошла ошибка, либо он перегружен. *Обычно происходит, когда сервер не отвечает за отведенное на запрос время.*

Форматы сообщений запроса/ответа

Структура сообщения, передаваемого сообщения через HTTP:

message = <**start-line**>
*(<**message-header**>)
CRLF
[<**message-body**>]

<**start-line**> = Request-Line | Status-Line

<**message-header**> = Field-Name ':' Field-Value

Форматы сообщений запроса/ответа

Между заголовком и телом сообщения должна обязательно присутствовать **пустая строка.**

Заголовков может быть несколько:

- Общие заголовки
- Заголовки запроса
- Заголовки ответа
- Заголовки сущностей

Тело ответа может содержать полную информацию или её часть, если активирована соответствующая возможность (Transfer-Encoding: chunked).

Идентификация сторон сетевого взаимодействия

Для обмена данными друг с другом участники сетевого взаимодействия, например Java-программы, должны однозначно идентифицировать друг друга в сети.

Первым шагом идентификации участников взаимодействия является идентификация в сети тех машин, на которых они расположены, для чего применяется адресация на основе **протокола IP (Internet Protocol)**.

Идентификация сторон сетевого взаимодействия

Она может существовать **в двух формах**:

- В виде **десятично-точечной нотации**, представленной четырьмя числами от 0 до 255, разделенными точками.
- Так как запоминание цифрового адреса компьютеров является для человека затрудненным, то применяется и **система буквенной идентификации**, например **www.google.com**. За преобразование между цифровыми и буквенными адресами отвечает **система доменных имен DNS (Domain Name System)**.

В обоих случаях, внутренне IP-адрес представляется **32-битным числом для протокола IPv4** (длина адреса – **4 байта**) или **128-битным числом для протокола IPv6** (длина адреса – **16 байт**).

IP-адреса в Java-программах

IP-адреса в Java-программах представляются посредством класса **InetAddress**, находящегося в пакете **java.net**.

Для получения адреса локальной или удаленной машины следует применять статические методы:

- **getLocalHost()** – возвращает объект **IP-адреса** локальной машины;
- **getByName(String name)** – возвращает объект **IP-адреса** машины по ее имени или десятично-точечной записи адреса.

Замечание: вызовы **getByName("localhost")**, **getByName(null)**, **getByName("127.0.0.1")** являются идентичными вызову **getLocalHost()**.

Участники сетевого взаимодействия: клиент и сервер

Применение IP-адресации позволяет участникам взаимодействия обмениваться данными друг с другом, но для начала диалога им необходимо обнаружить друг друга.

Машина, адрес которой известен до начала диалога, является **сервером**, а машина, инициирующая соединение, является **клиентом**.

Это различие существует только **до начала их взаимодействия**.

После соединения коммуникация протекает **в двух направлениях одновременно**, и уже не играет роли, кто инициировал это соединение.

Идентификация взаимодействующих программ

На каждом из компьютеров может быть запущено несколько приложений, выполняющих процедуру **приема/передачи данных по сети**. Поэтому для их различения дополнительно используется такая характеристика как **номер порта** (изменяющийся в диапазоне от **1** до **65535**).

Все порты разделены на три диапазона:

- **общеизвестные** (или **системные**, **0—1023**),
- **зарегистрированные** (или **пользовательские**, **1024—49151**)
- **динамические** (или **частные**, **49152—65535**).

Идентификация взаимодействующих программ

Для того, чтобы приложения-участники сетевого взаимодействия могли обмениваться данными, между ними должна быть достигнута договоренность об используемых номерах портов.

Для этой задачи наиболее распространенным сетевым сервисам (**HTTP-серверам**, **DNS-серверам**, **почтовым серверам** и т.п.) выделяются определенные номера портов из диапазона от **1** до **1024**, являющегося **служебным**.

Например, для **HTTP-службы** стандартным номером порта является **80**, для почтового протокола – **25** и **115** и т.д. В некоторых случаях приложение может переопределить номер порта, к которому оно привязывается, но это может привести к **неудаче всех попыток соединиться с ним**, если другие участники взаимодействия **не были уведомлены** о подобном изменении.

Концепция сокетов

В низкоуровневом сетевом программировании для взаимодействия сторон применяется концепция **сокетов (гнезд)**, которые **однозначно идентифицируют каждого из участников взаимодействия**.

Сокет можно рассматривать как **пару (адрес машины; порт)**, представляющую **конечную точку соединения**.

В Java сокет создается **для подключения к другому участнику взаимодействия**, после чего из сокета конструируются **потоки ввода** и **вывода**.

Классы для работы с Сокетами

Существуют два класса, ориентированных на потоковую коммуникацию:

- **ServerSocket** (используется для **ожидания входящих соединений**);
- **Socket** (применяется **для инициирования соединений**).

После того, как клиент инициирует соединение через экземпляр класса **Socket**, объект **ServerSocket** возвращает другой экземпляр класса **Socket**, который логически связанный с первым, через метод **accept()**.

Начиная с этого момента с помощью методов **getInputStream()** и **getOutputStream()**, вызываемых для каждого из сокетов, можно получить доступ к **потокам ввода-вывода**.

Пример программы-сервера

```
public class MySHMICQServer {
  public static final int PORT = 4488;
  public static void main(String[] args) throws IOException {
    ServerSocket s = new ServerSocket(PORT);
    try {
      Socket socket = s.accept();
      try {
        BufferedReader in = new BufferedReader(
          new InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(
          new BufferedWriter(
            new OutputStreamWriter(
              socket.getOutputStream())),true);
        while (true) {
          String str = in.readLine();
          if (str.equals("BYE")) break;
          out.println(str);
        }
      } finally { socket.close(); }
    } finally { s.close(); }
  }
}
```

Пример программы-клиента

```
public class MySHMICQClient {
  public static void main(String[] args) throws IOException {
    InetAddress addr = InetAddress.getByName(args[0]);
    Socket socket = new Socket(addr, MySHMICQServer.PORT);
    try {
      BufferedReader in =
        new BufferedReader(
          new InputStreamReader(socket.getInputStream()));
      PrintWriter out =
        new PrintWriter(
          new BufferedWriter(
            new
OutputStreamWriter(socket.getOutputStream())),true);
      out.println(System.in.readLine());
    } finally { socket.close(); }
  }
}
```

Недостатки однопоточного сервера

В рассмотренном примере сервер обладает одним, но весьма существенным **недостатком**:

- он способен обрабатывать одновременно запрос **только одного клиента**.

Для преодоления этого недостатка лучше всего использовать возможности **многопоточных вычислений.**

Как?

- **В главном потоке** происходит только **ожидание подключений клиентов** и **запуск их обработки**.
- **Обработка запросов каждого клиента** должна выполняться в **отдельном потоке**.

Пример многопоточной программы сервера

```
public class MySHMICQServerV2 {
  static final int PORT = 4488;
  public static void main(String[] args) throws IOException {
    ServerSocket s = new ServerSocket(PORT);
    try {
      while(true) {
        Socket socket = s.accept();
        try {
          new MySHMICQWorker(socket);
        } catch(IOException e) {
          socket.close();
        }
      }
    } finally {
      s.close();
    }
  }
}
```

Пример программы потока-обработчика запросов клиентов

```
class MySHMICQWorker extends Thread {
  private Socket socket;
  private BufferedReader in;
  private PrintWriter out;
  public MySHMICQWorker (Socket s) throws IOException {
    socket = s;
    in = new BufferedReader(
      new InputStreamReader(socket.getInputStream()));
    out = new PrintWriter(new BufferedWriter(
      new OutputStreamWriter(socket.getOutputStream())), true);
    start();
  }
  public void run() {
    try {
      while (true) {
        String str = in.readLine();
        if (str.equals("BYE")) break;
        out.println(str);
      }
    } finally {
      socket.close();
    }
  }
}
```

Домашнее задание

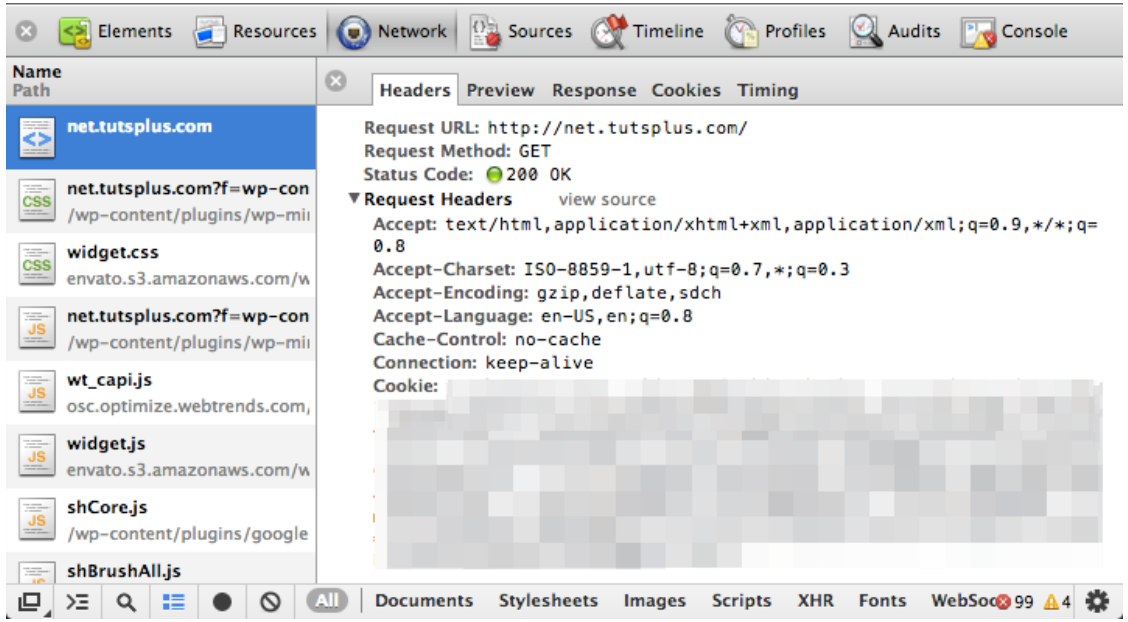
Существует множество инструментов для мониторинга HTTP-трафика (захвата и анализа сетевого трафика):

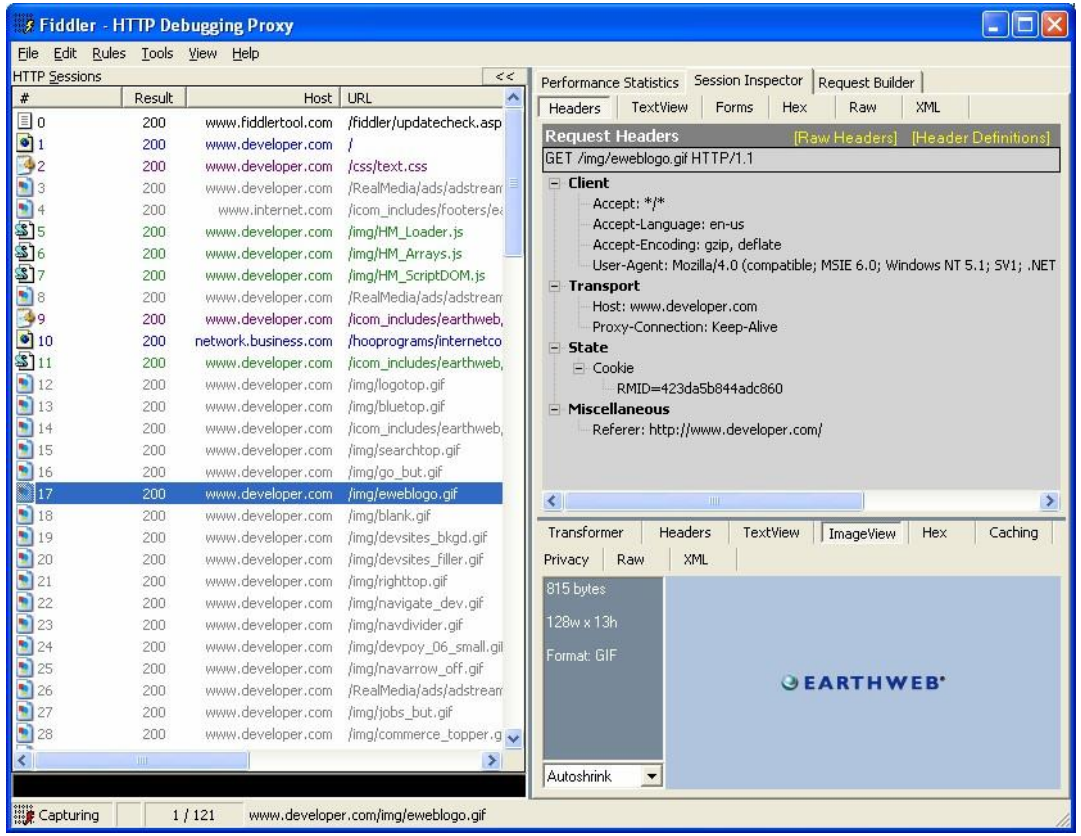
- **Chrome Developers Tools**
- **Fiddler**
- **WireShark**

Задача:

Используйте данные инструменты для анализа домашнего сетевого траффика. И подготовьте отчет к зачету в любой удобной для Вас форме.

Домашнее задание





На сегодня все!
Увидимся на зачете!