**Работа с сетевой моделью на транспортном уровне** (TCP и UDP) осуществляется за счет пакетов .net и .nio, и соответствующие им классы:

*В пакете .net:* Socket/ ServerSocket – протокол TCP DatagramSocket – протокол UDP

*В пакете .nio:* SocketChannel/ServerSocketChannel – протокол TCPDatagramChannel – протокол UDP

Классы описывают взаимодействие на транспортном уровне, не определяя стандарты взаимодействия для уровней выше.

|  |  |
| --- | --- |
| TCP | UDP |
| public class TCPClient {   public static void main(String[] args) throws IOException {  InetAddress address = Inet4Address.*getByName*("localhost"); //Класс для представления IP адресов  try (Socket socket = new Socket(address, 7777);  var outputStream = new DataOutputStream(socket.getOutputStream());  var inputStream = new DataInputStream(socket.getInputStream())) { //Созданеие клиентского сокета и получение потоков ввода и вывода сокета //для отправки запроса на сервер и получения ответа с помощью соответствующего потока  outputStream.writeUTF("Hello from client"); //отправка сообщения на сервер  System.*out*.println(inputStream.readUTF()); //обработка ответа от сервера   }  } } | public class UDPClient {   public static void main(String[] args) throws IOException {   try (DatagramSocket socket = new DatagramSocket()) {  //использует Datagram пакеты для отправки и получения сообщений, а не потоки  byte[] buffer = "Hello from UPD client".getBytes();  InetAddress address = Inet4Address.*getByName*("localhost");   DatagramPacket pack = new DatagramPacket(buffer, buffer.length, address, 7777);  //адрес передачи определяется в пакете, а не в сокете (в случае клиента)  // => используя один сокет можно отправлять пакеты разным серверам   socket.send(pack);  }  } } |
| public class TCPServer {   public static void main(String[] args) throws IOException {   try (ServerSocket serverSocket = new ServerSocket(7777);  Socket socket = serverSocket.accept(); //получение сокета клиента  DataOutputStream outputStream = new DataOutputStream(socket.getOutputStream());  DataInputStream inputStream = new DataInputStream(socket.getInputStream())) { //получение потоков клиентского сокета  System.*out*.println(inputStream.readUTF()); //обработка обращения  outputStream.writeUTF("Hello from server"); //Отправка ответа от сервера  }  } } | public class UDPServer {   public static void main(String[] args) throws IOException {  try (DatagramSocket socket = new DatagramSocket(7777)) { // в случае сервера нужно указывать порт  byte[] buf = new byte[512];  DatagramPacket pack = new DatagramPacket(buf, buf.length); //адрес передавать не нужно  socket.receive(pack); //получение переданного пакета данных  System.*out*.println(new String(pack.getData()));  }  } } |
| Блокирующие – сервер ожидает получения пакета, клиент ожидает ответа от сервера, если это TCP.  Для получения ответа от сервера необходимо на стороне сервера сформировать новый пакет и передать его клиенту (то есть клиент и сервер поменяются местами)  TCP и UPD порты – разные сущности, поэтому могут повторяться. | |

**Классы прикладного уровня (для клиента).**

Классы URL и классы пакета .net.http используются для работы с соединением на прикладном уровне и необходимы для клиентских Http-запросов к серверу.

URL предоставляет базовый функционал для работы с http – предоставляет минимальный набор функций для создания и управления http запросами.

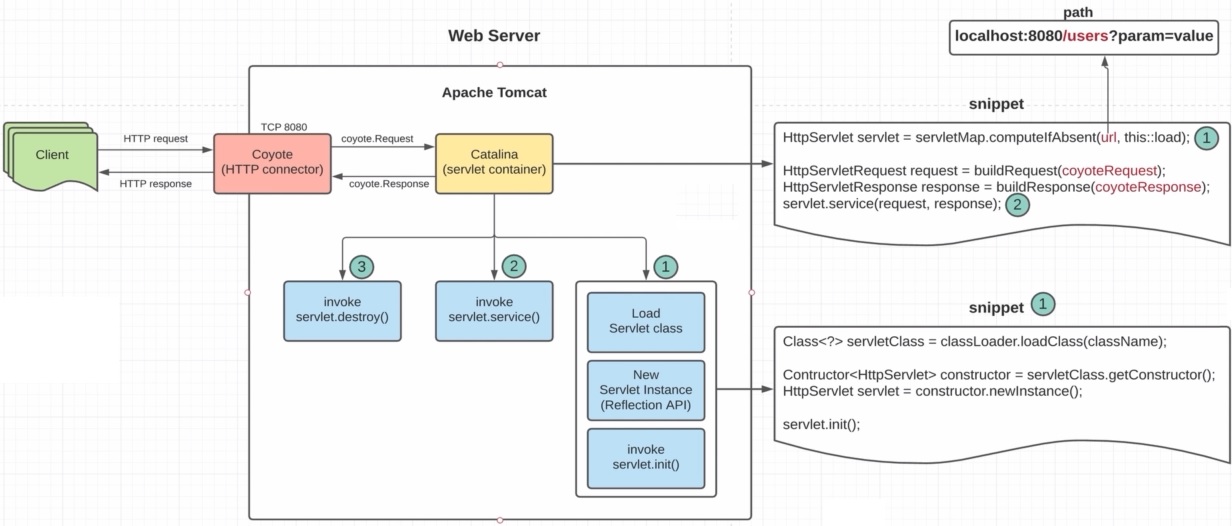
Классы пакета .net.http (HttpClient/HttpRequest/HttpResponse и др.) предоставляют более удобный API (декларативный стиль), гибкие настройки http запроса (легко сконфигурировать http запрос), поддержку асинхронных запросов (не блокирует основной поток выполнения - completableFuture).

**Apache TomCat** – веб-сервер и контейнер сервлетов. Состоит из модулей:

Catalina – контейнер сервлетов. Управляет жизненным циклом сервлетов, обеспечивает взаимодействие coyote и jasper (является промежуточным звеном, которое перенаправляет запрос из coyote в jasper).

Coyote – Http connector – принятие/отправка запросов/ответов через http соединение (промежуточное звено, которое принимает запрос и передает его Catalina для обработки, также принимает ответ от Catalina и передает его клиенту).

Jasper – JSP движок – отвечает за компиляцию JSP (Java service pages) страниц в сервлеты и их последующее выполнение.



**Обработка http запроса:**

Http запрос поступает от клиента на коннектор Coyote (работает на транспортном уровне, используя классы Socket и ServerSocket), который принимает эти запросы от клиентов и оборачивает его в coyote.Request, что дает возможность абстрагироваться от деталей протокола HTTP (например унифицировать обработку запроса независимо от версии HTTP) и стандартизировать запрос, что дает возможность работать с запросом независимо от его свойств. Далее запрос принимает Catalina и так же оборачивает request и response для дальнейшей передачи в метод service.

Catalina представляет из себя ассоциативный массив, где URL (путь в нем) – ключ, а значение определенный сервлет, который должен отрабатывать по данному пути в URL. Если объект сервлета еще не был создан, то запускается механизм инициализации: загрузка класса с помощью ClassLoader, создание инстанса сервлета с помощью механизма рефлексии, вызов метода init.

Сервлет реализует концепцию существования единственного объекта, т.к. каждому сервлету соответствует определенный путь в URL (не является singleton, т.к. имеет открытый конструктор, создание объекта контролирует контейнер сервлетов, а не сам класс)

**Конфигурация сервлетов:**

Для конфигурации сервлетов используется дескриптор развертывания – web.xml (устаревший вариант) либо аннотации в коде.

Оба способа позволяют конфигурировать маппинг сервлета на URL, определять параметры инициализации и т.д.

**Запуск и развертывание**

Webapps – директория для развертывания веб-проекта, при добавлении war-архива в эту папку происходит ее развертывание для возможности дальнейшего использования, добавление war-архива в директорию ROOT приводит к тому, что обращение к ресурсу будет доступно по корневому URL.

Запуск и остановка TomCat осуществляется в директории bin (startup.bat, shutdown.bat)

Запуск и развертывание возможны с помощью IDE – edit configuration. Для ускорения развертывания можно использовать war-exploded (распакованная версия).

**Сервлет** – java программа, которая работает на серверной части приложения, обрабатывает запросы от клиента и формирует динамический ответ.

**ЖЦ сервлетов**:

1) Инициализация при вызове метода init() – вызывается контейнером сервлетов только 1 раз – происходит создание сервлета и инициализация необходимых для него ресурсов (создается либо при первом обращении к сервлету, либо при запуске сервера), после его создания контейнером сервлетов выделяются потоки исполнения для каждого нового клиентского запроса.

2) Обработка клиентского запроса вызовом метода service() – определение типа HTTP запроса и перенаправление его соответствующему http методу (doGet, doPut и др)

3) Уничтожение сервлета вызовом метода destroy() – вызывается контейнером сервлетов и выполняет операции по освобождению ресурсов перед уничтожением сервлета.

4) Полное уничтожение сервлета сборщиком мусора Java (Garbage Collector).

**Алгоритм работы:**

1) Унаследоваться от HttpServlet (библиотека jakarta) и переопределить основные методы жизненного цикла сервлета (init, service, destroy) – метод service() не переопределяется напрямую, так как в суперклассе определено поведение по умолчанию для него (определяет тип запроса и вызывает соответствующий метод doX, у которых также определено поведение по умолчанию в суперклассе), вместо него необходимо переопределить соответствующие методы doGet, doPost и т.д.

2) Должен присутствовать конструктор по умолчанию, для возможности использовать рефлексию.

3) Замапить сервлет на URL (с помощью аннотаций или web.xml)

**Работа с заголовками (**Headers**)**:

Получение заголовков http запроса с помощью объекта HttpServletRequest req и вызов у него методов по ключ слову Header (геттеры)

Отправка заголовков http ответа с помощью объекта HttpServletRequest resp и вызов у него методов по ключ слову Header (сеттеры и геттеры).

**Обработка параметров запроса:**

*Передача через URL (GET-запрос):*

example.com?id=123&id=234&pass=some – отделяются от пути знаком вопроса, между собой разделены &, у одного параметра могут быть несколько значений.

Параметры такого вида передаются в GET-запросе и обрабатываются в сервлете посредством методов типа getParameter();

*Передача через тело запроса (POST) – относится к работе с телом запроса, что подразумевает передачу параметров:*

**form-data –** Используется как подтип MIME-type multipart с заголовком Content-type. Позволяет передавать несколько различных типов данных в теле одного сообщения посредством объявления и использования разделителя (boundary).

Каждая часть данных имеет свой заголовок, который включает в себя метаданные содержимого в виде заголовков и передаваемый контент.

Пример:

POST /upload HTTP/1.1  
 Host: example.com  
 Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW  
  
------WebKitFormBoundary7MA4YWxkTrZu0gW  
 Content-Disposition: form-data; name="username"  
  
 JohnDoe  
------WebKitFormBoundary7MA4YWxkTrZu0gW  
 Content-Disposition: form-data; name="avatar"; filename="avatar.jpg"  
 Content-Type: image/jpeg  
  
 (здесь находится содержимое файла avatar.jpg в виде бинарных данных)  
------WebKitFormBoundary7MA4YWxkTrZu0gW--

**x-www-form-urlencoded** – используется как подтип MIME-type application с заголовком Content-Type. При использовании этого заголовка в теле запроса POST передаются параметры запроса, в виде аналогичном URL-параметрам.

Пример:

POST /submit-form HTTP/1.1  
Host: example.com  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 44  
Authorization: Bearer your\_access\_token  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36

username=johndoe&password=123456&rememberme=true

**raw –** для передачи данных в их исходном формате, используется для передачи данных текстового формата в теле запроса (text, JavaScript, JSON, HTML, XML).

**Binary –** для передачи данных в бинарном формате (изображения, аудио, видео)

**graphQL**

Для обработки тела запроса у HttpServletRequest req вызываются методы getReader или getInputStreamReader, в зависимости от получаемых данных. Для обработки x-www-form-unlencoded можно использовать методы типа getParameter()

**Загрузка файла с сервера:**

Для инициализации загрузки контента сервером на стороне клиента необходимо установить заголовки Content-Type, Content-length и заголовок Content-Disposition attachment; filename=xxx, который и будет указывать на то, что отправленный контент не нужно отображать, а необходимо сохранить в виде файла.

*Способы передачи:*

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 resp.setContentType("application/json");  
 resp.setCharacterEncoding("UTF-8");  
 resp.setHeader("Content-Disposition", "attachment; filename=employees.json");  
  
 //Так как приложением является TomCat, а текущий проект является просто war архивом,  
 // то для корректной отправки ресурсов необходимо указывать либо полный путь к ресурсам,  
 // либо пометить нужную директорию как resources root и получить stream этого ресурса с помощью  
 // FirstServlet.class.getResourceAsStream("/employees.json");  
  
 //с помощью указания полного пути  
  
 var writer = resp.getOutputStream();  
 writer.write(Files.*readAllBytes*(Path.*of*("C:\\Users\\dimir\\IdeaProjects\\demo\\src\\main\\resources", "employees.json")));  
  
 //с помощью стрима  
 InputStream resourceAsStream = FirstServlet.class.getResourceAsStream("/employees.json");  
 var stream = resp.getOutputStream();  
 stream.write(resourceAsStream.readAllBytes());  
}

**Работа с cookies:**

**Cookies –** механизм для хранения состояния и информации о пользователе на стороне клиента.

На уровне http-сообщения серверу необходимо передать в ответе заголовок Set-cookie собязательнымипараметрами ключ-значение. Для более гибкого контроля над поведением cookie возможна установка опциональных параметров типа:

*Expires* (устанавливает дату истечения срока действия cookie – по умолчанию в течение всей сессии).  
*Max-Age* (время жизни в секундах, если установлен и Expires и Max-age, то приоритет на Max-age).  
*Domain* (домен для которого cookie должны быть отправлены, по умолчанию доступны для домена установившего cookie).  
*Path* (устанавливает путь для которого cookie должны быть доступны, по умолчанию для пути установившего cookie).  
*Secure* (только для https)  
*HttpOnly* (доступны только для http и недоступны для JavaScript)  
SameSite (контроль над cookie при крос-сайтовых запросах)

Средствами Java работа с cookie осуществляется с помощью класса Cookie, который позволяет конфигурировать отправляемый cookie методами типа set и получать клиентские с помощью методов типа get.

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
 Cookie[] cookies = req.getCookies(); //получение клиентских cookie  
  
 Cookie cookie = new Cookie("name", "value"); //установка обязательных параметров  
 cookie.setMaxAge(20); //установка доп. параметра cookie  
 resp.addCookie(cookie); //установка cookie в ответное сообщение от сервера  
}

**Работа с сессией**

*Сессия в контексте соединения* – время, в течение которого между двумя узлами поддерживается связь для обмена данными.

*Сессия в контексте клиент-серверной архитектуры* – описание текущего состояния взаимодействия клиента и сервера, сохраненное на стороне сервера, которое осуществляется путем идентификации пользователя и объединения всех его запросов в единый процесс.

*Поддержание сессии –* это способ переслать серверу уникальный session id, которому соответствует объект типа Session на стороне сервера.

Способы:

1) Аутентификация – для подтверждения себя пользователю необходимо вводить логин и пароль. Неудобно каждый раз логиниться.

2) Скрытое HTML поле – сервер отправляет session id путем внедрения его в HTML (добавляется скрытое поле, которое не рендерится браузером). Небезопасно, т.к. хранится в открытом виде.

3) URL Rewriting – передается с помощью параметра URL запроса. Небезопасно – передается открыто.

4) Cookies – передается с помощью cookie

5) Java Session Management API – механизм управления интернет-сессией (выбирает механизм управления между URL Rewriting или cookies, если отключены cookies) – TomCat использует этот механизм.

Взаимодействие с сессией осуществляется с помощью метода getSession() – позволяет настраивать атрибуты сессии (данные ассоциированные с пользовательской сессией) и время ее жизни.

Сессия на примере работы TomCat – если запрос от клиента инициируется впервые, то создается объект Session, генерируется session id и атрибуты сессии (Атрибуты используются для внутренней коммуникации сервлетов) сессия кладется в Map<String, Session>, где Sting – сгенерированный session id, этот id передается на клиент в виде cookie, который клиент затем передает на сервер, что позволяет ассоциировать его с определенной сессией.

*3 уровня атрибутов:*

ServletContext – глобальные настройки для приложения, доступны всем сервлетам, фильтрам, слушателям, сессиям.

HttpSession – доступны в рамках сессии.

ServletRequest – в рамках http запроса – используются при перенаправлении.

**Работа с файлами**

Для получения тела запроса используется getInputStream(), в таком случае будет взят весь контент, помещенный в тело запроса.

**getPart() –** Используется для получения части запроса из его тела, при использовании contentType multipart/form-data. Возвращает объект Part, который содержит все параметры полученной части запроса.

Для использования нужно использовать аннотацию @MultipartConfig(параметры)

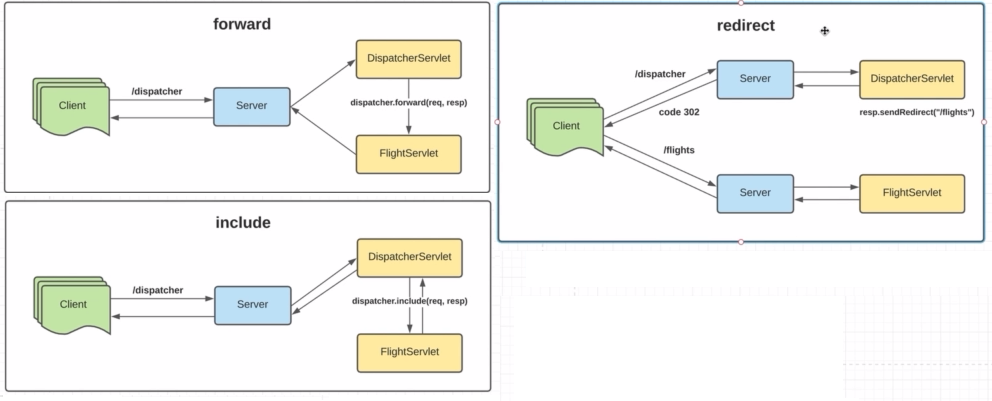
**Перенаправление запроса:**

Перенаправление может быть **серверное –** выполняется на стороне сервера (клиент не участвует в перенаправлении – не происходит перехода на другой URL). Используется для перенаправления запроса на другой сервлет в рамках одного приложения.

Серверное перенаправление осуществляется за счет интерфейса RequestDispatcher и его методов forward() и include().

forward() – передает управление указанному сервлету или другому серверному ресурсу (например, JSP странице). Новый ресурс обрабатывает запрос и формирует ответ для клиента. Оригинальный запрос и ответ передаются новому ресурсу.

include() – запрос передается на указанный ресурс, который обрабатывает запрос и передает результат обратно вызывающему сервлету, который включает результат его работы в свой ответ клиенту.



**Клиентское** – выполняется на стороне клиента. Сервер направляет сообщение о необходимости перехода на другой URL, что приводит к новому HTTP-запросу. Используется для перенаправления запроса на сервлет другого приложения или другой сервер.

Клиентское перенаправление осуществляется за счет resp.sendRedirect().

sendRedirect() – формирование ответа для клиента с кодом ответа 302 – redirect, что информирует его о необходимости перехода на указанный URL.

**JSP (Java Server Pages) –** технология для динамического создания веб-страниц, может содержать как статические (HTML), так и динамические компоненты (JSP-элементы).

JSP–страницы транслируются модулем Jasper в Java-классы, которые представляют из себя сервлеты.

**Директивы JSP**- управляют общими настройками и конфигурацией JSP страницы. Они предоставляют инструкции контейнеру JSP о том, как обрабатывать и управлять страницей**:**

*<%@page* – метаданные, конфигурация текущей JSP страницы (импорты, тип контента и др.)

*<%@Include* – для включения содержимого другой JSP-страницы в текущую

*<%@taglib* – подключение библиотеки тегов к JSP странице (тэгов, которые не относятся к html)

**JSP-элементы:**

*Скриптлеты* <% %> - для вставки многострочного Java-кода на стороне сервера.  
*Выражения* <%= %> - для вывода значений Java-выражений.   
*Декларации* <%! %> - для объявления классов, методов и полей.   
*Комментарии* <%-- --%> - для вставки комментариев в код.

\*Использование Скриплетов, выражений и деклараций в JSP не является хорошей практикой, так как в таком случае логика приложения перетекает на слой View, что нарушает MVC.

*EL (expression language) выражения* - ${} - является альтернативным и предпочтительным способом обращения с данными в JSP, т.к. при таком способе логика сервера не пишется в JSP, а происходит обращение к нужным переменным, которые содержат данные.

Контекст запроса JSP страница может получить с помощью различных видов scope (application/request/session/page) – и дополнительных объектов, определенных в requestScope – parameter, cookie, header.

Scope – используются для хранения и обмена данными между компонентами приложения (в частности между сервлетами). Контекст (данные) устанавливается посредством добавления атрибута (setAttribute()) к определенной области видимости.

${requestScope.user.name} – область видимости, получить объект по ключу, поле объекта (У поля должен быть геттер, иначе отвалится – Java Beans)

public class User {  
 private String name;  
   
 public String getName() {  
 return name;  
 }

${requestScope.SomeUser.name}

req.setAttribute("SomeUser", user);

*Теги JSTL* – библиотека стандартных тэгов для исполнения базовых задач на стороне jsp (некоторые функции, циклы, условия и др.). Используется для уменьшения и упрощения кол-ва кода на стороне JSP (для tomcat 10 2 зависимости: jakarta.servlet.jsp.jstl, jakarta.servlet.jsp.jstl-api)

@WebFilter()  
Сервлет-фильтр – сервлеты, реализующие интерфейс Filter, позволяют перехватывать запросы до их обработки сервлетом и модифицировать ответы после обработки сервлетом. Жизненный цикл схож с сервлетом, но вместо service() метода doFilter().

Цепочки фильтров: Использование аннотаций не гарантирует порядок вызова фильтров. Цепочки фильров создаются за счет указания адреса

Req – прямой порядок, resp – обратный порядок

Применение: установка глобальных атрибутов, логирование, аутентификация, шифрование

URL паттерн (“someURL/\*”)

Переменные метода doFilter представленные базовым классом ServletRequest, их можно приводить к HttpServletRequest, объект FilterChain для передачи запроса дальше по цепочке фильтров