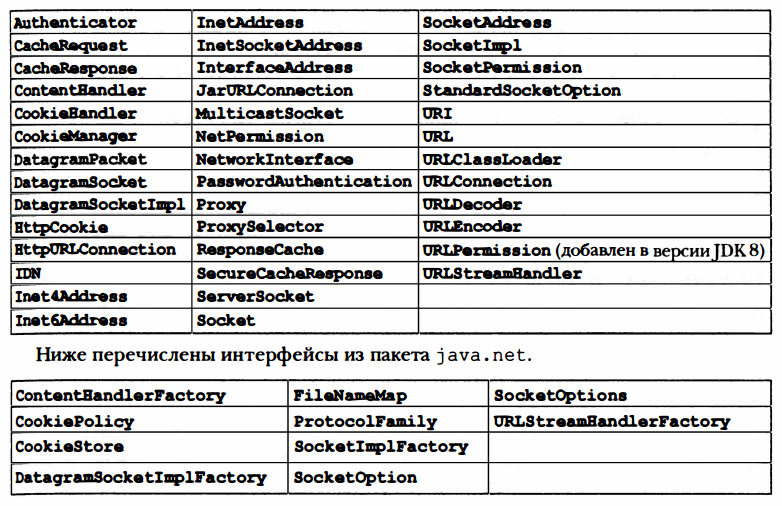
**java.net**

****

**Сокет -** кінечна точка в мережі. Дозволяє одночасно обслуговувати декілька клієнтів в мережі надаючи їм інформацію різного роду. Зв’язок між сокетами відбувається за допомогою визначеного мережевого протоколу.

**Порт -** нумерований сокет на окремій машині. Серверний процес прослуховує порт поки клієнт не з’єднається з ним. Сервер може прийняти багато різних запитів на один порт при чому всі ці запити будуть індивідуальними. Грубо кажучи, порти - це різного роду програми які запущені на ПК і чекають поки їх викличуть.

**Протокол IP (Internet Protocol)** - низкорівневий маршрутизований мережевий протокол, який розбиває данні на невеликі пакети і відправляє їх по мережі за певним адресом. Не гарантує доставки всіх пакетів, іноді може спостерігатися втрата пакетів.

**Протокол TCP (Transport Control Protocol) -** мережевий протокол більш високого рівня, який забезпечує зв’язання, сортування і повторну передачу пакетів аби удостовіритися у їх надійній доставці.

**Протокол UDP (User Datagram Protocol)** - протокол більш нижчого рівня ніж TCP. Використовується для швидкої, не потребуючої постійного підключення, передачі даних. В силу цього - не надійний.

Як тільки з’єднання встановлено, в дію вступає високорівневий протокол, тип якого залежить від порта який використовується. Протокол TCP/IP резервує перших 1024 порта для окремих протоколів (Наприклад: 21 - FTP, 23 - Telnet, 25 - email, 80 - HTTP і т.д.).

**Протокол HTTP** - використовується для передачі гіпертексту і графічних зображень. Наприклад, коли клієн запитує файл у HTTP-сервера, він (клієнт) відправляє його ім’я в визначений порт, сервер відповідає і клієнт читає зміст файла. Сервер також відправляє код стану щоб дати знати клієнту чи був обслужений запит і причину, якщо той був не обслужений.

Кожен комп’ютер в інтернеті має свій адрес. Схеми адресації є IPv4, IPv6 (має більший адресний простір) - для мережевого програмвання не варто заморочуватися яку схему використовувати бо проблема вирішується автоматично.

**Доменне ім’я** - місцезнаходження сайту інтернета (www.herbertshieldt.com). Взаємодією (зпівставленням) між доменним ім’ям і ІР-адресом займається служба DNS (Domain Name Service).

**Маска підмережі** - визначення підмережі ІР адрес. Наприклад, за допомогою маски підмережі можна сказати, що один діапазон ІР адресів знаходиться в одній підмережі, а інший - в іншій. Например, узел с IP-адресом 12.34.56.78 и маской подсети 255.255.255.0 находится в сети 12.34.56.0 с длиной префикса 24 бита.

**Клас InetAddress** - клас, який інкапсулює (скриває в собі) ІР-адрес і доменне ім’я. Для взаємодії з даним класом використовується ім’я ІР-хоста (**вузла мережі**), що є зручніше і зрозуміліше ніж використовувати числовий ІР-адрес. Може керувати адресами по протоколах IPv4 і IPv6. В цьому класі відсутні доступні конструктори. Для того аби створити екземпляр даного класу використовують фабричні методи.

**Фабричний метод** - це згода, за якою статичні методи класа повертають екземпляр цього класа. Це робиться замість перегрузки конструкторів з різними параметрами на користь викорестання імен методів які більш проясняють результат, який ми хочем отримати.

Найбільш вживані фабричні методи класу InetAddress:

**static InetAddress getLocalHost()** - повертає локальний хост. throw UnknownHostException.

**static InetAddress getByName(String hostName)** - повертає локальний хост по імені. throw UnknownHostException.

**static InetAddress[] getAllByName(String hostName)** - повертає в масив всі адреса в які перетворюється конкретне ім’я. В інтернеті одне ім’я може вказувати на декілька машин за рахунок чого веб-сервера можна масштабувати. throw UnknownHostException.

Приклад:

**import** java.net.\*;

**public** **class** InetAddressTest {

**public** **static** **void** main(String[] args) **throws** UnknownHostException{

InetAddress Address = InetAddress.*getLocalHost*();

System.***out***.println(Address);

Address = InetAddress.*getByName*("www.devid.info");

System.***out***.println(Address);

InetAddress[] SW = InetAddress.*getAllByName*("www.google.com.ua");

**for**(**int** i = 0; i < SW.length; i++){

System.***out***.println(SW[i]); } } }

Ще деякі методи класу InetAddress:

**boolean equals(Object obj)** - повертає істину, якщо ІР-адреси порівнюваних об’єктів одинакові.

**byte[] getAddress()** - повертає масив байтів, який представляє ІР-адрес в порядку послідовності байтів в мережі.

**String getHostAddress()** - повертає строку яка містить адрес хоста пов’язану з об’єктом типу InetAddress.

**String getHostName()** - повертає строку, яка містить ім’я хоста пов’язаного з об’єктом класу InetAddress.

**boolean isMulticastAddress()** - повертає істину, якщо адрес являється груповим.

**String toString()** - повертає строку яка містить одразу і ІР-адрес, і ім’я хоста.

**Класи Inet4Address і Inet6Address -** підкласи класа InetAddress які підтримують адреси протоколів IPv4 і IPv6. Але як праивло, клас InetAddress можна використовувати на пряму, оскільки він приспособлений для роботи по цим двом протоколам.

**Клас URL -** уніфікований вказівник ресурса.Специфікація URL адресу складається з 4-х частин. http://www.MHProfessional.com чи http://www.MHProfessional.com:80/index.html. Перша частина визначає мережевий протокол (HTTP, FTP, file, gopher), що відділяється ":" від іншої частини адреси. Правда більшість браузерів правильно інтерпритує адресу, навіть, якщо виключити з неї вказання протоколу http://. Друга частина: ІР-адреса або доменне ім’я хоста (відділяється зліва //, а зправа / чи :). Третя частина: номер порта. (необов’язковий. Відділяється зліва :, а зправа /). Четверта частина визначає дійсних шлях до ресурсу. (index.html. На більшості серверах прив’язана автоматично і не потребує фактичного вказання). URL є одноразовим, тобто ми не зможемо на ходу поміняти протокл, порт і т.п. Конструктори (можуть генерувати MalformedURLException):

**URL (String address)** - повертає об’єкт URL по адресу.

**URL (String protoclName, String hostName, int port, String path)** - те ж саме, тільки розбиває URL на частини.

**URL (String protocоlName, String hostName, int port)** - те ж саме, тільки розбиває URL на частини.

**URL(URL baseURL, String relativeAddress)** - вказує існуючий URL в якості базового, а потім створити з цього URL базового новий URL. Конкатенує відносний шлях з базовим, наприклад. ігнорує базовий, якщо в відносному вказано абсолюний шлях.

**snippet** - фрагмент, відривок. Фрагмент коду, який пригодний для повторного використання.

address чисто називають специфікатором URL.

**import** java.net.\*;

**public** **class** URLDemo {

**public** **static** **void** main(String[] args) **throws** MalformedURLException {

URL hp = **new** URL("http://www.herbschildt.com/articles");

System.***out***.println("Protocol: " + hp.getProtocol());

System.***out***.println("Port: " + hp.getPort());

System.***out***.println("Host: " + hp.getHost());

System.***out***.println("File: " + hp.getFile());

System.***out***.println("Full info: " + hp.toExternalForm()); } }

Пробіл в адресі:

URL url = new URL("http://example.com/hello%20world");

Якщо нам потрібно до такого вигляду привести декілька символів в адресі, то варто використати клас URI:

URI uri = new URI("http", "example.com", "/hello world/", "");

І конвертувати його в URL.

URL url = uri.toURL();

Приклад читання з ЮРЛ в консоль (Оракл):

**public class** URLTest {  
 **public static void** main(String[] args) **throws** IOException {  
 String urlAddress = **"http://oracle.com/"**;  
 URL oracleURL = **new** URL(urlAddress);  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(oracleURL.openStream()));  
  
 String inputString;  
 **while**((inputString = in.readLine()) != **null**){  
 System.***out***.println(inputString);  
 }  
 in.close();  
 }  
}

Після того, як ми успішно стоврили об’єкт URL то ми можемо викликати метод openConnection() щоб отримати об’єкти типу URLConnection чи якийсь з підкласів, напрклад, HttpURLConnection. Цей об’єкт URLConnection можна використати для налаштування параметрів і властивостей запитів які нам потрібні перед з’єднанням. З’єднання з віддаленим об’єктом за допомогою URL ініційоване тільки тоді, коли викликано метод URLConnection.connect() - коли ми це зробимо, ми ініціалізуємо з’єднання між Java програмою і URL в мережі. Приклад:

**try** {  
 String urlAddress = **"http://oracle.com/"**;

//create url  
 URL oracleURL = **new** URL(urlAddress);

//open url connection  
 URLConnection myConnection = oracleURL.openConnection();

//connect  
 myConnection.connect();  
  
}**catch**(MalformedURLException e){  
 e.printStackTrace();  
}**catch** (IOException e2){  
 e2.printStackTrace();  
}

Об’єкт URLConnection буде створюватися кожного разу, коли буде викликатися метод openConnection().

Тепер, коли URL з’єднання встановлено, можна використовувати об’єкт URLConnection для дій читання чи запису в з’єднання.

**Клас URLConnection** - призначений для доступу до атрибутів віддаленого хоста. Тобто, як тільки буде встановлено з’єднання з сервером, клас URLConnection дасть змогу продивитися атрибути об’єкта перш ніж ми перенесемо його локально. НТТР орієнтований. Методи:

**int getContentLength()** - повертає довжину змісту ресурса в байтах. Повертає -1, якщо довжина недоступна.

**String getContentType()** - повертає тип змісту ресурса. null - якщо тип недоступний.

**long getDate()** - повертає час і дату в мілісекундах з 1 січня 1970 року.

**long getExpiration()** - повертає час і дату сроку дії русурса в мілісекундах з 1970 року.

**String getHeaderField(int index) -** повертає значення заголовку по вказаному індексу. null якщо значення індексу перевищує кількість полів.

Заголовки складаються з пар ключ-значення.

**Map<String, List<String>> getHeaderFields()** - повертає відображення з всіма полями заголовка.

**long getLastModified()** - повертає час і дату останньої модифікації ресурса в мілісекундах.

**InputStream getInputStream()** - повертає поток вводу прив’язаний до ресурсу. Цей поток може використовуватися для отримання змісту ресурса.

Приклад:

**import** java.net.\*;

**import** java.io.\*;

**import** java.util.Date;

**public** **class** URLConnectionDemo {

**public** **static** **void** main(String[] args) {

**int** charForReading;

**try**{

URL link = **new** URL("http://www.internic.net/");

URLConnection URLLink = link.openConnection();

//get Date

**long** date = URLLink.getDate();

**if**(date==0)

System.***out***.println("Missing data.");

**else**

System.***out***.println("Data: " + **new** Date(date));

//get content type

System.***out***.println("Content type: " + URLLink.getContentType());

//get Data

date = URLLink.getExpiration();

**if**(date==0)

System.***out***.println("Missing data.");

**else**

System.***out***.println("Expired period: " + **new** Date(date));

//get last modified data

date = URLLink.getLastModified();

**if**(date==-1)

System.***out***.println("Missing data.");

**else**

System.***out***.println("Last modified time: " + **new** Date(date));

//get the size of content

date = URLLink.getContentLength();

**if**(date==-1)

System.***out***.println("Missing data.");

**else**

System.***out***.println("Content size: " + date);

**if**(date!=0){

System.***out***.println("Content: ");

InputStream inputStream = URLLink.getInputStream();

**while**((charForReading = inputStream.read()) != -1){

System.***out***.print((**char**) charForReading);

}

}

}**catch**(Exception e){

System.***out***.println("Error. Pls try again. Error message: " + e.getMessage());

}

}

}

Приклад програми читання безпосередньо з ЮРЛ наведеної вище, але з використанням URLConnection. Виводить той же результат (ісходнік оракл.ком), тільки замість того щоб брати вхідний потік InpuStream одразу з ЮРЛ, ця програма створює спочатку об’єкт URLConnection і знього бере InpuStream. Можна використовувати обидва варіанти для читання, але варінт з URLConnection є більш функціональним бо можна використовувати цей об’єкт і для інших задач одночасно:

**public class** URLTest {  
 **public static void** main(String[] args) **throws** IOException {  
 BufferedReader in = **null**;  
 **try** {  
 String urlAddress = **"http://oracle.com/"**;

//create url  
 URL oracleURL = **new** URL(urlAddress);

//open connection  
 URLConnection myConnection = oracleURL.openConnection();

//buffered reader from myConnection IS  
 in = **new** BufferedReader(**new** InputStreamReader(myConnection.getInputStream()));  
  
 String inStr;  
 **while**((inStr = in.readLine()) != **null**){  
 System.***out***.println(inStr);  
 }  
 }**catch**(MalformedURLException e){  
 e.printStackTrace();  
 }**catch** (IOException e2){  
 e2.printStackTrace();  
 }**finally** {  
 in.close();  
 }  
  
 }  
}

Для Java програми щоб взаємодіяти з процесами серверної сторони потрібно просто "писати" в ЮРЛ, таким чином забезпечуючи передачу даних на сервер. Це виконується наступною послідовністю:

1. Створити URL.

URL url = new URL(args[0]);

2. Отримати інстанс URLConnection.

URLConnection connection = url.openConnection();

3. Задати URLConnection властивість вихідного потоку.

connection.setDoOutput(true);

4. Відкрити конект до ресурсу.

5. Отримати вихідний потік з конекту.

OutputStreamWriter out = new OutputStreamWriter(connection.getOutputStream());

//метод getOutputStream throws UnknownServiceException

6. Писати в вихідний потік.

out.write("string=" + stringToReverse);

7. Закрити вихідний потік.

out.close();

Приклад невеликого сервлета (чи cgi-bin скрипта) і програми з дотриманням цієї послідовності:

Програма:

**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.io.OutputStreamWriter;  
**import** java.net.URL;  
**import** java.net.URLConnection;  
**import** java.net.URLEncoder;  
  
**public class** Reverse {  
 **public static void** main(String[] args) **throws** IOException {  
 String string = **"Hello"**;  
 String stringToReverse = URLEncoder.*encode*(string, **"UTF8"**);  
  
 String servletAddress = **"https://docs.oracle.com/javase/tutorial/networking/urls/examples/ReverseServlet.java"**;  
 URL servletURL = **new** URL(**"servletAddress"**);  
 URLConnection connection = servletURL.openConnection();  
 connection.setDoOutput(**true**);  
  
 *//get OS from URLConnection* OutputStreamWriter outputStreamWriter = **new** OutputStreamWriter(connection.getOutputStream());  
 outputStreamWriter.write(stringToReverse);  
 outputStreamWriter.close();  
  
 *//get IS from URLConnection* BufferedReader in = **new** BufferedReader(**new** InputStreamReader(connection.getInputStream()));  
 String decodedString;  
 **while**((decodedString = in.readLine()) != **null**){  
 System.***out***.println(decodedString);  
 }  
 in.close();  
 }  
}

Якщо в строці запиту є пробіли чи недозволені для строки запиту символи - використовується URLEncoder.

String stringToReverse = URLEncoder.*encode*(string, **"UTF8"**);

Сервлет, яким можна протестувати програму:

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

import java.net.\*;

public class ReverseServlet extends HttpServlet

{

private static String message = "Error during Servlet processing";

public void doPost(HttpServletRequest req, HttpServletResponse resp) {

try {

int len = req.getContentLength();

byte[] input = new byte[len];

ServletInputStream sin = req.getInputStream();

int c, count = 0 ;

while ((c = sin.read(input, count, input.length-count)) != -1) {

count +=c;

}

sin.close();

String inString = new String(input);

int index = inString.indexOf("=");

if (index == -1) {

resp.setStatus(HttpServletResponse.SC\_BAD\_REQUEST);

resp.getWriter().print(message);

resp.getWriter().close();

return;

}

String value = inString.substring(index + 1);

//decode application/x-www-form-urlencoded string

String decodedString = URLDecoder.decode(value, "UTF-8");

//reverse the String

String reverseStr = (new StringBuffer(decodedString)).reverse().toString();

// set the response code and write the response data

resp.setStatus(HttpServletResponse.SC\_OK);

OutputStreamWriter writer = new OutputStreamWriter(resp.getOutputStream());

writer.write(reverseStr);

writer.flush();

writer.close();

} catch (IOException e) {

try{

resp.setStatus(HttpServletResponse.SC\_BAD\_REQUEST);

resp.getWriter().print(e.getMessage());

resp.getWriter().close();

} catch (IOException ioe) {

}

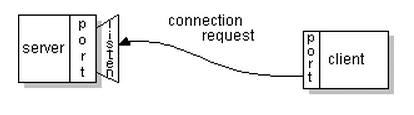
}

}

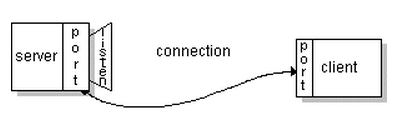
}

**Сокет -** кінечна двонапрямлена точка в мережі яка об’єднує дві програми. Дозволяє одночасно обслуговувати декілька клієнтів в мережі надаючи їм інформацію різного роду. Зв’язок між сокетами відбувається за допомогою визначеного мережевого протоколу. Класи Socket використовуються для побудови з’єднання між клієнтом і сервером. Пакет java.net представляє 2 класи для цього: Socket для клієнтської сторони і SrverSocket для серверної сторони відповідно. Грубо кажучи: ІР адреса + порт - це сокет.

Зазвичай, сервер запущено на конкретній машині і має сокет, який прив’язаний до конкретного номера порта. Сервер просто чекає і прослуховує порт, поки клієнт не встановить з’єднання. Клієнт знає адресу чи доменне ім’я сервера і порт де сервер прослуховує, на який потрібно звертатися. Також клієнт повинен ідентифікувати себе, що прив’язує до нього локальний порт, що будевикористовуватися через з’єднання.



Якщо все проходить добре, то сервер приймає з’єднання. Після приняття сервр отримує новий сокет, який прив’язаний до того ж порта і має адрес і номер порта клієнта. Це потрібно для того, аби оригінальной сокет міг і надалі прослуховувати порт і обслуговувати клієнтські запити в той час, коли ми працюємо з нашим клієнтом. TCP layer ідентифікує додаток дані якого призначені для відправки (кожне ТСР з’єднання може бути унікальним для двох enpoints, таким чином можна мати багато з’єднань між хостом і сервером).



**Endpoint** - IP address & port number.

**Читання і запис в сокет:**

Сокети по протоколу TCP/IP служать для надійної, двонапрямленої, постійної, потокової передачі даних між двома хостами в інтернеті. Сокет може слугувати для підключення системи вводу-виводу Джава до других програм, які можуть знаходитися як на локальній машині, так і на іншій машині в інтернеті.

В Джава підтримуються 2 різновиди сокетів: сокети клієнти і сокети сервера.

**Клас ServerSocket** - є приймачем, який очікує поки клієнт підключиться до нього, а потім починає з ним взаємодіяти.

**Клас Socket** - клас для сокетів клієнтів який слугує для підключення до серверних сокетів і ініціює обмін даними по мережевому протоколу. При створенні об’єкта класа Socket неявно створюється з’єднання клієнта з сервером. Конструктори:

**Socket(String socketName, int portNumber)** - throws UHE, IOE. Створює сокет який підключається до заданого імені і порта.

**Socket (InetAddress ipAdress, int portName)** - throws IOE. Створює сокет використовуючи вже існуючий об’єкт InetAddress і вказаний порт.

Методи класу Socket:

**InetAddress getInetAddress()** - повертає об’єкт типу InetAddress, пов’язаний з об’єктом Socket. null якщо сокет не підключено.

**int getPort() -** повертає віддалений порт, до якого прив’язаний об’єкт типу Socket. null якщо сокет не прив’язано.

**int getLocalPort() -** повертає локальний порт до якого прив’язаний об’єкт типу Socket. -1 якщо сокет не прив’язано.

Для доступа до потоків вводу\виводу пов’язаних з класом Socket використовують методи getInputSream(), getOutputStream(). Вони можуть генерувати IOE ,якщо сокет втрачає з’єднання. Ці потоки використовуються для передачі і прийому данних так само, як і звичайні потоки вводу\виводу.

**InpupStream getInputStream()** - повертає об’єкт типу InputStream, що пов’язаний з викликаючим сокетом.

**OutputStream getOutputStream()** - повертає об’єкт типу OutputStream, що пов’язаний з викликаючим сокетом.

**connect()** - створює нове підключення.

**boolean isConnected()** - true якщо сокет підключено до сервера.

**boolean isBound()** - true якщо сокет прив’язаний до адреси.

**boolean isClosed()** - true якщо сокет закрито.

**close()** - закрити сокет.

Клас Socket реалізує інтерфейс AutoCloseable, тому його можна використовувати в операторі try-з-ресурсами.

Приклад:

**import** java.net.\*;

**import** java.io.\*;

**public** **class** Whois {

**public** **static** **void** main(String[] args) {

**int** c;

**try**{

//socket connecting to adress with port 43 (whois)

Socket socket = **new** Socket("whois.internic.com", 43);

//get I\O streams from socket

InputStream inStream = socket.getInputStream();

OutputStream outStream = socket.getOutputStream();

//request string

String str ="MHProfessional.com";

//string to bytes for sending request

**byte** buff[] = str.getBytes();

//sending request

outStream.write(buff);

//reading answer

**while**((c = inStream.read()) != -1){

System.***out***.println( (**char**) c);

}

}**catch**(Exception e){

System.***out***.println("Writing error. " + e.getMessage());

}

}

}

Сокети Оракл:

Для написання більш низькорівенивих програм які дають можливості більші за URL і URLConnection пишуть клієнт-серверні додатки. Наприклад, дістати дані з бази чи сформувати якийсь прайс. Для такого з’єднання потрібен надійний протокол передачі даних, такий як ТСР. Для роботи через ТСР прграма на клієнті і програма на сервері повинні встановити з’єднання між собою. Кожна програма прив’язує до себе сокет до кінця з’єднання і запис чичитання відбувається в цей сокет.

**Клас HttpURLConnection** - нащадок класа URLConnection. Підтримує з’єднання по http протоколу. Щоб отримати об’єкт класу HttpURLConnection потрібно викликати метод openConnection() для об’єкта типу URL, а потім привести його до типу HttpURLConnection (при цьому потрібно переконатися, що з’єднання по протоколу HTTP установлено). Крім методів URLConnection має додатково свої:

**static boolean getFollowRedirects()** - повертає істину, якщо присутня автоматична переадресація.

**static void setFollowRedirects(boolean howTo)** - якщо вказати істину - переадресація відбуватиметься автоматично. В противному випадку - буде відключено. По дефолту автоматична переадресація ввімкнена.

**String getRequestMethod()** - повертає строкове представлення метода за допомогою якого відбувається запит по URL. По замовчуванню це GET (передає запит серверу використовуючи URL). Доступні і інші методи, в тому числі і POST (передає дані використовуючи тіло http запиту).

**void setRequestMethod(String method)** - можна задати метод яким робляться запити. (GET і POST). Кидає ProtocolException якщо вказано невірне значення.

**int getResponseCode()** - throws IOE. Повертає код запиту по HTTP. Повертає -1, якщо код недоступний.

**int getResponseMessage()** - throws IOE. повертає повідомлення запиту зв’язане з кодом.

**import** java.net.\*;

**import** java.io.\*;

**import** java.util.\*;

**public** **class** HttpURLDemo {

**public** **static** **void** main(String[] args) **throws** Exception {

URL link = **new** URL("http://www.google.com");

//setting to HttpURLConnection type

HttpURLConnection hpCon = (HttpURLConnection) link.openConnection();

System.***out***.println("Request method: " + hpCon.getRequestMethod());

System.***out***.println("The code of answer: " + hpCon.getResponseCode());

System.***out***.println("Response message: " + hpCon.getResponseMessage());

//getting fields to Map

Map<String, List<String>> hdrMap = hpCon.getHeaderFields();

//getting set of keys from map

Set<String> hdrFields = hdrMap.keySet();

System.***out***.println("Header: ");

//show all header keys

**for**(String k : hdrFields){

System.***out***.println("Key: " + k + " Value: " + hdrMap.get(k));

}

}

}

**Клас URІ –** універсальний ідентифікатор ресурса. Дуже схожий на URL, але насправді, URL є підмножиною URІ. URІ описує стандартний ідентифікатор до ресурсу, а URL ще може описувати доступ до ресурсу.

Серверні сокети по протоколу TCP/IP.

**Клас ServerSocket –** використовується для створення серверів, які можуть приймати запити через відкриті порти як від локальних клієнтів, так і від віддалених. Кидає IOE. Конструктори:

**ServerSocket(int port)** – створюює серверний сокет по вказаному порту. По дефолту дає значення черзі в 50 з’єднань.

**ServerSocket(int port, int connectMax)** – створюює серверний сокет по вказаному порту і задає максимальну кількість з’єднань.

**ServerSocket(int port, int connectMax, InetAddress localAddress)** - створюює серверний сокет по вказаному порту, задає максимальну кількість з’єднань. localAddress означає ІР адресу до якої прив’язано сокет, якщо система багатоканальна.

Методи:

**Socket accept()** – реалізує блокуючий виклик і чекає з’єднання від клієнта. Потім повертає об’єкт типу Socket, яким можна далі взаємодіяти з клієнтом.

**Датаграми –** порції даних які передаються між машинами. На відміну від протокола TCP, який є надійним, але містить в своїх запитах багато інформації типу алгоритмів адаптації до перегруженості мережі, реакції на втрати пакетів тощо. що сповільнює йогороботу, є швидшими, але ненадійними – нема гарантій, що дейтаграма дойде до кінечного пункту, чи що в кінці шляху взагалі буде хтось, хто прийме наші дані. Аналогічно, якщо дейтаграма приймається, то нема ніяких гарантій, що вона не була пошкоджена по дорозі. Дейтаграми реалізуютьсяповерх мережевого протоколу UDP за допомогою двох класів DatagramPacket (контейнер даних) і DatagramSocket (механізм для прийому і передачі даних типу DatagramPacket).

**Клас DatagramSocket**. Конструктори (кидають SocketException):

**DatagramSocket()** – створює об’єкт DatagramSocket пов’язаний з будь-яким незайнятим локальним портом в системі.

**DatagramSocket(int port)** – створює об’єкт DatagramSocket пов’язаний з вказаним портом.

**DatagramSocket(int port, InetAddress ipAddress)** – створює об’єкт DatagramSocket пов’язаний з вказаним портом і об’єктом InetAddress.

**DatagramSocket(SocketAddress address)** - створює об’єкт DatagramSocket пов’язаний з заданим об’єктом класу SocketAddress.

Найважливіші методи:

**void send(DatagramPacket packet)** – відправляє в порт вказаний пакет.

**void receive(DatagramPacket packet)** – очікує прийом через порт даного пакета і повертає отриманий результат.

**close()** – закриває сокет даного типа. Неактуальний для нового коду, оскільки DatagramSocket з 7 врчії Джави підтримує інтерфейс AutoCloseable.

**InetAddres getInetAddress()** – повертає адрес, якщо сокет підключено. Інакше null.

**int getLocalPort()** – повертає номер локального порта.

**int getPort()** – повертає номер порта до якого підключено сокет. Якщо сокет не підключено до жодного зпортів – повертає -1.

**boolean isBound()** – повертає істину, якщо заданий сокет прив’язано до адресу.

**boolean isConnected()** – повертає істину, якщо заданий сокет підключений до сервера.void **void setSoTimeout(long millis)** – задає період очікування сокета в мілісекундах. throws SocketException.

**Клас DatagramPacket.** Конструктори:

**DatagramPacket(byte[] data, int size)** – створює буфер, який буде приймати дані і розмір пакета. Слугує для прийому даних через сокет типу DatagramSocket.

**DatagramPacket(byte[] data, int shift, int size)** – дозволяє вказати зміщення де мають знаходитися дані в буфері.

**DatagramPacket(byte[] data, int size, InetAddress ipAddress, int port)** – дозволяє вказати цільовий адрес і порт який використовується в сокеті типу DatagramSocket для відправлення пакету до назначення.

**DatagramPacket(byte[] data, int shift, int size, InetAddress ipAddress, int port) –** організує передачу пакетів з вказаного зміщення в буфері.

Перших 2 метода можна розглядати як запаковування листа в конверт, а останні два, як запаковування листа в конверт і написання на конверті адреси.

Методи класу DatagramPacket:

**InetAddres getAddress()** – повертає адрес джерела чи адрес місця призначення. Інакше null.

**int getLength()** – повертає довжину достовірних даних, які лежать в байтовому масиві, який має повернути метод getData(). Ця довжина може не повністю співпадати здовжиною масива.

**void setLength(int size)** – встановлює довжину пакета.

**int getOffset()** – повретає початковий індекс даних.

**int getPort()** – повертає номер порта.

**void setPort(int port)** – втсановлює номер порта.

**void setAddress(InteAddress ipAddress)** – встановлює адрес по якому ввідправлятиметься пакет.

**byte[] getData()** – повертає масив даних який знаходиться в дейтаграмі. В основному використовуєтьсядля витягнення даних після прийому дейтаграми.

**viod setData(byte[] data)** – утановлює буфер в вигляді заданого масиву байт, знульовим зміщенням на довжину, і довжину рівну масиву байт.

**void setData(byte[] data, int index, int size) –** встановлює буфер в вигляді заданого масиву байт, зміщений по вказаному індексу, а довжину – по заданому розміру.

Приклад клінт-серверної програми. Сервер чекає на підключення і отримує строку, далі повертає її назад клієнту з модифіфкацією. Відповідно клієнт відправляє строку серверу і отримує її від нього.

Сервер:

**import** java.io.\*;  
**import** java.net.ServerSocket;  
**import** java.net.Socket;  
  
**public class** Server {  
 **public static void** main(String[] args) {  
 **int** port = 228; *//jaguar and АК-47 edition* **try**{  
 *//create serever socket* ServerSocket serverSocket = **new** ServerSocket(port);  
 System.***out***.println(**"Waiting for a client."**);  
  
 *//create socket and accept from server socket. server wait for connecting* Socket socket = serverSocket.accept();  
 System.***out***.println(**"Got a client."**);  
 System.***out***.println();  
  
 *//get IS and OS from socket* InputStream is = socket.getInputStream();  
 OutputStream os = socket.getOutputStream();  
  
 *//converting IS and OS to another type for simple sending text* DataInputStream dataInputStream = **new** DataInputStream(is);  
 DataOutputStream dataOutputStream = **new** DataOutputStream(os);  
  
 String line;  
 **while** (**true**) {  
 line = dataInputStream.readUTF(); *//wait for string from client* System.***out***.println(**"Client send to server: "** + line);  
 System.***out***.println(**"Sending it back."**);  
 dataOutputStream.writeUTF(line + **" :)"**); *//sending string back to client* dataOutputStream.flush();  
 System.***out***.println();  
 }  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Клієнт:

**import** java.io.\*;  
**import** java.net.InetAddress;  
**import** java.net.Socket;  
**import** java.net.UnknownHostException;  
  
**public class** Client{  
 **public static void** main(String[] args) {  
 **int** serverPort = 228; *//must be the same like on server* String address = **"127.0.0.1"**; *//server IP address. now its local machine.* **try** {  
 *//create ip address object. don't need "new".* InetAddress inetAddress = InetAddress.*getByName*(address);  
 System.***out***.println(**"Socket IP address: "** + address + **". Socket port number: "** + serverPort);  
  
 *//create socket using inetAddress object and server port.* Socket socket = **new** Socket(inetAddress, serverPort);  
  
 *//get IS & OS from socket.* InputStream inputStream = socket.getInputStream();  
 OutputStream outputStream = socket.getOutputStream();  
  
 *//converting IS and OS to another type for simple sending text* DataInputStream dataInputStream = **new** DataInputStream(inputStream);  
 DataOutputStream dataOutputStream = **new** DataOutputStream(outputStream);  
  
 *//create stream for reading from keyboard* BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader(System.***in***));  
  
 String line;  
 System.***out***.println(**"Type something:"**);  
 System.***out***.println();  
  
 **while**(**true**){  
 line = bufferedReader.readLine();  
 System.***out***.println(**"Sending this line to server."**);  
 dataOutputStream.writeUTF(line);  
 dataOutputStream.flush();  
  
 line = dataInputStream.readUTF(); *// waiting for server response* System.***out***.println(**"Server response: "** + line);  
 System.***out***.println();  
 }  
  
  
 } **catch** (UnknownHostException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
}

Компілюємо 2 файла в Джава класи (javac Client.java Server.java). Далі запускаємо 2 вікна консолі. В одному java Server – це буде сервер, в іншому java Client. Все, що ми писатимемо на клієнті, відправлятиметься серверу і ним повертатиметься назад на клієнт.

Розглянемо детальніше:

ServerSocket ss = new ServerSocket(port);  
Socket socket = ss.accept();

Клас ServerSocket відрізняється від класу Socket. Клас Socket - це і є сокет. Головною відмінністю є те, що клас SrverSocket вміє заставити програму чекати на з’єднання до нього. Коли ми створюємо екземпляр класу ServerSocket, ми вказуємо йому порт, потім створюємо об’єкт типу Socket якому присвоюємо результат методу accept() від об’єкту типу SrverSocket - цей метод є блокуючим і заставляє сокет чекати підключення, програма не виконуватиметься далі доти, доки не буде встановлено з’єднання. По суті, виконується метод accept() і його результат повертаєься змінній типу Socket, таким чином утворюючи вже нормальний сокет з яким можна виконувати всі потрібні дії.

Для створення сокета потрібен ІР адрес і порт. Адрес створювати більш рекомендовано в вигляді об’єкта InetAddress.

InetAddress ipAddress = InetAddress.getByName(address);

Socket socket = new Socket(ipAddress, serverPort);

Після створення сокета можна з нього взяти вхідний і вихідний потоки аби читати чи писати в них.

InputStream sin = socket.getInputStream();  
OutputStream sout = socket.getOutputStream();

Якщо ми працюємо з строками чи примітивами, то зручніше використовувати поток вводу-виводу типу DataInputStream і DataOutputStream.

DataInputStream in = new DataInputStream(sin);  
DataOutputStream out = new DataOutputStream(sout);

Далі реалізовуємо потрібний функціонал звичним нам способом.

Ще один приклад клієнт-серверного додатку, який повертає модифіковану строку. З оракла. Потоки повинні закриватися до закриття сокета:

1. Open a socket.
2. Open an input stream and output stream to the socket.
3. Read from and write to the stream according to the server's protocol.
4. Close the streams.
5. Close the socket.

Клієнт:

**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.io.PrintWriter;  
**import** java.net.InetAddress;  
**import** java.net.Socket;  
**import** java.net.UnknownHostException;  
  
**public class** EchoClient {  
 **public static void** main(String[] args) {  
 **if** (args.**length** != 2) {  
 System.***out***.println(**"Usage: \"server address\", \"server port\". "**);  
 System.*exit*(1);  
 }  
  
 String hostName = args[0];  
 **int** portNumber = Integer.*parseInt*(args[1]);  
  
 BufferedReader in = **null**;  
 BufferedReader inConsole = **null**;  
 PrintWriter out = **null**;  
  
 **try** {  
 *//creating socket with InetAddress* InetAddress hostAddress = InetAddress.*getByName*(hostName);  
 Socket socket = **new** Socket(hostAddress, portNumber);  
  
 *//getting IS, IS console, OS from socket* in = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));  
 inConsole = **new** BufferedReader(**new** InputStreamReader(System.***in***));  
 out = **new** PrintWriter(socket.getOutputStream());  
  
 System.***out***.println(**"Type text:"**);  
 System.***out***.println();  
  
 String userInputString;  
 **while** ((userInputString = inConsole.readLine()) != **null**) {  
 out.println(userInputString);  
 out.flush();  
 System.***out***.println(**"Echo form server: "** + in.readLine());  
 }  
  
 } **catch** (UnknownHostException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **finally** {  
   
 **if** (in != **null**) {  
 **try** {  
 in.close();  
 } **catch** (IOException e) {  
 *//NOP* }  
 }  
  
 **if** (inConsole != **null**) {  
 **try** {  
 in.close();  
 } **catch** (IOException e) {  
 *//NOP* }  
 }  
  
 **if** (out != **null**) {  
 **try** {  
 in.close();  
 } **catch** (IOException e) {  
 *//NOP* }  
 }  
  
  
 }  
  
 }  
}

Сервер:

**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.io.PrintWriter;  
**import** java.net.ServerSocket;  
**import** java.net.Socket;  
  
**public class** EchoServer {  
 **public static void** main(String[] args) {  
 **if** (args.**length** != 1) {  
 System.***out***.println(**"Usage: java EchoServer \"port number\""**);  
 System.*exit*(1);  
 }  
  
 **int** portNumber = Integer.*parseInt*(args[0]);  
 BufferedReader in = **null**;  
 PrintWriter out = **null**;  
  
 **try** {  
 *//creating server socket anf waiting for client connection* ServerSocket serverSocket = **new** ServerSocket(portNumber);  
 System.***out***.println(**"Waiting for client connection."**);  
 Socket socket = serverSocket.accept();  
 System.***out***.println(**"Connection established."**);  
  
 *//getting IS & OS from socket* in = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));  
 out = **new** PrintWriter(socket.getOutputStream(), **true**);  
  
 *//reading string and writing back to client* String inputLine;  
 **while** ((inputLine = in.readLine()) != **null**) {  
 out.println(inputLine + **" - it's from server."**);  
 }  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 } **finally** {  
  
 **if** (in != **null**) {  
 **try** {  
 in.close();  
 } **catch** (IOException e) {  
 *//NOP* }  
 }  
  
 **if** (out != **null**) {  
 **try** {  
 in.close();  
 } **catch** (IOException e) {  
 *//NOP* }  
 }  
 }  
 }  
  
}

**com.sun.net.httpserver -** встроєний в JDK простий HTTP сервер.

Дані HTTP йдуть всередині TCP/IP.

Нвіть якщо з браузера через ЯваСкріпт відправлено всього 2 символа на сервер, то це не означає, що відправлено тільки символьних 4 байта. Оскільки при запиті добавляється додаткова інформація в вигляді хедерів HTTP, хедерів TCP/IP тощо. Зв’язок між браузером і сервером відбувається по HTTP, а сам протокол в свою чергу зав’язаний на TCP/IP.

Ініціатором підключення є клієнт. Далі їхні ролі стають симетричними в обі сторони. Вони можуть як обмінюватися данними, так і переривати з’єднання.

UDP не робить з’єднання, а просто посилиає пакет даних по адресу. Тому не відомо чи дойшли дані.

TCP/IP протокл з звітом про успішну чи ні доставку пакетів.

В Джава нема API яке може працювати нижче протоколу TCP/IP. Наприклад, працювати напряму з IP.

**Tomcat** (Catalina) - контейнер срвлетів. RI reference implementation від ServletAPI. В той час як писалася специфікація ServletAPI по ній писався Томкет. Сидить на TCP/IP, розбирає його HTTP і видає нам зручний результат в вигляді класів тощо Джави. Реалізує JSP.

**Порт** - абстрактна сутність, оскільки ніде не існує фізично. Коли пакет приходить на сервер, то в нім міститься інформація з номером порта на який він відправлений (2 байта які містять порт в інтервалі 0 - 65535). Сервер ж налаштований на прослуховування певних портів. Відповідно потрібний пакет потрапляє в потрібний порт, а якщо пакет приходить з портом на який не налаштовано сервер, то мережева карта не віддасть цей пакет нікуди в ОС і т.д, а видалить його. Зроблено це так, аби один пакет не приходив на всю машину, тоді б один комп’ютер відповідав тільки одному додатку. Почті, тощо.

Як відбувається співставлення ІР і доменного імені: набираємо адрес в браузері, спочатку відправляється UDP над яким працює DNS, який повертає ІР-адрес даного доменного імені. Після цього через TCP/IP працює HTTP. При чому браузер деякий час зберігає локальний кеш, в якому зберігається дане співставлення і і воно більше не перепитуватиметься броузером у DNS сервера. Таким чином, можна поміняти ІР свого сервера і зберегти при цьому доменне ім’я.

Socket відповідає одному з’єднанню з одним клієнтом:

**while**(**true**){

Socket socket = servSocket.accept(); //accept() - socket waiting for client connection. Pause.

**new** BlockingHttpHandler(socket).call();

}

Тут socket буде чекати підключення клієнта і створювати екзкмпляр типу Socket. Потім виконувати BlockingHttpHandler(socket).call();. Сокет буде очікувати доки відпрацює call(); і знову очікуватиме запитів. Скільки клієнтів прийде - стільки екзкмплярів типу Socket буде створено.

**import** java.io.IOException;

**import** java.net.\*;

**public** **class** BlockingHttpServer {

**public** **static** **void** main(String[] args) {

**try** {

ServerSocket servSocket = **new** ServerSocket(80, 256); //256 - backlog. how many connections can be in queue.

**while**(**true**){

Socket socket = servSocket.accept(); //accept() - socket waiting for client connection. Pause.

**new** BlockingHttpHandler(socket).call();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

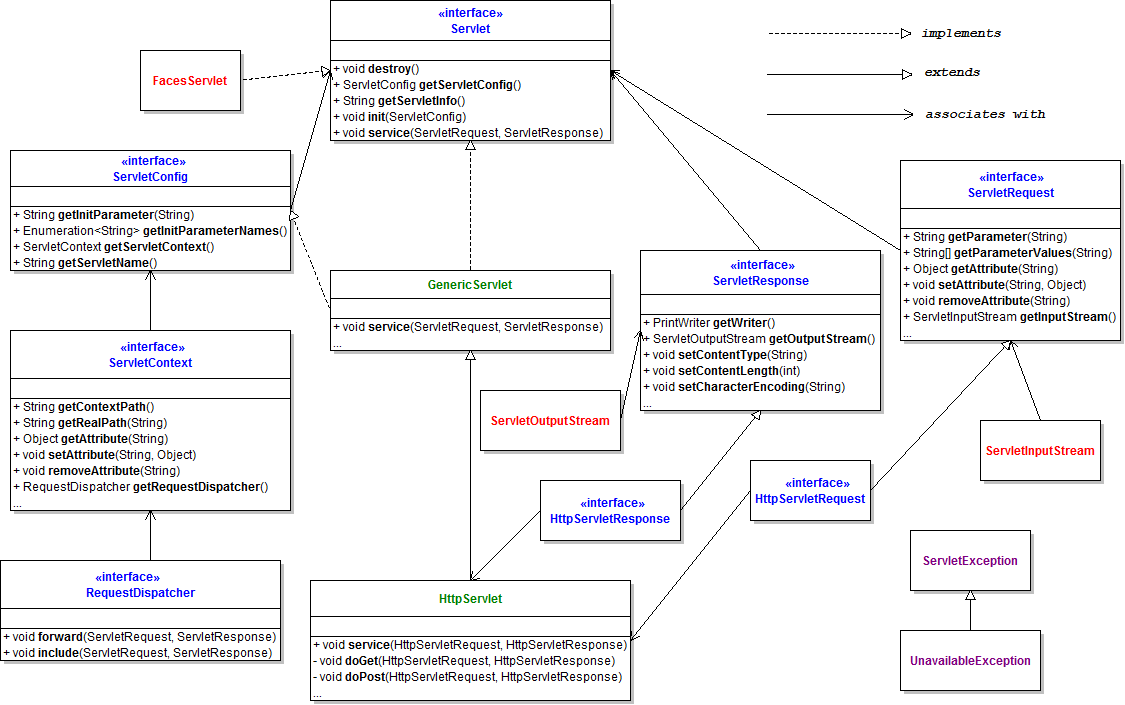
}

**Header** по суті це Map<String, List<String>> (мапа строк на лісти строк) Приклад - Host: localhost, Connection: keep-alive.

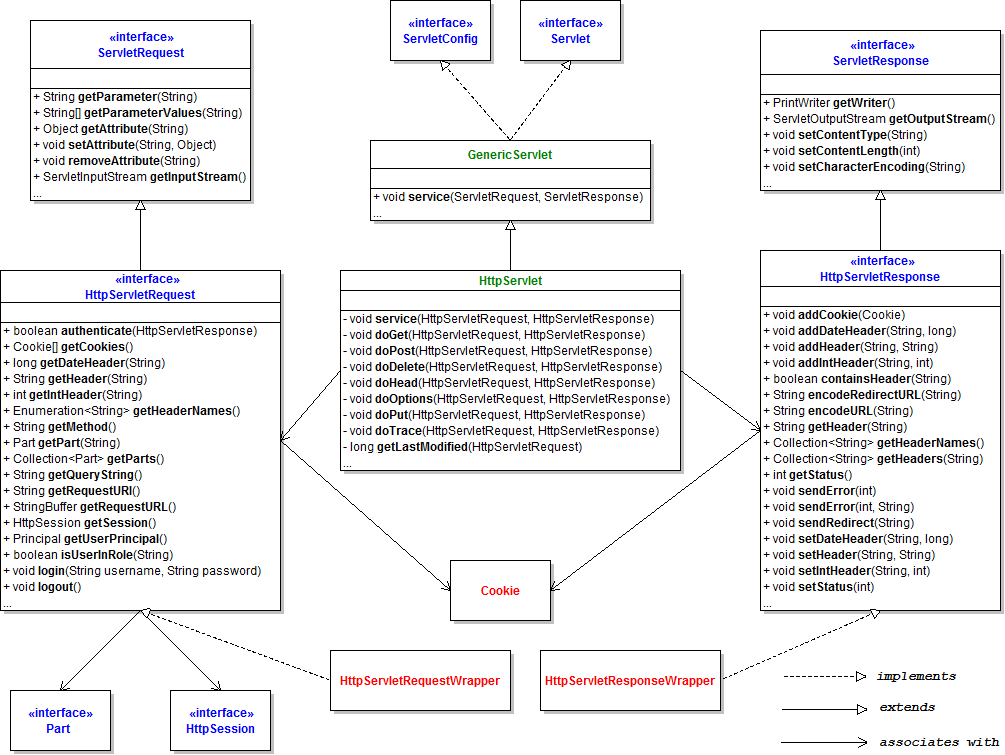
**Кешуючий проксі сервер**, це сервер, який моніторить запити і хешує сторінки. Якщо хтось інший звертається з такими же запитом, то проксі видає захешований результат одразу і не ходить в базу, викачує баннери і т.д. Має час життя який налаштовується.

**HTTPS** (HyperText Transfer Protocol Secure) — расширение [протокола](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [HTTP](https://ru.wikipedia.org/wiki/HTTP), поддерживающее [шифрование](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Данные, передаваемые по протоколу HTTPS, «упаковываются» в криптографический протокол [SSL](https://ru.wikipedia.org/wiki/SSL) или [TLS](https://ru.wikipedia.org/wiki/TLS). В отличие от HTTP, для HTTPS по умолчанию используется [TCP](https://ru.wikipedia.org/wiki/TCP)-[порт](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82_(TCP/IP)) 443.

**Servlets.**

****

**------------------------------------------------------------------------------------------------------------------------------**

****

**Сервлети** - невеликі програми, які виконуються на стороні веб-сервера підключення. Розширяють можливості веб сервера.

Як відбувається запит на сервер: користувач вводить URL в полі браузера, Браузер формує запит по протоколу HTTP відправляючи його до віповідного веб-сервера. Веб-сервер співставляє цей запит з конкретним файлом і повертає його браузеру в якості відповіді по тому ж протоколу HTTP. В заголовку цієї HTTP відповіді вказано тип контенту (стандарт MIME. Приклад типу MIME "text/html" - вихідний код html сторінки).

Життєвий цикл сервлета складають 3 основних метода: **init**(), **service**(), **destroy**(), які реалізовані сервлетом і в потрібний момент викликаються сервером.

**Сценарій взаємодії сервлета**: в вікні браузера вводиться URL запит, який формує HTTP запит і відправляє його до потрібного сервера. Далі цей HTTP запит приймає веб сервер і співставляє його з конкретним сервлетом і цей сервлет загружаєть в адресний простір веб сервера. Далі сервер викликає метод **init**() з сервлета і сервлет загружається в ОЗУ веб сервера. (якщо до цього вже викликався даний сервлет, то він не викликається знову, а використовується вже запущений в ОЗУ сервлет). Сервлету можна задавати параметри ініціалізації, тому він може самоналаштовуватися. Далі викликається метод **service**() з сервлета який займається обробкою HTTP запита. Сервлет може читати НТТР запит і формувати НТТР відповідь клієнту. Сервлет залишається в адресному просторі для будь яких інших НТТР запитів, а метод service() буде викликатися при кожному НТТР запиті. Щоб вигрузити сервлет з ОЗУ використовують метод **destroy**(), при чому дані можуть бути збережені в персистентне сховище.

**javax.servlet.\*** - пакет, який містить класи і інтерфейси потрібні для роботи сервлетів.

**Клас GenericServlet** - має функціонал, який упрощує створення сервлетів. Зокрема, він містить методи **init**() і **destroy**().

**CGI -** Common Gateway Interface - загальний шлюзовий інтерфейс - використовувався раніше для отримання окремими процесамти НТТР запитів і формування НТТР відповіді. Був замінений іншими методологіями, зокрема сервлетами, оскільки був не ефективний в плані ресурсів ОЗУ і процесора. Крім того, був платформо залежним і неефективно працював з БД.

Якщо tomcat **startup.bat** не запускається, то треба задати змінну оточення **JRE\_HOME** і вказати в ній шлях до JRE.

Класи і інтерфейси, які потрібні для створення сервлетів знаходяться в архівному файлі sevlet-api.jar в папці томката "lib". Для доступу до цього архівного файлу потрібно добавити змінну середовища **CLASSPATH** і добавити в неї шлях до файлу. (C:\Program Files\apache-tomcat-7.0.63\lib\servlet-api.jar) Що б не вказувати змінну середовища при компіляції програми (javac Example.java -classpath " C:\Program Files\apache-tomcat-7.0.63\lib\servlet-api.jar "), потрібно добавити її в налаштуваннях системи в графі CLASSPATH (або **set PATH=C:\Program Files\apache-tomcat-7.0.63\lib\servlet-api.jar**).

Якщо IDE не бачить імпорт javax.servlet.\*; потрібно добавити файл **servlet-api.jar** в проект. (add external JAR). File\ProjectStructure/Modules.

Також потрібно прослідкувати щоб IDE використовувала одинакову JDK в ProjectSrtructure і змінні середовища системи вели на ту саму версію джави (в консолі **java -version**).

**JAVA\_HOME** - C:\Program Files\Java\jdk1.7.0\_60\

**JRE\_HOME** - C:\Program Files\Java\jdk1.7.0\_60\jre

**Path** - C:\Program Files\Java\jdk1.7.0\_60\bin\ (для використання команд javac.exe java.exe, javadoc.exe)

Для томкета потрібно щоб були і JRE i JDK. Оскільки JRE потрібно бо Томкет сам написаний на джаві, а JDK йому потрібно бо він викликає компілятор Джави. Бо на льоту свої Жспхи Томкет своїм транслятором транслює в Джава сорци а потім компілює їх в клас файли (при чому їх можна вважати коректними сервлетами). Тому в класспас нам потрібно вказувати і servlet-api.jar, і шлях до JDK.

**CLASSPATH** - C:\Program Files\apache-tomcat-7.0.63\lib\servlet-api.jar\;C:\Program Files\Java\jdk1.7.0\_60\bin\ (вказує додатку і JDK tools де шукати класи користувача. Можна задати окремо для кожного додатка використавши ключ -ср. Щоб перевірити цю змінну з консолі - **echo %CLASSPATH%)**

Steps for adding external jars in IntelliJ IDEA:

1. Click File from File menu
2. Project Structure (CTRL + SHIFT + ALT + S on Windows/Linux, ⌘ + ; on Mac OS X)
3. Select Modules at the left panel
4. Dependencies tab
5. Add... → Project Library → Attach Jar

Приклад послідовності створення сервлета і запуску Томката:

**1.** Створюємо сервлет:

**import** javax.servlet.GenericServlet;  
**import** javax.servlet.ServletRequest;  
**import** javax.servlet.ServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.rmi.ServerException;  
  
**public class** HelloServlet **extends** GenericServlet {  
 @Override *//overriding method service() from GenericServlet class* **public void** service(ServletRequest request, *//getting request from client data* ServletResponse response) *//setting the response to client* **throws** ServerException, IOException {  
 response.setContentType(**"text/html"**); *//setting MIME type for HTML pages* PrintWriter writer = response.getWriter();  
 writer.write(**"Test passed."**); *//sending HTTP request* writer.close();  
 }  
}

Далі потрібо скомпілювати його і клас-файл скопіювати в директорію Томката examples (C:\Program Files\apache-tomcat-7.0.63\webapps\examples\WEB-INF\classes) або створити свою папку і скопіювати його туди. Далі, "зареєструвати сервлет" в файлі Томката web.xml (C:\Program Files\apache-tomcat-7.0.63\webapps\examples\WEB-INF) де в потрібній секції добавити інформацію:

<servlet>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>HelloServlet</servlet-class>

</servlet>

-----------------------------------------------------------------------

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

<url-pattern>/servlets/servlet/HelloServlet</url-pattern>

</servlet-mapping>

**2.** Потім запускаємо на виконання Томкат.

**3.** Запускаємо браузер і даємо запит на сервлет. (http://localhost:8080/examples/servlets/servlet/HelloServlet)

**ServletAPI**. (Tomcat)

Пакет javax.servlet містить в собі інтерфейси і класи які використовуються для роботи сервлетів. Найголовнішим з них є інтерфейс Servlet, всі інші сервлети повинні реалізовувати його або розширяти класи які його реалізують. Також важливими є інтерфейси ServletRequest і ServletResponse.

Основні інтерфейси пакета javax.servlet:

**Sevlet** - визначає методи життєвого циклу сервлета.

**ServletConfig** - дозволяє отримувати параметри ініціалізації сервлетів.

**ServletContext** - дозволяє реєструвати події сервлетів та звертатися до відомостей про їх оточеня.

**ServletRequest** - слугує для вводу даних з запроса клієнта.

**ServletResponse** - слугує для виводу даних в відповідь клієнту.

Основні класи пакету javax.servlet:

**GenericServlet** - реалізує інтерфейси Servlet і ServletConfig.

**ServletInputStream** - представляє потік для вводу даних з запиту клієнта.

**ServletOutputStream** - представляє потік для виводу даних клієнту у відповідь на запит.

**ServletException** - вказує на те, що в сервлеті відбулася помилка.

**UnavailableException** - вказує на те, що сервлет недоступний.

**Інтерфейс** **Servlet.** Всі сервлети повинні наслідувати цей інтерфейс, або той, який його реалізує. В ньому оголошуються методи init(), service(), destroy(). Крім них, містить ще методи які дозволяють отримувати параметри ініціалзації. Методи:

**void destroy()** - викликається при вигрузці сервлета.

**ServletConfig getServletConfig()** - отримумє об’єкт типу ServletConfig, що містить в собі будь-які параметри ініціалізації.

**String getServletInfo()** - повертає строку з описом сервлета. Можна переоприділити щоб надати, наприклад, ім’я автора і т.п.

**void init(ServletConfig config) throws ServletException** - викликається під час ініціалізації сервлета. Параметри конфігурації можуть бути задані в config. Кидає ServletException, якщо сервлет неможливо ініціювати.

**void service(ServletRequest request, ServletResponse response) throws ServletException, IOException** - викликається для обробки запиту клієнта. Запит клієнта можна отримати з request і сформувати відповідь в response, відповідно створивши вихідний потік. Кидає виключення, якщо при виконанні сервлета чи вводі\виводі виникає помилка.

**Інтерфейс ServletConfig.**  Дозволяє отримати дані конфігурації сервлета під час загрузки самого сервлета. Методи:

**ServletContext getServletContext()** - повертає "вміст" даного сервлета.

**String getInitParameter(String parameter)** - повертає значення параметра ініціалізації parameter.

**Enumeration getInitParameterNames()** - повертає перелік імен параметрів ініціалізації.

**String getServletName()** - повертає ім’я заданого сервлета.

**Інтерфейс ServletContext.**  Дозволяє сервлету отримувати відомості про середовище виконання сервлета. Методи:

**Object getAtribbute(String attribute)** - повертає значення вказаного атрибута сервера.

**void setAttribute(String attribute, Object value)** - задає заданному атрибуту значення.

**String getMimeType(String pathFile)** - повертає МІМЕ тип файлу.

**String getRealPath(String virtual path)** - повертає реальний шлях файлу пов’язаний з віртуальним шляхом.

**String getServerInfo()** - повертає відомості про сервер.

**void log(String s)** - записує вказану строку в журнал сервлета.

**void log(String s, Throwable e)** - записує вказану строку в журнал сервлета і стектрейс виключення яке вилетіло.

**Інтерфейс ServletRequest**. Дозволяє отримувати в сервлеті відомості про запит клієнта. Методи:

**Object getAtribbute(String attribute)** - повертає значення вказаного атрибута сервера.

**String getCharacterEncoding()** - повертає кодування символів в запиті.

**int getConetentLength()** - повертає довжину запиту. Повертає -1, якщо довжину неможливо визначити.

**String getContentType()** - повертає тип запиту. null, якщо тип запиту неможливо визначити.

**SevletInputStream getInputStream() throws IOE** - повертає вхідний потік, який читає байтові дані з потока. Якщо до цього вже викликався метод getReader(), то отримаємо виключення IllegalStateException.

**String getParameter(String parameter)** - повертає значення параметра parameter.

**Enumeration<String> getParameterNames()** - повертає перечислення імен параметрів по даному запиту.

**String[] getParameterValues(String parameter)** - повертає масив значень параметра parameter.

**String getProtocol()** - повертає значення протокола в строковому представленні.

**BufferedReader getReader() throws IOE** - повертає буферизований потік вводу, який може читати текст з запиту. Якщо до цього вже викликався метод getInputStream(), то отримаємо виключення IllegalStateException.

**String getRemoteAddr()** - повертає ІР представлення адреса клієнта.

**String getRemoteHost()** - повертає строкове представлення адресу клієнта.

**String getScheme() -** повертає схему передачі URL по якій робився запит. (http, ftp etc.)

**String getServerName()** - повертає ім’я сервера.

**int getServerPort()** - повертає номер порта сервера.

**Інтерфейс ServletResponse**. Дозволяє сформувати в сервлеті відповідь клієнту на його запит. Методи:

**String getCharacterEncoding()** - повертає кодування символів в запиті.

**ServletOutputStream getOutputStream() throws IOE** - повертає вихідний потік, який пише байтові дані в потік відповіді. Якщо до цього вже викликався метод getWriter() по цьому ж запиті, то отримаємо виключення IllegalStateException.

**PrintWriter getWriter() throws IOE** - повертає поток запису типу PrintWriter ,який можна використовувати для виводу символьних даних. Якщо до цього вже викликався метод getOutputStream () по цьому ж запиті, то отримаємо виключення IllegalStateException.

**void setContetnLength(int length)** - вказує довжину відповіді клієнту.

**void setContentType(String type)** - задає тип відповіді.

**Клас GenericServlet.** Представляє реалізації основних методів життєвого циклу сервлета оскільки реалізовує інтерфейси Servlet і ServletConfig. Також має метод для запису символьної строки в журнал логу сервера:

**void log(String s)** - записує вказану строку в журнал сервера.

**void log(String s, Throwable e)** - записує вказану строку в журнал сервера і стектрейс виключення яке вилетіло.

**Клас ServletInputStream**. Розширяє клас InputStream. Клас реалізується контейнером сервлетів і надає вхідний потік ,яким можна читати вхідні дані запиту. Має конструктор по замовчуванню, а також метод для читання байтів з потоку вводу:

**int readLine(byte[] buffer, int shift, int size) throws IOE** - тут bufferце масив який зберігає певну кількість (size) байтів починаючи з вказаного зміщення (shift). Повертає кількість прочитаних байтів. -1, якщо дані з потока закінчуться.

**Клас ServletOutputStream**. Розширяє клас OutputStream. Клас реалізується контейнером сервлетів і надає вихідний потік, яким можна писати вихідні дані на відповідь клієнту. Має конструктор по замовчуванню, а також методи print() і println() якими відбувається запис в вихідний потік.

**Клас ServletException.** В пакеті javax.servlet є два класа виключень. Це ServletException, який сповіщає про помилку під час виконання сервлета, і клас UnavailableException який розширяє клас ServletException і сповіщає, що сервлет недоступний.

Приклад обробки форми методом "POST" за допомогою сервлета:

HTML сторінка з формою:

<html>

<head>

<title>Post parameters page</title>

</head>

<body>

<center>

**<form name="Form1"**

**method="post"**

**action="http://localhost:8080/examples/servlets/servlet/PostParameterServlet">**

<table>

<tr>

<td><b>**Employee**</b></td>

<td><input type="textbox" name="e" size="20" value=""></td>

</tr>

<tr>

<td><b>**Phone**</b></td>

<td><input type="textbox" name="p" size="20" value=""></td>

</tr>

</table>

<br>

<input type="submit" value="**Submit**" align="left">

**</form>**

</center>

</body>

</html>

Сервлет який обробояє запит:

**import** javax.servlet.GenericServlet;  
**import** javax.servlet.ServletException;  
**import** javax.servlet.ServletRequest;  
**import** javax.servlet.ServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.util.Enumeration;  
  
**public class** PostParameterServlet **extends** GenericServlet{  
  
 @Override  
 **public void** service(ServletRequest servletRequest,  
 ServletResponse servletResponse) **throws** ServletException, IOException {  
  
 *//get the output writing stream* PrintWriter printWriter = servletResponse.getWriter();  
 *//get enumeration of parameters names* Enumeration enumeration = servletRequest.getParameterNames();  
  
 *//show parameter names and values* **while**(enumeration.hasMoreElements()){  
 String parameterName = (String) enumeration.nextElement();  
 printWriter.print(parameterName + **" = "**);  
 String parameterValue = servletRequest.getParameter(parameterName);  
 printWriter.println(parameterValue);  
 }  
 printWriter.close();  
 }  
}

**Пакет javax.servlet.http.** Призначений для роботи по протоколу НТТР і спрощує створення сервлетів, які обробляють запити по НТТР. Основні інтерфейси:

**HttpServletRequest** - дозволяє вводити дані з НТТР запиту.

**HttpServletRespose** - дозволяє пистати дані з сервлета в вихідний потік НТТР.

**HttpSession** - дозволяє вводити і виводити дані про сеанс зв’язку.

**Cookie** - Дозволяє зберігати дані стану на машині клієнта. (дані автентифікації, налаштувань юзера, статистика юзера тощо...)

**HttpServlet** - містить методи для оброки запитів і формування відповідей по НТТР.

**Інтерфейс** **HttpServletRequest**. Реалізований контейнером сервлетів. Дозволяє отримувати відомості про запит клієнта. Методи:

**String getAuthType()** - повертає тип автентифкації.

**Cookie[] getCookies()** - повертає масив, який містить кукі файл в даному запиті.

**long getDateHeader(String field)** - повертає значення вказаного поля з заголовка дати.

**String getHeader(String field)** - повертає значення поля в строковому еквіваленті.

**Enumeration<String> getHeaderNames()** - Повертає перелік імен заголовків.

**int getIntHader(String field)** - повертає цілочисельний еквівалент поля заголовка.

**String getMethod()** - повертає метод для НТТР запиту.

**String getPathInfo()** - повертає відомості про шлях після шляху до сервлета і до строки URL запиту.

**String getPathTranslated()** - повертає відомості про шлях після шляху до сервлета і до строки URL запиту після її перетворення в справжній шлях.

**String getServletPath()** - повертає ту частину адреси, яка вказує на сервлет.

**String getQueryString()** - повертає строку запиту URL.

**String getRemoteUser()** - повертає ім’я користувача, який відправив запит.

**String getRequestedSesssionId()** - повертає ідентифікатор сеансу зв’язку.

**String getRequestedURI()** - повертає URL.

**StringBuffer getRequestURL()** - повертає URL.

**HttpSesssion getSession()** - повертає сеанс зв’язку для даного запиту. Якщо сеанс не було створено, то створює його, а потім повертає.

**HttpSession getSession(boolean new)** - повертає сеанс зв’язку для даного запиту. Якщо параметр встановлено як істину і сеанс не було створено, то створює його, а потім повертає. Якщо фолс - використовує існуючий сеанс по даному запиту.

**boolean isRequestedSession-IdFromCookie()** - повертає істину, якщо кукі містять ідентифікатор сеансу зв’язку. Інакше - фолс.

**boolean isRequestedSession-IdFromURL()** - повертає істину, якщо URL містить інформацію про сеанс зв’язку. Інакше - фолс.

**boolean isRequestedSession-IdValid()** - повертає істинуя, якщо ідентифікатор сеансу є дійсним для даної сесії.

**Хедер (header)** - строки в НТТР-повідомленні, які містять в собі інформацію в вигляді пар ключ-значення розділені двокрапками. Основні хедери - включаються в всі запити і відповіді, Хедери запиту - використовують тільки в запитах клієнта, Хедери відповіді - використовують тільки в відповідях сервера, Хедери сутності - супроводжують кожну сутність повідомлення.

Content-Type: text/html;charset=windows-1251

Allow: GET,HEAD,OPTIONS

Content-Length: 1984

**Інтерфейс HttpSevletResponse.** Інтерфейс дозволяє стформувати відповідь клієнту на запит. Містить декілька констант, які відповідають різним станам НТТР-відповіді.

**int** SC\_CONTINUE = 100;  
**int** SC\_SWITCHING\_PROTOCOLS = 101;  
**int** SC\_OK = 200;  
**int** SC\_CREATED = 201;  
**int** SC\_ACCEPTED = 202;  
**int** SC\_NON\_AUTHORITATIVE\_INFORMATION = 203;  
**int** SC\_NO\_CONTENT = 204;  
**int** SC\_RESET\_CONTENT = 205;  
**int** SC\_PARTIAL\_CONTENT = 206;  
**int** SC\_MULTIPLE\_CHOICES = 300;  
**int** SC\_MOVED\_PERMANENTLY = 301;  
**int** SC\_MOVED\_TEMPORARILY = 302;  
**int** SC\_FOUND = 302;  
**int** SC\_SEE\_OTHER = 303;  
**int** SC\_NOT\_MODIFIED = 304;  
**int** SC\_USE\_PROXY = 305;  
**int** SC\_TEMPORARY\_REDIRECT = 307;  
**int** SC\_BAD\_REQUEST = 400;  
**int** SC\_UNAUTHORIZED = 401;  
**int** SC\_PAYMENT\_REQUIRED = 402;  
**int** SC\_FORBIDDEN = 403;  
**int** SC\_NOT\_FOUND = 404;  
**int** SC\_METHOD\_NOT\_ALLOWED = 405;  
**int** SC\_NOT\_ACCEPTABLE = 406;  
**int** SC\_PROXY\_AUTHENTICATION\_REQUIRED = 407;  
**int** SC\_REQUEST\_TIMEOUT = 408;  
**int** SC\_CONFLICT = 409;  
**int** SC\_GONE = 410;  
**int** SC\_LENGTH\_REQUIRED = 411;  
**int** SC\_PRECONDITION\_FAILED = 412;  
**int** SC\_REQUEST\_ENTITY\_TOO\_LARGE = 413;  
**int** SC\_REQUEST\_URI\_TOO\_LONG = 414;  
**int** SC\_UNSUPPORTED\_MEDIA\_TYPE = 415;  
**int** SC\_REQUESTED\_RANGE\_NOT\_SATISFIABLE = 416;  
**int** SC\_EXPECTATION\_FAILED = 417;  
**int** SC\_INTERNAL\_SERVER\_ERROR = 500;  
**int** SC\_NOT\_IMPLEMENTED = 501;  
**int** SC\_BAD\_GATEWAY = 502;  
**int** SC\_SERVICE\_UNAVAILABLE = 503;  
**int** SC\_GATEWAY\_TIMEOUT = 504;  
**int** SC\_HTTP\_VERSION\_NOT\_SUPPORTED = 505;

Методи інтерфейса HttpSevletResponse:

**void addCookie(Cookie cookie)** - вводить вказаний файл cookie в НТТР відповідь.

**boolean containsHeader(String header)** - повертає істину, якщо в заголовку НТТР-відповіді є вказаний хедер.

**String encodeURL(String url)** - визначає чи повинен ідентифікатор сеанса зв’язку бути закодованим в вказанному URL. Якщо повинен, то повертається змінена версія вказаного URL, інакше - сам вказаний URL. Цим методом повинні оброблятися **всі** **URL** які формує сервлет!

**String encideRedirectURL(String URL)** - визначає чи повинен ідентифікатор сеанса зв’язку бути закодованим в вказанному URL. Якщо повинен, то повертається змінена версія вказаного URL, інакше - сам вказаний URL. Цим методом повинні оброблятися **всі** **URL** які передаються методу send/redirect()!

**void sendRedirect(String url) throws IOE** - переадресовує клієнта по вказаному URL.

**void sendError(int errorCode) throws IOE**- повертає клієнту код помилки.

**void sendError(int errorCode, String errorMsg) throws IOE**- повертає клієнту код помилки і строку з параметра errorMsg.

**void setDateHeader(String str, long millis)** - вводить вказане поле в заголовок зі значенням дати в мілісекундах. (з 1970 року - ера UNIX)

**void setHeader(String field, String value)** - вводить в загаловок поле field зі значенням value.

**void setIntHeader(String field, int value)** - вводить в загаловок поле field із цілочисельним значенням value.

**void setStatus(int code)** - вказує код статусу даної НТТР відповіді.

**Інтерфейс HttpSession.** Дозволяє читати і записувати в сервлеті дані про стан отточення сервлета під час виконання з’язані з сеансом по протоколу НТТР. Методи (всі генерують IllegalStateException):

**Object getAttribute(String attribute)** - повертає значення атрибута attribute. null, якщо даний атрибут не знайдено.

**Enumeration<String> getAttributeNames()** - повертає перечислення імен атрибутів, які пов’язані з даним сеансом зв’язку.

**long getCreationTime()** - повертає час створення сеансу зв’язку. (з 1970)

**String getId()** - повертає ідентифкатор сенса зв’язку.

**long getLastAccesseTime()** - повертає час, який пройшов з того моменту, коли клієн в останнє робив запит в даному сенсі зв’язку. (з 1970)

**void invalidate()** - відміняє даний сеанс і видалє його з контексту.

**boolean isNew()** - повертає істину, якщо сервер створив сеанс зв’язку ще недоступний клієнту.

**void removeAttribute(String attribute)** - видаляє атрибут attribute з сеансу зв’язку.

**void seAttribute(String attribute, Object value)** - пов’язує значення value з іменем даного атрибута attribute.

**Клас Cookie.** Клас інкапсулює файл кукі, який зберігається на стороні клієнта і містить в собі дані стану. Користуватися кукі-файлами корисно для відслідковування активності клієнтів. В кукі файл можна зберегти ім’я клієнта, його адрес, телефон і іншу потрібну інформацію і клієнту не треба буде вводити все заново, якщо він знову зайде на ресурс. Сервлет може зберегти кукі файл на машині клієнта за допомогою метода **addCookie**() з інтерфейса **HttpServletResponse**. Дані з нього потім будуть включатися в заголовок НТТР запиту. Частина значень з кукі файлу:

- ім’я кукі файлу

- значення з кукі файлу

- срок дії кукі файлу. (Визначатиме коли кукі файл буде видалено з машини клієнта. Якщо срок дії не зазначено, тоді кукі будуть видалені одразу після завершення сенсу з’єднання (сесії).)

- домен і шлях до кукі файлу (визначають, коли кукі файл буде включено в НТТР запит. Якщо клієнт вводить URL де домен і шлях співпадають з нашими значеннями, то кукі файл передається серверу, інакше - не передається)

Cookie – якщо сервер післав відповідь де в хедері є кукі і його значення, то клієнт повинен зберегти це кукі. І кожен раз, коли клієнт посилає запит на сервер, то він повинен при запиті добавляти ці кукі.

Клас Cookie має тільки один конструктор: **Cookie(Sting attribute, String value)**

Методи класу Cookie:

**Object clone()** - повертає копію даного об’єкта.

**String getComment()** - повертає коментар.

**String getDomain()** - повертає домен.

**int getMaxAge()** - повертає максимально можливий срок кукі файлу в секундах.

**String getName()** - повертає ім’я.

**String getPath()** - повертає шлях.

**boolean getSecure()** - істину, якщо кукі файл є безпечним.

**String getValue()** - повертає значення.

**int getVersion()** - повертає версію.

**boolean isHttpOnly()** - істина, якщо кукі файл містить значення HttpOnly.

**void setComment(String comment)** - задає комент.

**void setDomain(String domain)** - задає домен.

**void setHttpOnly(boolean onlyHttp)** - якщо значення onlyHttp задається істинним, то в кукі файл добавляється параметер HttpOnly. Якщо задається фолс, то параметер HttpOnly видаляється з кукі файлу.

**void setMaxAge(int seconds) -** задає максимальний строк дії кукі в секундах. Після закінчення сроку кукі видаляються.

**void setPath(String path)** - задає вказаний шлях.

**void setSecure(boolean secure)** - встановлює безпечність кукі файла.

**void setValue(String value)** - задає значення.

**void setVersion(int version)** - задає версію.

**Клас HttpServlet .** Розширяє клас GenericServlet і використовується для прийому та обробки НТТР запитів. Методи класу, які перевизначаються для використання в сервлеті:

**void** **doDelete**(**HttpServletRequest** **request**, **HttpServletResponse** **response**) throws IOE, ServletException - обробляє НТТР запити типу **DELETE**.

**void** **doGet**(**HttpServletRequest** **request**, **HttpServletResponse** **response**) throws IOE, ServletException - обробляє НТТР запити типу **GET**.

**void** **doOptions**(**HttpServletRequest** **request**, **HttpServletResponse** **response**) throws IOE, ServletException - обробляє НТТР запити типу **OPTIONS**.

**void** **doPost**(**HttpServletRequest** **request**, **HttpServletResponse** **response**) throws IOE, ServletException - обробляє НТТР запити типу **POST**.

**void** **doPut**(**HttpServletRequest** **request**, **HttpServletResponse** **response**) throws IOE, ServletException - обробляє НТТР запити типу **PUT**.

**void** **doTrace**(**HttpServletRequest** **request**, **HttpServletResponse** **response**) throws IOE, ServletException - обробляє НТТР запити типу **TRACE**.

**Long getLastModified(HttpServletRequest request)** - повертає момент часу в мілісекундах, коли було в останнє змінено запрошуваний ресурс.

**void** **service**(**HttpServletRequest** **request**, **HttpServletResponse** **response**) throws IOE, ServletException - метод викликається сервлетом, коли до сервлета відбувається НТТР запит. Відповідно, request - працює з запитом, response - працює з відповіддю.

Приклад НТТР запиту типу GET:

HTML форма яка відправляє запит типу GET (відправить колір в строці URL - **http://localhost:8080/colorservlet?color=Red**):

<**html**>  
<**head**>  
 <**title**>Color Servlet</**title**>  
</**head**>  
<**body bgcolor="silver"**>  
<**form name="Form1" action="http://localhost:8080/examples/servlets/servlet/ColorGetServlet"**>  
 <**b**>Color:</**b**>  
 <**select name="color" size="1"**>  
 <**option value="Red"**>Red</**option**>  
 <**option value="Blue"**>Blue</**option**>  
 <**option value="Green"**>Green</**option**>  
 </**select**>  
 <**br**>  
 <**br**>  
 <**input type="submit" value="Send"**>  
</**form**>  
</**body**>  
</**html**>

Сервлет, який викликаєть після відправлення форми і обробляє запит типу GET:

**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
  
**public class** ColorGetServlet **extends** HttpServlet {  
  
 **public void** doGet(HttpServletRequest request,  
 HttpServletResponse response) **throws** ServletException, IOException {

//get color from request form  
 String color = request.getParameter(**"color"**);

//set content type to answer  
 response.setContentType(**"text/html"**);

PrintWriter writer = response.getWriter();  
 writer.println(**"<b>The selected color is: </b>"**);  
 writer.println(color);  
 writer.close();  
 }  
}

**Порядок запуску даного сервлета в IteliJIDEA + Tomcat:**

1. Створюємо проект. Вказуємо web application.

2. В index.jsp створюємо HTML форму з GET запитом:

<%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %>  
<**html**>  
<**head**>  
 <**title**>Color Servlet</**title**>  
</**head**>  
<**body bgcolor="silver"**>  
<**form name="Form1"  
 method="get"  
 action="http://localhost:8080/color"**>  
 <**b**>Color:</**b**>  
 <**select name="color" size="1"**>  
 <**option value="Red"**>Red</**option**>  
 <**option value="Blue"**>Blue</**option**>  
 <**option value="Green"**>Green</**option**>  
 </**select**>  
 <**br**>  
 <**br**>  
 <**input type="submit" value="Send"**>  
</**form**>  
</**body**>  
</**html**>

</**html**>

3. Створюємо сам сервлет, який буде оробляти цю фому:

**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.rmi.ServerException;  
  
**public class** ColorServlet **extends** HttpServlet {  
 **public void** doGet(HttpServletRequest request,  
 HttpServletResponse response) **throws** IOException, ServerException{  
 String color = request.getParameter(**"color"**);  
 response.setContentType(**"text/html"**);  
 PrintWriter writer = response.getWriter();  
 writer.println(**"<b>Color is: </b>"**);  
 writer.println(color);  
 writer.close();  
  
 }  
}

4. Реєструємо сервлет в файлі web.xml:

*<?***xml version="1.0" encoding="UTF-8"***?>*<**web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  
 version="3.1"**>  
  
 <**servlet**>  
 <**servlet-name**>ColorServlet</**servlet-name**>  
 <**servlet-class**>ColorServlet</**servlet-class**>  
 </**servlet**>  
  
 <**servlet-mapping**>  
 <**servlet-name**>ColorServlet</**servlet-name**>  
 <**url-pattern**>/colorservlet</**url-pattern**>  
 </**servlet-mapping**>  
  
</**web-app**>

5. Конфігуруємо Томкат, в вкладці Deployment додаємо наш сервлет як war exploded.

6. Запускаємо Томкат.

**MANIFEST файл (\*.MF, \*.manifest)** - перелік вмісту (файлів) пакету. Також може містити додаткову інформацію типу номера версії пакету (JAR), та точку розпакуання і крипографічний хеш.

Приклад НТТР запиту типу **POST**:

HTML форма яка відправляє запит типу POST (не відправить колір в строці URL, як при використанні метода GET. Колір буде відправлено в самому тілі НТТР запиту. **http://localhost:8080/colorservlet**):

<%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %>  
<**html**>  
<**head**>  
 <**title**>Color Servlet</**title**>  
</**head**>  
<**body bgcolor="silver"**>  
<**form name="Form1"  
 method="post"  
 action="http://localhost:8080/color"**>  
 <**b**>Color:</**b**>  
 <**select name="color" size="1"**>  
 <**option value="Red"**>Red</**option**>  
 <**option value="Blue"**>Blue</**option**>  
 <**option value="Green"**>Green</**option**>  
 </**select**>  
 <**br**>  
 <**br**>  
 <**input type="submit" value="Send"**>  
</**form**>  
</**body**>  
</**html**>

Сервлет, який викликаєть після відправлення форми і обробляє запит типу POST:

**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
  
**public class** ColorGetServlet **extends** HttpServlet {  
  
 **public void** doPost(HttpServletRequest request,  
 HttpServletResponse response) **throws** ServletException, IOException {

//get color from form request  
 String color = request.getParameter(**"color"**);

//set content type to answer  
 response.setContentType(**"text/html"**);

PrintWriter writer = response.getWriter();  
 writer.println(**"<b>The selected color is: </b>"**);  
 writer.println(color);  
 writer.close();  
 }  
}

**Приклад використання Cookie файлів**. Створюємо форму, в якій заповнюємо поле зі значенням куки. Сервлет AddCookieServlet обробляє запит по POST, створює новий Cookie об’єкт і повертає нам його значення. Інший сервлет GetCookieServlet може обробляти запит форми по GET.

index.jsp:

<%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %>  
<**html**>  
<**head**>  
 <**title**>Send cookie page</**title**>  
</**head**>  
<**body**>  
  
<**form name="cookieForm"  
 method="post"  
 action="http://localhost:8080/AddCookieServlet"**>  
 <**b**>Enter cookie value: </**b**>  
 <**br**>  
 <**input type="textbox" size="25" name="data" value=""**>  
 <**input type="submit" value="Send cookie"**>  
</**form**>  
  
</**body**>  
</**html**>

Сервлет AddCookieServlet:

**import** javax.servlet.http.Cookie;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.rmi.ServerException;  
  
**public class** AddCookieServlet **extends** HttpServlet {  
  
 **public void** doPost(HttpServletRequest request,  
 HttpServletResponse response) **throws** ServerException, IOException {  
  
 String data = request.getParameter(**"data"**);  
 *//creating cookie* Cookie cookie = **new** Cookie(**"MyCookie"**, data);  
  
 response.setContentType(**"text/html"**);  
 PrintWriter writer = response.getWriter();  
 writer.println(**"MyCookie has ben send to: "**);  
 writer.println(data);  
 writer.close();  
 }  
}

Сервлет GetCookieServlet:

**import** javax.servlet.ServletException;  
**import** javax.servlet.http.Cookie;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
  
**public class** GetCookieServlet **extends** HttpServlet {  
  
 **public void** doGet(HttpServletRequest request,  
 HttpServletResponse response) **throws** ServletException, IOException {  
  
 Cookie[] cookies = request.getCookies();  
  
 response.setContentType(**"text/html"**);  
 PrintWriter writer = response.getWriter();  
  
 **for** (**int** i = 0; i < cookies.**length**; i++) {  
 String name = cookies[i].getName();  
 String value = cookies[i].getValue();  
 writer.println(**"Cookie name = <b>"** + name + **"</b>. Cookie value = <b>"** + value + **"</b>"**);  
 }  
 writer.close();  
  
 }  
  
}

Мережевий протокол НТТР діє без збереження стану. Кожний наступний запит не залежить від попереднього. Але інколи потрібно зберігати стан взаємодії між клієнтом і сервером, тоді застосовують сеанси зв’язку. Сеанс зв’язку можна отримати за допомогою метода **getSession**() з інтерфейса **HttpServletRequest** і він повертає об’єкт типу **HttpSession**. Цей об’єкт може зберігати ряд імен прив’язаних до об’єктів. Цими привязками керують методи getAttribute(), setAttribute(), getAttributeNames() і removeAttributt() з і-са HttpSession.

**Приклад відслідковування сеансів зв’язку.** Метод getSession() отримує поточний сеанс звязку. Якщо сеанс до цього ще не був створений, то метод створює новий сеанс звязку. Метод getAttribute() отримує обєкт привязаний до параметра date, що містить в собі час останнього доступу до сесії. Потім створюється обєкт Date, який показує поточний час. І сервлет вивиодить дату останнього доступу сесії і поточний час. (при першому зверненні дата останнього доступу буде порожньою).

**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** javax.servlet.http.HttpSession;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.rmi.ServerException;  
**import** java.util.Date;  
  
**public class** DateServlet **extends** HttpServlet {  
 **public void** doGet(HttpServletRequest request,  
 HttpServletResponse response) **throws** ServerException, IOException {  
  
 *//get the session object* HttpSession session = request.getSession();  
  
 response.setContentType(**"text/html"**);  
 PrintWriter writer = response.getWriter();  
 writer.print(**"<b>"**);  
  
 *//date of last session action* Date date = (Date) session.getAttribute(**"date"**);  
 **if** (date != **null**) {  
 writer.println(**"Last action: "** + date + **"<br>"**);  
 }  
  
 *//current date* date = **new** Date();  
 session.setAttribute(**"date"**, date);  
 writer.println(**"Current date: "** + date);  
 }  
}

Всі класи\інтерфйси які є в ServletAPI є і в Tomcat + класи імплементації.

Томкет має свій класлоадер і кожен варнік має свій класлоадер який ніби наслідується від класлоадера Томкета. При запуску аплікухи, варнік шукає спочатку класи в предку класлоадера (який в свою чергу показує лише частину класів), а якщо там таких нема, то класлоадер варніка шукає класи в самому варніку.

При запиті на сервер для запиту виділяється один потік з пулу потока.

Екземпляр сервлета створюється коли на нього відбудеться запит, а не в момент старту аплікухи.

Також контейнер сервлетів має право видаляти сервлет, якщо на нього не звертаються певний час.

Сервлет контейнер має право створювати любу кількість екземпярів сервлета.

<load-on-startup>1</ load-on-startup > - тег в web.xml (чи метод onStartup() в джава конфіг) вказує на те, що екземпляр сервелета буде створено при старті аплікухи. По цифрі можна налаштовувати порядок створення сервлетів.

GC не оперує з недоступними об’єктами, він просто вважає, що це пустий сегмент пам’яті і дефрагментує його - за рахунок цього видалення об’єктів майже не вартує ресурсів (за вийнятком, якщо ми заовверрайдимо пустий метод finalize класу Object). Створення нових об’єктів в Джаві є відносно незатратним, за рахунок того, що GC є дефрагментуючим, ВМ розуміє, що вся пам’ять в ОЗУ "з ліва" зайнята, а "з права" вільна. Тому при створенні об’єкта просто інкрементується кількість потрібної пам’яті до вже зайнятої, що й дає продуктивність й швидкість при створенні.

Метод fianlize() вважаєть не дуже вдалим і замість нього пропонують використовувати phantom references.

Категорично не рекомендується у себе заводити поля HttpServletRequest i HttpServletResponse. Це порушує потокостійкість, і може призвести до ситуації, коли запит буде оброблятися в запам’ятованому полі, а з пула реквестів іншому запиту видадуть цей же тимчасовий елемент пула, тоді 2 реквеста будуть міняти одне поле.

Рівні зберігання, які представляє сервлет контейнер: Request, Session, ServletContext.

З сервлета можна передавати пару: request/response.