

Java Networking

OVERVIEW OF NETWORKING

WHAT IS JAVA NETWORKING?

COMMON NETWORK PROTOCOLS

CREATING URL

JAVA NETWORKING TERMINOLOGY

SOCKET PROGRAMMING

NETWORK INTERFACE

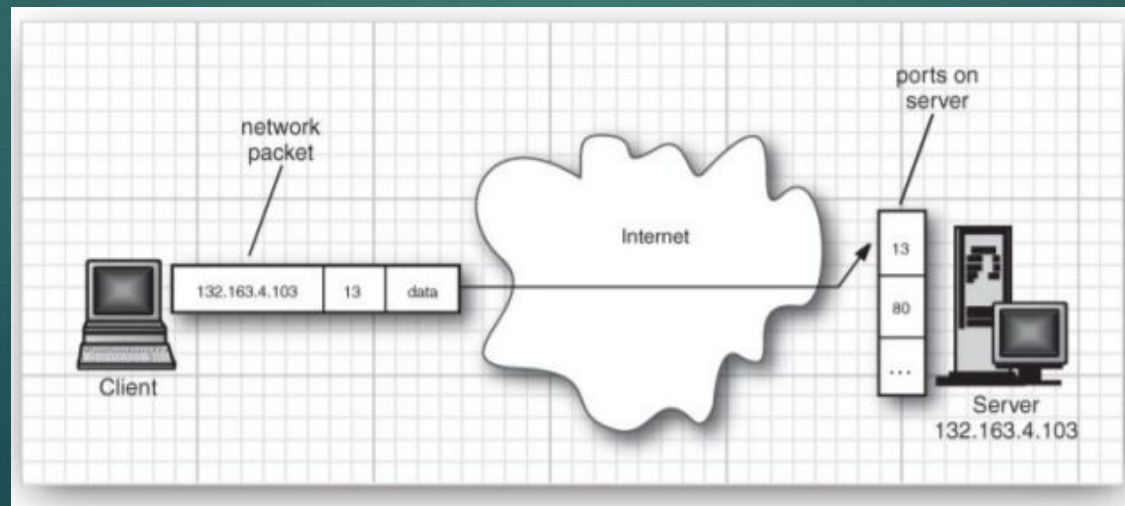
POSTING FORM DATA

CLIENT SERVER

OVERVIEW OF SOCKET DIRECT PROTOCOL

Overview Of networking

- ▶ When computing devices such as laptops, desktops, servers, smartphones, and tablets and an eternally-expanding arrangement of IoT gadgets such as cameras, door locks, doorbells, refrigerators, audio/visual systems, thermostats, and various sensors are sharing information and data with each other is known as networking.
- ▶ In simple words, the term network programming or networking associates with writing programs that can be executed over various computer devices, in which all the devices are connected to each other to share resources using a network.



What is Java Networking?

- ▶ Networking supplements a lot of power to simple programs. With networks, a single program can regain information stored in millions of computers positioned anywhere in the world. Java is the leading programming language composed from scratch with networking in mind. Java Networking is a notion of combining two or more computing devices together to share resources.
- ▶ All the Java program communications over the network are done at the application layer. The **java.net** package of the J2SE APIs comprises various classes and interfaces that execute the low-level communication features, enabling the user to formulate programs that focus on resolving the problem.

Common Network Protocols

- ▶ The **java.net.*** package provides support for the two common network protocols –
- ▶ **Transmission Control Protocol (TCP)** – TCP or Transmission Control Protocol allows secure communication between different applications. TCP is a connection-oriented protocol which means that once a connection is established, data can be transmitted in two directions. This protocol is typically used over the Internet Protocol. Therefore, TCP is also referred to as TCP/IP. TCP has built-in methods to examine for errors and ensure the delivery of data in the order it was sent, making it a complete protocol for transporting information like still images, data files, and web pages.
- ▶ **User Datagram Protocol (UDP)** – UDP or User Datagram Protocol is a connection-less protocol that allows data packets to be transmitted between different applications. UDP is a simpler Internet protocol in which error-checking and recovery services are not required. In UDP, there is no overhead for opening a connection, maintaining a connection, or terminating a connection. In UDP, the data is continuously sent to the recipient, whether they receive it or not.

Creating URL

- ▶ The easiest way to create a URL object is from a String that represents the human-readable form of the URL address. This is typically the form that another person will use for a URL. In your Java program, you can use a String containing this text to create a URL object:
- ▶ `URL myURL = new URL("http://example.com/");`
- ▶ The URL object created above represents an absolute URL. An absolute URL contains all of the information necessary to reach the resource in question. You can also create URL objects from a relative URL address.

Reading directly from URL

- ▶ After you've successfully created a URL, you can call the URL's `openStream()` method to get a stream from which you can read the contents of the URL. The `openStream()` method returns a `java.io.InputStream` object, so reading from a URL is as easy as reading from an input stream.
- ▶ The following small Java program uses `openStream()` to get an input stream on the URL `http://www.oracle.com/`. It then opens a `BufferedReader` on the input stream and reads from the `BufferedReader` thereby reading from the URL. Everything read is copied to the standard output stream:


```
import java.net.*;
import java.io.*;

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

        URL oracle = new URL("http://www.oracle.com/");
        BufferedReader in = new BufferedReader(
            new InputStreamReader(oracle.openStream()));

        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Connecting to URL

- ▶ After you've successfully created a URL object, you can call the URL object's `openConnection` method to get a `URLConnection` object, or one of its protocol specific subclasses, e.g. `java.net.HttpURLConnection`
- ▶ You can use this `URLConnection` object to setup parameters and general request properties that you may need before connecting. Connection to the remote object represented by the URL is only initiated when the `URLConnection.connect` method is called. When you do this you are initializing a communication link between your Java program and the URL over the network. For example, the following code opens a connection to the site `example.com`:


```
try {
    URL myURL = new URL("http://example.com/");
    URLConnection myURLConnection = myURL.openConnection();
    myURLConnection.connect();
}
catch (MalformedURLException e) {
    // new URL() failed
    // ...
}
catch (IOException e) {
    // openConnection() failed
    // ...
}
```

Reading from URL Connection

- ▶ The following program performs the function of Reading Directly from a URL.
- ▶ However, rather than getting an input stream directly from the URL, this program explicitly retrieves a `URLConnection` object and gets an input stream from the connection. The connection is opened implicitly by calling `getInputStream`. Then, this program creates a `BufferedReader` on the input stream and reads from it.

```
import java.net.*;
import java.io.*;

public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL oracle = new URL("http://www.oracle.com/");
        URLConnection yc = oracle.openConnection();
        BufferedReader in = new BufferedReader(new InputStreamReader(
            yc.getInputStream()));

        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```


Working with URL example

```
// Java program to demonstrate working of URL
import java.net.MalformedURLException;
import java.net.URL;

public class URLclass1
{
    public static void main(String[] args)
        throws MalformedURLException
    {
        // creates a URL with string representation.
        URL url1 =
            new URL("https://www.google.co.in/?gfe_rd=cr&ei=ptYq" +
                "WK26I4fT8gfth6CACg#q=geeks+for+geeks+java");

        // creates a URL with a protocol,hostname,and path
        URL url2 = new URL("http", "www.geeksforgeeks.org",
            "/jvm-works-jvm-architecture/");

        URL url3 = new URL("https://www.google.co.in/search?" +
            "q=gnu&rlz=1C1CHZL_enIN71" +
            "4IN715&oq=gnu&aqs=chrome..69i57j6" +
            "9i6015.653j0j7&sourceid=chrome&ie=UTF" +
            "-8#q=geeks+for+geeks+java");

        // print the string representation of the URL.
        System.out.println(url1.toString());
        System.out.println(url2.toString());
        System.out.println();
        System.out.println("Different components of the URL3-");

        // retrieve the protocol for the URL
        System.out.println("Protocol:- " + url3.getProtocol());

        // retrieve the hostname of the url
        System.out.println("Hostname:- " + url3.getHost());

        // retrieve the default port
        System.out.println("Default port:- " +
            url3.getDefaultPort());
```

```
        // retrieve the query part of URL
        System.out.println("Query:- " + url3.getQuery());

        // retrieve the path of URL
        System.out.println("Path:- " + url3.getPath());

        // retrieve the file name
        System.out.println("File:- " + url3.getFile());

        // retrieve the reference
        System.out.println("Reference:- " + url3.getRef());
    }
}
```

https://www.google.co.in/?gfe_rd=cr&ei=ptYqWK26I4fT8gfth6CACg#q=geeks+for+geeks+java
<https://www.geeksforgeeks.org/jvm-works-jvm-architecture/>

Different components of the URL3-

Protocol:- https

Hostname:- www.google.co.in

Default port:- 443

Query:- q=gnu&rlz=1C1CHZL_enIN714IN715&oq=gnu&aqs=chrome..69i57j69i6015.653j0j7&sourceid=chrome&ie=UTF-8#q=geeks+for+geeks+java


Path:- /search

File:- /search?q=gnu&rlz=1C1CHZL_enIN714IN715&oq=gnu&aqs=chrome..69i57j69i6015.653j0j7&sourceid=chrome&ie=UTF-8#q=geeks+for+geeks+java


Reference:- q=geeks+for+geeks+java

Java networking terminology

- ▶ IP Address
- ▶ Port Number
- ▶ Protocol
- ▶ Mac Address
- ▶ Socket
- ▶ Communication oriented and communication less

- 
- ▶ IP Address - An IP address is a unique address that distinguishes a device on the internet or a local network. IP stands for “Internet Protocol.” It comprises a set of rules governing the format of data sent via the internet or local network. IP Address is referred to as a logical address that can be modified. It is composed of octets. The range of each octet varies from 0 to 255.
 - Range of the IP Address – 0.0.0.0 to 255.255.255.255
 - For Example – 192.168.0.1
 - ▶ Port Number - A port number is a method to recognize a particular process connecting internet or other network information when it reaches a server. The port number is used to identify different applications uniquely. The port number behaves as a communication endpoint among applications. The port number is correlated with the IP address for transmission and communication among two applications. There are 65,535 port numbers, but not all are used every day.

- 
- ▶ Protocol - A network protocol is an organized set of commands that define how data is transmitted between different devices in the same network. Network protocols are the reason through which a user can easily communicate with people all over the world and thus play a critical role in modern digital communications. For Example – TCP, FTP, POP, etc.
 - ▶ Mac Address - MAC address stands for Media Access Control address. It is a bizarre identifier that is allocated to a NIC (Network Interface Controller/ Card). It contains a 48 bit or 64-bit address, which is combined with the network adapter. MAC address can be in hexadecimal composition. In simple words, a MAC address is a unique number that is used to track a device in a network.

- 
- ▶ **Socket** - A socket is one endpoint of a two-way communication connection between the two applications running on the network. The socket mechanism presents a method of inter-process communication (IPC) by setting named contact points between which the communication occurs. A socket is tied to a port number so that the TCP layer can recognize the application to which the data is intended to be sent.
 - ▶ **Communication oriented and communication less** - In a connection-oriented service, the user must establish a connection before starting the communication. When the connection is established, the user can send the message or the information, and after this, they can release the connection. However, In connectionless protocol, the data is transported in one route from source to destination without verifying that the destination is still there or not or if it is ready to receive the message. Authentication is not needed in the connectionless protocol.
 - Example of Connection-oriented Protocol – Transmission Control Protocol (TCP)
 - Example of Connectionless Protocol – User Datagram Protocol (UDP)

Java networking classes

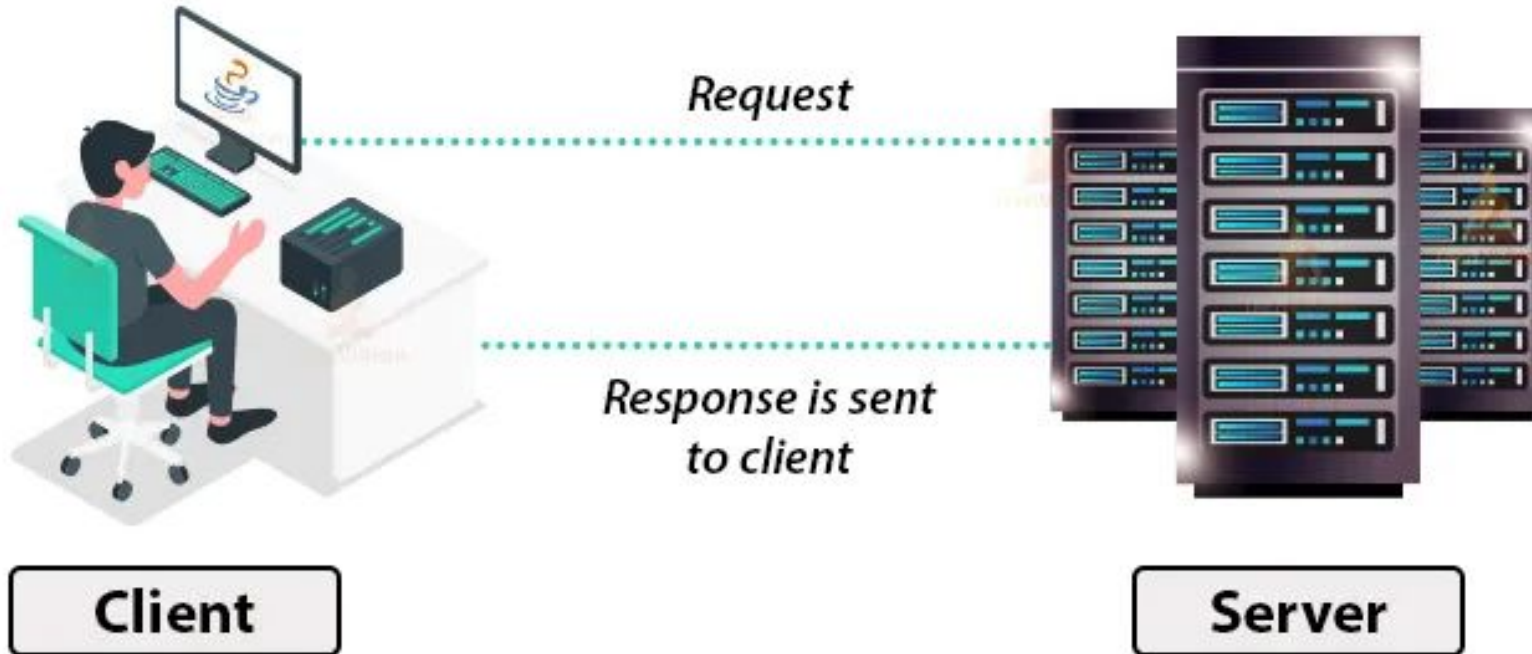
- ▶ CacheRequest
- ▶ CookieHandler
- ▶ CookieManager
- ▶ DatagramPacket
- ▶ InetAddress
- ▶ Server Socket
- ▶ Socket
- ▶ DatagramSocket
- ▶ Proxy
- ▶ URL
- ▶ URLConnection

Socket Programming

- ▶ Java Socket Programming is practiced for communication between the applications working on different JRE. Sockets implement the communication tool between two computers using TCP. Java Socket programming can either be connection-oriented or connection-less. In Socket Programming, Socket and ServerSocket classes are managed for connection-oriented socket programming. However, DatagramSocket and DatagramPacket classes are utilized for connection-less socket programming.
- ▶ A client application generates a socket on its end of the communication and strives to combine that socket with a server. When the connection is established, the server generates an object of socket class on its communication end. The client and the server can now communicate by writing to and reading from the socket.
- ▶ Visit the link for brief description of classes and methods used by server socket :
https://www.tutorialspoint.com/java/java_networking.htm

Socket Programming

Java Socket Programming Process



Network Interface

- ▶ You can access network parameters about a network interface beyond the name and IP addresses assigned to it
- ▶ You can discover if a network interface is “up” (that is, running) with the `isUP()` method. The following methods indicate the network interface type:
 - `isLoopback()` indicates if the network interface is a loopback interface.
 - `isPointToPoint()` indicates if the interface is a point-to-point interface.
 - `isVirtual()` indicates if the interface is a virtual interface.
- ▶ The `supportsMulticast()` method indicates whether the network interface supports multicasting.
- ▶ The `getHardwareAddress()` method returns the network interface's physical hardware address, usually called MAC address, when it is available.
- ▶ The `getMTU()` method returns the Maximum Transmission Unit (MTU), which is the largest packet size.


```

import java.io.*;
import java.net.*;
import java.util.*;
import static java.lang.System.out;

public class ListNetsEx {

    public static void main(String args[]) throws SocketException {
        Enumeration<NetworkInterface> nets = NetworkInterface.getNetworkInterfaces();
        for (NetworkInterface netint : Collections.list(nets))
            displayInterfaceInformation(netint);
    }

    static void displayInterfaceInformation(NetworkInterface netint) throws SocketException
    {
        out.printf("Display name: %s\n", netint.getDisplayName());
        out.printf("Name: %s\n", netint.getName());
        Enumeration<InetAddress> inetAddresses = netint.getInetAddresses();

        for (InetAddress inetAddress : Collections.list(inetAddresses)) {
            out.printf("InetAddress: %s\n", inetAddress);
        }

        out.printf("Up? %s\n", netint.isUp());
        out.printf("Loopback? %s\n", netint.isLoopback());
        out.printf("PointToPoint? %s\n", netint.isPointToPoint());
        out.printf("Supports multicast? %s\n", netint.supportsMulticast());
        out.printf("Virtual? %s\n", netint.isVirtual());
        out.printf("Hardware address: %s\n",
            Arrays.toString(netint.getHardwareAddress()));
        out.printf("MTU: %s\n", netint.getMTU());
        out.printf("\n");
    }
}

```

The following is sample output from the example program:

```

Display name: bge0
Name: bge0
InetAddress: /fe80:0:0:0:203:baff:fef2:e99d%2
InetAddress: /129.156.225.59
Up? true
Loopback? false
PointToPoint? false
Supports multicast? false
Virtual? false
Hardware address: [0, 3, 4, 5, 6, 7]
MTU: 1500

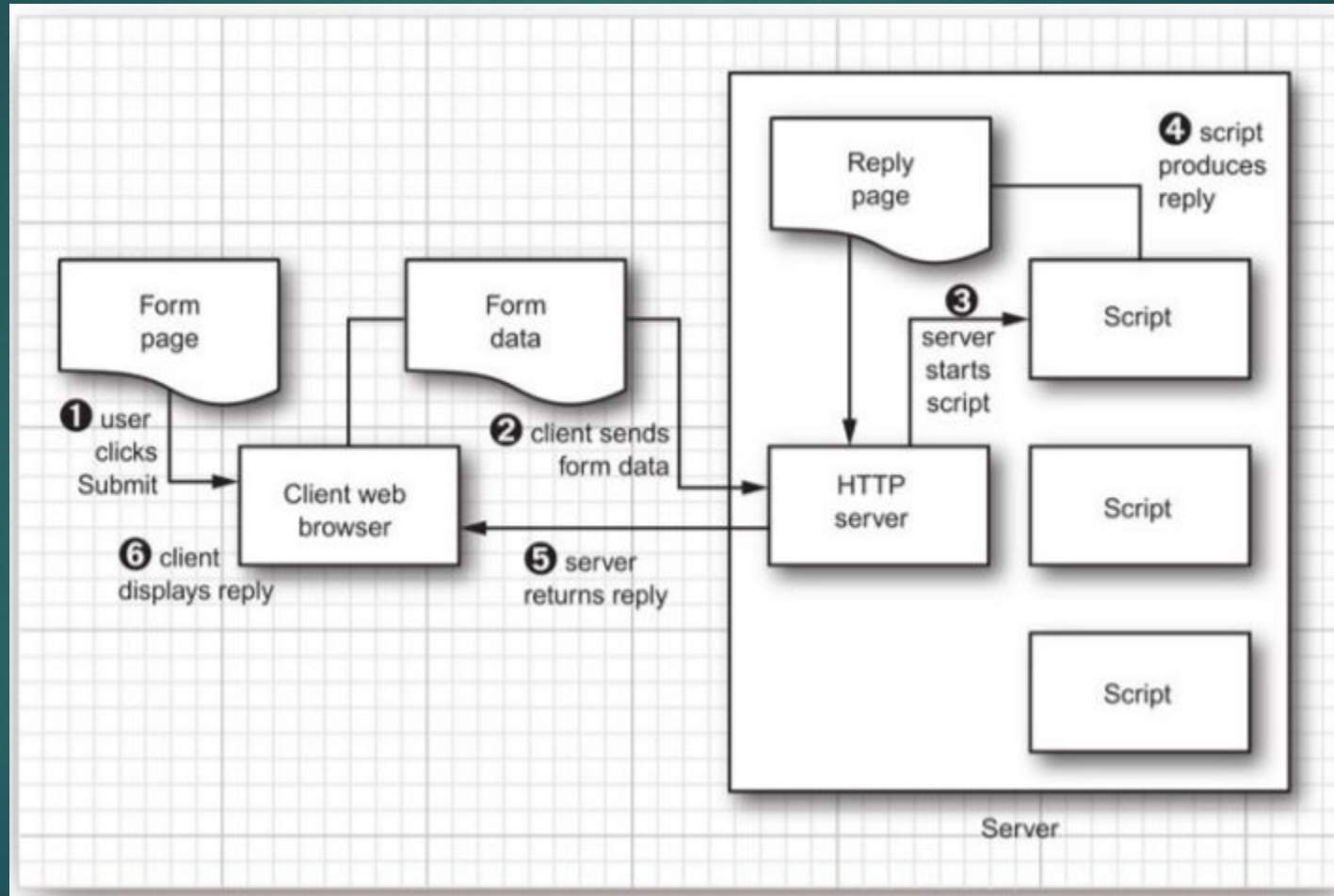
```

```

Display name: lo0
Name: lo0
InetAddress: /0:0:0:0:0:0:0:1%1
InetAddress: /127.0.0.1
Up? true
Loopback? true
PointToPoint? false
Supports multicast? false
Virtual? false
Hardware address: null
MTU: 8232

```

Posting Form Data



Understanding the Sockets Direct Protocol

- ▶ For high performance computing environments, the capacity to move data across a network quickly and efficiently is a requirement. Such networks are typically described as requiring high throughput and low latency. High throughput refers to an environment that can deliver a large amount of processing capacity over a long period of time. Low latency refers to the minimal delay between processing input and providing output, such as you would expect in a real-time application.
- ▶ In these environments, conventional networking using socket streams can create bottlenecks when it comes to moving data. Introduced in 1999 by the InfiniBand Trade Association, InfiniBand (IB) was created to address the need for high performance computing. One of the most important features of IB is Remote Direct Memory Access (RDMA). RDMA enables moving data directly from the memory of one computer to another computer, bypassing the operating system of both computers and resulting in significant performance gains.
- ▶ The Sockets Direct Protocol (SDP) is a networking protocol developed to support stream connections over InfiniBand fabric. SDP support was introduced to the JDK 7 release of the Java Platform, Standard Edition ("Java SE Platform") for applications deployed in the Solaris Operating System ("Solaris OS") and on Linux operating systems. The Solaris OS has supported SDP and InfiniBand since Solaris 10 5/08. On Linux, the InfiniBand package is called OFED (OpenFabrics Enterprise Distribution). The JDK 7 release supports the 1.4.2 and 1.5 versions of OFED.

Understanding the Sockets Direct Protocol

- ▶ SDP support is essentially a TCP bypass technology.
- ▶ When SDP is enabled and an application attempts to open a TCP connection, the TCP mechanism is bypassed and communication goes directly to the IB network. For example, when your application attempts to bind to a TCP address, the underlying software will decide, based on information in the configuration file, if it should be rebound to an SDP protocol. This process can happen during the binding process or the connecting process (but happens only once for each socket).
- ▶ There are no API changes required in your code to take advantage of the SDP protocol: the implementation is transparent and is supported by the classic networking (`java.net`) and the New I/O (`java.nio.channels`) packages. See the Supported Java APIs section for a list of classes that support the SDP protocol.
- ▶ SDP support is disabled by default. The steps to enable SDP support are:
 - ▶ Create an SDP configuration file.
 - ▶ Set the system property that specifies the location of the configuration file.