Advanced Network Programming

<Block 1: Threads>

1.1. Introduce.

Define Thread A process is isolated, has its own independent memory and it is associated with a ~~sig~~ single application. A Thread exist within one process and share the same memory, each have their own stack.

code 创建 Thread

```
① Class EX implement Runnable{ public void run(){...}}.
main→ EX ex = new EX();
      Thread thread = new Thread(ex);
      thread.start();
```

implement 使用 Runnable 接口
实现 run方法.

```
② Class EX extends Thread{ run(){...}}.
   EX ex = new EX();
   ex.start();
```

使用(extends) Thread类
实现 run.

Note 只有Thread才可start(). JAVA不允许多重extends, 所以推荐用Runnable.

code n种方法

```
static       Thread.sleep(ms)   毫秒.
non-static   thread2.join()     当前线程等thread2执行完再执行
             wait()
static       yield();           当前线程自动停止,进入ready状态.
```

Note sleep, join, wait 都进入block状态,有可能抛出异常. throws InterruptedException. 所以这三种方法应在 try{}环境中,并用 catch(InterruptedException e)捕获异常. yield()并不进入阻塞. (static 指Thread类的method).

1.2. Interrupting and terminating threads.

code 中断

```
static       interrupted()    查询当前线程是否中断,并设置中断状态为false.
non-static   interrupt()      设置某线程为中断;如果在中断状态,抛出异常,并设置中断状态为false
non-static   isInterrupted()  查询某线程是否中断.
```

Note 因为 ~~interrupted~~ interrupt 为非静态方法, 所以只能使用 Thread.currentThread.interrupt(). Interrupted 只可查当前线程, 每个线程都有 interrupt flag, 默认为false, 被中断了就是true, 中断之后无事发生, 看我们如何处理.

code 中断线程并停止

① private volatile boolean pleaseFinish = false;
   public void finishThreadPlease() { pleaseFinish = true; }
   在 run() 内周期检查 pleaseFinish flag 并采取行动.

② 在当前线程中创建了另一个线程 Thread newt = new Thread()
   使用 newt.setDaemon(true) 作为守护线程，当前线程终止，newt 自动终止

③ 利用 interrupt 中断 flag，在 catch (InterruptedException e) 中
   ~~中断~~使线程停止，其它线程调用 interrupt().

Note 1) 方法①中，必须 volatile.
2) 方法②中，setDaemon(true) 必须在 start 之前写下.
3) 方法③要注意 interrupt() 会在进入中断的同时设置 flag 为 false，注意手动设定下一步操作，有时没有 block 不会进入 InterruptedException，可以自己周期性检查 interrupt flag.
4) 使用 isAlive() 可检查线程是否存活

1.3. Volatile variables and locks.

key word volatile

Note 1) 当一个 variable 可能被多个 Threads 修改时，用 volatile，此时 load, store, read, write 可以视为 atomic action，++，-- 不是原子(对于 volatile).
2) 对一个对象使用 volatile，它的引用是 volatile 的，但是内部的属性或对象仍无保证.
3) 每个 ~~vari~~ Thread 都有自己的 cache，volatile 让每个线程把 volatile variable 直接 load/read/write from "main memory"，而不是在 Thread ~~cha~~ cache 内操作，但是 volatile 不适合复杂操作.
4). volatile helps JAVA keep variable safe.

Define Thread-safe. 一段代码 guarantee multiple threads 可以安全的执行关于 shared data 的 manipulates (operations) 这段 code 就是 thread safe 的.

Define Critical sections. Code segments within a program that access the same data with parallel threads are known as critical sections.

可以用 synchronized 标识临界区，是有概念 critical code，它是 synchronized 的作用：The code can only be accessed by a ~~sig~~ single thread at any given time.

Code
Intrinstic locks

① public synchronized int getNextValue().

② Synchronized(Object object){ 同步体 }

Note 1) Synchronized 是对 Object 而言的. 显然, 可通过声明 method 为 Synchronized, 但一个对象中所有的 Synchronized 方法 (方法) 共享一个 lock 的, 注意, 而 Synchronized 的方法 即使 线程没有 lock 也可以用. (locks apply to Objects).

2) Synchronized method 可访问同一 Object 的其它 Synchronized 方法, 和可以调用非 Synchronized 方法.

3) Synchronized method 不继承, ~~该~~ 对比 volatile, 只有基本变量可被声明 volatile, 且 volatile 只保证 atomic operation safe, 且不会导致 block 问题.

1.4. The monitor.

Define Monitor 只使用 Synchronized, 可使 Thread 相互排斥. A ~~lock~~ lock that supports Cooperation through the wait() & notify() methods is called a Monitor.

即, 使用 Synchronized, wait(), notify 这个 Object ~~...~~ 的 lock 才是 Monitor.

Code
Monitor 操作

wait() 持有 lock 的线程主动 release lock 进入 Wait set.

notify() 持有 lock 的线程随机叫醒 wait set 中的某个 thread, 使它可以参与下一次资源获得.

notifyAll() 叫醒所有 wait set 中的 Thread.

Note 1) wait, notify, notifyAll 都是 Object 类的方法, 可对任意对象使用, 但都是对当前线程使用! 且必须有锁, 三者都必须在 Synchronized 环境中使用.

2) wait() 主动释放 lock 进入 wait set, Notify 和 notifyAll 中使用线程不释放 lock, 也不进入 wait set, 只是叫醒其它的.

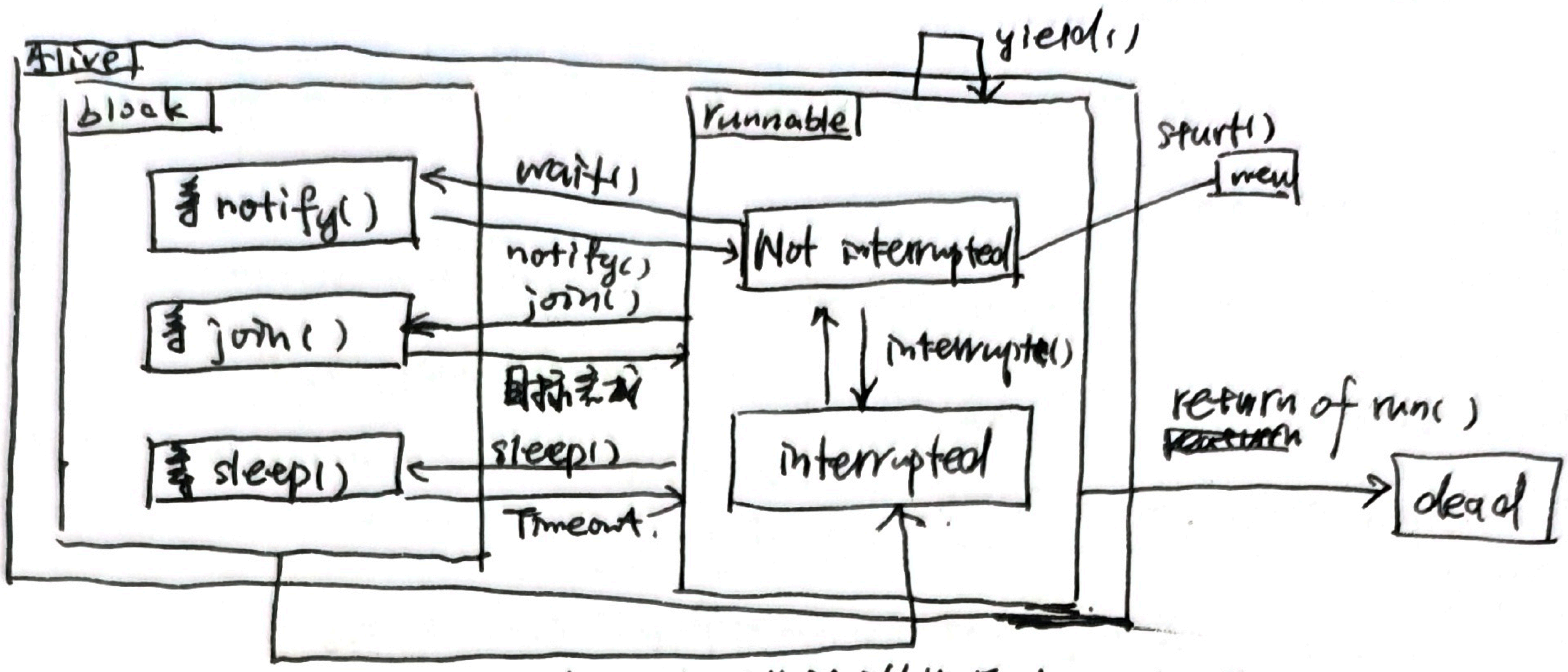
3) Coordination with monitors 要比 join() 更加灵活, ~~虽~~ 虽然, 二者都可以让线程相互协作, 但 monitors 可让线程交替运行, 更灵活.

Define Deadlock. Two or more threads waiting for two or more locks to be freed, and the circumstances in the program is such that the locks will never be freed. 多个线程等待永不被释放的多个 lock.

Solution: 1) Detection and Recovery periodically. 2) avoid ~~method Synches~~ Synchronized. 3) allow interrupt others to get lock. 4) 不让一个 Thread 拥有多个 locks.

Summary

生命周期.



interrupte()将会抛出 InterruptedException.

注： Hidden states in runnable : ready and running

<Block 2 : Sockets, HTTP, HTML and JS>.

2.1. Sockets.

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

Define Stream: A Stream is a connection to a source of data to a destination of data. It can represent any data, a stream is a sequence of bytes that flow from source to destination.

并不只是 byte! Byte stream: InputStream, OutputStream.
Character stream: Reader, Writer.

Define Socket: A Socket is an abstraction of a network interface. → represent the "terminals" of connection (可以用来帮 Server 区分不同 clients)

Code IP

```
String domainName = "www.notsierier.github.io";
InetAddress a = InetAddress.getByName(domainName);
InetAddress addr = InetAddress.getByName(null);
              or InetAddress.getByName("localhost");
              or InetAddress.getByName("127.0.0.1");
```
} 获取localhost的IP.

Note: IP可在Internet中找到主机. IP:port 可以找到主机上的一个 server 或 client.
IP是 32 bit, Port是 0→1023 (1024以后) 80 for http, 443 for https.

Code Socket Server

```
public static final int PORT = 8080;

ServerSocket s = new ServerSocket(PORT);
Socket socket = s.accept();
BufferedRead in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(
                      socket.getOutputStream())), true);
String str = in.readLine();
out.println(str);
socket.close();
s.close();
```

```
Socket socket = new Socket("localhost", 8080); // 客户端创建 Socket
```

Code

Socket

Client

```
InetAddress str = InetAddress.getByName(null);
Socket socket = new Socket(addr, PORT);   // PORT → 8080.
// 创建 BufferedReader, PrintWriter 最后关闭
```

Note 1) 上述忽略了一些代码，与stream相关，都要带 IOException.

PrintWriter → BufferedWriter → OutputStreamWriter → Socket.getOutputStream.

BufferedReader → InputStreamReader → socket.getInputStream.

切记最后关闭socket.

2) 上述代码包含了两种Socket：ServerSocket 和 Socket.

ServerSocket 使用 accept() 方法 接受 Socket.

Socket (Client) 要素需要建立连接的 InetAddress 和 PORT.

2.2. HTTP and the Web.

Some Concepts.

1) HTTP: HyperText Transfer Protocol, A application layer protocol used to transfer web Pages (or other content) from server to Client. And HTTP is based on TCP.

2) TCP: Transmission Control Protocol. ①Connection-oriented protocol. ②provide a reliable connection, end-to-end. 当我们在建立Socket时，就是在建立TCP连接. (HTTP/1.0, 1.1, 2.0 都在TCP之上).

3) URL: Uniform Resource Locator, A way of locating a reasource on the Internet:

protocol://hostname[:post]/path/filename#section.

ex. http://www.company.com/index.html → 使用protocol的default port.

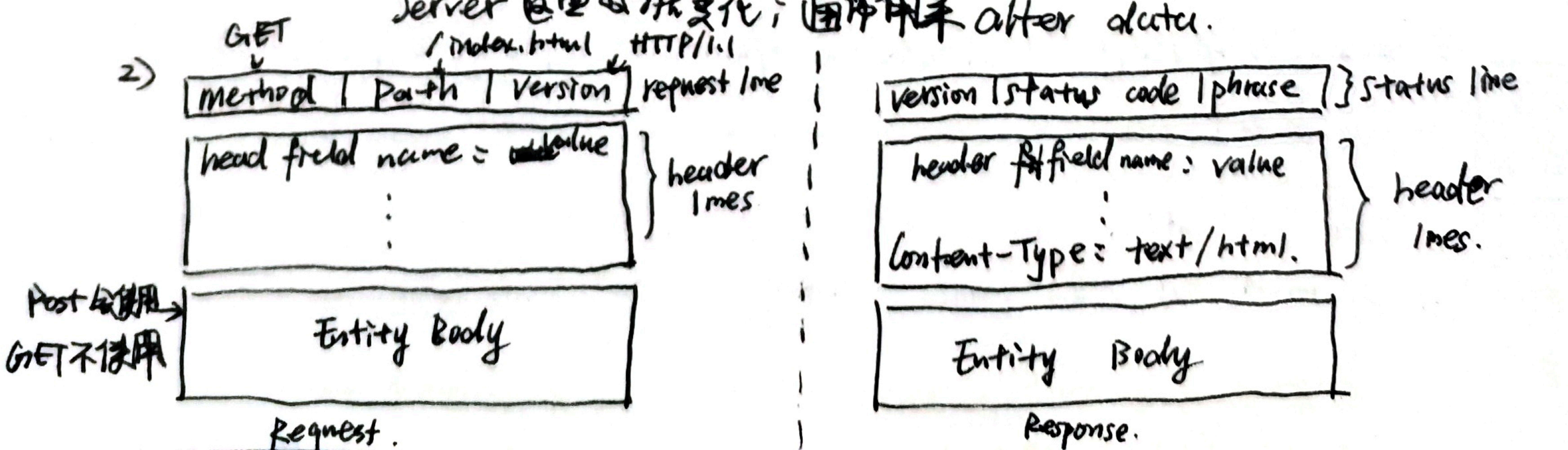
Note HTTP是stateless protocol (无状态协议)，每次连接发送文件后断开，发送新的时，还得多次建立Connetion发送.

Details of HTTP

1) Methods (请求): ① GET: Query string 会包含在URL中; 发送多个GET不会对server造成影响; Cachable, 本地Browser缓存.

② POST: Query string 在 HTTP request body中; 多次POST可能会使Server的数据变化; 通常用来 alter data.

2)



3) 原始的 HTTP 每个 ~~Resource~~ Resource 都建立一个 TCP session，使用
① pipelining ② parallel TCP connections 可以加快数据传输(其实①②是同一个东西).

4) Header: Content-Type = text/html, Set-cookie: xxxx (Server 设置).
cookie: xxxx (Client 发送), Content-Length: xxxx

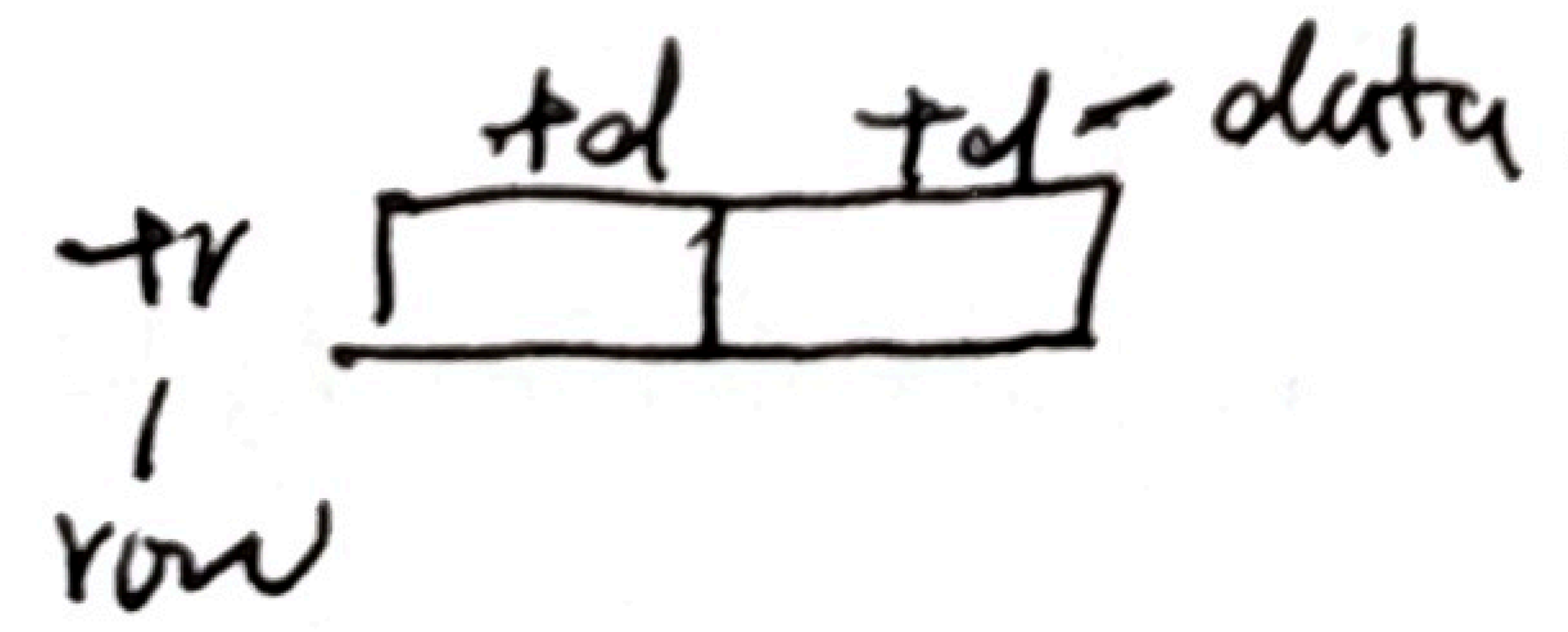
<u>Others</u> 1) HTTPs: HTTP Secure, Run over TLS (Transport Layer Security), Port 443.
缺点: ~~Incr~~ Increase connection setup time; increase page load time.
2) HTTP 2.0: focus on reducing page load times. 和 SPDY 很像.
技术: multiplexing, Universal encryption, Server push / hint, Content send order

2.3、HTML.

Define HTML: Hyper Text Markup Language. Tells the Web browser how to display the page.

Code 基本显示语法

```
<html> ... </html>
<head> ... </head>, <body> ... </body>.
<p> ... </p>, <h1> to <h6> 标题, <title> 网站标题 </title>.
<hr> 横向分隔线, <br> 回车, <b> 粗体 bold, <i> 斜体
转义字符: < : &lt, > : &gt, & : &amp, " : &quot.
<a href = "link.html"> 在同路径下寻找 </a>
<a href = "http://www. ....."> 在不同路径寻找 </a>.
<table border = "1"> → 1为有边框, 0无.
    <tr>
        <td> row1, cell1 </td>
        <td> row2, cell2 </td>
    </tr>
</table>
<ul> unordered list          • list
    <li> list </li>          • list
    <li> list </li>
</ul>
<ol>
    <li> list </li>          1、list
    <li> list </li>          2、list.
</ol>
```



Code Form.

```
<form name = "input" action = "html.jsp" method = "get">.
    <input type = "?" name = "name" id = "id">     } ? 可以是 text, password, radio, checkbox, submit, reset, button
    <label for = "id"> 点击激活该字段, id 为 input </label>.
    <select name = "cars">
        <option value = "1"> 1 </option>            } 下拉菜单.
    </select>
    <textarea row = "10" cols = "30"> 文本域 </textarea>.
```

Note 1) form 被 submit 时，name 会被提交，id 不会被提交，但 id 对每个 tag 来说是唯一的。
2) 对于 "radio" type，同 name 的选项只能选一个，不同任意。
3) 补充一点，<fieldset><legend></legend></fieldset> 定义 fieldset 和 caption.

CSS concept Cascading Style Sheets. Separates the layout from the document structure.

SVG Concept Scalable Vector ~~Form~~ Graphics 可伸缩矢量图。
可直接嵌入 html，Not lose any quality if they are zoomed or resized.

2.4. JavaScript (I).

JS A Scripting language most often used for client-side web development.
应用：1) create interactive user interface in webpage.
2) manipulate web dynamically 3) form validation.

Code Basic

```
<script> 所有JS code都在script 环境中 </script>  - 可放在<head>中或在单独.js文件.
document.write("html代码");
document.writeln("只是在结尾后加回车符，不换行得<br>");
function functionName(){...}.
alert("弹窗");
document.getElementById("Id").value;
document.getElementByName("name").value;
运算符: +, -, ==, !=, ===, !==.
x = eval("123.35*67"); //用来求 num value, 无法转换则 NaN
parseInt("  123xyz") → 123, parseFloat(""), isNaN().
```

a-b 无法转换成 num时 value → NaN, type → number "NaN"也是number!

Note 1) JS中 variable 无强制类型. "+" 操作符优先字符拼接 (num + boolean, boolean + boolean 除外).
"-" 操作则等价于数值转换再运算 (不可转换字符串 NaN), "==", "!=" 先数值转换 (type conversion) 之后再比较, "===", "!==" 直接比较.
2) eval 需要 string 的参数，返回表达式的值，但是并不会直接把值打印出来，如果 eval() 内是 string，原封不动 ~~返回~~ 返回.
3) parseInt(), parseFloat() 返回最前面的 Int 或 Float, 忽略空格，忽略最前面 Int 或 Float 之后的字符，如果第一个非空格字符无法转换 → NaN. (可指明底数).
4) isNaN() 先进行数值转换，不可转换到数值则为 true.
5) 注意 x=null 和 undefined (var x;, 不赋值) 不同.
6) 注意使用括号 () 改变或合理设置 "+" "-" 运算顺序.

<Block3：JS and Servlets>

3.1. JavaScript (Ⅱ).

Code
form validation

```
< form action = "someURL"   onSubmit = "return validate()">.
// onclick = "validate()", onfocus, onblur, onchange <Body onload="start()">
//validate() 验证表单, return false 或 true.
element.submit()
element.focus()          } element=document.getElementByName()
element.innerHTML="内容".
value = document.getElementByName("Name").value
      = document.forms[0].name.value.  ↳ 其实可直接用id调用 id.value.
if(value == "" && value.charAt(i) ≤ 'A' && value.length<9).
```

Note 1) 使用 .innerHTML可以灵活修改 <p id=""> </p>中的内容,不像 document.write(). 仅在HTML加载完成后,调用该write()后的Html会被覆盖.

2) 在<form>中一定要"return validation()",如果只单独"validation()",即使 return false,也不会阻止form提交,而会直接提交上去.

Code
Sparse Array & Scope

```
//声明array的三种方式
list = ["1", "2"];
array = new Array(length);
array[i] = "1";
Arr = new Array("1", "2");
len = Arr.length ;//typeof(Arr) -> Object.
var y = .... ;
```

Note 1) JS的数组是sparse数组,初始化时长度为0,如果只在~~第5个位~~ arr[5]填一个元素, arr.length=6, 数组自动扩展长度,前5个为空.

2) 在函数内部使用var,变量在函数内局部,其余所有情况皆全局.

Code
Cookie

```
document.cookie = "version" + 4 + "; expires = " expireData.toGMTString();
```

Note 1) Store information between browser sessions.

2) document.cookie以Name-value pair的方式存储cookie.

3.2. Servlet (I).

What Servlets

1) program run on web server 2) used for client request
3) generate dynamic web pages.

Server ↦ JVM ↦ apache (CGI) ↦ Tomcat ↦ servlet.

Tomcat 中有很多 web app，servlet 在每个 web-app 中都有

Code Servlet 基本结构

```
import jakarata.servlet.*;
import jakarta.servlet.http.*;
import java.io.*;
public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws
                      IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String Username = request.getParameter("fname");
        out.println(username);
        out.close();
    }
}
```

可换为 doPost，一模一样.

因为定义了 ContentType("text/html")，println 返回的是 HTML 代码. 其实应该写：

```
out.println("<html><body>"+username+"</body></html>");
```

Note
1) request 包含了请求相关的所有信息，request.getParameter(String name)
(Server 通过 request 阅读 client 的请求). request.getMethod().
2) response 包含生成并发送响应的所有信息.
response.setContentType(String type) ; response.getWriter();
("text/html")
3) request.getparameter() 得到的 return 都是 String 类型，JAVA 是强类型语言，注意使用 Integer.parseInt() 转换.
4) no main(), no constructor!
5) servlet 优点：Efficient, Convenient, Powerful.

generic servlet

Servlet ◁— GenericServlet ◁— HttpServlet ◁— MyServlet

生命周期：constructor() → init(ServletConfig config) → Service(req, res) → destroy()

Note 1) 存在 main 并且 working all the time.

2) 每个 web-app 会有一个 Servlet Object (第一次 request 时 constructor & init()).
每个 request 都会 create 一个 ~~thread~~ 并使用该 Servlet.

3) Service() 只是 servlet 的工作阶段, 会根据不同的 ~~ep~~ request method 调用不同的方法 response (ex. doPost(), doGet()).

(4) 对于 extends ~~HttpSer~~ HttpServlet 的我们来说, 只要 override doGet() 和 doPost() method 即可, 且必须 rewrite, ~~因为~~ 虽然 HttpServlet 不和 GenericServlet 一样是 abstract class, 但它的 doPost(), doGet 不中用.

Code
web.xml

Deployment Descripter (DD)

// 假设 Client 表单 <form action = "/servlet/MyTest.do" >
// Serlet的 class 路径为 WEB-INF/classes/a/Echo.class ——— 都在 Tomcat 目录内.

webapps
  app1 ┬ WEB-INF ┬ classes .class
       │   ├ src
       │   └ web.xml
       └ .html或.jsp
还有其它 app

src ——— 不必需, 只需将产生 .class的路径 放在 classes 中即可.

.xml

```
<web-app>
    <Servlet>  ……小写
        <Servlet-name> Random </servlet-name>
        <Servlet-class> a.Echo </servlet-class>    …… 用"."分割路径
        <init-param>
            <param-name> exparam </param-name>
            <param-value> *** </param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name> Random </servlet-name>
        <url-pattern> /myTest.do </url-pattern>    …… 用"/"分割路径
    </servlet-mapping>
    <context-param>
        <param-name> EXparam </param-name>
        <param-value> *** </param-value>
    </context-param>
</web-app>
```

String (name/value pair)

被 ServletConfig 包含, 由 Tomcat 创建.

被 ServletContext 包含.

JAVA

```
out.println( getServletConfig().getInitParameter("exparam"));
out.println( getServletContext().getInitParameter("EXparam"));
```

Note 1) creat Servlet 的过程:
Tomcat 读 DD ⟶ creat ServletConfig (以及 init-param) ⟶ creat 并 init Servlet(config)

2) Tomcat 中的 web-app 中可能有很多 servlet, 每个 servlet 只被 create 一次, 每个 ServletConfig 对应一个 Servlet, 每个 ServletContext 对应一个 web-app, 即对于一个 web 的所有 Servlet 都是可访问的 (全局的).

3) Initparam 都是 Name-value (Both String) pair, 但是对于 ServletContext 来说, 它还拥有 Attributes, 可以是 (String - Object) 对.

访问: getAttribute(String) 设置: setAttribute(String, Object).

4) 因为多个 request 创建的 thread 可能共享一个 Servlet, 因此会有 Thread safe 的问题: ① Never use instance variables in servlet ② use Synchronization.

code 多值提取

```
Enumeration paramNames = req.getParameterName();
while (paramNames.hasMoreElements()) {
    String paramName = (String) paramNames.nextElement();
    String[] paramValues = req.getParameterValue(paramName);
  // 随后根据 paramValues.length 判断单值还是多值再操作.
```

<Block 4: Servlet, JSP, JavaBean, MVC>

4.1. Servlet (II)

code 重定向与转发.

```
~~response.sendRedirect~~
response.sendRedirect("new URL")
request.getRequestDispatcher("display.jsp").forward(request, response)
```

("new URL" 旁注: 不包括 Http, 例如该 URL 可能是 www..... 因为 Redirect 只可重定向到 Http GET Request.)

(response / request 旁注: 不同的调用对数据共享的影响.)

Note

1) 对于 Redirect, 不管原 Request 是 POST 还是 GET, 都会以 GET 形式重定向, 这是在浏览器 (Client) 端进行的跳转, 用户的 URL 直接变化, 两次访问的数据不共享.

2) 对于 forward (Request Dispatch) 是服务器端完成的跳转, 用户只知道一次请求, 不知道跳转, 数据是共享的

3) Redirect 可跨 app 跳转, forward 只在同一个 app 下进行 .jsp .html 等的跳转.

4) 使用 sendRedirect, 会向 Client 发送 "302" status code, 表示 server moved temporarily.

Code HTTP Header

```
// Request Header
req.getHeader(String)
req.getHeader("Content-Type") ⇔ req.getContentType()
Enumeration var = req.getHeaderNames()
String types = req.getHeader("Accept"); // 期望 response 的格式.
```

// Response Header

① Status line (version, status code, message)

response.setStatus(int); // 设置status code, 基本自动补, 最常用.

response.sendError(404, "Not found"); // 404 (基本自动生成) + message.

response.sendRedirect(String url); // 302

② Other

res.setHeader(String header, String value)

res.setContentType("text/html") ⇔

res.setHeader("Content-Type", "text/html");

res.setHeader("Refresh", "30"); // 每30s刷新网页

Note 1) 常见控制的 Header { Req: Content-Type, Accept.
Res: Content-Type, Refresh.

2) setStatus() 设置的 status code 设置代码必须在所有向 PrintWriter out response 回的代码之前设置, 下面是常用 status code:

① 200-OK = The request send by client was successful

② 302- Moved temporarily ③ 404- Not found.

④ 303- see other: 表示的在其他URL处, 应该使用 GET 去 new URL

⑤ 400-Bad Request = Not understood by Server.

3) 关于 Content-Type, 记住 text/html, text/plain 之不做任何格式转换的返回

Goole Session

```
import jakarta.servlet.http.*; // 引入让我们去操作的处理 cookie 和 URL rewritten.
                                                    session tracking
HttpSession session = request.getSession();
session.isNew();
session.setAttribute("userage", age);   → 等号之前的从 req.getParameter() 得到
String age = (String) session.getAttribute("userage");
```

Note 1) SessionTracker (属于Tomcat) uses a sessionID (JSESSIONID) to match users up with Session objects on the server side. attach a session id to the end of URL.
优先使用 cookie 设用户记住 sessionID, 如果不支持 cookie, URL rewriting

2) session = request.getSession(), HttpSession 对象在 Server 中有的是一个 hash tables, 包含 value 与 Attribute 的信息, 及 session id 当在 cookie 的键值.

3) sessionID 对应为得到 session Object, 从而找内部信息.

4) ~~Session~~ memory is held on the Session object, the objects are held by Tomcat. 所以 Session 可以在同一个 Tomcat 的不同 Servlet 中使用.

Code

URL rewritting & Encoding

1. when URLs are embedded in a generated web page.

```
String orignalURL = 原始URL;
String encodeURL = res.encodeURL(orignalURL);
out.println("<A Href=\" " + encodeURL + " \">...</A>");
```

} 前后查询的替换 背后有复杂的规则.

2. when placed in location header.

```
String encodeURL = res.encodeRedirectURL(orignalURL);
res.sendRedirect(encodeURL);
```

Note 1) 1中的encodeURL有两个用处:

① 如果browser支持cookies, then URL is not rewritting

② 如果URL需要Rewriting, session ID 被 inserted 进 URL并返回 rewritting 后的URL

2) 两种encode. encode(URL), encodeRedirectURL() 分别对应了两个用途, URL-rewritting for insert session ID 和 Redirect.

3) 如果Browser不支持cookie且user点击了un-rewritting URL, session lost.

4.2. Java Server Pages (JSPs) & JAVA Bean (I).

Motivation JSPs allow static HTML to be mixed with dynamically generated content from servlets. (在HTML中直接写JAVA).

与Servlet对比 Servlet 是 HTML 用 out.println 嵌入JAVA | JSP: 是以某种格式将JAVA嵌入HTML.

优势: 1) 写HTML比Servlet方便, 不需要写一堆 out.println.

2) 相比于html, 可以 dynamically generated content.

3) 相比于JS, 可以在 Server-side 运行, 且继承了JAVA的Powerful.

JSP生命周期 JSP本身是Servlet! JSP Page →(translate) Generated Servlet →(Compiled / loaded) Compiled Servlet

对于从 web Brower 收到"JSP"请求是翻译为Servlet后再编译出来的, 用户所谓"JSP", Server是用Compiled Servlet对request做答复的.

再解释 (JspInit() ⟶ _JspService() ⟶ JSPDestroy().

rep ↑ ↓ res 其本质是Servlet.

Server返回的是JSP编译后对应的html页面!

Servlet

code
JSP Syntax

1. Scripting element.
   ① expression <%= expression %> //value 被转化为 ~~vector~~ string 插入.
   ② scriptlets <% code %> //多行的 JAVA code
   ③ declarations <%! private int acc = 0 %> //声明.
2. directives (指令)
   <%@ directive attribute = "value" %> //另有 isErrorPage, errorPage
   (directive → page, include)
   ex. <%@ page contentType = "text/plain" %> //JSP默认 "text/html"
   <%@ page session = "true" %> <%@ page import = "....."%>
3. Action <jsp:action attribute="value"/> useBean, setProperty, getProperty

Note
1) HTML注释 <!-- --> 会被 source 看到. JSP注释 <%-- --%> 不会被 source 看到.
2) Script elements 中的 Scriptlets code 直接被插入到 _JSPService() 中, 可以使用 pre-define 的 variables. declarations code 被插入到 _jspService() 之外. 不可使用预定义变量. 是 Servlet的 Instance variable, 需要使用 Synchronize 去访问.
3) 一些 pre-define variables: request, response, out, session, config, application. 其中 out 是 JSPWriter类而不是 PrintWriter类, application 就是 ServletContext, 这些变量只可在 _jspService() 中使用
4) directives 中有 page, include 等 type。page 用来 import, 设置 content-type 等 include 用来 Insert a file into JSP at translation time

4.3. JSP & JavaBeans (五).

JavaBean JavaBean 只是满足一些要求下的 JAVA class:
① 必须在一个 package 里 ② 实现 (implement) Serializable 接口
③ 必须含有一个 empty constructor ④ 所有实例变量都不是 public (一般用 private).
⑤ 一般, 所有 public 方法都是 setXxx(), getXxx() 或 isXxx(), setXxx() (专门针对于 Boolean 类型).

Code
JavaBean

```
<jsp:useBean id="Beanname" class="packagename.classname"/>  //声明
set { <jsp:setProperty name="Beanname" property="Someproperty" param="value"/>   (param ⇔ value)
    { <jsp:setProperty name="Beanname" property="Sumproperty"/>
    { <jsp:setProperty name="Beanname" property="*"/>
get { <jsp:getProperty name="Beanname" property="Propertyname"/>
```

Note 1) jsp:useBean 封装 JSP action 声明一个 JavaBean 类，但并不完全，应为：

<jsp:useBean id="beanname" class="..." scope="page" />

在该说明该 Bean 的存在 scope：

① Application, stored in ServletContext.

② Session, stored in HttpSession.

③ request, stored in HttpServletRequest.

④ page (default), stored in PageContext.

2) setProperty 中使用了两个表示取值的 value 的 attribute. value 与 param.

param 多常用，因为它为以帮助 type conversion，且直接 param="Paramname" 即为 value 后面 value='<% request.getParameter("Paramname")%>.

注：当 Property 的 name 和 表单中的 name 相同时，使用 <jsp:setproperty name="..." property="..."/> 即为，不需要再 param="..."，因为此时为自动获取.

3) <jsp:getProperty name="beanname" property="height"/>

⇔ <%= beanname.getheight() %>.

例 有时用 <%= beanname.getXxx() %>，<% beanname.setXxx() %> 也比较直观.

Code Handling errors

ex.jsp 中：<%@ page errorPage="errorPage.jsp" %>

errorPage.jsp 中：<%@ page isErrorPage="true" %>

Note ex.jsp 中发生 error (JSP, servlet 或 beans 的 error)，跳转到 errorPage.jsp.

Code include file

1. 静态包含 (include directive)

<%@ include file="相对 URL" %>

2. 动态包含 (action = jsp:include)

<jsp:include page="相对 URL" flush="true" />

Note 静态包含在 translation time 时加载，不会自动刷新 (如果 file change 了，JSP 必须 update)

动态包含在 request time 时加载，modified without retranslating JSP page.

即为以自动更新.

转发：<jsp:forward page="Relative URL" />

⇔ request.getRequestDispatcher("Relative URL").forward(request, response);

RequestDispatcher: Defines an object that receives request from clients and sends them to any resource on the server.

必须在任何对 Client 的响应之前调用 Redirect 或 forward，否则 Exception.

总结： JSP directives： <%@ page/include ... %>

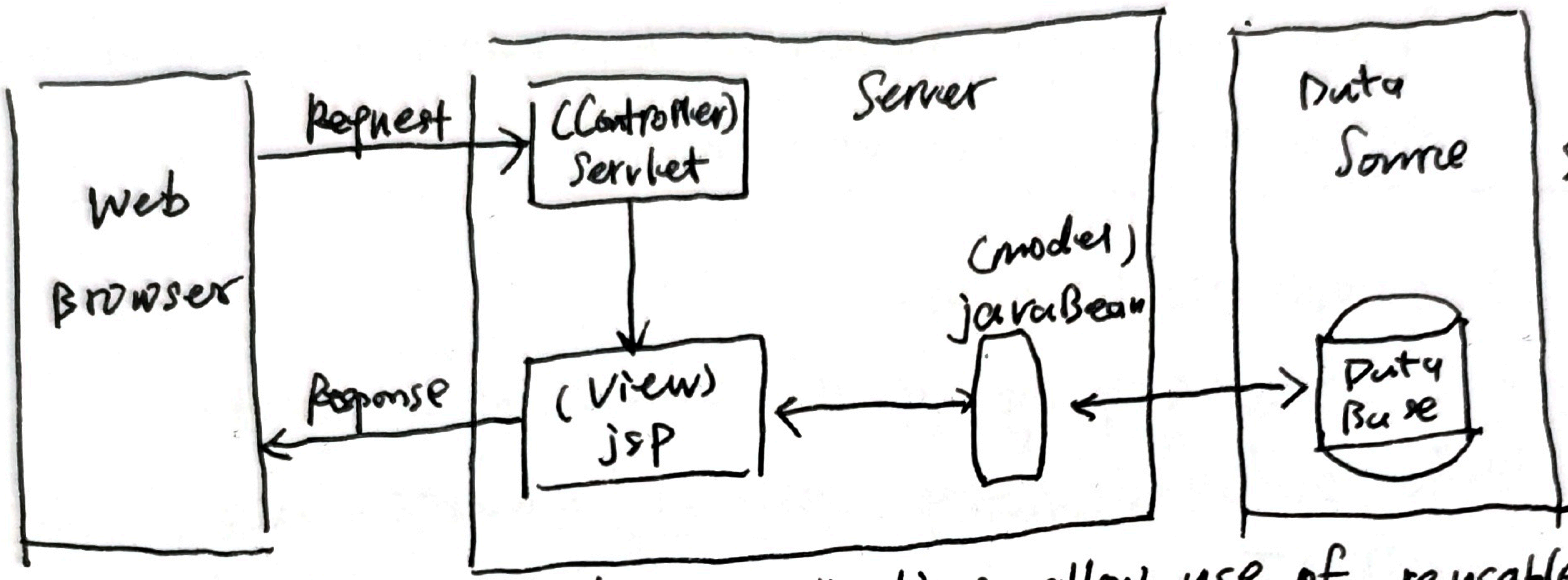
isErrorpage, errorpage, contentType, session, import, include: file

JSP action = <jsp:action ..../>

useBean, setProperty, getProperty, include (page), forward (page).

4.4. Model View Controller (MVC) 模型视图控制器.

MVC approach



Scalable!

Note 1) Advantages: Suitable for large applications, allow use of reusable software components, easy to maintain and test.

2) Servlet (Controller) 处理 Request, setProperty to JavaBean, 并 forward 到 .jsp 进行显示, jsp 不进行任何数据处理, 只用 jsp: getProperty

Model: Represent the logic behaviour of data, app's business logic.

View: Transforms the state of the model into readable HTML.

Controller: Accepts inputs data, instructs the view and model to perform action.

No.

Date

网络编程

B1.1. Introduction to Thread

a. Thread的两种实现方法 { implement Runable / extend Thread

- implement Runable → 主要是run()方法，执行 .start()
- extend Thread → 可以利用 extends，JAVA不支持 muti extends.

b. Static方法 sleep() → milliseconds.

Static方法 yield() 当前运行线程自动停止，进入ready状态（非block）.

非静态方法 join()，等待这个线程执行完，再执行.

wait(), Sleep(), join() 都 throws InterruptedExcepection.

Sleep(), join(), wait() 可使 Thread 进入 Block.

B.1.2、Interrupting & Terminating

a. static方法 interrupted()，查询中断flag (boolean) 并清除中断状态，即flag置为false，因为static，注意使用Thread.currentThread.interrupted().

非static方法 isInterrupted 查询对应Thread的中断flag状态.

非static方法 interrupt()，设置对应Thread的中断 flag 为 true，如果对应正在 sleep(), waiting（即Block状态），立刻抛出 InterruptedExcepection异常并 set the flag to false

（旁注：处理异常或 interrupted()都可 flag→false.）

（interrupt()只是设置flag而已，它什么都没做，不会关线程）

b. isAlive()可检查Thread是否存活，返回boolean，Thread终结的3种方式：

① 运行完毕，自然死亡

② 其它线程A，创造了一个线程B，在用start线程B之前，使用代码：B.setDaemon(true); 然后start()，把B变成了A的守护线程 (daemon thread) A终止，daemon thread forcefully terminated.

③ 使用interrupt()方法使 Block线程抛出 InterruptedExcepection，在 catch Block中直接给线程终止；或用 interrupt()方法设置标志位 flag 为 true，然后被设置线程检查自己的 flag，检查到 true的时候自动关闭（周期检查，routinely check）.

B.1.3. keep code safe.

a. volatile 关键字（→只有定义基本类型） 1) help java keep variable safe 2) 同一时间，只有一个线程可以访问该变量，但它并不 atomic. 3) 保证数据的可见性. ~~4) 保障线程安全~~.（每个 thread 都可以看到 volatile 数据变化）.

b. Thread safe：多个 Thread 可以同时安全执行的方式共享数据结构.

code is → Critical section（临界区）：程序从独立的单个 Thread 中访问相同的数据的临界区.

Critical code：只能同时被一个 thread 执行的代码.

Synchronized ↦ Intrinsic locks.（只能对 Object 使用）.

访问器无须 Synchronized，多个 Thread 想同时 Access 一个 Synchronized block，一个拿到了，其它会被 block. Synchronized 方法不继承.

volatile 变量永远不会导致 block

B.1.4. Monitor.

a. A lock that supports Cooperation through the wait() & notify() methods is called a Monitor.

Entry set — The Owner — Wait Set

- wait() 方法，释放锁进入 wait Set. InterruptedException.
- notify() 方法，不放锁，随机叫醒一个 waiting Thread 进入下一轮竞争.
- notifyAll() 方法，叫醒所有 waiting set 的 Thread.

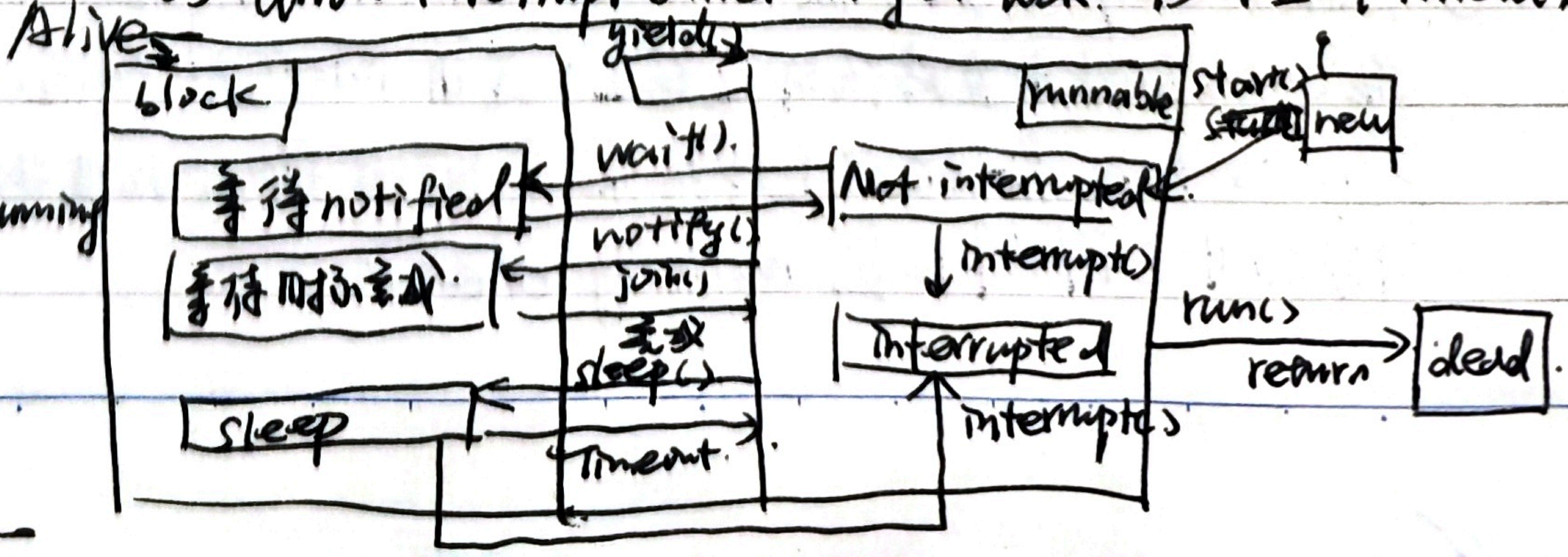
→ 必须在 Synchronized 环境中，都是 Object 的方法.

b. 多个线程等待多个永不会被释放的锁（至少两个）→ 死锁. Deadlock.

Solution： 1) Check periodically 2) avoid mutual exclusion 3) allow interrupt other to get lock. 4) 不让一个 Thread 有一堆 lock.

状态：

runnable 中有 ready and running 状态.



No.
Date

B.2.1. Socket. import java.net.*;

a. A Socket is an abstraction of network interface & presents a stream to ours. |有stream的地方都有throws IOException.

Static InetAddress.getByName() 静态方法(参数为域名字符串(DNS字符串)) www....

InetAddress addr = ~~InetAddr~~ → InetAddress.getByName { (null) / ("localhost") / ("127.0.0.1") } 也可用其它DNS获取IP.

获取本地IP

b. Socket: The Server need a way to distinguish between clients and keeping their communication separate.

ServerSocket类: 只有port(port). 与accept()方法联合,帮助Server获得Socket. → 返回socket

Socket类: address和port参数. (add, port).

PrintWriter → BufferedWriter → OutputStreamWriter → socket.getOutputStream, true (自动刷新)

BufferedReader → InputStreamReader → socket.getInputStream.

└→ 搭配 out.println()/print() └→ 搭配 in.readLine()

切记关闭ServerSocket和Socket, 流会自动关闭.

B.2.2. HTTP.

a. Hyper Text Transfer Protocol.

使用于Server传webpage给Client的应用层协议. 以Byte为头部.

Based on TCP (Transmission Control Protocol). Reliable, Connection-Oriented.

建立一个Socket就是在建立一条TCP连接, TCP可以output Byte.

b. URL. Uniform ~~Reso~~ Resource Locator

A way of locating a reasource on the Internet.

protocol:// hostname[:port] /path/filename#section.

无port时, URL自动选择合适protocol.

c. Http details. 仍属于不保存连接, stateless.

建立HTTP连接 → 用户通过URL向server发请求 → server处理请求, 发给Client, 并告诉Client关闭连接 → Client收到请求关闭连接.

→ Client解析请求 → Client向server请求referenced objects

KOKUYO

No.

Date

Request Method.
1) GET. Query string incorporated in the request URL.
Idempotent：多个同样请求有相同effect.
~~可~~Cachable，且只读，只会获取信息，传输信息量受限.
2) POST. Query string placed in the body of HTTP request.
Non-idempotent. 每次request内容数据变化.
在改变server-side中的信息使用，更安全.

报文. 1) request.

Method	path	version	request line

head field name : value
⋮
— header lines (key = value).

Entity Body — Body（Post用，Get不用）.

ex. GET /path/... HTTP/1.1

2) Response.

version	status code	phrase	status line.

head field name : value
⋮
— header lines

Entity body — Body（通常是HTML）.

ex. HTTP/1.1 200 OK

a) pipelining; b) parallel TCP connection；（加快数据传输方法）还可有 c) persistent connection

• HTTP是一个MIME protocol（媒体类型）MIME在Content-Type和Accept中使用.
例如：Accept：text/html. Content-Type：text/html.

Some Headers：Content-Length，~~Cha~~ Cache-Control，Expires（过期时间）
Set-Cookie（Server设置，Client维护并每次发送给Server）.

d. Other.
<https>. HTTP/1.0, /1.1, /2.0 都运行在TCP之上. 所有流量（payload和header / ALL!）加密.
HTTP Secure运行在TLS上（Transport Layer Security）.
URLs: ~~http~~ https://hostname[:port]/path/filename#section.
http port ⇒ 80, https → 默认为443（TLS）
Issues: extra overhead, ~~in~~ increase setup time, increase page load time
<HTTP/2>. ~~focus on Google's SPDY~~.
接替谷歌内部SPDY作为 focus on reducing page load times.
多路复用，一致加密，优先级，Server push/hint.

§2.3 HTML. Markup language.

使用 ~~tag~~ tags <> </> 来定义 elements；大小写不敏感.

<html> <head> <title> <body>

<p> <h1>(标题)
(换行) <hr>(水平线) | 可给tag加属性 ex. <h1 align="center">.

<b>(加粗) <em>(强调) <i>(斜体) <small>(小字) | 转义字符: &lt (<), &gt (>), &amp (&), &quot (")

link: <a href="path.html"> 跟着当前 URL找 path.html

<a href="http://www.path.ac.uk"> 在全网找.

可添加 target="_blank" 在新窗口打开.

<table border="1"> (有边框) <tr> (行) <td> (列). | <tr> <td> 1 </td> <td> 2 </td> </tr> [1 | 2].

表头用 <th> 与 <td> 地位一致, 不过文字(字)加粗居中.

List: 无序 ... | 有序1,2,3 <ol> .

<Forms & Input>:

forms is an area that can contain form elements to allow enter information

<input type="text" name="fname"> [ ] 文本框.

type = password [****]

- <input type="radio" name="same name" value="1">1 ○1 ○2 ○3.
  单选框, 同name只可选一个.
- <input type="checkbox" ....> 多选框. ☐1 ☐2 随便选.
- <label> 可以与一个<input>关联. <label for="id">, label的for必须与关联input的id相同, 这样, 点击<label>等于点击<input>.
- <form name="n" action="html_action.jsp" method="get">.
  <input type="submit" value="submit"> 提交的 button
  action attribute 定义提交到的 file., method定义用什么method提交.
- <select name="select"> <option value="1">1</option> </select>.
  在<form>里不一定需要, 为select选择值 [▽] 下拉表单.
- <textarea> 文本输入框 <input type="button"> 按钮.
  与"submit"不同, 点"button"不会提交.

No.

Date

· id 只在前端中多使用，真正提交的是 name = value.

<other>.

a. HTML5.

b. CSS. Separates the layout from the document structure.

c. Scalable Vector Format (SVG) 可伸缩矢量图.

Define the graphics in XML format. 放大后，缩放时质量不下降.

B.2.4. JavaScript (I).

— A scripting language most often used for client-side web development.

用途：1. interactive user interface. 2. 动态操纵 web.

3. 动态生成 html. 4. Form validation.

重要 method. document.getElementById("id").value.

document.write(" "); ← 注意引号!

可以在 <FORM> 中写属性 onSubmit = "return validate()"，提交时调用函数.

变量类型 使用 "+" 对变量运算，除了 boolean 外，优先字符拼接.
使用 "-" 优先数值运算，无法转换则为 NaN.（注：运算具有从左向右的优先级）

null 与 undefined 不一致，undefined 是未定义时的变量 value.

注：使用 writeln() 比 write() 多了一个换行符，但在 html 中换行符不管用，还是得用
，如果不加
，只会多一个 (\n) 字符.

比较运算符 == != 比较前先数值转换.

=== !== 严格比较，不进行数值转换.

一些 method. eval("") 执行 "" 内的代码.

parseInt("") 从开头开始extract，然后向后查找，把最前面的 Int 提出来，第一个字符不是数字，NaN.

isNaN() 先数值转换，再判断是不是 NaN.

注 form 提交的 value 类型与 type 相关，而不是数据的类型，在 script 处理时使用 document.getElementById("id").value 获得后可能需要先数据类型转换再处理.

No.
Date

B.3.1 JavaScript (II).

a、表单验证. Event-Driven Programming: On event 处理 event 事件

点击"button"⟺ onclick = "clickfunction()"　修改input组件内容使用 value =

注: 可以用 document.forms[0].submit() 来提交, button 也有 submit 功能

注: 使用 getElementById("id").innerHTML = Date() 可以实现改变内容　一般修改<p> innerText 也可以

点击"submit"⟺ onSubmit = "return validate()" 一定记得 return (true 才提交)

注: 使用 getElementById 后, 也可以直接修改 value, 使用 document.forms[0].name of element. ···· 也同样操作. focus 可以在指向输入栏

b.对象、数组. 1) 产生数组方式 1 list = ["", ""]　2 list = new Array(length)

list[0] = ""; list[1] = "";　3 list = new Array("1", "2")

employee.length 数组 length 所记录是 sparse 的, 第 6 个插入数, 前 5 个为空, 总长度为 6.

typeof(数组) = Object. 且数组和对象的内容命名声明可混用

例. A = New Object(); A["red"] = 127 ⟺ A.red = 127, 也可使用 new array() 来替.

2) Scope. 函数外声明变量, 全为全局可见.

函数内声明变量, 加 var 局部, 不加 var 全局.

c. Cookies. Store information between browser sessions, set expiry date.

document.cookie = "version = v4; expires = " + expireDate ";　↳ .toGMTString()

Cookie 存储 name = value 对. 注意 存储的 value 都是 string.

使用 value = parseInt(cookieVal("name of pair")) 可以提取 value

name = value

Cookie name 不可有奇怪符号, 可以使用 encodeURIComponent(value) 来存储 value, 提取时 decodeURIComponent().

补充: <Body onload = "doSomething()"> 页面加载完调用 dosomething

声明函数, 在 <script> 中: function fname(参数) {Body};　↳ Html, JS 中参数无类型!

onblur 从某处离开处理事件
onfocus 点击某处处理事件
} 都是 <Input> 的属性.

onload, onblur, onfocus, onClick, onSubmit.

KOKUYO

No.

Date . .

B3.2 Servlet. import jakarta.servlet.*; jakarta.servlet.http.*; java.io.*;

a. Intr Servlets are Java's answer to CGI.

1) run on a web server 2) use for client requests 3) generate dynamic web page

Server → JVM → CGI → Tomcat (容器) → Servlets.

重写method doGet(HttpServletRequest request, HttpServletResponse response)

doPost 一样 throws IOException, ServletException { }.

[illegible]code response.setContentType("text/html")

PrintWriter out = response.getWriter();

渲染只返回 string. → out.println("<html> 代码").

String username=request.getParameter("name"). (<input>的name属性)

out.close();

注：如果 extends HttpServlet, 无须写构造器和main(), 只需要覆盖doGet() doPost().

b. 生命周期 Constructor → init(config) → service() → destroy()

循环处理，会自动选择合适的处理方法.

接口 Servlet ~~service~~ ← 类 GenericServlet ← 类 HttpServlet ← 类自己的Servlet

只需重写doPost, doGet.

init()之后 Servlet才可以发挥作用.

每个 request 都会建立一个线程，多个线程（访问同一个server web-app）会共享一个Servlet对象.

c. Deployment Descriptor 部署描述符 (DD). 在 web.xml 文件中.

<web-app>
<Servlet>
<servlet-name> 随便 A </servlet-name>
<Servlet-class> 包名.类名 </servlet-class>.
</Servlet>
<servlet-mapping>
<servlet-name> A </servlet-name>
<url-pattern> /path </url-pattern>
</servlet-mapping>
</web-app>.

只有一个Servlet实例被创建(一个app)，不同请求有不同Thread，访问这一个servlet，

ServletConfig 在 Servlet init() 之前被创建，Servlet 有对应的 Config.

在<servlet>中

```
<init-param>
    <param-name> name </param-name>
    <param-value> value </param-value>
</init-param>.
```

获取param： getServletConfig().getInitParameter("name").

注：防止 Thread Risk，在 serlet 中不使用 instance variable.

更general ⇒ ServletContext. Config是 one per servlet.(contain init param).

↳ One per web app.

```
<context-param>
    <param-name> ... <param-value>
</context-param>
```

value 都是 string

在<servlet>之外

<web-app>之内.

getServletConfig().getServletContext().getInitParameter("名称");

ServletContext 有 attributes，与 param不同，(param: string = string)，attributes是 string = Object.

访问 getAttribute(String) 设置 setAttribute(String, Object).

ex. 获取 param的Name.

```
    Enumeration paramNames = req.getParameterNames()
while (paramName.hasMoreElements()).
    String paramName = (String)paramNames.nextElement();
    String[] paramValue = req.getParameterValue(paramName).
```

paramValue的length就是其中存了几个value.