# 第1章 线程和多线程

## 1. 相关概念

程序 静态的概念

进程

线程

## 2. 线程生命周期：新建、就绪（可执行状态）、运行、死亡、阻塞

## 3. 创建线程2种方法

1. 继承Thread类
2. 实现Runnable接口

区别：1 继承Thread类不能再继承其他类，实现简单，不方便资源共享

2实现Runnable接口还可以继承其他类，便于功能扩展，便于资源共享

## 4. 启动线程：start()

## 其他的方法：

sleep(毫秒数)、

yield()线程让步、

join()、join(毫秒值)

属性：name 存储线程名称 getName() setName(“”)

priority 范围（1~10）getPriority() setPriority()

得到线程对象方法：this 仅限于用在Thread类中

Thread.currentThread() 通用的方法

## 6.线程同步synchronized

1) 给方法加锁 (权限修饰符之后 返回值之前)

2) 给当前对象加锁 synchronized(this){ }

3) 给类的所有对象加锁 synchronized(类.class){ }

4) 给属性加锁 synchronized(new 类().属性)

## 7. 死锁必要条件

互斥、请求和保持、不可剥夺、环路等待

## 8. 死锁出现的情况

相互排斥、部分分配、缺少优先级、循环等待

# 第2章 网络编程

## 1.常见的类

URL：构造方法 URL（String url）

获取网络输入流：InputStream openStream()

获取网络连接的：URLConnection openConnection()

关闭连接：close()

URLConnection(HTTPURLConnection)：

获取网络输入流：InputStream getInputStream()

获取网络输出流：OutputStream getOutputStream()

关闭连接：close()

Socket：用于客户端类，构造方法 Socket(String addr, int port)

获取网络输入流：InputStream getInputStream()

获取网络输出流：OutputStream getOutputStream()

关闭连接：close()

ServerSocket：用于服务端监听端口的类，ServerSocket(int port)

接收客户端连接的方法：Socket accept()

获取网络输入流：InputStream getInputStream()

获取网络输出流：OutputStream getOutputStream()

关闭连接：close()

DatagramSocket：

构造方法： DatagramSocket()、DatagramSocket(int port)

发送数据包：send(DatagramPacket p)

接收数据包：receive(DatagramPacket p)

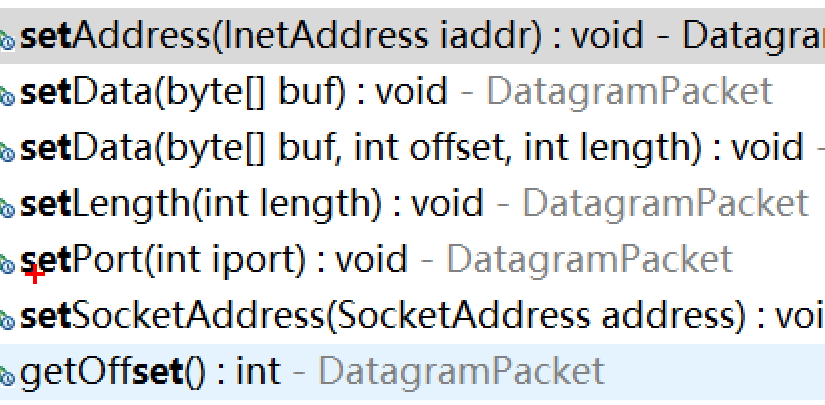
DatagramPacket：

构造方法：DatagramPacket(byte[] bytes, int length)

DatagramPacket(byte[] bytes, int length,InetAddress assr, int port)

InetAddress：获取对象方法：InetAddress. getByName(String host)

设置数据包参数方法：

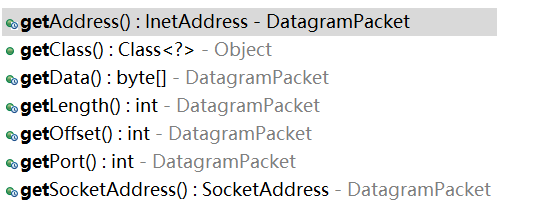


setData(byte[] bytes) 设置数据

setInetAddress(InetAddress addr) 设置连接的目标主机

setPort(int port) 设置端口号

获取数据包参数方法：



InetAddress getAddress() 获取发送方地址

Int getPort() 获取发送方的端口号

byte[] getData() 获取发送的数据

备注：

通过InetAddress对象获取主机IP地址的方法：

String getHostAddress()方法

## 2. Socket网络编程步骤（TCP）

客户端：

1. 建立连接
2. 获取流
3. 流的读写
4. 关闭连接

服务端：

1. 监听端口
2. 获取客户端连接
3. 获取流
4. 流的读写
5. 关闭连接

## 3. 常用的流

字节流：

InputStream、OutputStream：网络中获取到的流

FileInputStream、FileOutputSteam：操作本地文件的流

字符流：

BufferedReader（String readLine()）、

BufferedWriter（write(String str)、newLine()）

高级缓冲字符流使用注意事项：

1. 避免中文乱码，指定编码方式
2. 不会自动写入换行符

## 4. UDP网络编程步骤

1）构造DatagramSocket对象

2）构造DatagramPacket对象

3）发送/接收数据

4）关闭

# 第4-5章 Servlet模型

## 1. 站点结构

站点名称的文件夹：

WEB-INF：classes(字节码文件)、lib、web.xml

静态页面（html）或动态资源（JSP）

## 2. web.xml中对Servlet配置

1）Servlet别名

2）Servlet被访问的虚拟路径

<servlet>

<servlet-name>别名</servlet-name>

<servlet-class>包含完整包目录的Servlet类</servlet-class>

<servlet>

<servlet-mapping>

<servlet-name>别名</servlet-name>

<url-patten>虚拟路径</url-patten>

<servlet-mapping>

## 3. web.xml中welcome-list节点：配置服务器运行站点时默认加载的页面

## 4. 虚拟路径配置方法（3种）

1）完整路径：/hello 、/abc/hello

2）/目录/\*：/first/\*

3）\*.后缀名：\*.docc

## 5. 根目录：站点根目录（配置的站点文件夹）、源文件根目录（src目录）

## 6. 常用的类（接口）及方法

Servlet、GenericServlet、HttpServlet（公有的service、保护的service）

HttpServlet中公有service方法的实现：

ServletRequest、ServletResponse——>HttpServletRequest、HttServletResponse

调用do方法（doGet、doPost）

方法：

请求中的方法：String getParameter(String param)用于获取表单、超链接(href的值不可以有空格，如href=”hello?paramName=paramValue”)

获取输出流：PrintWriter response.getWriter()、getOutputStream()

Request的Attribute参数：request.setAttribute(String name, String value)，request.getAttribute(String name)

编码方式的设置：

setCharacterEncoding(String charset)

响应页面的MIME类型设置：

response.setContentType(“text/html”) “image/jpg”

response.setContentType(“text/html;charset=utf-8”)

## 7. Servlet实例化时间设置 < load-on-startup>

负数或不设置表示用到时加载；

非负数表示服务器启动后加载；

## 8. 页面跳转

转发（forward）：request.getRequestDispather(String url).forward(request, respond)

重定向（sendRedirect）:response.sendRedirect(String url)

两种方式的区别：

1. 速度：请求转发的速度快，请求转发对于用户来说是1次请求，请求重定向对于用户来说是2次请求
2. 地址栏：请求转发时地址栏没变化、请求重定向时地址栏改变
3. 参数共享：请求转发可以共享参数、请求重定向不可以

## 9. Servlet的生命周期

加载与实例化、初始化、服务（service方法的执行）、销毁

无参构造实例化、init(ServletConfig config)、service(ServletRequest，ServletResponse)、destroy()

在Servlet生命周期中服务器自动产生了哪些对象：

Servlet本身实例、request、response、config（ServletConfig）、context（ServletContext）

## 10. 2个特殊的接口，可以用来封装初始化数据

1）ServletConfig：用于封装Servlet的初始化数据

服务器创建的

获取ServletConfig对象：servlet.getServletConfig()

获取初始化数据：config.getInitParameter(String name)

2）ServletContext：用于封装全局的初始化数据

获取ServletContext对象：servlet.getServletContext()、config.getServletContext()

获取初始化数据：context.getInitParameter(String name)

设置/获取全局化参数：setAttribute(String name, String value)、getAttribute(String name)

## 11. 通常使用注解的方式来配置Servlet

别名、虚拟路径、加载时机、初始化参数

@WebServlet(

name=”别名”,

urlPatterns=”虚拟路径”,

loadOnStartup=0/-1, //≥0：服务器启动就加载 负数/不配置：需要的时候加载

initParams={@WebInitParam(name=”K”, value=”V”),

@WebInitParam(name=”KEY”, value=”VALUE)

}

)

## 12. 作用域

request Attribute

context Attribute

\*session 会话 Attribute

# 第6章 会话

## 1. 会话

一个用户与服务器端连续不断的请求/响应的一系列过程。

## 2. Cookie+Session

## 3. HTTP协议特点（了解）

请求/响应模式、无连接、无状态

## 4. Cookie用法

(以键值对形式进行存储，一个键值对表示一个Cookie)

创建：new Cookie(String name, String value)

存储：response.addCookie(Cookie c)

获取：Cookie[] request.getCookies()

遍历：for(Cookie c : cookies){

c.getName();//获取当前Cookie的名称（Key）

c.getValue();//获取当前Cookie的值(Value)

}

其他操作：设置Cookie存储有效期setMaxAge(int n),参数单位是秒

## 5. Cookie的优点缺点

优点：键值对存储，方便、简单

存储在客户端，不占用服务器的内存资源

缺点：大小受限

不适合存储敏感性数据，有风险

## 6. Session的原理：

借助于Cookie来实现的，用Cookie保存Session的ID（JSSESSIONID）

## 7. Session的用法（HttpSession）

获取Session的方法：request.getSession()

获取SessionID的方法：session.getId()

设置/获取Attribute参数：session.setAttribute(String key, String value); session.getAttribute(String key)

设置超时期的方法：session.setMaxInactiveInterval()

## 8. Session用途

购物车

# 第7章 过滤器

## 过滤器：一个用于拦截数据源和目的地之间消息的对象

## 特点：线程不安全

## 定义过滤器：实现过滤器接口（Filter）

## 生命周期的方法

init(FilterConfig config)

doFilter(ServletRequest request,

ServletResponse response,

FilterChain chain){

……用于拦截请求

chain.doFilter(request, response);//分水岭

……用于拦截响应

}

destroy()

## 过滤器的功能

拒绝请求：自己产生一个响应，返回给客户端

传递请求：将请求传递给过滤器链中的下一个过滤器，如果该过滤器是过滤器链中的最后一个过滤器，就将请求传递给目标资源。

篡改请求：直接将请求转发给其他资源

## 过滤器的配置

web.xml: <filter> <filter-mapping>

注解：@webFilter(

filterName

urlPatterns

initParams{ @webInitParam(),…}

)

## 过滤器的执行顺序

拦截请求的顺序：

web.xml中配置的过滤器的正序，注解配置的过滤器的类名称的自然顺序

拦截响应的顺序：拦截请求的顺序的逆序

## \*过滤器应用场合

门卫、前台、秘书

# 第8章 监听器

<http://naotu.baidu.com/file/8d6320077f445b5f6d7ee98cf8677a17?token=01ad21df6ec01e06>

密码为  caMx

## 监听器的概念（了解）

监听器是一个用于监听特定对象或特定对象的特定方法调用或属性改变的事件的对象。当对应的事件发生时，将自动触发事件中的相应方法的执行。

## 监听器的类型

1. 按照监听的作用域区分

ServletContext相关的监听器：

ServletContextListener、ServletContextAttributeListener

HttpSession相关的监听器：

HttpSessionListener、HttpSessionAttributeListener、HttpSessionBindingListener（唯一一个不用配置就可以使用的监听器）

ServletRequest相关的监听器：

ServletRequestListener、ServletRequestAttributeListener

1. 按照监听的对象类型的不同区分

监听作用域对象本身的监听器

监听作用域内属性变化的监听器

## 事件

作用域对象本身相关的事件：ServletContextEvent、HttpSessionEvent、ServletRequestEvent

属性事件：ServletContextAttributeEvent、HttpSessionBindingEvent、ServletRequestAttributeEvent

## 事件中的方法

作用域对象本身事件中的方法：contextInitialized()、contextDestroyed()

sessionInitialized()、sessionDestroyed()

requestInitialized()、requestDestroyed()

属性事件中的方法：attributeAdded()、attributeReplaced()、attributeRemoved()

# Servlet规范总结

<http://naotu.baidu.com/file/e656dac9c8c2ab9b29de5b3b28a6b62c?token=0c07b01996154a0c>

密码为 MeA4

# 第9、10章 反射、JDBC

## 1. 反射中用到的重要的类

Class：类类型

Field：属性

Constructor：构造方法

Method：普通方法

## 常用方法

getDeclaredXxx()

创建对象：newIntance()

触发方法：invoke()

## JDBC

基本步骤：

1. 加载驱动 Class.forname(“驱动字符串”)
2. 和数据库建立连接（获取数据库连接对象）

Connection DriverManager.getConnection(“连接字符串”, “用户名”, “密码”)

1. 获取操作数据库的Statement或PreparedStatement对象

Statement conn.createStatement()

PreparedStatement conn.preparedStatement(String sql)

1. 通过上一步获取的操作数据的对象来执行相应的sql语句

增、删、改：int executeUpdate()

查：ResutSet executeQuery()

boolean execute()

1. 关闭连接

conn.close()

## PreparedStatement与Statement的区别

1. 语法不同：PreparedStatement操作的sql语句允许带有占位符（？，注意占位符不需要加引号）
2. SQL语句预处理能力：PreparedStatement具备预处理能力
3. 性能上：PreparedStatement性能相对高一些
4. 安全性：PreparedStatement相对安全
5. \*批处理：两者都具备批处理操作能力，addBatch()、executeBatch()，其中PreparedStatement加入批处理操作时，不带有参数，而Statement加入批处理操作时，需要带有SQL语句的字符串参数。