# Cookie

## 概述

Cookie的原意是小甜点。Cookie由服务端创建并通过Servlet发送给客户端浏览器并存储在客户端（通过Cookie可以存储少量的信息）。当客户端访问服务器时会将Cookie发送回服务器。通过Cookie服务器可以有效地识别每一个客户端。

|  |
| --- |
| Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server. A cookie's value can uniquely identify a client, so cookies are commonly used for session management. |

## 使用Cookie

|  |
| --- |
| [创建Cookie]  //1、创建Cookie  Cookie c1 = **new** Cookie("c1",**new** Date().~~toLocaleString~~());  Cookie c2 = **new** Cookie("c2",**new** Date().~~toLocaleString~~());  Cookie c3 = **new** Cookie("c3",**new** Date().~~toLocaleString~~());  Cookie c4 = **new** Cookie("c4",**new** Date().~~toLocaleString~~());  Cookie c5 = **new** Cookie("c5",**new** Date().~~toLocaleString~~());    //2、设置Cookie的过期时间  //setMaxAge（）的参数如果为负值则表示Cookie不被存储（浏览器关闭则Cookie立即删除）  //超过过期时间的Cookie会被立即删除  //-1为默认值  //setMaxAge（）参数的单位为秒  c2.setMaxAge(-1);  //设置C3的过期时间为90秒  c3.setMaxAge(90);  //设置C4的过期时间为一周  c4.setMaxAge(60\*60\*24\*7);  //设置C5的过期时间和可访问路径  c5.setMaxAge(60\*60\*24\*7);    //3、设置Cookie的访问路径  //设置只有指定路径下的资源可以访问当前Cookie  //路径格式：/项目名/路径/资源名  c5.setPath("/03CookieServlet/car/rc2");    //4、将Cookie添加到Response（发送给客户端）  resp.addCookie(c1);  resp.addCookie(c2);  resp.addCookie(c3);  resp.addCookie(c4);  resp.addCookie(c5); |
| [读取Cookie]  //读取Cookie  Cookie[] cookies = req.getCookies();    //输出Cookie数组  **if**(cookies != **null**){  //遍历cookies数组集合  **for**(**int** i = 0;i<cookies.length;i++){  //得到当前cookies集合中的Cookie对象  Cookie c = cookies[i];    //输出Cookie的名字和值  System.*out*.println(c.getName() + " = " + c.getValue());  }  }  **else**{  System.*out*.println("当前系统中不存在任何Cookie");  } |

## 常用方法

* setMaxAge() ：设置Cookie的过期时间，默认值为-1。表示不存储Cookie，浏览器关闭后立即删除
* setPath() ：设置Cookie的可访问路径，只有指定路径下的Servlet才能对其进行读取

## Cookie的作用

* 在客户端存储少量信息
* 标识每一个客户端(解决了Http无状态的特点)

## 缺点

* Cookie可以被阻止
* Cookie存在一定的安全隐患

## 应用

购物车、十天面登录

Cookie一般用于安全性要求较低，且可以提高用户体验性的环境中

# Session

## 概述

Session通称被称为会话(从浏览器打开到关闭的一次访问)，Session在服务器端创建并存储在服务器端。Session通常用于存储当前用户的信息。一个Session通常对应于一个用户。服务器端可以通过Cookie或URL重写的方式去区分每一个用户。

## 创建Session

|  |
| --- |
| //1、创建Session  //如果Session不存在(读取Session关联信息失败，则创建一个Session)  //如果存在Session则返回一个和当前请求相关联的Session(JSessionId)  HttpSession session = req.getSession(); |

## 设置Session的值

|  |
| --- |
| //2、保存Session的值  //session的值是以键指对的形式存在的  session.setAttribute("user", "Mike");  //如果session的名称已经存在，在此使用会替换原有session的值  session.setAttribute("user", "Rose"); |

## 读取Session的值

|  |
| --- |
| //读取Session的值  String user = (String)session.getAttribute("user"); |

## 常见设置

|  |
| --- |
| //3、设置Session的过期时间，以秒为单位  //如果其值设置为0则表示永不过期  //Session的特点：如果在Session过期时间达到前用户有过新的请求时，过期时间将重新计算    //过期时间设置方式：代码方式、配置文件（Tomcat >> conf >> web.xml >> session-configure.单位：分钟）  session.setMaxInactiveInterval(20);    //4、使Session失效  //session使无法直接删除的，只能过期或失效  session.invalidate(); |

## 特点

* 安全性更高（服务器内存中）
* 读取效率高（读内存更快）

## 应用

记录当前用户的信息

# Session与Cookie【问】

## 关系

Session创建后，会将SessionId存储到Cookie中，用户每次访问时可以通过请求中提交的Cookiee读取到Session的信息。

## 相同点

* + 都可以在web项目中存储一定量的数据
  + 都用于解决Http协议无状态

## 不同点

* Cookie在服务器端创建，存储在客户端；Session在服务器端创建，存储在服务器端
* Cookie的安全性较低；Session的安全性较高
* Cookie的读取效率偏低；Session的读取效率较高
* Cookie一般存储少量信息，具有大小限制；Session如果不去指定一般没有大小限制

# ServletConfig

## 作用

通过ServletConfig可以读取Servlet所配置的初始化参数。

## 参数配置

|  |
| --- |
| <servlet>  <servlet-name>ConfigServlet</servlet-name>  <servlet-class>cn.hl.servlet.ConfigServlet</servlet-class>  <!--  Servlet的初始化参数  init-param所配置的参数只能被当前Servlet所使用  -->  <init-param>  <param-name>eMail</param-name>  <param-value>zhanzhang@163.com</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>ConfigServlet</servlet-name>  <url-pattern>/conf</url-pattern>  </servlet-mapping> |

同过init-param配置的参数只能被所属的Servlet所使用。

## 读取参数

|  |
| --- |
| //获取ServletConfig对象  //通过该对象可以获取到Servlet的初始化参数  ServletConfig config = **this**.getServletConfig();    //获取初始化参数  String str = config.getInitParameter("eMail");  System.*out*.println(str); |

# ServletContext

## 概述

同过ServletContext对象可以读取context-parm（可以被任何servlet所访问）所配置的参数。以及提供其他常用方法。如：getRealPath()

## 参数配置

|  |
| --- |
| <!--  上下文参数  上下文参数可以在任意位置被读取  -->  <context-param>  <param-name>ip</param-name>  <param-value>192.168.10.1</param-value>  </context-param> |

## 读取参数

|  |
| --- |
| //1、获取对象  //方式一(推荐使用。一般在Servlet中使用)  ServletContext c1 = **this**.getServletContext();  //方式二  ServletContext c2 = **this**.getServletConfig().getServletContext();  //方式三  ServletContext c3 = req.getSession().getServletContext();  //方式四  ServletContext c4 = req.getServletContext();    System.*out*.println(c1 == c2 && c1 == c3 && c1 == c4);    //2、读取配置参数  String ip = c1.getInitParameter("ip");  System.*out*.println("ip = " + ip); |

## 获取物理路径

|  |
| --- |
| //3、获取指定路径对应的物理路径  //文件上传、下载时经常使用  String path = c2.getRealPath("/"); |

## 获取资源的输入流

|  |
| --- |
| //4、获取指定文件的IO流  InputStream is = c3.getResourceAsStream("/index.jsp"); |