# 会话安全性

## 会话劫持和防御

每次用户通过 HTTP 连接访问网站或应用程序时，该服务都会在打开通信线路并提供访问权限之前对用户进行身份验证（例如，通过用户名和密码）。 但是，HTTP 连接本身是“无状态的”，这意味着用户执行的每个操作都是独立查看的。 因此，如果我们仅依赖 HTTP，用户将不得不为他们执行的每个操作或查看的页面重新验证自己。也就是说攻击者通过伪装成合法用户，获取信息，并以假冒身份采取行动。

在会话劫持中攻击Cookies和Session IDs是常见的攻击方式。

·***Cookies:***

攻击者通常通过窃取用户的会话 cookie（因此也称为 cookie 劫持）或诱使用户单击包含预测会话 ID 的恶意链接来获得此访问权限。则攻击者就可以伪装成正常用户，和获得未授权权限。

·***Cookies防御：***

1、给Cookie添加HttpOnly属性：通过设置HttpOnly属性，可以防止JavaScript脚本访问Cookie，从而降低XSS攻击的风险。

2、在Cookie中添加校验信息：为了检测Cookie是否被篡改，可以在Cookie中添加一个与用户外置环境相关的校验信息。

3、定期更换Session ID：为了防止Session劫持，可以定期更换Session ID。当用户登录成功后，服务器生成一个新的Session ID，并将其存储在Cookie中。同时，服务器端的Session数据也要更新对应的Session ID。这样，即使攻击者窃取了旧的Session ID，也无法继续使用。

4、使用安全的Cookie传输方式：在传输敏感数据时，应使用HTTPS协议来加密数据传输过程。这样可以防止数据在传输过程中被截获或篡改。

5、限制Cookie的生命周期：将Cookie的有效期限制在较短的时间内，可以降低被劫持的风险。

·***Session ID劫持：***

通过获取用户Session ID后，使用该Session ID登录目标账号的攻击方法，此时攻击者实际上是使用了目标账户的有效Session。会话劫持的第一步是取得一个合法的会话标识来伪装成合法用户，因此需要保证会话标识不被泄漏。

·***Session ID防御:***

1. 更改Session名称。PHP中Session的默认名称是PHPSESSID，此变量会保存在Cookie中，如果攻击者不分析站点，就不能猜到Session名称，阻挡部分攻击。
2. 关闭透明化Session ID。透明化Session ID指当浏览器中的Http请求没有使用Cookie来存放Session ID时，Session ID则使用URL来传递。
3. 设置HttpOnly。通过设置Cookie的HttpOnly为true，可以防止客户端脚本访问这个Cookie，从而有效的防止XSS攻击。
4. 关闭所有phpinfo类dump request信息的页面。
5. 验证HTTP头部信息。

## 跨站式脚本攻击(XSS)和防御

***(a)跨站脚本攻击***

XSS是一种常见的网络应用安全漏洞，它允许攻击者将恶意脚本注入到网页中，进而由受害者的浏览器执行。这些脚本可以窃取用户的会话凭证、篡改网页内容、重定向用户至恶意站点，甚至进行钓鱼攻击。XSS是一种通过浏览器点击后就会执行恶意脚本窃取用户的Cookies和Session ID，以此窃取用户的个人信息和植入木马病毒。

***(b)跨站式脚本防御***

1，输入验证与净化：对所有不受信任的输入进行严格检查，去除或转义特殊字符，如尖括号、双引号、斜线等。

2，输出编码：在向浏览器输出数据时，确保正确使用HTML实体编码、JavaScript字符串编码或CSS编码。

3，HTTPOnly Cookie：设置session cookie为HTTPOnly属性，防止通过JavaScript访问。

4，Content Security Policy (CSP)：实施CSP策略，限制网页加载的资源来源，避免引入恶意脚本。

5，前端安全库：利用诸如DOMPurify这样的库对DOM操作进行安全过滤。

## （3）跨站请求伪造(CSRF)和防御

***（a）跨站请求伪造(CSRF)***

CSRF，是跨站请求伪造（Cross Site Request Forgery）的缩写，是一种劫持受信任用户向服务器发送非预期请求的攻击方式。通常情况下，CSRF 攻击是攻击者借助受害者的 Cookie 骗取服务器的信任，在受害者毫不知情的情况下以**受害者名义伪造请求**发送给受攻击服务器，从而在并未授权的情况下执行在权限保护之下的操作。

***（b）******跨站请求伪造(CSRF)防御***

1，Referer头部防御：服务器检查HTTP请求的Referer头部，确保请求来源于信任的域名。

2，验证码：要求用户在执行敏感操作前输入验证码，以证明是真人操作。

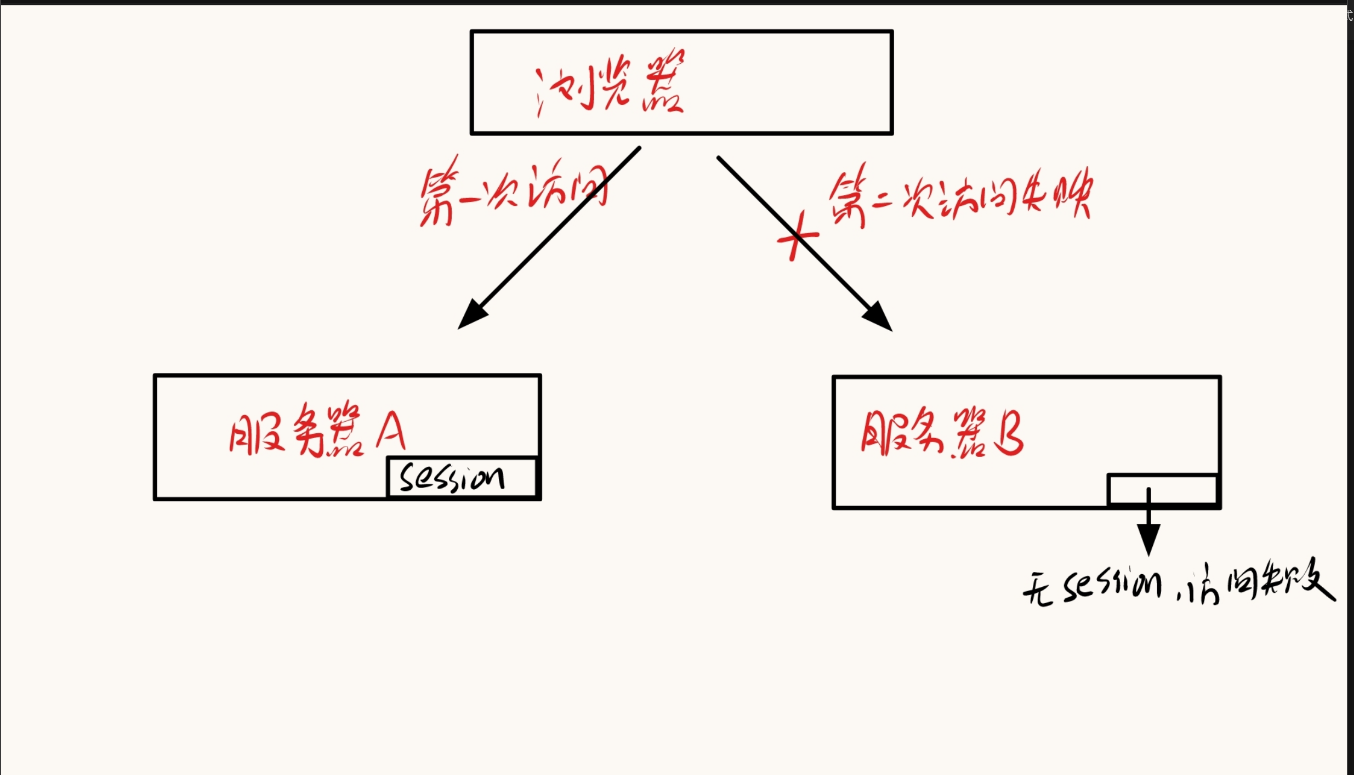
3，Token：在表单或AJAX请求中包含一个服务器生成的Token，服务器端验证Token的有效性。

4，自定义请求头：要求所有敏感请求必须包含一个特定的自定义HTTP头部。

# 2、分布式会话管理

## （1）分布式环境下的会话同步问题

由于HTTP协议是无状态的，在开发中我们可以将用户的信息存储在服务器的session中，并生成与之相对应的JSESSIONID通过cookie返回给浏览器。浏览器下次访问，cookie会自动携带上次请求存储的数据(JSESSIONID)到服务器中，服务器根据JSESSIONID找到对应的session，从而获取用户的信息。这种机制如果是在单一的应用中使用是没有问题的，但是如果使用的是分布式环境，当用户使用同一系统不同的部分时（如不同的服务器，不同的服务功能），只有一个服务器拥有用户的session，而其他服务器没有用户session，就会导致用户仅有一个部分的访问权限，从而造成会话不同步，造成用户体验差和数据混乱等问题。



## （2）Session集群解决方案

解决方案有**Nginx的IP\_Hash策略**，**Session复制**，Session共享和Session集中存储。

***(a)Nginx的IP\_Hash策略***：***同一个客户端每次的请求都会被同一个服务器处理。***

优点：

·只需要修改nginx配置，不需要修改应用程序代码

·可以支持web-server水平扩展

缺点：

·session还是存在web-server中的，所以web-server重启可能导致部分session丢失，影响业务，如部分用户需要重新登录

·如果web-server水平扩展，rehash后session重新分布，也会有一部分用户路由不到正确session。

***(b)Session复制：服务器将自己的session数据传送给其他服务器，使得每个服务器都拥有全量的数据。***

优点：

tomcat原生支持，只需要修改配置文件即可。、

缺点：

·session同步需要数据传输，会占用大量带宽，降低服务器集群的业务处理能力

·任意一台web-server保存的都是所有web-server的session总和，浪费了大量的空间，且受内存限制无法水平扩展更多的web-server

·大型分布式集群情况下，由于所有web-server都要全量保存数据，所以此方案不可取。

***（C）统一存储：将用户的信息存储在第三方中间件上，做到统一存储，如redis中，所有的服务都到redis中获取用户信息，从而实现session共享。***

优点：

·没有安全隐患

·可以水平扩展

·服务器重启或扩容都不会造成session的丢失

缺点：

·增加了一次网络调用，速度有所下降

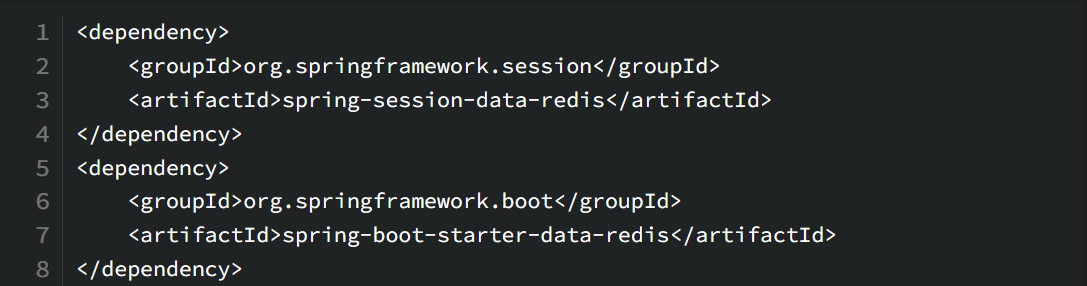
·需要修改应用程序代码，如将所有的getSession方法替换为Redis查数据的方式。但这个问题可以通过spring session完美解决

## （3）使用Redis等缓存技术实现分布式会话

***（a）认识Redis：***REmote DIctionary Server(Redis)是一个由 Salvatore Sanfilippo 写的 key-value 存储系统，是跨平台的非关系型数据库。Redis 是一个开源的使用 ANSI C 语言编写、遵守 BSD 协议、支持网络、可基于内存、分布式、可选持久性的键值对(Key-Value)存储数据库，并提供多种语言的 API。Redis 通常被称为数据结构服务器，因为值（value）可以是字符串(String)、哈希(Hash)、列表(list)、集合(sets)和有序集合(sorted sets)等类型。

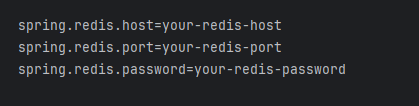
***（b）使用Redis缓存技术实现分布式会话***

1.添加依赖



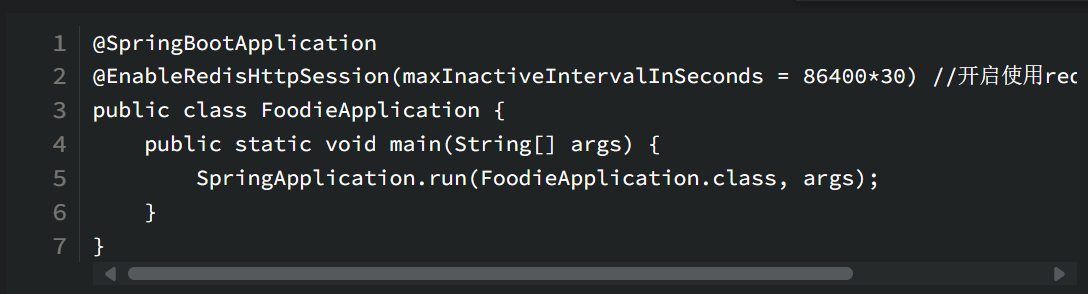
\*注意如果使用的是***Maven***时***，***pom.xml文件中添加此依赖

1. 配置Redis



\*在**application.propertie**s中配置Redis

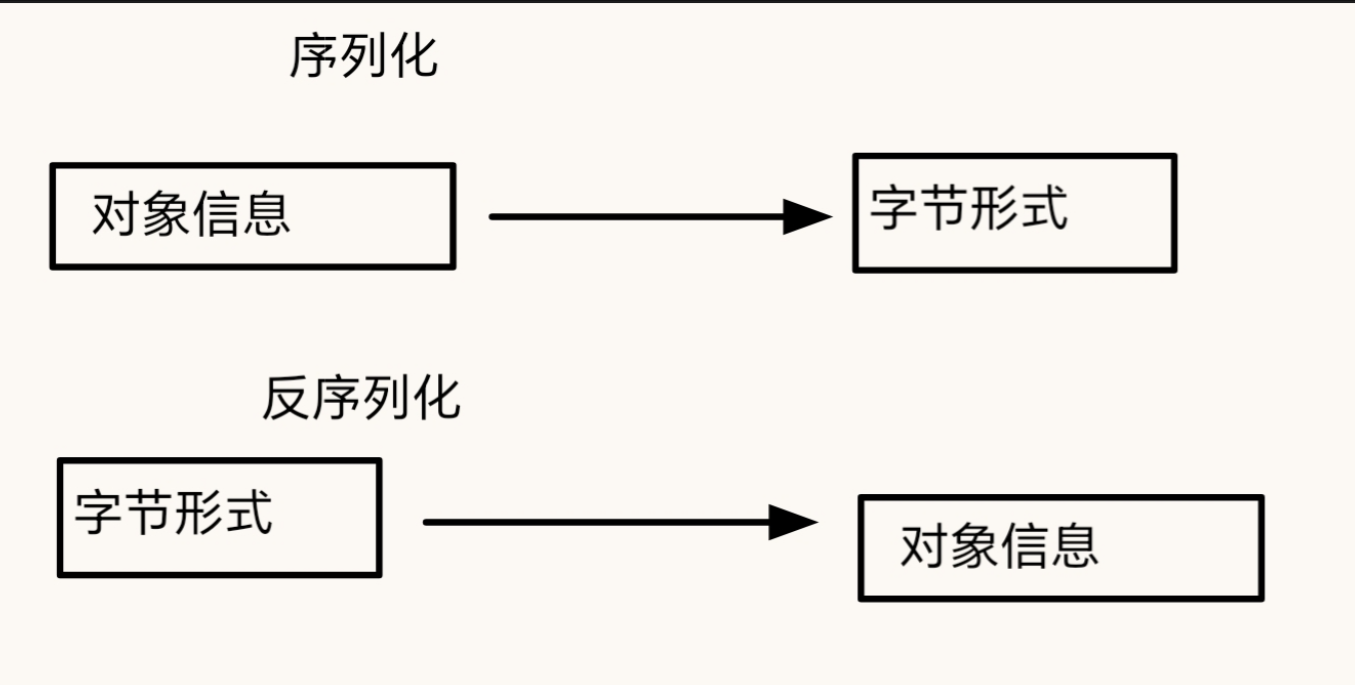
1. 添加注解@EnableRedisHttpSession



# 3、会话状态的序列化和反序列化

## （1）会话状态的序列化和反序列化

· ***会话状态的序列化和反序列化：*序列化**是指将对象的状态信息转换为可以**存储或传输的形式**的过程。这通常涉及到将数据结构或对象转换成**字节流**或**字符串**格式。**反序列化**则是序列化的逆过程，即将序列化后的**数据转换回原始的数据结构或对象**。

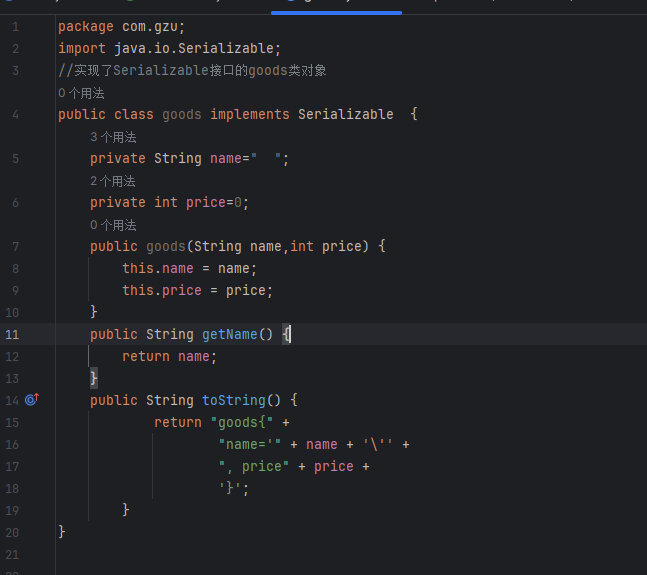


## （2）为什么需要序列化会话状态

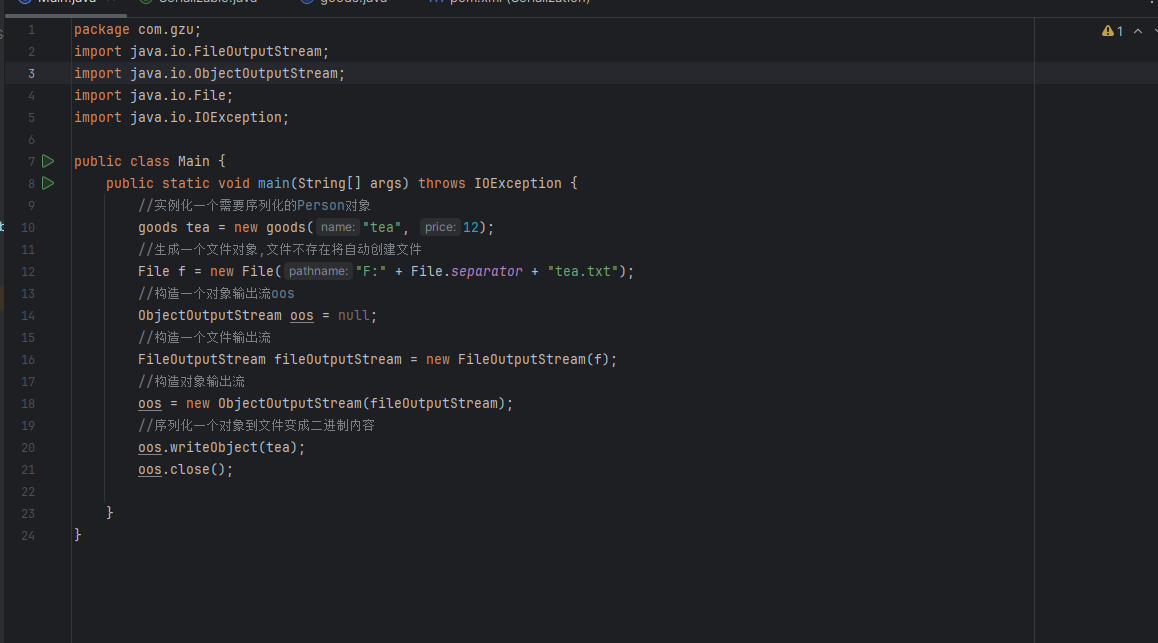
为我们提供一种**便捷的方式**，让我们可以在不同场景下轻松地**保存和恢复**对象的状态。比如说，当我们需要将对象的状态保存到文件或数据库中，或者通过网络传输对象时，序列化和反序列化就派上了用场。此外，在Web应用中，Session对象的序列化也大大减轻了服务器的压力。

## Java对象序列化

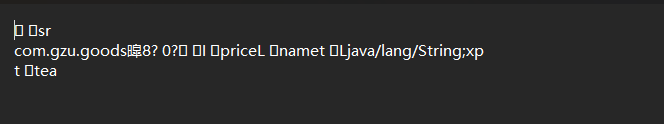
***（a）实现了Serializable接口的Person类对象，这个对象现在就已经具备了“可序列化”的能力。***



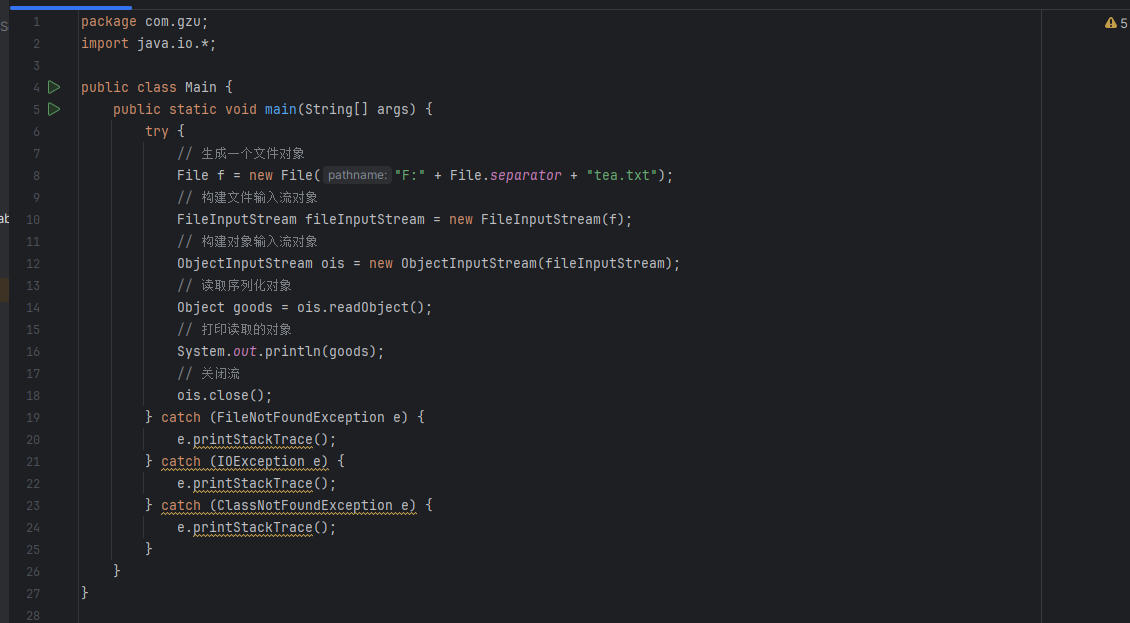
***（b）使用“对象输出流”来序列化一个Java对象。***



输出文件：



***（c)使用ObjectInputStream流把我们刚刚序列化的Person对象给反序列化出来***



## 自定义序列化策略

