一文搞懂Cookie、Session、Token、Jwt以及实战

一文搞懂Cookie、Session、Token、Jwt

Cookie

Cookie是存储在客户端（用户浏览器）的**小块数据**，可以用来记住用户的相关信息，例如登录凭证或偏好设置。它们随每个HTTP请求发送给服务器，并且可以被服务器读取以维持会话或个性化用户体验。

例如： 想象用户登录银行网站。服务器创建一个包含会话标识符的Cookie，并通过Set-Cookie头部发送回用户的浏览器。浏览器存储此Cookie，并在随后的请求中将其发送回服务器，允许服务器识别用户并在多个页面加载中保持他们的登录状态。

Session

会话用于跟踪用户在多个页面请求期间的状态。它们通常存储在**服务器端**，并且与唯一的会话标识符（通常是会话ID）相关联，会话ID作为Cookie发送给客户端。会话允许服务器在用户访问期间记住有关用户的信息。

例如： 用户在电子商务网站上购物。服务器为用户创建一个会话，存储他们的购物车项目和其他相关信息。会话ID作为Cookie发送给用户的浏览器。随着用户在网站上导航，Cookie中的会话ID允许服务器访问用户会话数据，使用户能够无缝购物体验。

Token

Token是一种无状态认证形式，客户端拥有一个**令牌**，通常是一串字符串，用于认证向服务器的请求。Token不要求服务器跟踪用户的状态，因为所有必要的信息都编码在令牌本身中。

例如： 用户希望通过移动应用程序访问他们的电子邮件。应用程序向电子邮件提供商的服务器发送带有用户凭据的请求。成功认证后，服务器发出一个访问令牌。应用程序存储此令牌，并在随后的API请求中使用它来访问用户的电子邮件。

JWT (JSON Web Tokens)

JWT是一种紧凑、安全的表示双方之间传输声明的方法。JWT是一个包含头部、负载和签名的**JSON对象**。JWT可用于认证和授权用户，它们是自包含的，意味着验证它们所需的所有信息都包含在令牌本身中。

例如： 开发人员创建了一个具有单点登录功能的Web应用程序。用户登录后，服务器生成一个包含用户身份和权限的JWT。这个JWT发送给客户端并存储在本地。当用户想要访问受保护的资源时，客户端在HTTP请求的Authorization头部中包含JWT。服务器验证JWT，如果有效，则授予资源访问权限。

四者的区别

下面是一个图表从各个方面说明了他们的区别

特性	Cookie	Session	Token	JWT
定义	服务器发送到浏览器的数据，用于跟踪状态	服务器端的会话状态记录	安全令牌，用于身份验证和信息交换	基于JSON的轻量级认证机制
存储位置	客户端	服务器端	客户端（LocalStorage或Cookie）	客户端（LocalStorage或Cookie）
安全性	较低，易被窃取或篡改	较高，数据不在客户端暴露	较高，尤其是加密Token	较高，包含签名，验证数据完整性
跨域支持	默认不支持，可通过设置实现	不支持，依赖Cookie	支持，不依赖Cookie	支持，不依赖Cookie
大小限制	约4KB	无大小限制	无大小限制	通常较小，但受JSON大小限制
生命周期	可设置过期时间	通常在用户关闭浏览器或超时后失效	可设置过期时间	可设置过期时间
无状态支持	不支持，依赖于Cookie	支持，但Session需基于Cookie	支持，服务端无状态	支持，服务端无状态

适用场景	简单的会话跟踪，用户偏好设置	需要服务器记住用户状态的场景	移动应用、API身份验证、跨域请求	Web应用、移动应用、单点登录
跨域问题	存在跨域限制	无跨域问题，但需处理集群部署的Session共享	无跨域问题，适合跨域认证	无跨域问题，适合跨域认证
服务器压力	无	高并发时会增加服务器压力	低，适合大规模部署	低，适合大规模部署
数据类型	只支持字符串	可以存储任意数据类型	可以存储任意数据类型	可以存储非敏感信息

下面我们从他的优点和缺点来介绍他们四个的区别

机制	简介	优点	缺点	适用场景
Cookie	在客户端存储小型文本文件	简单易用、支持跨域	有限存储容量、易受CSRF攻击	存储少量不敏感信息，如用户偏好设置等
Session	在服务器上存储关联特定用户会话的数据	安全性更高、可存储敏感信息	服务器负载增加、需要维护会话状态	存储较多敏感信息，如用户登录状态、购物车内容等
Token	用于身份验证和授权的令牌	无状态、可扩展、跨域	需要额外的安全措施来保护令牌、增加网络传输负载	API身份验证，特别是在分布式系统中
JWT	一种基于JSON的开放标准，用于安全传输信息	可扩展、自包含、无需服务器状态	一旦签发无法撤销、增加网络传输负载	跨域认证，特别是在分布式系统和单点登录（SSO）场景中

汇总：Cookie 和 Session 是传统的基于服务器的会话管理机制，而 Token 和 JWT 则是更为灵活和安全的身份验证和授权机制，适用于分布式系统和前后端分离的应用场景。JWT 是 Token 的一种实现方式，具有更高的可移植性和可扩展性。

项目实战

这里是在springboot中实战的一些区别

```
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.util.Date;

@SpringBootApplication
@RestController
public class AuthApplication {

    private final String SECRET_KEY = ”secretkey123”; // JWT加密密钥

    public static void main(String[] args) {
        SpringApplication.run(AuthApplication.class, args);
    }

    @GetMapping(”/setCookie”)
    public String setCookie(HttpServletResponse response) {
        Cookie cookie = new Cookie(”user”, ”john_doe”);
        cookie.setMaxAge(3600); // 设置Cookie的生命周期为1小时
        response.addCookie(cookie);
        return ”Cookie设置成功！”;
    }

    @GetMapping(”/setSession”)
    public String setSession(HttpServletRequest request) {
        HttpSession session = request.getSession();
        session.setAttribute(”user”, ”john_doe”);
        return ”Session设置成功！”;
    }

    @GetMapping(”/getCookie”)
    public String getCookie(HttpServletRequest request) {
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if (cookie.getName().equals(”user”)) {
                    return ”从Cookie获取用户信息：” + cookie.getValue();
                }
            }
        }
        return ”未找到Cookie！”;
    }

    @GetMapping(”/getSession”)
```

```
    public String getSession(HttpServletRequest request) {
        HttpSession session = request.getSession();
        String user = (String) session.getAttribute("user");
        if (user != null) {
            return "从Session获取用户信息：" + user;
        }
        return "未找到Session！";
    }

    @PostMapping("/login")
    public String login(@RequestBody UserCredentials credentials) {
        // 在实际应用中，这里应该是对用户进行验证，比如检查数据库中的用
户名和密码是否匹配
        if (credentials.getUsername().equals("john_doe") &&
credentials.getPassword().equals("password123")) {
            // 生成Token
            String token = Jwts.builder()
                    .setSubject(credentials.getUsername())
                    .setIssuedAt(new Date())
                    .setExpiration(new Date(System.currentTimeMillis() +
3600000)) // 设置Token过期时间为1小时
                    .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                    .compact();
            return "登录成功！生成的Token：" + token;
        } else {
            return "用户名或密码错误！";
        }
    }

    @GetMapping("/secure")
    public String secure(HttpServletRequest request) {
        // 在实际应用中，这里应该是验证Token的有效性
        String token = request.getHeader("Authorization").replace("Bearer
", "");
        String user =
Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).getBo
dy().getSubject();
        return "受保护的端点访问成功，用户：" + user;
    }

    static class UserCredentials {
        private String username;
        private String password;

        // Getters and setters
        public String getUsername() {
            return username;
        }
```

```
        public void setUsername(String username) {
            this.username = username;
        }

        public String getPassword() {
            return password;
        }

        public void setPassword(String password) {
            this.password = password;
        }
    }
}
```

相信看了这个代码你应该已经明白了最基本的区别了。

之后我推荐一下在实战中的一些我认为的最佳实战（不代表为最好，在我这里为最好的，如果有错误也欢迎各位来评论区讨论）

首先，你需要添加Spring Security和JWT的依赖项到你的`pom.xml`文件中：

```
<dependencies>
    <!-- Spring Security -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <!-- JSON Web Token Support -->
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
    </dependency>
</dependencies>
```

然后，你可以创建一个JWT工具类来生成和验证JWT：

```
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.function.Function;

@Component
public class JwtUtils {
    private String secret = "yourSecretKey"; // 你的密钥

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() +
3600000)) // 1小时后过期
                .signWith(SignatureAlgorithm.HS256, secret)
                .compact();
    }

    public Boolean validateToken(String token, String username) {
        final String usernameFromToken = getUsernameFromToken(token);
        return (usernameFromToken.equals(username) &&
!isTokenExpired(token));
    }

    public String getUsernameFromToken(String token) {
        return getClaimFromToken(token, Claims::getSubject);
    }

    public Date getExpirationDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getExpiration);
    }

    public <T> T getClaimFromToken(String token, Function<Claims, T>
claimsResolver) {
        final Claims claims = getAllClaimsFromToken(token);
        return claimsResolver.apply(claims);
    }

    private Claims getAllClaimsFromToken(String token) {
        return
Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
    }
```

```
    private Boolean isTokenExpired(String token) {
        final Date expiration = getExpirationDateFromToken(token);
        return expiration.before(new Date());
    }
}
```

接下来，创建一个控制器来处理登录请求，并生成JWT：

```
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenti
cationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.User;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtUtils jwtUtils;

    @PostMapping("/signin")
    public String signin(@RequestBody User user) throws
AuthenticationException {
        authenticationManager.authenticate(
                new
UsernamePasswordAuthenticationToken(user.getUsername(),
user.getPassword())
        );
        return jwtUtils.generateToken(user.getUsername());
    }

    @GetMapping("/info")
    public String info(@RequestParam String token) {
```

```
        if (jwtUtils.validateToken(token, "user")) {
            return "Token is valid!";
        } else {
            return "Token is not valid!";
        }
    }
}

```

最后，你需要配置Spring Security来使用JWT进行认证：

```
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.A
uthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity
;
import
org.springframework.security.config.annotation.web.configuration.Enable
WebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSe
curityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .authorizeRequests()
            .antMatchers("/auth/**").permitAll()
            .anyRequest().authenticated()
            .and()
            .httpBasic(); // 使用基本认证
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        auth.inMemoryAuthentication()
```

```
            .withUser("user")
            .password("password")
            .roles("USER");
    }
}
```

这个只能作为你自己学习的时候一个方案，如果是上线的项目，还需要考虑很多安全性的问题。

下面是一些措施：

安全措施

使用HTTPS

为了保护数据在客户端和服务器之间传输的安全性，你应该使用HTTPS。HTTPS通过SSL/TLS对数据进行加密，防止中间人攻击和数据泄露。

在Spring Boot中启用HTTPS:

1.在`application.properties`或`application.yml`中配置服务器的SSL属性

```
server.port=8443
server.ssl.key-store=classpath:keystore.jks
server.ssl.key-store-password=yourpassword
server.ssl.key-password=yourkeypassword
```

2.创建一个密钥库文件(`keystore.jks`)并配置适当的密码。

3.确保你的应用程序可以通过8443端口访问，这是HTTPS的默认端口。

密钥管理

对于JWT，密钥管理是至关重要的。你应该使用一个安全的方式来存储和访问签名密钥，并且定期更换密钥。

密钥管理最佳实践:

1. 不要在代码中硬编码密钥。
2. 使用专门的密钥管理系统，如AWS KMS、HashiCorp Vault或其他。
3. 定期更换密钥，并确保旧密钥不再被用于签名新的JWT。

防止CSRF攻击

跨站请求伪造(CSRF)是一种攻击，攻击者可以利用用户已经认证的身份在用户不知情的情况下执行非预期的操作。

在Spring Security中防止CSRF:

1. 确保所有敏感操作都通过POST请求执行，而不是GET。
2. 使用Spring Security的`@csrfProtection`注解来启用CSRF保护。
3. 在表单提交时使用`_csrf`令牌。

```
@PostMapping("/some-protected-action")
@csrfProtection
public String someProtectedAction(@ModelAttribute SomeData data,
@RequestParam("csrfToken") String csrfToken) {
    // 你的业务逻辑
}
```

其他安全措施

* 使用最新的安全框架和库。
* 定期更新依赖项以修复已知的安全漏洞。
* 实施输入验证来防止注入攻击。
* 实施输出编码来防止跨站脚本(XSS)攻击。
* 限制密码尝试次数来防止暴力破解。

* 实施访问控制列表(ACL)来限制对敏感资源的访问。
原文链接: https://juejin.cn/post/7353543714151776295