**Spring Boot+Spring Security+JWT 实现 RESTful Api 权限控制**

摘要：用spring-boot开发RESTful API非常的方便，在生产环境中，对发布的API增加授权保护是非常必要的。现在我们来看如何利用JWT技术为API增加授权保护，保证只有获得授权的用户才能够访问API。

一：开发一个简单的API

在IDEA开发工具中新建一个maven工程，添加对应的依赖如下：

1. <dependency>
2. <groupId>org.springframework.boot</groupId>
3. <artifactId>spring-boot-starter</artifactId>
4. </dependency>
6. <dependency>
7. <groupId>org.springframework.boot</groupId>
8. <artifactId>spring-boot-starter-test</artifactId>
9. <scope>test</scope>
10. </dependency>
12. <dependency>
13. <groupId>org.springframework.boot</groupId>
14. <artifactId>spring-boot-starter-web</artifactId>
15. </dependency>
17. *<!-- spring-data-jpa -->*
18. <dependency>
19. <groupId>org.springframework.boot</groupId>
20. <artifactId>spring-boot-starter-data-jpa</artifactId>
21. </dependency>
23. *<!-- mysql -->*
24. <dependency>
25. <groupId>mysql</groupId>
26. <artifactId>mysql-connector-java</artifactId>
27. <version>5.1.30</version>
28. </dependency>
30. *<!-- spring-security 和 jwt -->*
31. <dependency>
32. <groupId>org.springframework.boot</groupId>
33. <artifactId>spring-boot-starter-security</artifactId>
34. </dependency>
35. <dependency>
36. <groupId>io.jsonwebtoken</groupId>
37. <artifactId>jjwt</artifactId>
38. <version>0.7.0</version>
39. </dependency>

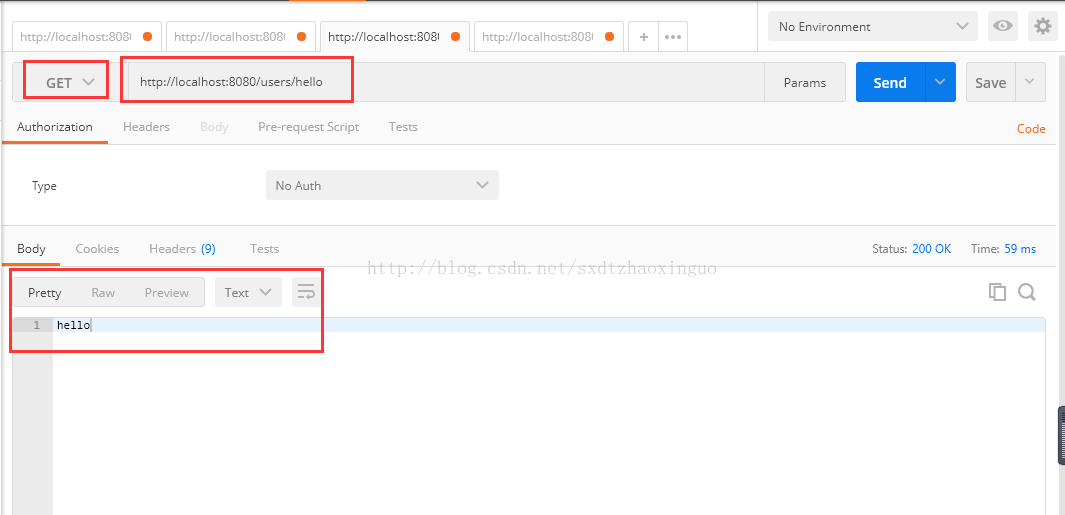
新建一个UserController.java文件，在里面在中增加一个hello方法：

1. @RequestMapping("/hello")
2. @ResponseBody
3. public String hello(){
4. return "hello";
5. }

这样一个简单的RESTful API就开发好了。

现在我们运行一下程序看看效果，执行JwtauthApplication.java类中的main方法：

等待程序启动完成后，可以简单的通过curl工具进行API的调用，如下图：



至此，我们的接口就开发完成了。但是这个接口没有任何授权防护，任何人都可以访问，这样是不安全的，下面我们开始加入授权机制。

二：增加用户注册功能

首先增加一个实体类User.java：

1. package boss.portal.entity;
3. import javax.persistence.\*;
5. */\*\**
6. *\* @author zhaoxinguo on 2017/9/13.*
7. *\*/*
8. @Entity
9. @Table(name = "tb\_user")
10. public class User {
12. @Id
13. @GeneratedValue
14. private long id;
15. private String username;
16. private String password;
18. public long getId() {
19. return id;
20. }
22. public String getUsername() {
23. return username;
24. }
26. public void setUsername(String username) {
27. this.username = username;
28. }
30. public String getPassword() {
31. return password;
32. }
34. public void setPassword(String password) {
35. this.password = password;
36. }
37. }

然后增加一个Repository类UserRepository，可以读取和保存用户信息：

1. package boss.portal.repository;
3. import boss.portal.entity.User;
4. import org.springframework.data.jpa.repository.JpaRepository;
6. */\*\**
7. *\* @author zhaoxinguo on 2017/9/13.*
8. *\*/*
9. public interface UserRepository extends JpaRepository<User, Long> {
11. User findByUsername(String username);
13. }

在UserController类中增加注册方法，实现用户注册的接口：

1. */\*\**
2. *\* 该方法是注册用户的方法，默认放开访问控制*
3. *\* @param user*
4. *\*/*
5. @PostMapping("/signup")
6. public void signUp(@RequestBody User user) {
7. user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
8. applicationUserRepository.save(user);
9. }

其中的@PostMapping("/signup")

这个方法定义了用户注册接口，并且指定了url地址是/users/signup。由于类上加了注解 @RequestMapping(“/users”)，类中的所有方法的url地址都会有/users前缀，所以在方法上只需指定/signup子路径即可。

密码采用了BCryptPasswordEncoder进行加密，我们在Application中增加BCryptPasswordEncoder实例的定义。

1. package boss.portal;
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
5. import org.springframework.context.annotation.Bean;
6. import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
8. @SpringBootApplication
9. public class JwtauthApplication {
11. @Bean
12. public BCryptPasswordEncoder bCryptPasswordEncoder() {
13. return new BCryptPasswordEncoder();
14. }
16. public static void main(String[] args) {
17. SpringApplication.run(JwtauthApplication.class, args);
18. }
19. }

三：增加JWT认证功能

用户填入用户名密码后，与数据库里存储的用户信息进行比对，如果通过，则认证成功。传统的方法是在认证通过后，创建sesstion，并给客户端返回cookie。现在我们采用JWT来处理用户名密码的认证。区别在于，认证通过后，服务器生成一个token，将token返回给客户端，客户端以后的所有请求都需要在http头中指定该token。服务器接收的请求后，会对token的合法性进行验证。验证的内容包括：

1. 内容是一个正确的JWT格式
2. 检查签名
3. 检查claims
4. 检查权限

处理登录

创建一个类JWTLoginFilter，核心功能是在验证用户名密码正确后，生成一个token，并将token返回给客户端：

1. package boss.portal.web.filter;
3. import boss.portal.entity.User;
4. import com.fasterxml.jackson.databind.ObjectMapper;
5. import io.jsonwebtoken.Jwts;
6. import io.jsonwebtoken.SignatureAlgorithm;
7. import org.springframework.security.authentication.AuthenticationManager;
8. import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
9. import org.springframework.security.core.Authentication;
10. import org.springframework.security.core.AuthenticationException;
11. import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
13. import javax.servlet.FilterChain;
14. import javax.servlet.ServletException;
15. import javax.servlet.http.HttpServletRequest;
16. import javax.servlet.http.HttpServletResponse;
17. import java.io.IOException;
18. import java.util.ArrayList;
19. import java.util.Date;
21. */\*\**
22. *\* 验证用户名密码正确后，生成一个token，并将token返回给客户端*
23. *\* 该类继承自UsernamePasswordAuthenticationFilter，重写了其中的2个方法*
24. *\* attemptAuthentication ：接收并解析用户凭证。*
25. *\* successfulAuthentication ：用户成功登录后，这个方法会被调用，我们在这个方法里生成token。*
26. *\* @author zhaoxinguo on 2017/9/12.*
27. *\*/*
28. public class JWTLoginFilter extends UsernamePasswordAuthenticationFilter {
30. private AuthenticationManager authenticationManager;
32. public JWTLoginFilter(AuthenticationManager authenticationManager) {
33. this.authenticationManager = authenticationManager;
34. }
36. *// 接收并解析用户凭证*
37. @Override
38. public Authentication attemptAuthentication(HttpServletRequest req,
39. HttpServletResponse res) throws AuthenticationException {
40. try {
41. User user = new ObjectMapper()
42. .readValue(req.getInputStream(), User.class);
44. return authenticationManager.authenticate(
45. new UsernamePasswordAuthenticationToken(
46. user.getUsername(),
47. user.getPassword(),
48. new ArrayList<>())
49. );
50. } catch (IOException e) {
51. throw new RuntimeException(e);
52. }
53. }
55. *// 用户成功登录后，这个方法会被调用，我们在这个方法里生成token*
56. @Override
57. protected void successfulAuthentication(HttpServletRequest req,
58. HttpServletResponse res,
59. FilterChain chain,
60. Authentication auth) throws IOException, ServletException {
62. String token = Jwts.builder()
63. .setSubject(((org.springframework.security.core.userdetails.User) auth.getPrincipal()).getUsername())
64. .setExpiration(new Date(System.currentTimeMillis() + 60 \* 60 \* 24 \* 1000))
65. .signWith(SignatureAlgorithm.HS512, "MyJwtSecret")
66. .compact();
67. res.addHeader("Authorization", "Bearer " + token);
68. }
70. }

该类继承自UsernamePasswordAuthenticationFilter，重写了其中的2个方法：

attemptAuthentication ：接收并解析用户凭证。

successfulAuthentication ：用户成功登录后，这个方法会被调用，我们在这个方法里生成token。

授权验证

用户一旦登录成功后，会拿到token，后续的请求都会带着这个token，服务端会验证token的合法性。

创建JWTAuthenticationFilter类，我们在这个类中实现token的校验功能。

1. package boss.portal.web.filter;
3. import io.jsonwebtoken.Jwts;
4. import org.springframework.security.authentication.AuthenticationManager;
5. import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
6. import org.springframework.security.core.context.SecurityContextHolder;
7. import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;
9. import javax.servlet.FilterChain;
10. import javax.servlet.ServletException;
11. import javax.servlet.http.HttpServletRequest;
12. import javax.servlet.http.HttpServletResponse;
13. import java.io.IOException;
14. import java.util.ArrayList;
16. */\*\**
17. *\* token的校验*
18. *\* 该类继承自BasicAuthenticationFilter，在doFilterInternal方法中，*
19. *\* 从http头的Authorization 项读取token数据，然后用Jwts包提供的方法校验token的合法性。*
20. *\* 如果校验通过，就认为这是一个取得授权的合法请求*
21. *\* @author zhaoxinguo on 2017/9/13.*
22. *\*/*
23. public class JWTAuthenticationFilter extends BasicAuthenticationFilter {
25. public JWTAuthenticationFilter(AuthenticationManager authenticationManager) {
26. super(authenticationManager);
27. }
29. @Override
30. protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws IOException, ServletException {
31. String header = request.getHeader("Authorization");
33. if (header == null || !header.startsWith("Bearer ")) {
34. chain.doFilter(request, response);
35. return;
36. }
38. UsernamePasswordAuthenticationToken authentication = getAuthentication(request);
40. SecurityContextHolder.getContext().setAuthentication(authentication);
41. chain.doFilter(request, response);
43. }
45. private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {
46. String token = request.getHeader("Authorization");
47. if (token != null) {
48. *// parse the token.*
49. String user = Jwts.parser()
50. .setSigningKey("MyJwtSecret")
51. .parseClaimsJws(token.replace("Bearer ", ""))
52. .getBody()
53. .getSubject();
55. if (user != null) {
56. return new UsernamePasswordAuthenticationToken(user, null, new ArrayList<>());
57. }
58. return null;
59. }
60. return null;
61. }
63. }

该类继承自BasicAuthenticationFilter，在doFilterInternal方法中，从http头的Authorization 项读取token数据，然后用Jwts包提供的方法校验token的合法性。如果校验通过，就认为这是一个取得授权的合法请求。

SpringSecurity配置

通过SpringSecurity的配置，将上面的方法组合在一起。

1. package boss.portal.security;
3. import boss.portal.web.filter.JWTLoginFilter;
4. import boss.portal.web.filter.JWTAuthenticationFilter;
5. import org.springframework.boot.autoconfigure.security.SecurityProperties;
6. import org.springframework.context.annotation.Configuration;
7. import org.springframework.core.annotation.Order;
8. import org.springframework.http.HttpMethod;
9. import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
10. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
11. import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
12. import org.springframework.security.core.userdetails.UserDetailsService;
13. import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
15. */\*\**
16. *\* SpringSecurity的配置*
17. *\* 通过SpringSecurity的配置，将JWTLoginFilter，JWTAuthenticationFilter组合在一起*
18. *\* @author zhaoxinguo on 2017/9/13.*
19. *\*/*
20. @Configuration
21. @Order(SecurityProperties.ACCESS\_OVERRIDE\_ORDER)
22. public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
24. private UserDetailsService userDetailsService;
26. private BCryptPasswordEncoder bCryptPasswordEncoder;
28. public WebSecurityConfig(UserDetailsService userDetailsService, BCryptPasswordEncoder bCryptPasswordEncoder) {
29. this.userDetailsService = userDetailsService;
30. this.bCryptPasswordEncoder = bCryptPasswordEncoder;
31. }
33. @Override
34. protected void configure(HttpSecurity http) throws Exception {
35. http.cors().and().csrf().disable().authorizeRequests()
36. .antMatchers(HttpMethod.POST, "/users/signup").permitAll()
37. .anyRequest().authenticated()
38. .and()
39. .addFilter(new JWTLoginFilter(authenticationManager()))
40. .addFilter(new JWTAuthenticationFilter(authenticationManager()));
41. }
43. @Override
44. public void configure(AuthenticationManagerBuilder auth) throws Exception {
45. auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder);
46. }
48. }

这是标准的SpringSecurity配置内容，就不在详细说明。注意其中的

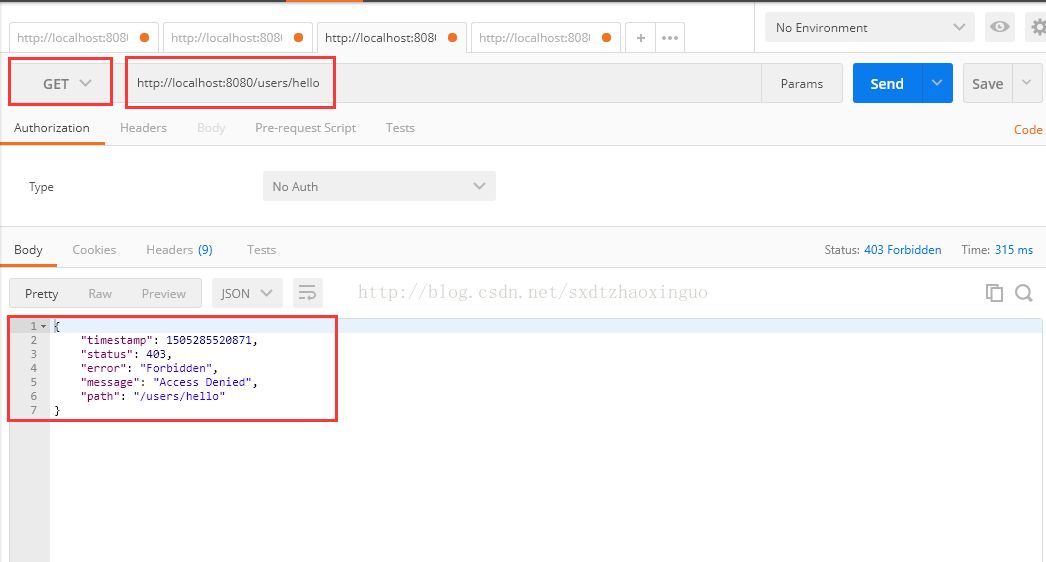
.addFilter(new JWTLoginFilter(authenticationManager()))   
.addFilter(new JwtAuthenticationFilter(authenticationManager()))

这两行，将我们定义的JWT方法加入SpringSecurity的处理流程中。

下面对我们的程序进行简单的验证：

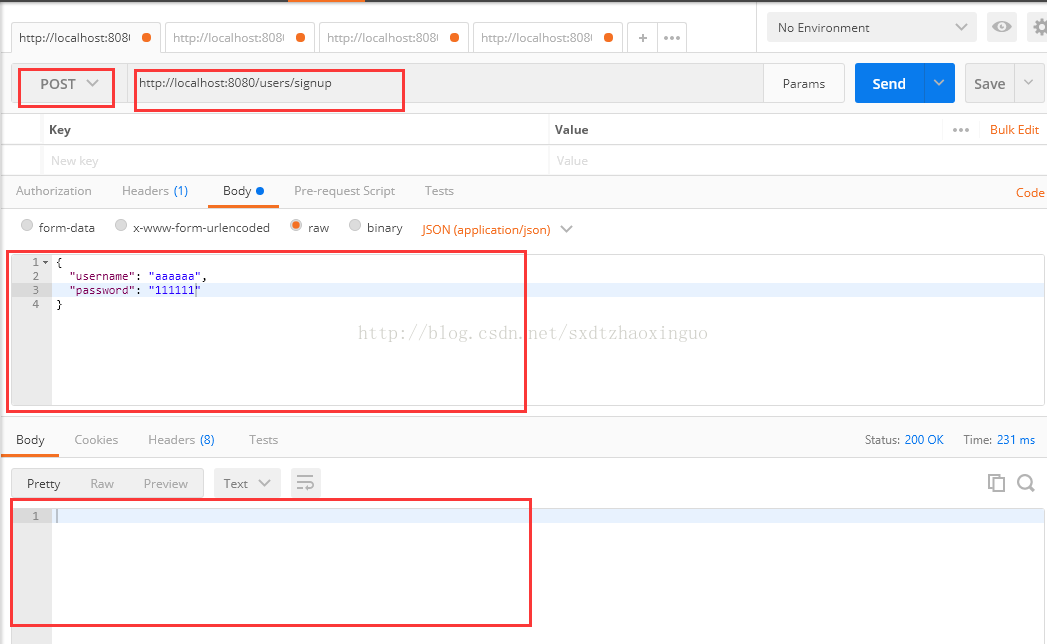
# 请求hello接口，会收到403错误，如下图：

curl http:*//localhost:8080/users/hello*

**

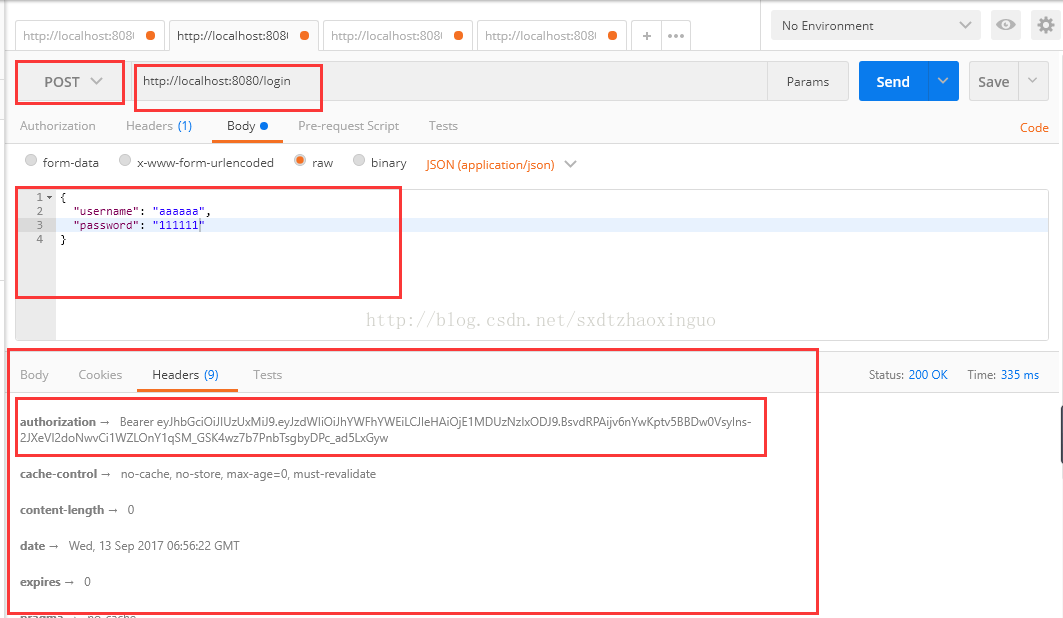
*# 注册一个新用户curl -H"Content-Type: application/json" -X POST -d '{"username":"admin","password":"password"}' http://localhost:8080/users/signup*

*如下图：*

**

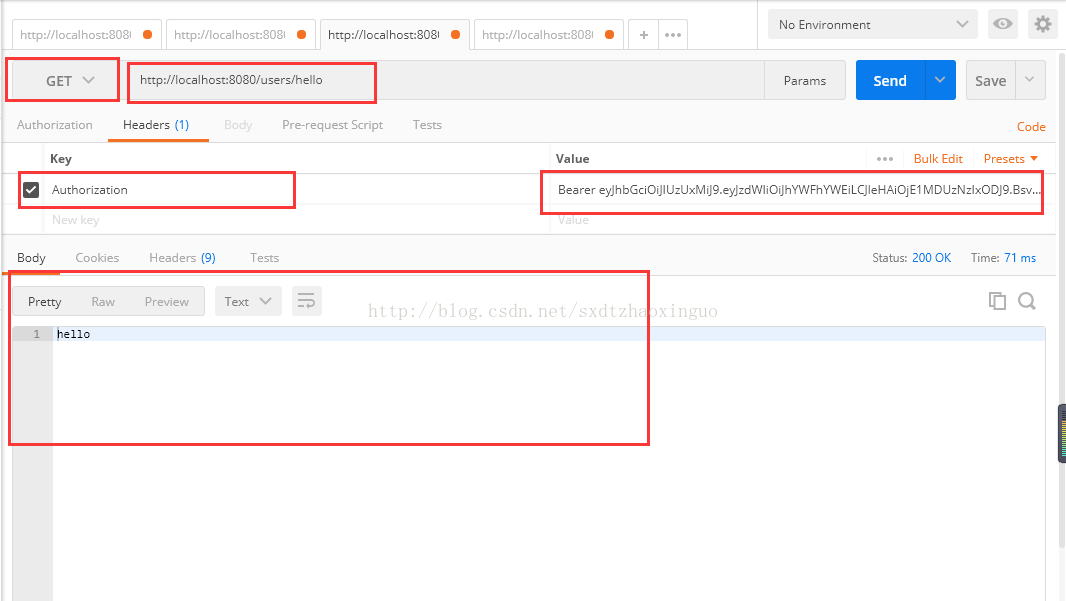
*# 登录，会返回token，在http header中，Authorization: Bearer 后面的部分就是tokencurl -i -H"Content-Type: application/json" -X POST -d '{"username":"admin","password":"password"}' http://localhost:8080/login*

*如下图：*

**

*# 用登录成功后拿到的token再次请求hello接口# 将请求中的XXXXXX替换成拿到的token# 这次可以成功调用接口了curl -H"Content-Type: application/json" \-H"Authorization: Bearer XXXXXX" \"http://localhost:8080/users/hello"*

*如下图：*

**

*五：总结*

至此，给SpringBoot的接口加上JWT认证的功能就实现了，过程并不复杂，主要是开发两个SpringSecurity的filter，来生成和校验JWT token。

JWT作为一个无状态的授权校验技术，非常适合于分布式系统架构，因为服务端不需要保存用户状态，因此就无需采用redis等技术，在各个服务节点之间共享session数据。