[回主页](https://www.cnblogs.com/ymstars/)

# [曾俊杰](https://www.cnblogs.com/ymstars/)

## 诚挚之心

* [博客园](https://www.cnblogs.com/)
* [首页](https://www.cnblogs.com/ymstars/)
* [新随笔](https://i.cnblogs.com/EditPosts.aspx?opt=1)
* [联系](https://msg.cnblogs.com/send/%E6%9B%BE%E4%BF%8A%E6%9D%B0%E7%9A%84%E4%B8%93%E6%A0%8F)
* [订阅](https://www.cnblogs.com/ymstars/rss)
* [管理](https://i.cnblogs.com/)

# [最简单易懂的Spring Security 身份认证流程讲解](https://www.cnblogs.com/ymstars/p/10626786.html)

最简单易懂的Spring Security 身份认证流程讲解

### 导言

相信大伙对Spring Security这个框架又爱又恨，爱它的强大，恨它的繁琐，其实这是一个误区，Spring Security确实非常繁琐，繁琐到让人生厌。讨厌也木有办法呀，作为JavaEE的工程师们还是要面对的，在开始之前，先打一下比方（比方好可怜）：

Spring Security 就像一个行政服务中心，如果我们去里面办事，可以办啥事呢？可以小到咨询简单问题、查询社保信息，也可以户籍登记、补办身份证，同样也可以大到企业事项、各种复杂的资质办理。但是我们并不需要跑一次行政服务中心，就挨个把业务全部办理一遍，现实中没有这样的人吧。

啥意思呢，就是说选择您需要的服务（功能），无视那些不需要的，等有需要的时候再了解不迟。这也是给众多工程师们的一个建议，特别是体系异常庞大的Java系，别动不动就精通，撸遍源码之类的，真没啥意义，我大脑的存储比较小，人生苦短，没必要。

回到正题！本文会以一种比较轻松的方式展开，不会是堆代码。

### 关于身份认证

Web 身份认证是一个后端工程师永远无法避开的领域，身份认证Authentication，和授权Authorization是不同的，Authentication指的是用户身份的认证，并不介入这个用户能够做什么，不能够做什么，仅仅是确认存在这个用户而已。而Authorization授权是建立的认证的基础上的，存在这个用户了，再来约定这个用户能补能够做一件事，这点大家要区分开。本文讲的是Authentication的故事，并不会关注权限。

热热身，让我们来温习一下身份认证的方式演变：

* 先是最著名的入门留言板程序，相信很多做后端的工程师都做过留言板，那是一个基本没有框架的阶段，回想一下是怎么认证的。表单输入用户名密码Submit，然后后端取到数据数据库查询，查不到的话无情地抛出一个异常，哦，密码错了；查到了，愉快的将用户ID和相关信息加密写入到Session标识中存起来，响应写入Cookie，后续的请求都解密后验证就行了，对吧。是的，身认证真可以简单到仅仅是匹配Session标识而已。令人沮丧的是现代互联网的发展早已经过了 Web2.0 的时代，客户端的出现让身份认证更加复杂。我们继续
* 随着移动端的崛起，Android和ios占据主导，同样是用户登录认证，取到用户信息，正准备按图索骥写入Session回写Cookie的时候，等等！啥？Android不支持Cookie？这听起来不科学是吧，有点反人类是吧，有点手足无措是吧。

嘿嘿，聪明的人儿也许想到了办法，嗯，Android客户端不是有本地存储吗？把回传的数据存起来不就行了吗？又要抱歉了，Android本地存储并没有浏览器Cookie那么人性化，不会自动过期。没事，再注明过期时间，每次读取的时候判断就行啦，貌似可以了。

等等。客户端的Api接口要求轻量级，某一天一个队友想实现个性化的事情，竟然往Cookie了回传了一串字符串，貌似很方便，嗯。于是其他队友也效仿，然后Cookie变得更加复杂。此时Android队友一声吼，你们够了！STOP！我只要一个认证标识而已，够简单你们知道吗？还有Cookie过期了就要重新登陆，用户体验极差，产品经理都找我谈了几十次了，用户都快跑光了，你们还在往Cookie里加一些奇怪的东西。

* Oauth 2.0来了

有问题总要想办法解决是吧。客户端不是浏览器，有自己特有的交互约定，Cookie还是放弃掉了。这里就要解决五个问题：

* [ ] 只需要简单的一个字符串标识，不需要遵守Cookie的规则
* [ ] 服务器端需要能够轻松认证这个标识，最好是做成标准化
* [ ] 不要让用户反复输入密码登录，能够自动刷新
* [ ] 这段秘钥要安全，从网络传输链路层到客户端本地层都要是安全的，就算被中途捕获，也可以让其失效
* [ ] 多个子系统的客户端需要独立的认证标识，让他们能够独立存在（例如淘宝的认证状态不会影响到阿里旺旺的登录认证状态）

需求一旦确定，方案呼之欲出，让我们来简单构思一下。

* [x] 首先是标识，这个最简单了，将用户标识数据进行可逆加密，OK，这个搞定。
* [x] 然后是标识认证的标准化，最好轻量级，并且让她不干扰请求的表现方式，例如Get和Post数据，聪明的你想到了吧，没错，就是Header，我们暂且就统一成 Userkey 为Header名，值就是那个加密过的标识，够简洁粗暴吧，后端对每一个请求都拦截处理，如果能够解密成功并且表示有效，就告诉后边排队的小伙伴，这个家伙是自己人，叫xxx，兜里有100块钱。这个也搞定了。
* [x] 自动刷新，因为加密标识每次请求都要传输，不能放在一起了，而且他们的作用也不一样，那就颁发加密标识的时候顺便再颁发一个刷新的秘钥吧，相当于入职的时候给你一张门禁卡，这个卡需要随身携带，开门签到少不了它，此外还有一张身份证明，这证明就不需要随身携带了，放家里都行，门禁卡掉了，没关系，拿着证明到保安大哥那里再领一张门禁卡，证明一次有效，领的时候保安大哥贴心的再给你一张证明。
* [x] 安全问题，加密可以加强一部分安全性。传输链路还用说吗？上Https传输加密哟。至于客户端本地的安全是一个哲学问题，嗯嗯嗯。哈哈。我们暂时认为本地私有空间存储是安全的的，俗话说得好，计算机都被人破解了，还谈个鸡毛安全呀（所以大家没事还是不要去ROOT手机了，ROOT之后私有存储可以被访问侬造吗）
* [x] 子系统独立问题，这个好办了。身份认证过程再加入一个因子，暂且叫 Client 吧。这样标识就互不影响了。

打完收工，要开始实现这套系统了。先别急呀，难道没觉得似曾相识吗？没错就是 Oauth 2.0 的 password Grant 模式！

### Spring Security 是怎么认证的

先来一段大家很熟悉的代码：

http.formLogin() .loginPage("/auth/login") .permitAll() .failureHandler(loginFailureHandler) .successHandler(loginSuccessHandler);

Spring Security 就像一个害羞的大姑娘，就这么一段鬼知道他是怎么认证的，封装的有点过哈。不着急先看一张图：

这里做了一个简化，

根据JavaEE的流程，本质就是Filter过滤请求，转发到不同处理模块处理，最后经过业务逻辑处理，返回Response的过程。

当请求匹配了我们定义的Security Filter的时候，就会导向Security 模块进行处理，例如UsernamePasswordAuthenticationFilter，源码献上:

public class UsernamePasswordAuthenticationFilter extends AbstractAuthenticationProcessingFilter { public static final String SPRING\_SECURITY\_FORM\_USERNAME\_KEY = "username"; public static final String SPRING\_SECURITY\_FORM\_PASSWORD\_KEY = "password"; private String usernameParameter = "username"; private String passwordParameter = "password"; private boolean postOnly = true; public UsernamePasswordAuthenticationFilter() { super(new AntPathRequestMatcher("/login", "POST")); } public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws AuthenticationException { if (this.postOnly && !request.getMethod().equals("POST")) { throw new AuthenticationServiceException("Authentication method not supported: " + request.getMethod()); } else { String username = this.obtainUsername(request); String password = this.obtainPassword(request); if (username == null) { username = ""; } if (password == null) { password = ""; } username = username.trim(); UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(username, password); this.setDetails(request, authRequest); return this.getAuthenticationManager().authenticate(authRequest); } } protected String obtainPassword(HttpServletRequest request) { return request.getParameter(this.passwordParameter); } protected String obtainUsername(HttpServletRequest request) { return request.getParameter(this.usernameParameter); } protected void setDetails(HttpServletRequest request, UsernamePasswordAuthenticationToken authRequest) { authRequest.setDetails(this.authenticationDetailsSource.buildDetails(request)); } public void setUsernameParameter(String usernameParameter) { Assert.hasText(usernameParameter, "Username parameter must not be empty or null"); this.usernameParameter = usernameParameter; } public void setPasswordParameter(String passwordParameter) { Assert.hasText(passwordParameter, "Password parameter must not be empty or null"); this.passwordParameter = passwordParameter; } public void setPostOnly(boolean postOnly) { this.postOnly = postOnly; } public final String getUsernameParameter() { return this.usernameParameter; } public final String getPasswordParameter() { return this.passwordParameter; } }

有点复杂是吧，不用担心，我来做一些伪代码，让他看起来更友善，更好理解。注意我写的单行注释

public class UsernamePasswordAuthenticationFilter extends AbstractAuthenticationProcessingFilter { public static final String SPRING\_SECURITY\_FORM\_USERNAME\_KEY = "username"; public static final String SPRING\_SECURITY\_FORM\_PASSWORD\_KEY = "password"; private String usernameParameter = "username"; private String passwordParameter = "password"; private boolean postOnly = true; public UsernamePasswordAuthenticationFilter() { //1.匹配URL和Method super(new AntPathRequestMatcher("/login", "POST")); } public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws AuthenticationException { if (this.postOnly && !request.getMethod().equals("POST")) { //啥？你没有用POST方法，给你一个异常，自己反思去 throw new AuthenticationServiceException("Authentication method not supported: " + request.getMethod()); } else { //从请求中获取参数 String username = this.obtainUsername(request); String password = this.obtainPassword(request); //我不知道用户名密码是不是对的，所以构造一个未认证的Token先 UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(username, password); //顺便把请求和Token存起来 this.setDetails(request, token); //Token给谁处理呢？当然是给当前的AuthenticationManager喽 return this.getAuthenticationManager().authenticate(token); } } }

是不是很清晰，问题又来了，Token是什么鬼？为啥还有已认证和未认证的区别？别着急，咱们顺藤摸瓜，来看看Token长啥样。上UsernamePasswordAuthenticationToken:

public class UsernamePasswordAuthenticationToken extends AbstractAuthenticationToken { private static final long serialVersionUID = 510L; private final Object principal; private Object credentials; public UsernamePasswordAuthenticationToken(Object principal, Object credentials) { super((Collection)null); this.principal = principal; this.credentials = credentials; this.setAuthenticated(false); } public UsernamePasswordAuthenticationToken(Object principal, Object credentials, Collection<? extends GrantedAuthority> authorities) { super(authorities); this.principal = principal; this.credentials = credentials; super.setAuthenticated(true); } public Object getCredentials() { return this.credentials; } public Object getPrincipal() { return this.principal; } public void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException { if (isAuthenticated) { throw new IllegalArgumentException("Cannot set this token to trusted - use constructor which takes a GrantedAuthority list instead"); } else { super.setAuthenticated(false); } } public void eraseCredentials() { super.eraseCredentials(); this.credentials = null; } }

一坨坨的真闹心，我再备注一下：

public class UsernamePasswordAuthenticationToken extends AbstractAuthenticationToken { private static final long serialVersionUID = 510L; //随便怎么理解吧，暂且理解为认证标识吧，没看到是一个Object么 private final Object principal; //同上 private Object credentials; //这个构造方法用来初始化一个没有认证的Token实例 public UsernamePasswordAuthenticationToken(Object principal, Object credentials) { super((Collection)null); this.principal = principal; this.credentials = credentials; this.setAuthenticated(false); } //这个构造方法用来初始化一个已经认证的Token实例，为啥要多此一举，不能直接Set状态么，不着急，往后看 public UsernamePasswordAuthenticationToken(Object principal, Object credentials, Collection<? extends GrantedAuthority> authorities) { super(authorities); this.principal = principal; this.credentials = credentials; super.setAuthenticated(true); } //便于理解无视他 public Object getCredentials() { return this.credentials; } //便于理解无视他 public Object getPrincipal() { return this.principal; } public void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException { if (isAuthenticated) { //如果是Set认证状态，就无情的给一个异常，意思是： //不要在这里设置已认证，不要在这里设置已认证，不要在这里设置已认证 //应该从构造方法里创建，别忘了要带上用户信息和权限列表哦 //原来如此，是避免犯错吧 throw new IllegalArgumentException("Cannot set this token to trusted - use constructor which takes a GrantedAuthority list instead"); } else { super.setAuthenticated(false); } } public void eraseCredentials() { super.eraseCredentials(); this.credentials = null; } }

搞清楚了Token是什么鬼，其实只是一个载体而已啦。接下来进入核心环节，AuthenticationManager是怎么处理的。这里我简单的过渡一下，但是会让你明白。

AuthenticationManager会注册多种AuthenticationProvider，例如UsernamePassword对应的DaoAuthenticationProvider，既然有多种选择，那怎么确定使用哪个Provider呢？我截取了一段源码，大家一看便知：

public interface AuthenticationProvider { Authentication authenticate(Authentication var1) throws AuthenticationException; boolean supports(Class<?> var1); }

这是一个接口，我喜欢接口，简洁明了。里面有一个supports方法，返回时一个boolean值，参数是一个Class，没错，这里就是根据Token的类来确定用什么Provider来处理，大家还记得前面的那段代码吗？

//Token给谁处理呢？当然是给当前的AuthenticationManager喽 return this.getAuthenticationManager().authenticate(token);

因此我们进入下一步，DaoAuthenticationProvider，继承了AbstractUserDetailsAuthenticationProvider，恭喜您再坚持一会就到曙光啦。这个比较复杂，为了不让你跑掉，我将两个复杂的类合并，摘取直接触达接口核心的逻辑，直接上代码，会有所删减，让你看得更清楚，注意看注释：

public class DaoAuthenticationProvider extends AbstractUserDetailsAuthenticationProvider { //熟悉的supports，需要UsernamePasswordAuthenticationToken public boolean supports(Class<?> authentication) { return UsernamePasswordAuthenticationToken.class.isAssignableFrom(authentication); } public Authentication authenticate(Authentication authentication) throws AuthenticationException { //取出Token里保存的值 String username = authentication.getPrincipal() == null ? "NONE\_PROVIDED" : authentication.getName(); boolean cacheWasUsed = true; //从缓存取 UserDetails user = this.userCache.getUserFromCache(username); if (user == null) { cacheWasUsed = false; //啥，没缓存？使用retrieveUser方法获取呀 user = this.retrieveUser(username, (UsernamePasswordAuthenticationToken)authentication); } //...删减了一大部分，这样更简洁 Object principalToReturn = user; if (this.forcePrincipalAsString) { principalToReturn = user.getUsername(); } return this.createSuccessAuthentication(principalToReturn, authentication, user); } protected final UserDetails retrieveUser(String username, UsernamePasswordAuthenticationToken authentication) throws AuthenticationException { try { //熟悉的loadUserByUsername UserDetails loadedUser = this.getUserDetailsService().loadUserByUsername(username); if (loadedUser == null) { throw new InternalAuthenticationServiceException("UserDetailsService returned null, which is an interface contract violation"); } else { return loadedUser; } } catch (UsernameNotFoundException var4) { this.mitigateAgainstTimingAttack(authentication); throw var4; } catch (InternalAuthenticationServiceException var5) { throw var5; } catch (Exception var6) { throw new InternalAuthenticationServiceException(var6.getMessage(), var6); } } //检验密码 protected void additionalAuthenticationChecks(UserDetails userDetails, UsernamePasswordAuthenticationToken authentication) throws AuthenticationException { if (authentication.getCredentials() == null) { this.logger.debug("Authentication failed: no credentials provided"); throw new BadCredentialsException(this.messages.getMessage("AbstractUserDetailsAuthenticationProvider.badCredentials", "Bad credentials")); } else { String presentedPassword = authentication.getCredentials().toString(); if (!this.passwordEncoder.matches(presentedPassword, userDetails.getPassword())) { this.logger.debug("Authentication failed: password does not match stored value"); throw new BadCredentialsException(this.messages.getMessage("AbstractUserDetailsAuthenticationProvider.badCredentials", "Bad credentials")); } } } }

到此为止，就完成了用户名密码的认证校验逻辑，根据认证用户的信息，系统做相应的Session持久化和Cookie回写操作。

Spring Security的基本认证流程先写到这里，其实复杂的背后是一些预定，熟悉了之后就不难了。

Filter->构造Token->AuthenticationManager->转给Provider处理->认证处理成功后续操作或者不通过抛异常

有了这些基础，后面我们再来扩展短信验证码登录，以及基于Oauth 2.0 的短信验证码登录。

分类: [spring](https://www.cnblogs.com/ymstars/category/1432695.html" \t "_blank)

[**好文要顶**](javascript:void(0);) [**关注我**](javascript:void(0);) [**收藏该文**](javascript:void(0);) **[https://common.cnblogs.com/images/icon_weibo_24.png](javascript:void(0);)** **[https://common.cnblogs.com/images/wechat.png](javascript:void(0);)**

[曾俊杰的专栏](https://home.cnblogs.com/u/ymstars/)  
[关注 - 0](https://home.cnblogs.com/u/ymstars/followees)  
[粉丝 - 2](https://home.cnblogs.com/u/ymstars/followers)

[+加关注](javascript:void(0);)

0

0

[«](https://www.cnblogs.com/ymstars/p/10508175.html)上一篇：[Laravel 和 Spring Boot 两个框架比较创业篇（二：人工成本）](https://www.cnblogs.com/ymstars/p/10508175.html)  
[»](https://www.cnblogs.com/ymstars/p/10629285.html)下一篇：[关于Jpa和Mybatis的一些看法](https://www.cnblogs.com/ymstars/p/10629285.html)

posted @ 2019-03-30 13:45 [曾俊杰的专栏](https://www.cnblogs.com/ymstars/) 阅读(984) 评论(0) [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=10626786) [收藏](https://www.cnblogs.com/ymstars/p/10626786.html)

[刷新评论](javascript:void(0);)[刷新页面](https://www.cnblogs.com/ymstars/p/10626786.html)[返回顶部](https://www.cnblogs.com/ymstars/p/10626786.html#top)

**注册用户登录后才能发表评论，请**[**登录**](javascript:void(0);)**或**[**注册**](javascript:void(0);)**，**[**访问**](http://www.cnblogs.com/)**网站首页。**

[【推荐】超50万C++/C#源码: 大型实时仿真组态图形源码](http://www.ucancode.com/index.htm" \t "_blank)  
[【前端】SpreadJS表格控件，可嵌入系统开发的在线Excel](https://www.grapecity.com.cn/developer/spreadjs?utm_source=cnblogs&utm_medium=blogpage&utm_term=bottom&utm_content=SpreadJS&utm_campaign=community" \t "_blank)  
[【推荐】码云企业版，高效的企业级软件协作开发管理平台](https://gitee.com/enterprises?from=bky-1" \t "_blank)  
[【推荐】程序员问答平台，解决您开发中遇到的技术难题](https://q.cnblogs.com/" \t "_blank)

**相关博文：**  
· [[JavaWeb]SpringSecurity-OAuth2.0 统一认证、资源分离的配置，用于分布式架构、模块化开发的认证体系](https://www.cnblogs.com/wuxinzhe/p/6739204.html" \t "_blank)  
· [ASP.NET Core 身份认证 (Identity、Authentication）](https://www.cnblogs.com/woailibian/p/6579850.html" \t "_blank)  
· [ASP.NET身份认证基础](https://www.cnblogs.com/laojiefang/archive/2012/05/05/2484501.html" \t "_blank)  
· [ASP.NET Forms身份认证](https://www.cnblogs.com/lilongsheng/archive/2012/11/30/3226075.html" \t "_blank)  
· [细说ASP.NET Forms身份认证](https://www.cnblogs.com/Alex80/p/5234131.html" \t "_blank)

**最新新闻**：  
· [宝马戴姆勒宣布共同开发自动驾驶 整合1200名工程师](https://news.cnblogs.com/n/627801/" \t "_blank)  
· [GitHub上的那些奇葩项目](https://news.cnblogs.com/n/627800/" \t "_blank)  
· [Facebook修复三大APP全球服务中断 向用户道歉](https://news.cnblogs.com/n/627799/" \t "_blank)  
· [美法院：华为前员工的确违约，但试图窃密的是华为](https://news.cnblogs.com/n/627798/" \t "_blank)  
· [天文学家首次精确定位来自太空的快速射电暴信号](https://news.cnblogs.com/n/627796/" \t "_blank)  
» [更多新闻...](http://news.cnblogs.com/" \o "IT新闻" \t "_blank)

### 公告

昵称：[曾俊杰的专栏](https://home.cnblogs.com/u/ymstars/)  
园龄：[4个月](https://home.cnblogs.com/u/ymstars/)  
粉丝：[2](https://home.cnblogs.com/u/ymstars/followers/)  
关注：[0](https://home.cnblogs.com/u/ymstars/followees/)

[+加关注](javascript:void(0);)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | [<](javascript:void(0);) | 2019年7月 | [>](javascript:void(0);) | | | | | | | |
| **日** | **一** | **二** | **三** | **四** | **五** | **六** |
| 30 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |

### 搜索

  

  

### 我的标签

* [java](https://www.cnblogs.com/ymstars/tag/java/)(1)
* [laravel](https://www.cnblogs.com/ymstars/tag/laravel/)(1)
* [php](https://www.cnblogs.com/ymstars/tag/php/)(1)
* [spring boot](https://www.cnblogs.com/ymstars/tag/spring%20boot/)(1)

### 随笔分类

* [laravel(2)](https://www.cnblogs.com/ymstars/category/1403921.html)
* [spring(2)](https://www.cnblogs.com/ymstars/category/1432695.html)

### 随笔档案

* [2019年3月 (5)](https://www.cnblogs.com/ymstars/archive/2019/03.html)

### 最新评论

* [1. Re:Laravel 和 Spring Boot 两个框架比较创业篇（一：开发效率）](https://www.cnblogs.com/ymstars/p/10480934.html#4201942)
* 在吗？ 1028 189 900 去去
* --代理ip
* [2. Re:Laravel 和 Spring Boot 两个框架比较创业篇（一：开发效率）](https://www.cnblogs.com/ymstars/p/10480934.html#4196793)
* @追疯真的人不,你错了,考虑到作者的起步资金,一个3年java(薪资),熟悉spring boot >= 2个php => 1.5个.net,如果是从其它语言转到java的,上手spring boot......
* --提伯斯
* [3. Re:Laravel 和 Spring Boot 两个框架比较创业篇（一：开发效率）](https://www.cnblogs.com/ymstars/p/10480934.html#4196753)
* @提伯斯能熟练使用spring boot的,至少两年工作经验以上=====================================如果是应届生，是需要至少两年，但是已经有了两三年的语言的开发经......
* --追疯真的人
* [4. Re:Laravel 和 Spring Boot 两个框架比较创业篇（一：开发效率）](https://www.cnblogs.com/ymstars/p/10480934.html#4196706)
* askjdjabdah
* --代理ip
* [5. Re:Laravel 和 Spring Boot 两个框架比较创业篇（一：开发效率）](https://www.cnblogs.com/ymstars/p/10480934.html#4196601)
* net core 完爆两者  
  ef狂拽酷霸屌炸天
* --艾尼路

### 阅读排行榜

* [1. Laravel 和 Spring Boot 两个框架比较创业篇（一：开发效率）(3923)](https://www.cnblogs.com/ymstars/p/10480934.html)
* [2. 最简单易懂的Spring Security 身份认证流程讲解(984)](https://www.cnblogs.com/ymstars/p/10626786.html)
* [3. 关于Jpa和Mybatis的一些看法(842)](https://www.cnblogs.com/ymstars/p/10629285.html)
* [4. Spring Boot JPA Entity Jackson序列化触发懒加载的解决方案(633)](https://www.cnblogs.com/ymstars/p/10473425.html)
* [5. Laravel 和 Spring Boot 两个框架比较创业篇（二：人工成本）(451)](https://www.cnblogs.com/ymstars/p/10508175.html)

### 评论排行榜

* [1. Laravel 和 Spring Boot 两个框架比较创业篇（一：开发效率）(19)](https://www.cnblogs.com/ymstars/p/10480934.html)

### 推荐排行榜

* [1. Laravel 和 Spring Boot 两个框架比较创业篇（一：开发效率）(6)](https://www.cnblogs.com/ymstars/p/10480934.html)
* [2. Spring Boot JPA Entity Jackson序列化触发懒加载的解决方案(1)](https://www.cnblogs.com/ymstars/p/10473425.html)

Copyright ©2019 曾俊杰的专栏