一 Spring security 概括

1 spring security 历史

Spring Security 开始于 2003 年年底，“spring 的 acegi 安全系统”。起因是 Spring 开发者邮件列表中的一个问题，有人提问是否考虑提供一个基于 spring 的安全实现。在当时 Spring 的社区相对较小（尤其是和今天的规模比！），其实 Spring 本身是从 2003 年初才作为一个 source-forge 的项目出现的。对这个问题的回应是，这的确是一个值得研究的领域，虽然限于时间问题阻碍了对它的继续研究。

有鉴于此，一个简单的安全实现建立起来了，但没有发布。几周之后，spring 社区的其他成员询问安全问题，代码就被提供给了他们。 随后又有人请求，在 2004 年一月左右，有 20 人在使用这些代码。另外一些人加入到这些先行者中来，并建议在 source-forge 上建立一个项目，项目在 2004 年 3 月正式建立起来。

在早期，项目本身没有自己的认证模块。认证过程都是依赖容器管理安全的，而 acegi 则注重授权。这在一开始是合适的，但随着越来越多用户要求提供额外的容器支持，基于容器认证的限制就显现出来了。还有一个有关的问题，向容器的 class-path 中添加新 jar，常常让最终用户感到困惑，又容易出现配置错误。

随后 acegi 加入了认证服务。大约一年后，acegi 成为 spring 的官方子项目。经过了两年半在许多生产软件项目中的活跃使用和数以万计的改善和社区的贡献，1.0.0 最终版本发布于 2006 年 5 月。

acegi 在 2007 年年底，正式成为 spring 组合项目，被更名为“Spring Security”。
现在，Spring Security 成为了一个强大而又活跃的开源社区。 在 Spring Security 支持论坛上有成千上万的信息。有一个积极的核心开发团队专职开发，一个积极的社区定期共享补丁并支持他们的同伴。

2 为什么用 spring seucrity

2.A 认证与授权

认证：authentication /ɔːˌθentɪˈkeɪʃn/。
授权：authorization /ˌɔːθərəˈzeɪʃn/。
英语如此烂，傻傻分不清楚。

认证是判断一个用户是谁的问题。

授权是判断一个用户能干什么的问题，授权一般都会绑定认证过程，并且在认证之后才能授权。

例如：
高铁飞机买票，到站后使用身份证验证身份就是认证。证明你是你，而不是你二舅，你三姑。
认证完成后打印机票登机牌，上飞机要看登机牌就是授权。说明有了登机牌你就有了上飞机这个操作的权限。但是仅限于上飞机权限，你不可能拿着登机牌去机场饭店白吃白喝。

2.B 整合多种功能

- oauth2

授权协议，可以实现第三方授权。例如：使用虎牙时候可以通过微信 qq 微博账号登录。

- spring social

也是依赖 spring 的一个框架，它是一个专门用户连接社交平台，实现 oauth 服务共享的框架。例如 facebook，Twitter，微信，新浪微博都提供了相关服务。

2.C 强大的依赖后盾

spring 框架之强大，应用之广泛，更新之迅速，对应应用场景之丰富自不必多说。自从 springboot 问世，使得早期 spring 基于 java 语言的各种不便几乎消失殆尽。spring security 从功能上足以应对几乎任何对安全框架的需求，又基于这样一个强大的后盾框架，可以说即使不使用这个技术，学习它也是有很多好处的。

二 Spring Security 入门案例

1 创建 spring-boot 项目并导入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.13.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>security</artifactId>
    <dependencies>
        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2 创建启动类

```
@SpringBootApplication
public class StarterSecurity {
    public static void main(String[] args) {
        SpringApplication.run(StarterSecurity.class,args);
    }
}
```

3 创建配置文件

```
#
配置默认端口 8080
server.port=9090
```

4 编写一些测试的 controller

```
@RestController
public class AdminController {

    @RequestMapping("/admin/write")
    public String write(){
        return "写入数据";
    }

    @RequestMapping("/admin/update")
    public String update(){
        return "更新数据";
```

```
    }

    @RequestMapping("/admin/delete")
    public String delete(){
        return "删除数据";
    }

    @RequestMapping("/user/read")
    public String read(){
        return "读取数据，不需要权限";
    }
}
```

5 测试

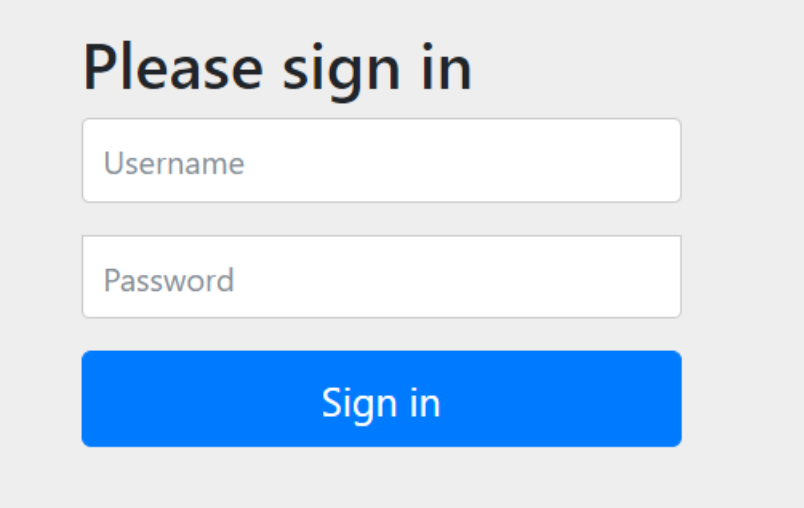
访问：http://localhost:9090/login
http://localhost:9090/user/read

…

登陆之前访问任何的 controller 中的接口都会跳到登陆界面

登陆后可访问任意接口，未做权限控制

（会自动创建账号 user、密码控制台打印，可在这里登陆）



6 思考问题

? 为什么当前搭建的工程没有做任何设置，spring security 看起来就生效了呢?

通过 springboot 自动配置,根据依赖 starter-security 提供的一些条件满足的类依赖,实现的自动配置.

? 用户名密码应该自己定义吧?

7 重写 WebSecurityConfig 权限控制自定义用户

spring security 提供给用户一个在整个过滤链中，方便实现自定义逻辑的配置类

a） 创建自定义配置类，

b） 继承 WebSecurityConfigurerAdapter，

c） 定义密码加密器

d） 重写 configure 的二个重载方法

e） 类上添加@EnableWebSecurity注解

```
@EnableWebSecurity
public class MySecurity extends WebSecurityConfigurerAdapter {
    //定义一个加密器和 4.x 不太一样 需要手动定义加密器
    //明文加密器 ,4.X 默认就是这个 5.x 的 security 没有手动定义会报错
    @Bean
    public PasswordEncoder myEncoder(){
        return NoOpPasswordEncoder.getInstance();
    }

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

        //admin 管理员用户 密码 123456 权限：增删查改所有权限
       auth.inMemoryAuthentication()
               .withUser("admin")
               .password("123456")
               .authorities("write","read","update","delete");
       //user 密码 123456 权限 查
        auth.inMemoryAuthentication()
                .withUser("user")
                .password("123456")
                .authorities("read");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //所有请求到当前系统都需要经过认证授权逻辑
        http.authorizeRequests()
                //当请求地址以/user/开始时，用户权限必须有 read
                .antMatchers("/user/**")
                .hasAuthority("read")

                //请求地址为"/admin/write"时，用户权限必须有 write
                .antMatchers("/admin/write")
                .hasAuthority("write")

                //请求地址为"/admin/update"时，用户权限必须有 update
                .antMatchers("/admin/update")
                .hasAuthority("update")

                //请求地址为"/admin/delete"时，用户权限必须有 delete
                .antMatchers("/admin/delete")
                .hasAuthority("delete")
                //其他的资源请求，只要登录就能访问
                .anyRequest().authenticated();
        //要求以表单填写用户名密码为认证入口
        http.formLogin();
    }
}
```

7.A 覆盖父类方法 configure(AuthenticationManagerBuilder auth)：

在这里我们利用内存数据重新定义了 2 个用户，由于 spring security5.x 不再使用 NoOpPasswordEncoder 作为默认密码加密器，所以需要自定义一个（myEncoder()）暂时使用的密码加密方式，这里使用的是“不加密”。后续会说怎么进行加密。

? 如何控制不同用户不同权限访问的资源呢

7.B 继续覆盖父类方法 configure(HttpSecurity http)实现 http 请求的与认证授权相关的拦截逻辑。

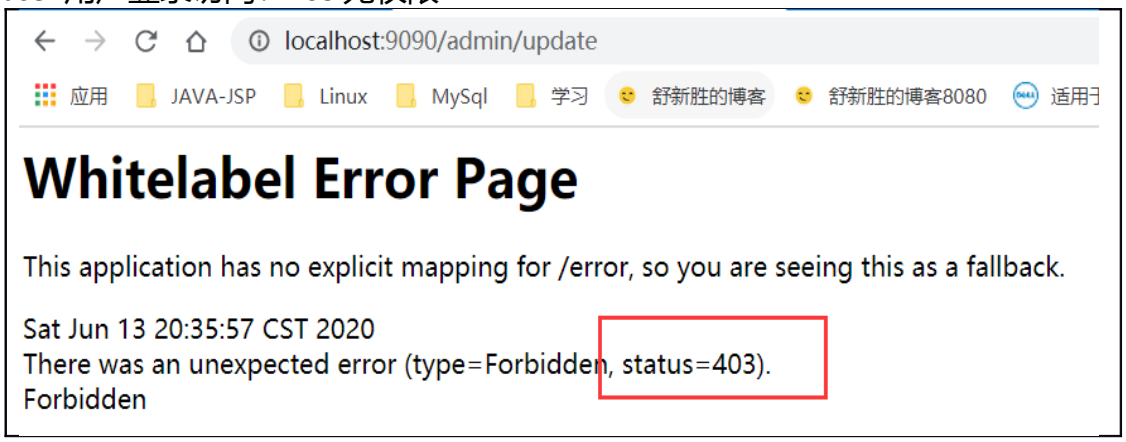
在上述方法中，antMathcers("")是比对请求地址，可以使用 ANT 匹配，也可以直接给准确地址。是细致地对请求地址做授权。/user/开始请求地址，需要有 read 权限，而 write，update，delete 分别对应一个权限。

而 anyRequest()表示其他地址，authenticated（）表示只要认证通过就可以访问。也就是说除了上面定义的地址外，只要已经登录，访问其它地址都可以。

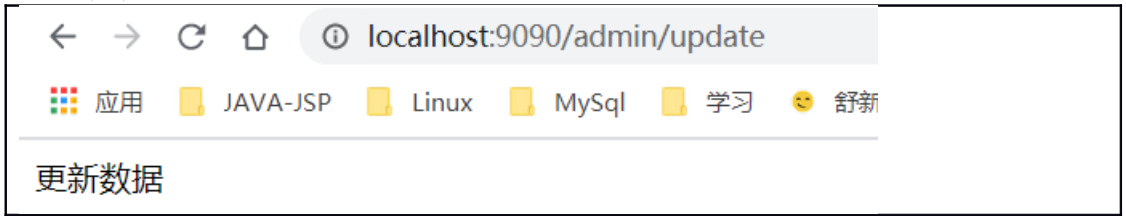
formLogin()表示认证以表单填写用户名密码方式进行。

7.C 权限测试

user 用户登录访问：403 无权限



admin 用户登录访问:正常访问



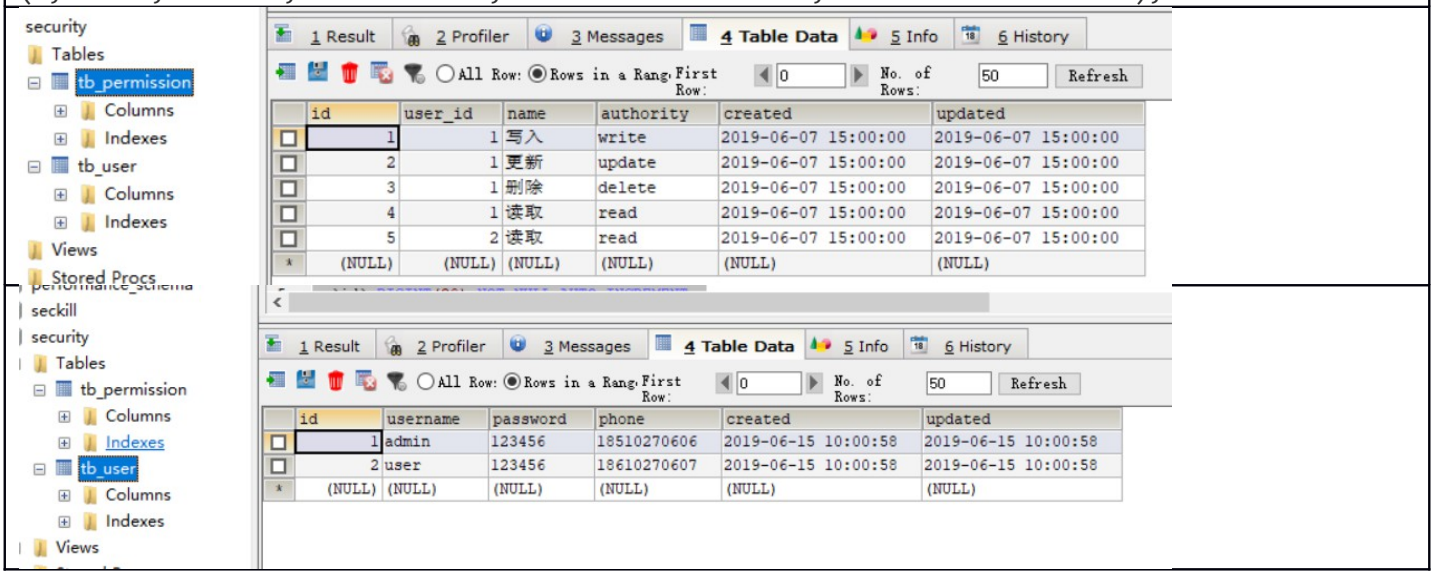
三　Spring Security 进阶（获取数据库用户）

前面我们通过 spring security 入门案例了解了 spring security 的认证授权逻辑。而且手动创建的模拟内存用户，实际项目中用户及用户权限是保存在数据库中。这里我们学习一下如何从数据库读用户表格数据。

1　导入测试数据到数据库

```
CREATE DATABASE /*!32312 IF NOT EXISTS*/`security` /*!40100 DEFAULT CHARACTER SET utf8 */;

USE `security`;

DROP TABLE IF EXISTS `tb_permission`;

CREATE TABLE `tb_permission` (
 `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
 `user_id` BIGINT(20) DEFAULT NULL COMMENT '父权限',
 `name` VARCHAR(64) NOT NULL COMMENT '权限名称',
 `authority` VARCHAR(64) NOT NULL COMMENT '权限英文名称',
 `created` DATETIME NOT NULL,
 `updated` DATETIME NOT NULL,
 PRIMARY KEY (`id`)
) ENGINE=INNODB AUTO_INCREMENT=49 DEFAULT CHARSET=utf8 COMMENT='权限表';

/*Data for the table `tb_permission` */

INSERT  INTO `tb_permission`(`id`,`user_id`,`name`,`authority`,`created`,`updated`) VALUES
(1,1,'写入','write','2019-06-07 15:00:00','2019-06-07 15:00:00'),(2,1,'更新','update','2019-06-
07 15:00:00','2019-06-07 15:00:00'),(3,1,'删除','delete','2019-06-07 15:00:00','2019-06-07
15:00:00'),(4,1,'读取','read','2019-06-07 15:00:00','2019-06-07 15:00:00'),(5,2,'读
取','read','2019-06-07 15:00:00','2019-06-07 15:00:00');

/*Table structure for table `tb_user` */

DROP TABLE IF EXISTS `tb_user`;

CREATE TABLE `tb_user` (
 `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
 `username` VARCHAR(50) NOT NULL COMMENT '用户名',
 `password` VARCHAR(64) NOT NULL COMMENT '密码，加密存储',
 `phone` VARCHAR(20) DEFAULT NULL COMMENT '注册手机号',
 `created` DATETIME NOT NULL,
 `updated` DATETIME NOT NULL,
 PRIMARY KEY (`id`),
```

```
 UNIQUE KEY `username` (`username`) USING BTREE,
 UNIQUE KEY `phone` (`phone`) USING BTREE
) ENGINE=INNODB AUTO_INCREMENT=39 DEFAULT CHARSET=utf8 COMMENT='用户表';

/*Data for the table `tb_user` */

INSERT  INTO `tb_user`(`id`,`username`,`password`,`phone`,`created`,`updated`) VALUES
(1,'admin','123456','18510270606','2019-06-15 10:00:58','2019-06-15 10:00:58'),
(2,'user','123456','18610270607','2019-06-15 10:00:58','2019-06-15 10:00:58');
```



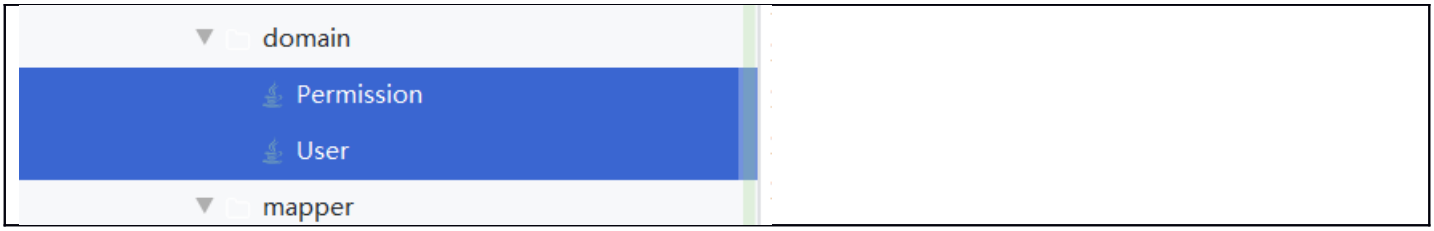
2 完成业务层持久层逻辑

2.A 添加持久层依赖

```
<!--mysql-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
<!--mybatis-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.2</version>
</dependency>
<!--JDBC-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

2.B 创建 domain 类

和数据库字段一致的实体 bean 类



2.C 创建 mapper 接口类

```
@Repository
public interface UserMapper {
//根据用户名获取用户信息
    @Select("SELECT * from tb_user where username=#{username}")
    public User selectUserByName(@Param("username") String username);
}
```

```
@Repository
public interface PermissionMapper {
//根据用户 id 获取用户权限信息
    @Select("SELECT * from tb_permission where user_id=#{userId}")
    public List<Permission> selectPermisssionsByUserid(@Param("userId")Long
userId);
}
```

2.D 启动类添加 mapper 扫描

```
@SpringBootApplication
@MapperScan("cn.shu.mapper")
public class StarterSecurity {
    public static void main(String[] args) {
        SpringApplication.run(StarterSecurity.class,args);
    }
}
```

2.E 配置文件中添加数据源信息

#

```
配置则默认端口 8080
server.port=9090
#datasource 配置
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql:///security?serverTimezone=Asia/Shanghai
spring.datasource.username=root
spring.datasource.password=admin
#mybatis
#开启驼峰命名转 bean
mybatis.configuration.map-underscore-to-camel-case=true
```

3 创建自定义的 UserDetailService

在 spring security 中，用户认证时查询用户数据可以自定义类继承 UserDetailsService 然后实现自定义的数据库表格结构查询用户信息。

所有的 user 认证过程都是这个 UserDetailsService 实现类完成.

- 创建一个类实现 UserDetailsService
- 实现 loadUserByUsername 方法

实现方法就一个 loadUserByUsername，方法参数就是 username 数据。可以判断，这种验证用户合法逻辑是通过 username 先查询出用户表格信息，然后通过提交的 password 加密后对比查询 password，如果相等则认证成功，不相等则认证失败。

方法的返回对象是 UserDetails，这个接口定义了 Spring security 查询的用户所有可扩展的规范。可以非常灵活的定义里面的数据，但是主要就是 3 个，username，password，authorities。前两个是用户名密码，最后是封装的权限数据，它是一个集合，因为一个用户的权限可以有多个。

```
public class MyUserDetailService implements UserDetailsService {
    @Autowired
    private UserMapper userMapper;
    @Autowired
    private PermissionMapper permissionMapper;
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        //查询 user 查询 permission 封装一个 UserDetails 对象结构返回
        //从数据库获取用户信息
        User user = userMapper.selectUserByName(username);
        //从数据库获取用户权限信息
        List<Permission> permissions =
permissionMapper.selectPermisssionsByUserid(user.getId());
        //封装用户权限信息
        List<GrantedAuthority> authorities=new ArrayList<>();
        for (Permission permission : permissions) {
            SimpleGrantedAuthority simpleGrantedAuthority = new
SimpleGrantedAuthority(permission.getAuthority());
            authorities.add(simpleGrantedAuthority);
        }
        //按照 security 要求封装 UserDetails 返回 User 为实现类
        return new
org.springframework.security.core.userdetails.User(username,user.getPassword(),authori
ties);

    }
}
```

上述方法的实现过程都是利用的已有现成的对象，也可以自定义接口的实现类，如 UserDetails 自定义实现和 GrantedAuthority 接口实现。

4 引入自定义 UserDetailsService 实现类

```
    @Bean
    public UserDetailsService initMyUserDetailsService(){
```

```
        return new MyUserDetailService();
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(initMyUserDetailsService());
/*        //admin 管理员用户 密码123456 权限：增删查改所有权限
      auth.inMemoryAuthentication()
              .withUser("admin")
              .password("123456")
              .authorities("write","read","update","delete");
      //user 密码123456 权限 查
       auth.inMemoryAuthentication()
              .withUser("user")
              .password("123456")
              .authorities("read");*/
    }
```

上述代码中将 auth 对象配置的内存用户信息注释掉，改用 userDetailsService 实现类来调用用户信息查询。

自定义的 MyUserDetailService 类可以通过注解如@Component 交给 Spring 管理，然后注入使用

这里是直接使用的@Bean，因为@EnableWebSecurity 注解的类也是配置类

四　Spring Security 进阶（用户密码加密）

1　spring security 的加密

在 spring security 中提供一个密码加密的接口 PasswordEncoder。

spring security 已经帮我们实现了一些常用的，安全性较高的密码加密实现类。比如，

StandartPasswordEncoder，BCryptPasswordEncoder 等。

如果想要使用其中任何一个，需要在配置类中使用@Bean 将其创建为容器对象，这样自定义的加密对象会成为 spring security 加密算法使用的对象。比如我们在之前代码中使用的明文加密 NoOpPasswordEncoder。

这里我们可以尝试使用别的加密算法。比如安全性较高的 BCryptPasswordEncoder。

```
//定义一个加密器和 4.x 不太一样 需要手动定义加密器
//明文加密器 ,4.X 默认就是这个 5.x 的 security 没有手动定义会报错
@Bean
public PasswordEncoder myEncoder(){
  /*  return NoOpPasswordEncoder.getInstance();*/
    return new BCryptPasswordEncoder();
}
```

这里需要注意，如果是在内存中的用户数据，需要指定 password 的时候利用加密器的 encode 方法实现赋值，如果是从数据库读取用户数据，那么密码这列就需要使用加密字符串。

2 自定义加密方式

除了使用 spring security 提供的密码加密实现类，我们也可以自定义实现这个类。主要就是实现 PasswordEncoder 这个接口的 2 个方法。

```
public class MyPasswordEncoder implements PasswordEncoder {
    //加密计算方法,把明文加密成密文  password 就是明文
    @Override
    public String encode(CharSequence password) {
        //CharSequence 类型方便进行加密计算
        String s = password.toString();
        //加密在明文前后添加一个字符串
        return "easymall_"+s+"haha";
    }
    //对比明文和 userDetails 中的密文中使用
    //凭什么明文和密码能相等 返回 true 表示 password 对比成功，false 表示失败
    //String s 的就是加密后数据库保存的秘密
    //CharSequence 就是登录时用户输入的秘密
    @Override
    public boolean matches(CharSequence charSequence, String s) {
        //用户输入秘密加密
        String encode = encode(charSequence);
        //与数据库的值比对
        return s.equals(encode);
    }
}
```

```
@Bean
public PasswordEncoder myEncoder(){
  /*  return NoOpPasswordEncoder.getInstance();*/
   /* return new BCryptPasswordEncoder();*/
    return new MyPasswordEncoder();
}
```

- encode 方法：

  该方法参数是明文字符串，使用 CharSequence 处理是为了更方便各种加密计算类型的转化。返回的 String 就是加密后的密码。

- matches 方法:

  这个方法就是在认证时判断用户密码与系统管理的密码是否匹配，一般判断非空后直接调用 encode 判断即可。

```
//String s 的就是加密后数据库保存的秘密
//CharSequence 就是登录时用户输入的秘密
```

五　SpringSecurity 高阶(会话管理)

1　理解会话

会话（session）就是解决无状态 HTTP 无法保存用户状态一种解决方案。
HTTP 本身的无状态使得用户在与服务器的交互过程中，每个请求之间都没有关联性。这意味着用户的访问没有身份记录。站点也无法为用户提供个性化的服务。session 的诞生解决了这个难题，服务器通过与用户约定每一个请求都携带一个 id 类的信息，从而让不同请求之间有了关联，而 id 又可以很方便的绑定具体用户，所以我们可以把不同请求归类到同一个用户。基于这个方案，为了让用户每一个请求都携带一个 id，在不妨碍体验的情况下，cookie 是很好的载体。当用户首次访问系统时，系统会为该用户生成一个 sessionId,并添加到 cookie 中。在该用户的会话期内，每个请求都自动携带 cookie，因此系统可以很轻松的识别出这事来自哪个用户的请求。

2　会话并发控制(登录顶替)

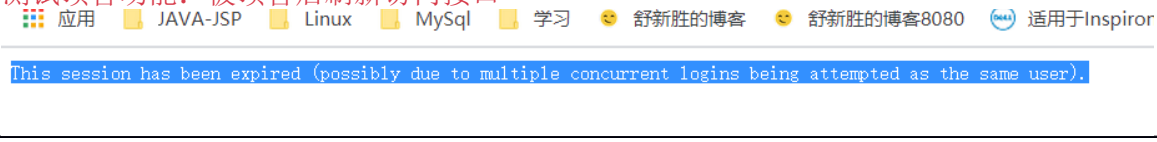
在 spring security 提供管理会话的功能，其中会话的并发控制是比较完善的。
当一个用户在 spring security 工程做了认证登录，客户端浏览器存储 sessionId 值，那么并不影响其他不同客户端使用相同用户名密码访问系统，这样一来同一个认证用户可以在不同的客户终端同时使用享受用户权限。这种情况对一些软件和系统来讲是不允许的。例如，购买了 vip 权限的视频账号，可以同时有 500 个人在线享受 vip 权限，却只需要购买一次。所以需要会话并发来控制。

实现 Spring security 的会话并发是非常容易的。只需要在过滤授权 http 方法链上添加管理会话的内容即可。

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    //所有请求到当前系统都需要经过认证授权逻辑
    http.authorizeRequests()
            //当请求地址以/user/开始时，用户权限必须有 read
            .antMatchers("/user/**")
            .hasAuthority("read")

            //请求地址为"/admin/write"时，用户权限必须有 write
            .antMatchers("/admin/write")
            .hasAuthority("write")

            //请求地址为"/admin/update"时，用户权限必须有 update
            .antMatchers("/admin/update")
            .hasAuthority("update")

            //请求地址为"/admin/delete"时，用户权限必须有 delete
            .antMatchers("/admin/delete")
            .hasAuthority("delete")
```

```
            //其他的资源请求，只要登录就能访问
            .anyRequest().authenticated();
    //要求以表单填写用户名密码为认证入口
    http.formLogin();
    //会话并发管理 表示同时只能一人登录同一账号
    http.sessionManagement().maximumSessions(1);
}
```

测试顶替功能：被顶替后刷新访问接口



这里有个陷阱

如果我们使用内存 user，或者使用 org.springframework.security.core.userdetails.User 来作为用户认证使用的对象，这个会话并发控制是有效的，如果我们使用自定义 UserDetails，必须重写 equals()和 hashCode()方法.原因是因为 spring security 底层判断 2 次登录是否是同一个用户，使用的是一个 map 对象。而 map 对象的 value 就是最大会话并发的一个 set，key 值就是 user 对象。判断对象是否是 map 的同一个 key 值，当然使用的是 equals 和 hashCode 方法，所以自定义 UserDetails 的话不重新定义这 2 个方法，永远在同一个用户登录时，底层判断是不相等的。

上面的例子中是可以的。

3 集群会话管理

spring security 作为认证登录服务器，使用 session 的管理控制并发整个过程结束后，我们不难联想，session 作为服务器内存数据，一定会在集群时出现共享的问题。那么很庆幸，spring 解决了 session 共享的问题，使用的技术叫做 spring session（说 spring 强大，从其技术涵盖范围可见一般）。

六 SpringSecurity 高阶（spring session）

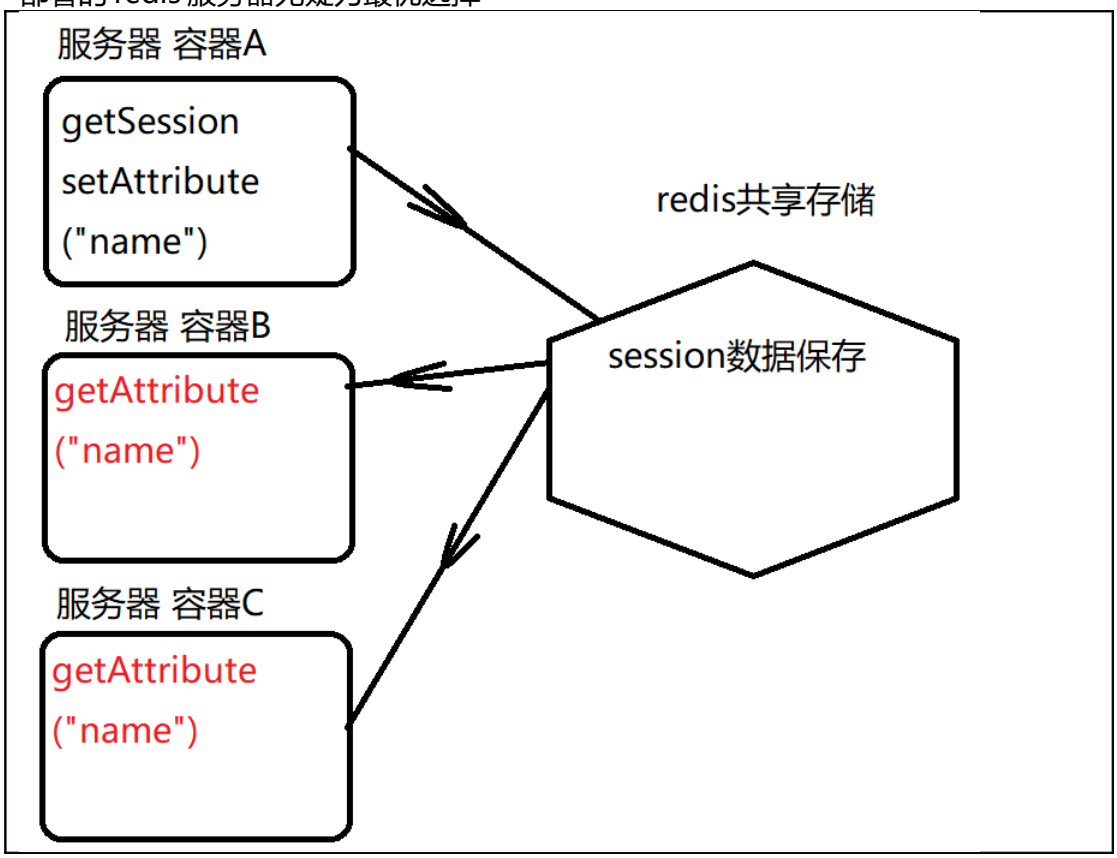
官网描述：Spring Session makes it trivial to support clustered sessions without being tied to an application container specific solution

Spring 会话使得支持集群会话而不必绑定到特定于应用程序容器的解决方案变得很简单。换句话将，想要集群共享 session，又和单节点 session 使用的一样，需要 Spring session

1 spring sessin 实现结构

spring session 实现集群会话共享的过程，就是将 session 数据保存到一个第三方存储中，而不是存储在容器内存里。所以可以选择数据库或者 redis。因为独立的数据存储增加了网络交互，数据存储的读/写性能，稳定性以及网络 I/O 速度都成为性能的瓶颈。基于这些问题，尽管理论

上使用任何存储介质都可以实现 session 共享，但是在网络环境中，尤其是内网环境，高可用部署的 redis 服务器无疑为最优选择



如图所示，在 A 容器中依然是调用常用的 session 方法 setAttribute，BC 容器依然调用常用的 session 方法 getAttribute 就能把 A 容器中设置的 session 域属性获取过来实现数据共享。

2 spring session 整合 springbootweb 应用

2.A 创建项目 pom.xml

依赖中 spring-session-data-redis 中既有 session-core 核心包也有 spring-data-redis，都会满足 springboot 自动配置（又是自动配置）

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.13.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>cn.shu</groupId>
    <artifactId>springboot-session-demo01</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springboot-session-demo01</name>
    <description>Demo project for Spring Boot</description>
    <properties>
```

```
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <!--spring session-->
        <dependency>
            <groupId>org.springframework.session</groupId>
            <artifactId>spring-session-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>redis.clients</groupId>
            <artifactId>jedis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2.B application.properties

```
#表示 session 数据用 redis 存储
spring.session.store-type=redis
#session 地址 给集群、cluster 都可以
spring.redis.host=10.42.175.170
#SESSION 端口
spring.redis.port=6380
```

2.C 启动类

```
@SpringBootApplication
public class StarterSession {
    public static void main(String[] args) {
        SpringApplication.run(StarterSession.class, args);
    }
}
```

2.D 测试 controller

等会调用这个接口写入读取 session 数据

```
@RestController
public class SessionController {
    @RequestMapping("session/add")
    public String add(HttpServletRequest req, String value){
        HttpSession session = req.getSession();
        session.setAttribute("name",value==null?
value:"cn.shu");
        return "success";
    }

    @RequestMapping("session/get")
    public String get(HttpServletRequest req){
        return (String)
req.getSession().getAttribute("name");
    }

}
```

2.E 测试

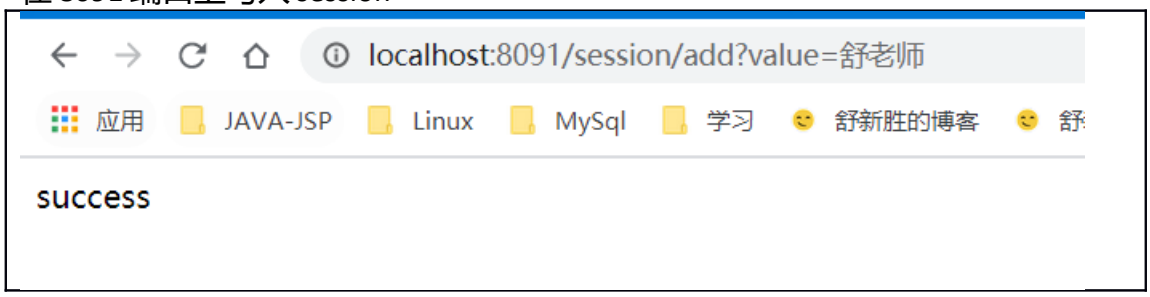
使用 idea 启动多个实例，修改端口以不同端口运行

测试前启动 redis（遇到的坑，默认启动开启了安全模式，外部 ip 无法访问）

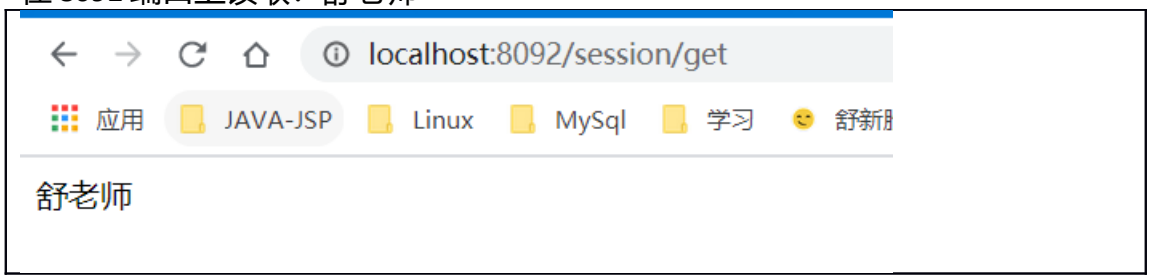
```
redis-server /home/software/redis-3.2.11/6380/redis.conf
```

测试写入数据：舒老师

在 8091 端口上写入 session



在 8092 端口上读取：舒老师



3 redis 管理的数据结构

```
127.0.0.1:6380> keys *
1) "spring:session:sessions:expires:f54172fd-9759-4afa-addb-b83ab49d2676"
2) "spring:session:sessions:f54172fd-9759-4afa-addb-b83ab49d2676"
3) "spring:session:expirations:1592065140000"
127.0.0.1:6380>
```

3.A hash 结构记录 2)

key 格式： spring:session:sessions:[sessionId]，对应的 value 保存 session 的所有数据包括：creationTme，maxInactiveInterval，lastAccessedTime，attribute；

```
127.0.0.1:6380>hkeys     spring:session:sessions:expires:f54172fd-9759-4afa-
addb-b83ab49d2676
1) "lastAccessedTime"
2) "sessionAttr:name"
3) "creationTime"
4) "maxInactiveInterval"
```

3.B set 结构记录 3)

key 格式：spring:session:expirations:[过期时间]，对应的 value 为 expires:[sessionId]列表，有效期默认是 30 分钟，即 1800 秒；

```
127.0.0.1:6380> scard spring:session:expirations:1592065140000
(integer) 1
127.0.0.1:6380> ttl spring:session:expirations:1592065140000
(integer) 610
127.0.0.1:6380>
```

3.C string 结构记录 1)

key 格式：spring:session:sessions:expires:[sessionId]，对应的 value 为空；该数据的 TTL 表示 sessionId 过期的剩余时间；这是一个保证过期的引用值。

七 spring security 整合 session

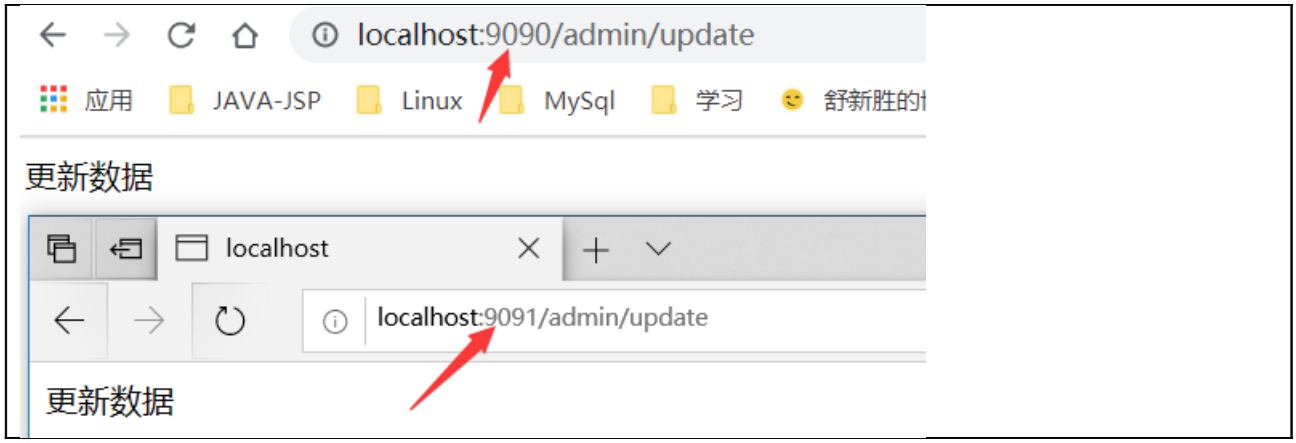
使用上述 security 的测试案例,代码中添加 session 相关依赖.唯一要考虑的代码开发底层原理,security 中有一个 sesssion 注册器.自定义注册器的使用,将其绑定到 spring session

1 整合前测试

开启二个 security 应用，端口不同，登录同一账户操作，结果之前设置的同时只能一个用户登录的逻辑没有生效。原因：集群 session

共享

//会话并发管理 表示同时只能一人登录同一账号
http.sessionManagement().maximumSessions(1);



2 pom 文件添加依赖

```
<!--spring session-->
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
</dependency>
```

3 application.properties 添加配置

```
#表示 session 数据用 redis 存储
spring.session.store-type=redis
#session 地址
spring.redis.host=10.42.175.170
#SESSION 端口
spring.redis.port=6380
```

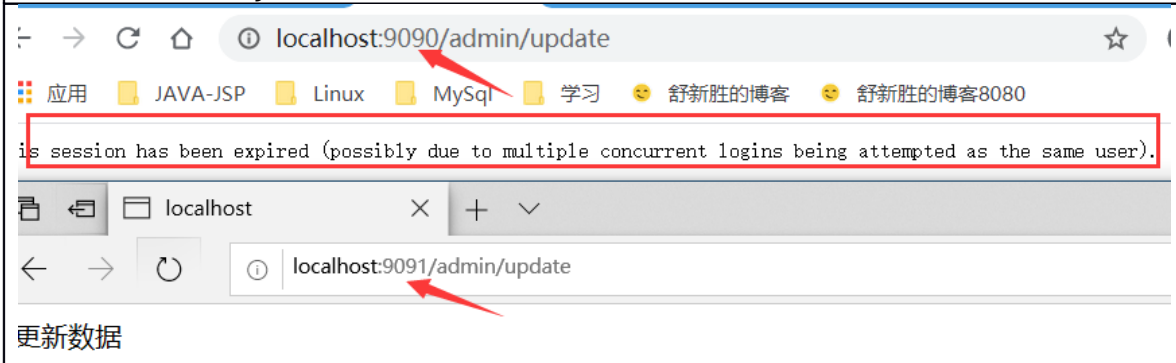
4 开启 spring-session

```
@EnableWebSecurity
//开启 redis session
@EnableRedisHttpSession
public class MySecurity extends WebSecurityConfigurerAdapter {
    //定义一个加密器和 4.x 不太一样 需要手动定义加密器
    //明文加密器 ,4.X 默认就是这个 5.x 的 security 没有手动定义会报错
    @Bean
    public PasswordEncoder myEncoder(){
      /*  return NoOpPasswordEncoder.getInstance();*/
       /* return new BCryptPasswordEncoder();*/
        return new MyPasswordEncoder();
    }
    @Autowired
    private FindByIndexNameSessionRepository sessionRepository;
    @Bean
    public SpringSessionBackedSessionRegistry sessionRegistry(){
        return new SpringSessionBackedSessionRegistry(sessionRepository);
    }
```

```
    //定义几个内存的user对象
    @Bean
    public UserDetailsService initMyUserDetailsService(){
        return new MyUserDetailService();
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
        auth.userDetailsService(initMyUserDetailsService());
/*        //admin 管理员用户 密码123456 权限：增删查改所有权限
       auth.inMemoryAuthentication()
               .withUser("admin")
               .password("123456")
               .authorities("write","read","update","delete");
       //user 密码123456 权限 查
        auth.inMemoryAuthentication()
               .withUser("user")
               .password("123456")
               .authorities("read");*/
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //所有请求到当前系统都需要经过认证授权逻辑
        http.authorizeRequests()
                //当请求地址以/user/开始时，用户权限必须有read
                .antMatchers("/user/**")
                .hasAuthority("read")

                //请求地址为"/admin/write"时，用户权限必须有write
                .antMatchers("/admin/write")
                .hasAuthority("write")

                //请求地址为"/admin/update"时，用户权限必须有update
                .antMatchers("/admin/update")
                .hasAuthority("update")

                //请求地址为"/admin/delete"时，用户权限必须有delete
                .antMatchers("/admin/delete")
                .hasAuthority("delete")
                //其他的资源请求，只要登录就能访问
                .anyRequest().authenticated();
        //要求以表单填写用户名密码为认证入口
        http.formLogin();

        http.sessionManagement()
                //会话并发管理 表示同时只能一人登录同一账号
                .maximumSessions(1)
                //spring session
                .sessionRegistry(sessionRegistry());
    }
}
```

5 测试分布式登录顶替功能

开启二个 security 应用，测试同一账号登录，另一账号被顶替掉并出现提示



spring security 整合 spring session 除了能解决集群共享 session 问题,还可以处理授权逻辑中多个 security 框架工程之间共享认证关系.