原创

springBoot整合shiro

yushiwh 关注

2018-06-13 15:38:39 18202人阅读 0人评论

依赖包

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.3.2</version>
</dependency>
```

数据库表

一切从简，用户 user 表，以及角色 role 表

id	username	password	role
1	howie	123456	user
2	swit	1234567	user

id	role
1	user

Shiro 相关类

Shiro 配置类

```
@Configuration
public  class ShiroConfig {
    @Bean
    public ShiroFilterFactoryBean shirFilter(SecurityManager securityManager) {
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
        // 必须设置 SecurityManager
        shiroFilterFactoryBean.setSecurityManager(securityManager);
        // setLoginUrl 如果不设置值，默认会自动寻找Web工程根目录下的"/login.jsp"页面 或 "/login" 映射
        shiroFilterFactoryBean.setLoginUrl("/notLogin");
        // 设置无权限时跳转的 url;
        shiroFilterFactoryBean.setUnauthorizedUrl("/notRole");
```

```
        //游客，开发权限
        filterChainDefinitionMap.put("/guest/**", "anon");
        //用户，需要角色权限 “user”
        filterChainDefinitionMap.put("/user/**", "roles[user]");
        //管理员，需要角色权限 “admin”
        filterChainDefinitionMap.put("/admin/**", "roles[admin]");
        //开放登陆接口
        filterChainDefinitionMap.put("/login", "anon");
        //其余接口一律拦截
        //主要这行代码必须放在所有权限设置的最后，不然会导致所有 url 都被拦截
        filterChainDefinitionMap.put("/**", "authc");

        shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);
        System.out.println("Shiro拦截器工厂类注入成功");
        return shiroFilterFactoryBean;
    }

    /**
     * 注入 securityManager
     */
    @Bean
    public SecurityManager securityManager() {
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        // 设置realm.
        securityManager.setRealm(customRealm());
        return securityManager;
    }

    /**
     * 自定义身份认证 realm;
     * <p>
     * 必须写这个类，并加上 @Bean 注解，目的是注入 CustomRealm，
     * 否则会影响 CustomRealm类 中其他类的依赖注入
     */
    @Bean
    public CustomRealm customRealm() {
        return new CustomRealm();
    }
}
```

注意：里面的 SecurityManager 类导入的应该是 `import org.apache.shiro.mgt.SecurityManager;` 但是，如果你是复制代码过来的话，会默认导入 `java.lang.SecurityManager` 这里也稍稍有点坑，其他的类的话，也是都属于 shiro 包里面的类

shirFilter 方法中主要是设置了一些重要的跳转 url，比如未登陆时，无权限时的跳转；以及设置了各类 url 的权限拦截，比如 /user 开始的 url 需要 user 权限，/admin 开始的 url 需要 admin 权限等

权限拦截 Filter

当运行一个Web应用程序时，Shiro将会创建一些有用的默认 Filter 实例，并自动地将它们置为可用，而这些默认的 Filter 实例是被 DefaultFilter 枚举类定义的，当然我们也可以自定义 Filter 实例，这些在以后的文章中会讲到

```
public enum DefaultFilter {

    anon(AnonymousFilter.class),
    authc(FormAuthenticationFilter.class),
    authcBasic(BasicHttpAuthenticationFilter.class),
    logout(LogoutFilter.class),
    noSessionCreation(NoSessionCreationFilter.class),
    perms(PermissionsAuthorizationFilter.class),
    port(PortFilter.class),
    rest(HttpMethodPermissionFilter.class),
    roles(RolesAuthorizationFilter.class),
    ssl(SslFilter.class),
    user(UserFilter.class);                    @51CTO博客
```

Filter	解释
anon	无参，开放权限，可以理解为匿名用户或游客
authc	无参，需要认证
logout	无参，注销，执行后会直接跳转到 `shiroFilterFactoryBean.setLoginUrl();` 设置的 url
authcBasic	无参，表示 httpBasic 认证
user	无参，表示必须存在用户，当登入操作时不做检查
ssl	无参，表示安全的URL请求，协议为 https
perms[user]	参数可写多个，表示需要某个或某些权限才能通过，多个参数时写 perms["user, admin"]，当有多个参数时必须每个参数都通过才算通过
roles[admin]	参数可写多个，表示是某个或某些角色才能通过，多个参数时写 roles["admin，user"]，当有多个参数时必须每个参数都通过才算通过
rest[user]	根据请求的方法，相当于 perms[user:method]，其中 method 为 post，get，delete 等
port[8081]	当请求的URL端口不是8081时，跳转到schemal://serverName:8081?queryString 其中 schmal 是协议 http 或 https 等等，serverName 是你访问的 Host，8081 是 Port 端口，queryString 是你访问的 URL 里的？后面的参数

常用的主要就是 anon，authc，user，roles，perms 等

注意：anon, authc, authcBasic, user 是第一组认证过滤器，perms, port, rest, roles, ssl 是第二组授权过滤器，要通过授权过滤器，就先要完成登陆认证操作（即先要完成认证才能前去寻找授权) 才能走第二组授权器（例如访问需要 roles 权限的 url，如果还没有登陆的话，会直接跳转到 `shiroFilterFactoryBean.setLoginUrl();` 设置的 url ）

自定义 realm 类

我们首先要继承 AuthorizingRealm 类来自定义我们自己的 realm 以进行我们自定义的身份，权限认证操作。记得要 Override 重写 doGetAuthenticationInfo 和 doGetAuthorizationInfo 两个方法（两个方法名很相似，不要搞错）

```
public  class CustomRealm extends AuthorizingRealm {
    private UserMapper userMapper;

    @Autowired
    private void setUserMapper(UserMapper userMapper) {
        this.userMapper = userMapper;
```

```
     * 获取身份验证信息
     * Shiro中，最终是通过 Realm 来获取应用程序中的用户、角色及权限信息的。
     *
     * @param authenticationToken 用户身份信息 token
     * @return 返回封装了用户信息的 AuthenticationInfo 实例
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws Aut
        System.out.println("————身份认证方法————");
        UsernamePasswordToken token = (UsernamePasswordToken) authenticationToken;
        // 从数据库获取对应用户名密码的用户
        String password = userMapper.getPassword(token.getUsername());
        if (null == password) {
            throw new AccountException("用户名不正确");
        } else if (!password.equals(new String((char[]) token.getCredentials()))) {
            throw new AccountException("密码不正确");
        }
        return new SimpleAuthenticationInfo(token.getPrincipal(), password, getName());
    }

    /**
     * 获取授权信息
     *
     * @param principalCollection
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        System.out.println("————权限认证————");
        String username = (String) SecurityUtils.getSubject().getPrincipal();
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
        //获得该用户角色
        String role = userMapper.getRole(username);
        Set<String> set = new HashSet<>();
        //需要将 role 封装到 Set 作为 info.setRoles() 的参数
        set.add(role);
        //设置该用户拥有的角色
        info.setRoles(set);
        return info;
    }
}
```

重写的两个方法分别是实现身份认证以及权限认证，shiro 中有个作登陆操作的 `Subject.login()` 方法，当我们把封装了用户名，密码的 token 作为参数传入，便会跑进这两个方法里面（不一定两个方法都会进入）

其中 doGetAuthorizationInfo 方法只有在需要权限认证时才会进去，比如前面配置类中配置了 `filterChainDefinitionMap.put("/admin/**", "roles[admin]");` 的管理员角色，这时进入 /admin 时就会进入 doGetAuthorizationInfo 方法来检查权限；而 doGetAuthenticationInfo 方法则是需要身份认证时（比如前面的 `Subject.login()` 方法）才会进入

再说下 UsernamePasswordToken 类，我们可以从该对象拿到登陆时的用户名和密码（登陆时会使用 `new UsernamePasswordToken(username, password);` ），而 get 用户名或密码有以下几个方法

```
token.getUsername()  //获得用户名 String
token.getPrincipal() //获得用户名 Object
token.getPassword()  //获得密码 char[]
token.getCredentials() //获得密码 Object
```

注意：有很多人会发现，UserMapper 等类，接口无法通过 @Autowired 注入进来，跑程序的时候会报 NullPointerException，网上说了很多诸如是 Spring 加载顺序等原因，但其实有一个很重要的地方要大家注意，CustomRealm 这个类是在 shiro 配置类的 `securityManager.setRealm()` 方法中设置进去的，而很多人直接写 `securityManager.setRealm(new CustomRealm());` ,这样是不行的，必须要使用 @Bean 注入 MyRealm，不能直接 new 对象：

```
@Bean
public SecurityManager securityManager() {
    DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
    // 设置realm.
    securityManager.setRealm(customRealm());
    return securityManager;
}

@Bean
public CustomRealm customRealm() {
    return new CustomRealm();
}
```

道理也很简单，和 Controller 中调用 Service 一样，都是 SpringBean，不能自己 new

当然，同样的道理也可以这样写：

```
@Bean
public SecurityManager securityManager(CustomRealm customRealm) {
    DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
    // 设置realm.
    securityManager.setRealm(customRealm);
    return securityManager;
}
```

然后只要在 CustomRealm 类加上个类似 @Component 的注解即可

功能实现

本文的功能全部以接口返回 json 数据的方式实现

根据 url 权限分配 controller

```
游客
@RestController
@RequestMapping("/guest")
public class GuestController{
    @Autowired
    private final ResultMap resultMap;

    @RequestMapping(value = "/enter", method = RequestMethod.GET)
    public ResultMap login() {
        return resultMap.success().message("欢迎进入，您的身份是游客");
    }

    @RequestMapping(value = "/getMessage", method = RequestMethod.GET)
    public ResultMap submitLogin() {
        return resultMap.success().message("您拥有获得该接口的信息的权限！");
```

普通登陆用户

```
@RestController
@RequestMapping("/user")
public class UserController{
    @Autowired
    private final ResultMap resultMap;

    @RequestMapping(value = "/getMessage", method = RequestMethod.GET)
    public ResultMap getMessage() {
        return resultMap.success().message("您拥有用户权限，可以获得该接口的信息！");
    }
}
```

管理员

```
@RestController
@RequestMapping("/admin")
public class AdminController {
    @Autowired
    private final ResultMap resultMap;

    @RequestMapping(value = "/getMessage", method = RequestMethod.GET)
    public ResultMap getMessage() {
        return resultMap.success().message("您拥有管理员权限，可以获得该接口的信息！");
    }
}
```

突然注意到 CustomRealm 类那里抛出了 AccountException 异常，现在建个类进行异常捕获

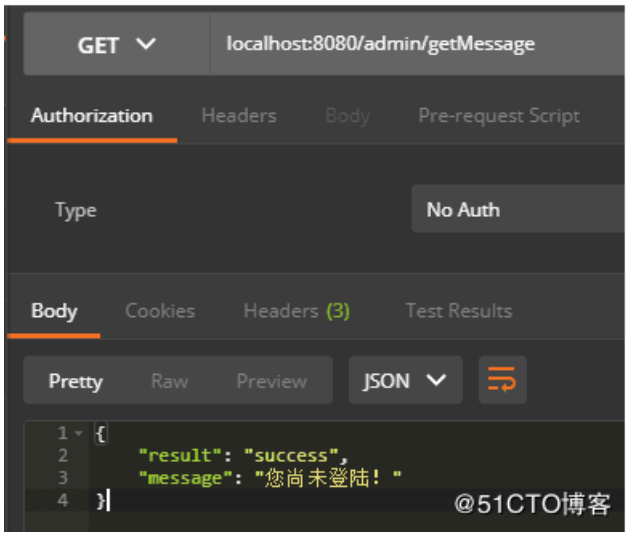
```
@RestControllerAdvice
public class ExceptionController {
    private final ResultMap resultMap;

    @Autowired
    public ExceptionController(ResultMap resultMap) {
        this.resultMap = resultMap;
    }

    // 捕捉 CustomRealm 抛出的异常
    @ExceptionHandler(AccountException.class)
    public ResultMap handleShiroException(Exception ex) {
        return resultMap.fail().message(ex.getMessage());
    }
}
```

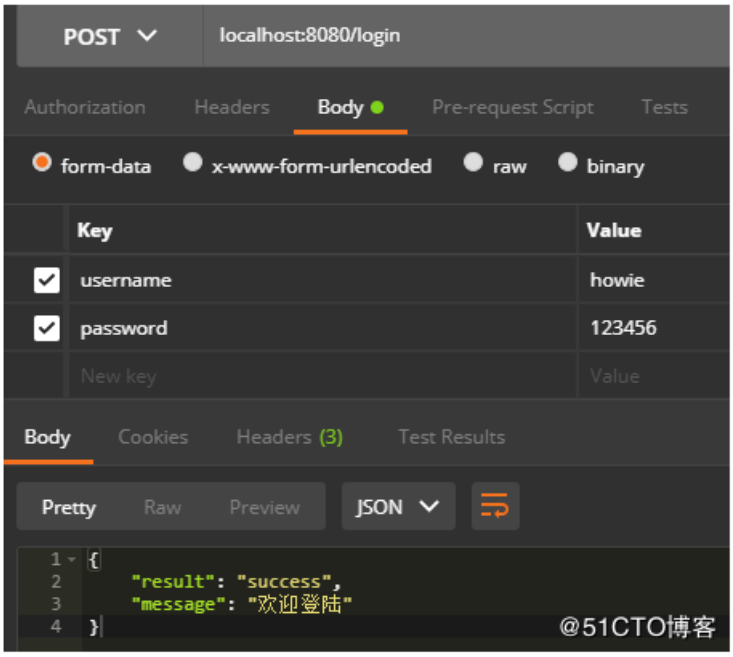
还有进行登陆等处理的 LoginController

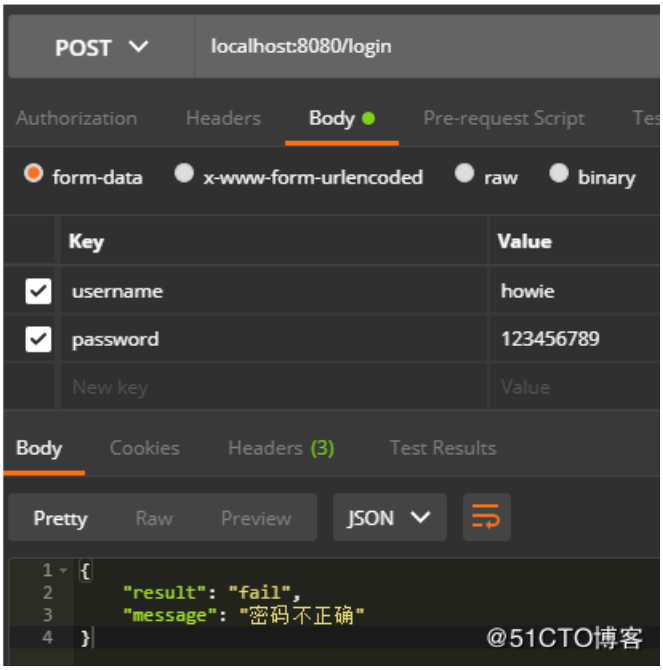
```
@RestController
public class LoginController {
    @Autowired
    private ResultMap resultMap;
    private UserMapper userMapper;

    @RequestMapping(value = "/notLogin", method = RequestMethod.GET)
    public ResultMap notLogin() {
        return resultMap.success().message("您尚未登陆！");
    }
```

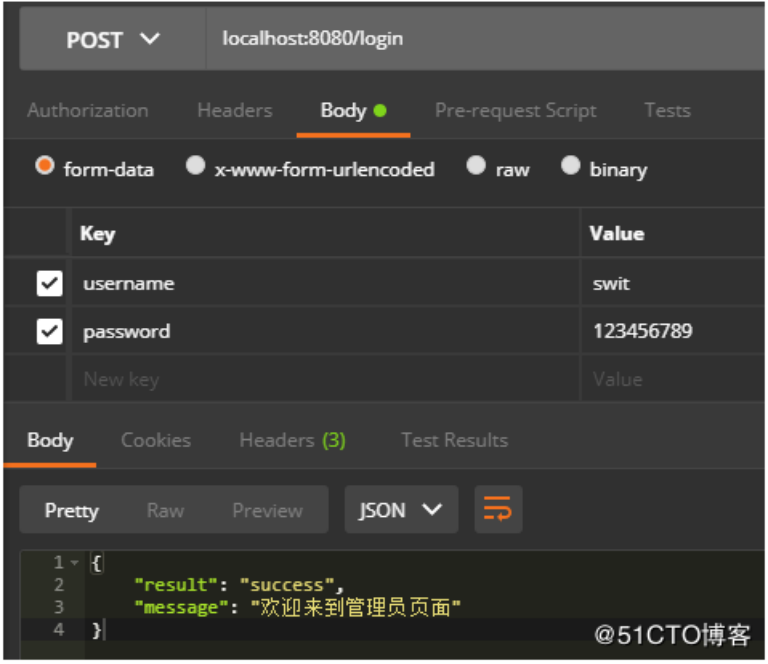
```
    }

    @RequestMapping(value = "/logout", method = RequestMethod.GET)
    public ResultMap logout() {
        Subject subject = SecurityUtils.getSubject();
        //注销
        subject.logout();
        return resultMap.success().message("成功注销！");
    }

    /**
     * 登陆
     *
     * @param username 用户名
     * @param password 密码
     */
    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public ResultMap login(String username, String password) {
        // 从SecurityUtils里边创建一个 subject
        Subject subject = SecurityUtils.getSubject();
        // 在认证提交前准备 token（令牌）
        UsernamePasswordToken token = new UsernamePasswordToken(username, password);
        // 执行认证登陆
        subject.login(token);
        //根据权限，指定返回数据
        String role = userMapper.getRole(username);
        if ("user".equals(role)) {
            return resultMap.success().message("欢迎登陆");
        }
        if ("admin".equals(role)) {
            return resultMap.success().message("欢迎来到管理员页面");
        }
        return resultMap.fail().message("权限错误！");
    }
}
```

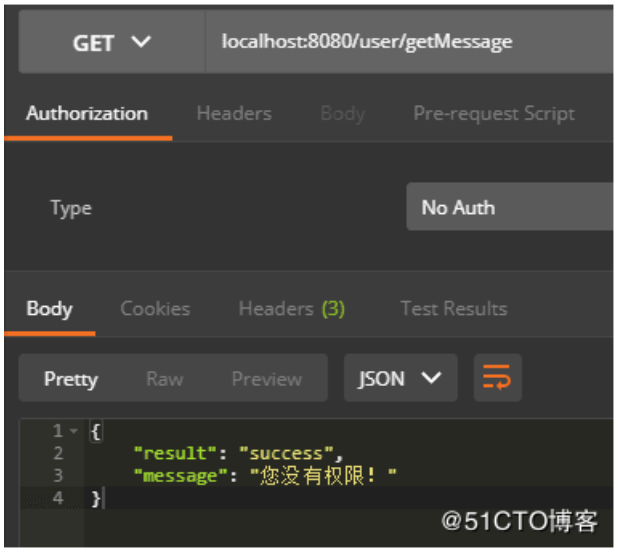
测试

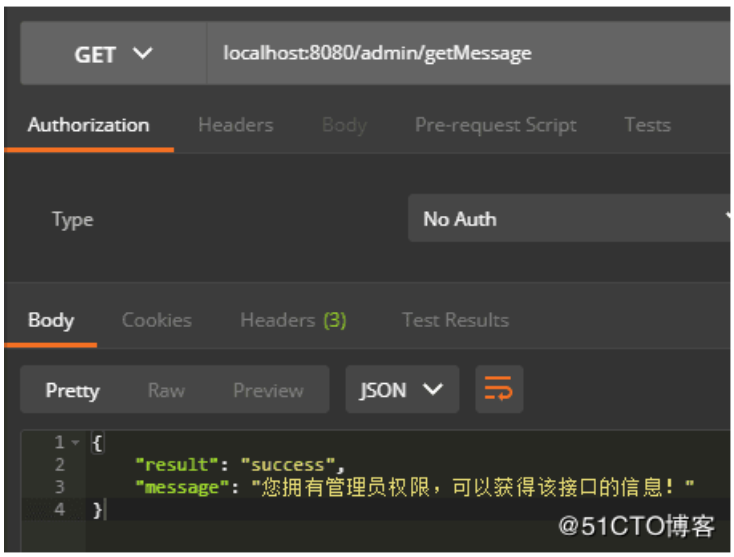


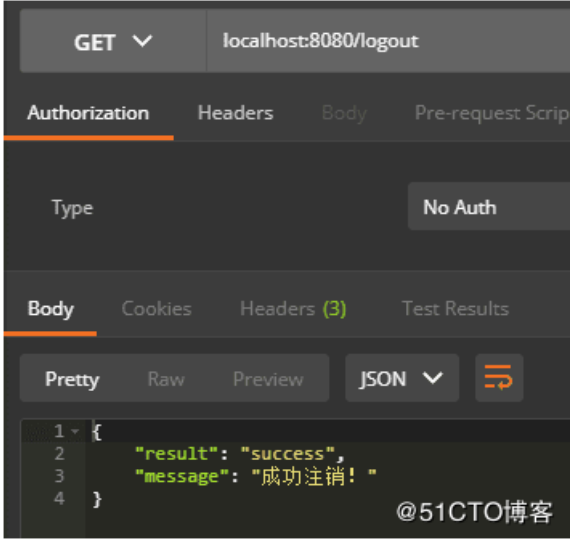

POST
localhost:8080/login
Authorization
Headers
Body
Pre-request Script
Tests
form-data
x-www-form-urlencoded
raw
binary
Key
Value
username
howie
password
123456
New key
Value
Body
Cookies
Headers (3)
Test Results
Pretty
Raw
Preview
JSON
{
"result": "success",
"message": "欢迎登陆"
}
@51CTO博客


POST
localhost:8080/login
Authorization
Headers
Body
Pre-request Script
form-data
x-www-form-urlencoded
raw
binary
Key
Value
username
howie
password
123456789
New key
Value
Body
Cookies
Headers (3)
Test Results
Pretty
Raw
Preview
JSON
{
"result": "fail",
"message": "密码不正确"
}
@51CTO博客


POST
localhost:8080/login
Authorization
Headers
Body
Pre-request Script
Tests
form-data
x-www-form-urlencoded
raw
binary
Key
Value
username
swit
password
123456789
New key
Value
Body
Cookies
Headers (3)
Test Results
Pretty
Raw
Preview
JSON
{
"result": "success",
"message": "欢迎来到管理员页面"
}
@51CTO博客


GET
localhost:8080/user/getMessage
Authorization
Headers
Body
Pre-request Script
Type
No Auth
Body
Cookies
Headers (3)
Test Results
Pretty
Raw
Preview
JSON
{
"result": "success",
"message": "您没有权限！"
}
@51CTO博客

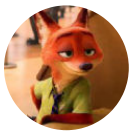

GET
localhost:8080/admin/getMessage
Authorization
Headers
Body
Pre-request Script
Tests
Type
No Auth
Body
Cookies
Headers (3)
Test Results
Pretty
Raw
Preview
JSON
{
"result": "success",
"message": "您拥有管理员权限，可以获得该接口的信息！"
}
@51CTO博客



shiro　　整合　　spring boot

0　　收藏　　分享

上一篇：linux下RocketMQ的安...　　下一篇：springboot+shiro...



yushiwh
55篇文章，29W+人气，4粉丝
程序员队伍中打杂、救火、背锅专业户

关注

提问和评论都可以，用心的回复会被更多人看到和认可

Ctrl+Enter 发布　　取消　　发布

在线客服

推荐专栏　　更多

1　　分享　　yushiwh　　关注

带你玩转高可用

前百度高级工程师的架构高可用实战

共15章 | 曹林华

订 阅

¥51.00 487人订阅

微服务技术架构和大数据治理实战

大数据时代的微服务之路

共18章 | 纯洁微笑

订 阅

¥51.00 681人订阅

猜你喜欢

java实现线索化二叉树的前序、中序、后续的遍历（完…

spring 通过注解实现工具类injection Service方法

php类库到sublime完成

敏捷开发思想及Scrum实践

敏捷开发&传统开发

《高效程序员的45个习惯：敏捷开发修炼之道》摘记一

JAVA伴我行——项目篇（一）：开发模型，敏捷开发和…

软件开发模式对比(瀑布、迭代、螺旋、敏捷)

关于敏捷开发的26个心得

敏捷开发简单思路

SpringBoot注解大全

spring boot拦截器WebMvcConfigurerAdapter，以及高…

Ansible基于服务树进行分组全量接口调用

开发自己的分布式监控Prometheus Exporter时遇到的坑

敏捷开发中产品负责人与团队如何协作

敏捷开发之Scrum扫盲篇

手机软件开发管理过程中，如何采用敏捷开发模式

敏捷开发的文档

敏捷开发：如何通过回顾保持学习状态

《高效程序员的45个习惯 敏捷开发修炼之道》 - 书摘精要

在线客服