# 概念

## Java异常类

首先从继承方面来说，Error和Exception都是Throwable的子类。所以从根本上你可以理解他们是差不多的，程序出了问题就抛出。然后一般意义来讲，区别就是Error比Exception更严重，不易通过后续代码处理和修复，不应该被代码catch到的错误。而Exception是指严重程度较轻的程序问题，相较于Error更容易被后续代码处理和修复的，有些需要通过try catch代码处理，有些不建议通过try catch代码处理。  
知识延伸：  
Exception分两种，checked和unchecked。checked Exception就是在写代码的时候，IDE(比如Eclipse)会要求你写try catch的那种Exception，比如IOException。这种Exception是Java的设计者要求你的程序去处理的。这种异常一般不会影响程序的主体，容易手动诊断修复，所以Java要求你在catch下面写出处理的代码，以保证程序遇到此类exception之后还可以正常运行。unchecked这一类就是你在代码处理了checked exception之后，你在运行时候依然会遇到的exception，所以又叫做RunTimeException，比如NullPointerException, IndexOutOfBoundsException。此类exception相较于前面那种更容易影响程序运行，从设计者角度不提倡从程序中catch出来并处理，当然你也可以这么做。  
  
Error就是程序运行时候抛出的最严重级别的错误了，如VirtualMachineError，ThreadDeath。抛出了Error的程序从Java设计的角度来讲，程序基本不可以通过后续代码修复，从而理应终止。当然，从语法上来讲，所有这些都可以被写进catch里面，但是Error因为上述原因，不应该被代码处理。

## Spring

Spring的两大特性IOC（DI）、Aop。

### 注解

1. Qualifier： qualifier的意思是合格者，通过这个标示，表明了哪个实现类才是我们所需要的。它之前仍然需要注解Autowired

<https://www.cnblogs.com/smileLuckBoy/p/5801678.html>

### 测试类注解

1. Runwith:

@RunWith就是一个运行器

@RunWith(JUnit4.class)就是指用JUnit4来运行

@RunWith(SpringJUnit4ClassRunner.class),让测试运行于Spring测试环境

@RunWith(Suite.class)的话就是一套测试集合，

@ContextConfiguration Spring整合JUnit4测试时，使用注解引入多个配置文件

单个文件

@ContextConfiguration(Locations="classpath：applicationContext.xml")

@ContextConfiguration(classes = SimpleConfiguration.class)

多个文件时，可用{}

@ContextConfiguration(locations = { "classpath:spring1.xml", "classpath:spring2.xml" })

https://blog.csdn.net/weixin\_41315469/article/details/79675044

### spring junit测试用例

https://blog.csdn.net/gongzi2311/article/details/70613438?utm\_source=itdadao&utm\_medium=referral

### spring事务

<https://www.cnblogs.com/yepei/p/4716112.html>

spring支持编程式事务管理和声明式事务管理两种方式。

## springboot

### 概念

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Spring Boot致力于在蓬勃发展的快速应用开发领域（rapid application development）成为领导者。

### 使用

#### Eclipse 搭建springboot

1. Eclipse集成sts（spring tool suit）插件

## 数据库连接池

### dbcp

### c3p0

### druid

## 事务

事务是指作为单个逻辑工作单元执行的一系列[操作](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C/33052)，要么完全地执行，要么完全地不执行。

### 事务的特性

1. 原子性（Atomicity）：事务是一个原子操作，由一系列动作组成。事务的原子性确保动作要么全部完成，要么完全不起作用。
2. 一致性（Consistency）：一旦事务完成（不管成功还是失败），系统必须确保它所建模的业务处于一致的状态，而不会是部分完成部分失败。在现实中的数据不应该被破坏。
3. 隔离性（Isolation）：可能有许多事务会同时处理相同的数据，因此每个事务都应该与其他事务隔离开来，防止数据损坏。
4. 持久性（Durability）：一旦事务完成，无论发生什么系统错误，它的结果都不应该受到影响，这样就能从任何系统崩溃中恢复过来。通常情况下，事务的结果被写到持久化存储器中。

### 事务与spring结合

Spring不负责管理事务，但提供了各种事务管理器，为jdbc和heibernate等不同平台提供了对应的事务管理器。Spring中事务的配置方式有如下两种：

#### 编程式事务

当系统需要明确的，细粒度的控制各个事务的边界，应选择编程式事务。

#### 声明式事务

当系统对于事务的控制粒度较粗时，应该选择申明式事务，通过<tx>标签和<aop>切面形式在xml中进行配置。可通过spring提供的jdbc或heibernate事务管理器进行管理。

##### Jdbc事务

如果应用程序中直接使用JDBC来进行持久化，DataSourceTransactionManager会为你处理事务边界。为了使用DataSourceTransactionManager，你需要使用如下的XML将其装配到应用程序的上下文定义中：

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource" />

</bean>

##### Hibernate事务

如果应用程序的持久化是通过Hibernate实习的，那么你需要使用HibernateTransactionManager。对于Hibernate3，需要在Spring上下文定义中添加如下的<bean>声明：

<bean id="transactionManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

##### Spring中事务的传播行为

传播行为有七种最常见为以下两种其它请自行百度：https://blog.csdn.net/mini\_jike/article/details/46275079

PROPAGATION\_REQUIRED：支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。如：增删改

PROPAGATION\_SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行。如：查看

##### Spring中事务的隔离级别：

https://blog.csdn.net/zht741322694/article/details/78676964

isolation\_default使用数据库默认的事务隔离级别

isolation\_read\_uncommitted允许读取尚未提交的修改，可能导致脏读、幻读和不可重复读

isolation\_read\_committed 允许从已经提交的事务读取，可防止脏读、但幻读，不可重复读仍然有可能发生

isolation\_repeatable\_read 对相同字段的多次读取的结果是一致的，除非数据被当前事务自生修改。可防止脏读和不可重复读，但幻读仍有可能发生

isolation\_serializable完全服从acid隔离原则，确保不发生脏读、不可重复读、和幻读，但执行效率最低。

##### Spring中<tx:method/>配置解析

| **属性** | **是否需要？** | **默认值** | **描述** |
| --- | --- | --- | --- |
| name | 是 |  | 与事务属性关联的方法名。通配符（\*）可以用来指定一批关联到相同的事务属性的方法。如：'get\*'、'handle\*'、'on\*Event'等等。 |
| propagation | 不 | REQUIRED | 事务传播行为 |
| isolation | 不 | DEFAULT | 事务隔离级别 |
| timeout | 不 | -1 | 事务超时的时间（以秒为单位） |
| read-only | 不 | false | 事务是否只读？ |
| rollback-for | 不 |  | 将被触发进行回滚的 Exception(s)；以逗号分开。 如：'com.foo.MyBusinessException,ServletException' |
| no-rollback-for | 不 |  | *不* 被触发进行回滚的 Exception(s)；以逗号分开。 如：'com.foo.MyBusinessException,ServletException' |

##### 利用aop将配置的事务进行切入

<!--1.声明式 配置事务管理器:注意这里的dataSource和SqlSessionFactoryBean的dataSource要一致，不然事务就没有作用了 -->

<!-- 配置事务的传播特性 -->

<tx:advice id=*"userTxAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<tx:method name=*"delete\*"* propagation=*"REQUIRED"* read-only=*"false"*

rollback-for=*"java.lang.Exception"* no-rollback-for=*"java.lang.RuntimeException"* />

<tx:method name=*"insert\*"* propagation=*"REQUIRED"* read-only=*"false"*

rollback-for=*"java.lang.RuntimeException"* />

<tx:method name=*"update\*"* propagation=*"REQUIRED"* read-only=*"false"*

rollback-for=*"java.lang.Exception"* />

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* />

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* />

<tx:method name=*"select\*"* propagation=*"SUPPORTS"* />

<tx:method name=*"batch\*"* propagation=*"REQUIRED"* read-only=*"false"* />

</tx:attributes>

</tx:advice>

<!-- 切面配置 -->

<aop:config>

<aop:pointcut id=*"transactionPointcut"*

expression=*"execution(\* com.future.service.\*.impl.\*.\*(..))"* />

<aop:advisor pointcut-ref=*"transactionPointcut"*

advice-ref=*"userTxAdvice"* />

</aop:config>

<!--2.注解式事务配置,即哪些方法需要事务就在那个方法上写一个@Transactional注解）-->

<!-- 开启事务注解，标注@Transactional的类和方法将具有事务性 -->

<tx:annotation-driven transaction-manager=*"transactionManager"* />

## Shiro

### 概念

<https://www.w3cschool.cn/shiro/>

<https://www.cnblogs.com/tohxyblog/p/6870261.html>

https://blog.csdn.net/oppoppoppo/article/details/53432202

shiro缓存和会话的区别： **shiro每个授权都会通过realm获取权限信息，为了提高访问速度需要添加缓存，第一次从realm中读取权限数据，之后不再读取。**

缓存失效逻辑：**如果用户正常退出 缓存清空。如果用户非正常退出比如关闭浏览器 缓存清空。如果session到期，缓存清空，设置一般在缓存的配置文件中。如果修改了用户的权限，而用户不退出系统，修改的权限不会生效，需要手动调用realm的clearCache方法清除。**

### Shiro与spring集成

<https://www.cnblogs.com/nuccch/p/6790408.html>

#### 操作步骤

1. 添加pom依赖，例如：

shiro-spring/shiro-core/shiro-web/shiro-aspectj/shiro-ehcache/slf4j-api/slf4j-log4j12/log4j【shiro使用slf4j作为日志框架，所以必需配置slf4j，同时，使用log4j作为底层的日志实现框架。】

1. 集成shiro到spring-mvc的框架中

Web.xml配置shiroFilter，例如：

<filter>

<filter-name>shiroFilter</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

<init-param>

<param-name>targetFilterLifecycle</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>shiroFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

1. 配置shiro-ehcache缓存配置文件，如下：

<?xml version="1.0" encoding="UTF-8"?>

<ehcache name="shirocache">

<diskStore path="java.io.tmpdir"/>

<!-- 登录记录缓存 锁定10分钟 -->

<cache name="passwordRetryCache"

maxEntriesLocalHeap="2000"

eternal="false"

timeToIdleSeconds="3600"

timeToLiveSeconds="0"

overflowToDisk="false"

statistics="true">

</cache>

<cache name="authorizationCache"

maxEntriesLocalHeap="2000"

eternal="false"

timeToIdleSeconds="3600"

timeToLiveSeconds="0"

overflowToDisk="false"

statistics="true">

</cache>

<cache name="authenticationCache"

maxEntriesLocalHeap="2000"

eternal="false"

timeToIdleSeconds="3600"

timeToLiveSeconds="0"

overflowToDisk="false"

statistics="true">

</cache>

<cache name="shiro-activeSessionCache"

maxEntriesLocalHeap="2000"

eternal="false"

timeToIdleSeconds="3600"

timeToLiveSeconds="0"

overflowToDisk="false"

statistics="true">

</cache>

</ehcache>

1. 创建类去从新实现角色和权限的控制，例如：

public class AnyPermissionsAuthorizationFilter extends PermissionsAuthorizationFilter{

@Override

public boolean isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue)

throws IOException

{

Subject subject = getSubject(request, response);

String[] perms = (String[])mappedValue;

if (perms == null || perms.length == 0) {

return true;

}

for (String perm : perms) {

if (subject.isPermitted(perm)) {

return true;

}

}

return false;

}

}

public class AnyRolesAuthorizationFilter extends RolesAuthorizationFilter {

@Override

public boolean isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue)

throws IOException {

final Subject subject = getSubject(request, response);

final String[] rolesArray = (String[]) mappedValue;

if (rolesArray == null || rolesArray.length == 0) {

return true;

}

for (String roleName : rolesArray) {

if (subject.hasRole(roleName)) {

return true;

}

}

return false;

}

}

1. 使用spring IOC容器托管shiro，如下：(集成了shiro及cas)

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:util="http://www.springframework.org/schema/util"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="

<http://www.springframework.org/schema/beans> http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="com.shiro.dao,com.shiro.cas" />

<!-- 缓存管理器 -->

<bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">

<property name="cacheManagerConfigFile" value="classpath:ehcache.xml"/>

</bean>

<!-- 配置为基于表单认证的过滤器 -->

<bean id="formAuthenticationFilter" class="org.apache.shiro.web.filter.authc.FormAuthenticationFilter">

</bean>

<!-- 重新封装角色过滤器，由and关系转为or -->

<!-- <bean id="anyRolesAuthorizationFilter" class="com.future.shiro.realm.AnyRolesAuthorizationFilter"/> -->

<!-- 重新封装权限过滤器，由and关系转为or -->

<bean id="anyPermissionsAuthorizationFilter" class="com.future.shiro.realm.AnyPermissionsAuthorizationFilter"/>

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

<property name="securityManager" ref="securityManager"/>

<property name="loginUrl" value="http://casServer /cas-server-web/login?service=http://project/index.jsp"/>

<property name="unauthorizedUrl" value="/unauthorized.jsp"/>

<!--

登录 CAS 服务端地址,服务端的返回地址 -->

<property name="filters">

<util:map>

<!-- 添加casFilter到shiroFilter -->

<entry key="casFilter" value-ref="casFilter"/>

<entry key="logout" value-ref="logoutFilter"/>

<entry key="casLogout" value-ref="casLogoutFilter" />

<entry key="authc" value-ref="formAuthenticationFilter"/>

<!-- <entry key="roles" value-ref="anyRolesAuthorizationFilter" /> --> <!-- 重写roles方法，只要满足其中一个即可 -->

<entry key="perms" value-ref="anyPermissionsAuthorizationFilter"></entry> <!-- 重写perms方法，只要满足其中一个即可 -->

</util:map>

</property>

<property name="filterChainDefinitions">

<value>

<!-- Anon：不指定过滤器,这个过滤器是空的，什么都没做，跟没有一样。 Authc:验证，这些页面必须验证后才能访问，也就是我们说的登录后才能访问。

.jsp\*:后面的\*代表 url的参数，比如 index.jsp?id=123 perms[SECURITY\_ACCOUNT\_VIEW] 表示用户需要提供值为“SECURITY\_ACCOUNT\_VIEW”Permission

信息。 /login.do\* = anon -->

/spring/systemMessage/\*\* = anon

/spring/serviceCatalogApply/newServiceCatalogFeedback = anon

/unauthorized.jsp = anon

/main.jsp = casLogout,casFilter

/logout = logout

<!--验证所有请求，如果shiro中不存在用户信息，则返回到loginUrl的登录页面 -->

/\*\* =casLogout,user

/images/\*\* = anon

/css/\*\* = anon

/js/\*\* = anon

<!-- / = authc,roles[admin,paas,daas,iaas,itsm,itsec] --> <!-- 根据角色匹配 -->

/ = authc,perms[审计管理] <!-- 根据权限匹配 -->

<!-- /\*\* = authc -->

</value>

</property>

</bean>

<!-- CAS Filter 校验失败跳转的页面 -->

<bean id="casFilter" class="org.apache.shiro.cas.CasFilter">

<property name="failureUrl" value="/unauthorized.jsp"/>

</bean>

<bean id="casLogoutFilter" class="com.future.cas.session.CasLogoutFilter">

<property name="sessionManager" ref="sessionManager"/>

</bean>

<bean id="logoutFilter" class="org.apache.shiro.web.filter.authc.LogoutFilter">

<property name="redirectUrl" value="http:// casServer /cas-server-web/logout?service=http://project/"/>

</bean>

<!-- Realm实现 -->

<bean id="userRealm" class="com.shiro.cas.MyCasRealm">

<property name="cachingEnabled" value="true"/>

<property name="authenticationCachingEnabled" value="true"/>

<property name="authenticationCacheName" value="authenticationCache"/>

<property name="authorizationCachingEnabled" value="true"/>

<property name="authorizationCacheName" value="authorizationCache"/>

<property name="casServerUrlPrefix" value="http://ip/cas-server-web"/>

<property name="casService" value="http://project/index.jsp"/>

</bean>

<!-- 配置SecurityManager的管理 -->

<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

<!-- 配置你需要使用的Realms -->

<property name="realm" ref="userRealm"/>

<!-- 定义要使用的session管理器 -->

<!-- <property name="sessionManager" ref="sessionManager"/> -->

<property name="cacheManager" ref="cacheManager"/>

<property name="subjectFactory" ref="casSubjectFactory"/>

</bean>

<bean id="casSubjectFactory" class="org.apache.shiro.cas.CasSubjectFactory"/>

<!-- 会话ID生成器 -->

<bean id="sessionIdGenerator" class="org.apache.shiro.session.mgt.eis.JavaUuidSessionIdGenerator"/>

<!-- 会话Cookie模板 -->

<bean id="sessionIdCookie" class="org.apache.shiro.web.servlet.SimpleCookie">

<!-- 在Tomcat运行下默认使用的Cookie的名字为JSESSIONID -->

<constructor-arg value="sid"/>

<!-- 保证该系统不会受到跨域的脚本操作供给 -->

<property name="httpOnly" value="true"/>

<!-- 定义Cookie的过期时间，单位为秒，如果设置为-1表示浏览器关闭，则Cookie消失 -->

<property name="maxAge" value="-1"/>

</bean>

<!-- 会话DAO -->

<bean id="sessionDAO" class="org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO">

<!-- 设置session缓存的名字，这个名字可以任意 -->

<property name="activeSessionsCacheName" value="shiro-activeSessionCache"/>

<!-- 定义该Session DAO操作中所使用的ID生成器 -->

<property name="sessionIdGenerator" ref="sessionIdGenerator"/>

</bean>

<!-- 会话管理器 -->

<bean id="sessionManager" class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">

<!-- 定义的是全局的session会话超时时间，此操作会覆盖web.xml文件中的超时时间配置 -->

<property name="globalSessionTimeout" value="1800000"/>

<!-- 删除所有无效的Session对象，此时的session被保存在了内存里面 -->

<property name="deleteInvalidSessions" value="true"/>

<!-- 定义要使用的无效的Session定时调度器 -->

<property name="sessionValidationScheduler" ref="sessionValidationScheduler"/>

<!-- 需要让此session可以使用该定时调度器进行检测 -->

<property name="sessionValidationSchedulerEnabled" value="true"/>

<!-- 定义Session可以进行序列化的工具类 -->

<property name="sessionDAO" ref="sessionDAO"/>

<!-- 所有的session一定要将id设置到Cookie之中，需要提供有Cookie的操作模版 -->

<property name="sessionIdCookie" ref="sessionIdCookie"/>

<!-- 定义sessionIdCookie模版可以进行操作的启用 -->

<property name="sessionIdCookieEnabled" value="true"/>

</bean>

<!-- 配置session的定时验证检测程序类，以让无效的session释放 -->

<bean id="sessionValidationScheduler" class="org.apache.shiro.session.mgt.quartz.QuartzSessionValidationScheduler">

<!-- 设置session的失效扫描间隔，单位为毫秒 -->

<property name="sessionValidationInterval" value="1800000"/>

<!-- 随后还需要定义有一个会话管理器的程序类的引用 -->

<property name="sessionManager" ref="sessionManager"/>

</bean>

<!-- Shiro生命周期处理器-->

<bean id="lifecycleBeanPostProcessor" class="org.apache.shiro.spring.LifecycleBeanPostProcessor"/>

<!-- 相当于调用SecurityUtils.setSecurityManager(securityManager) -->

<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">

<property name="staticMethod" value="org.apache.shiro.SecurityUtils.setSecurityManager"/>

<property name="arguments" ref="securityManager"/>

</bean>

<!-- <context:property-placeholder location="classpath:conf.properties"/> -->

</beans>

1. 登录、调用，例如：

//获取当前登录名

PrincipalCollection principalCollection = SecurityUtils.getSubject().getPrincipals();

String loginname = (String) principalCollection.getPrimaryPrincipal();

#### shiro配置文件详解

<https://blog.csdn.net/baicp3/article/details/45077331>

##### securityManagerx【Shiro的安全管理器】

<!-- shiro安全管理器 设置cacheManage,下列属性有实现CacheManagerAware接口的,都会自动注入缓存管理器-->

<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

<property name="cacheManager" ref="cacheManager" />

<property name="realm" ref="monitorRealm" />

<property name="sessionManager" ref="shiroSessionManager" />

<property name="rememberMeManager" ref="rememberMeManager"/>

</bean>

sessionManager再去管理sessionDAO会话DAO 和sessionIdCookie会话ID生成器和sessionValidationScheduler会话验证调度器。cacheManager通过使用Ehcache实现，Realm通过自己自定义或者其他方式的权限存储来实现，比如登录等.

##### loginURL

没有登录的用户请求需要登录的页面时自动跳转到登录页面，不是必须的属性，不输入地址的话会自动寻找项目web项目的根目录下的”/login.jsp”页面。

##### successUrl

登录成功默认跳转页面，不配置则跳转至”/”。如果登陆前点击的一个需要登录的页面，则在登录自动跳转到那个需要登录的页面。不跳转到此。

##### unauthorizedUrl

没有权限默认跳转的页面。

##### filterChainDefinitions

anon:例子/admins/\*\*=anon 没有参数，表示可以匿名使用。

authc:例如/admins/user/\*\*=authc表示需要认证(登录)才能使用，没有参数，必须是登录之后才能进行访问，不包括remember me。

roles：例子/admins/user/\*\*=roles[admin],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，当有多个参数时，例如admins/user/\*\*=roles["admin,guest"],每个参数通过才算通过，相当于hasAllRoles()方法。默认的判断是&，需要重写过滤器用来继承AuthorizationFilter了，将里面的判断方法由并且转换成或者的关系。

perms：例子/admins/user/\*\*=perms[user:add:\*],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，例如/admins/user/\*\*=perms["user:add:\*,user:modify:\*"]，当有多个参数时必须每个参数都通过才通过，想当于isPermitedAll()方法。

rest：例子/admins/user/\*\*=rest[user],根据请求的方法，相当于/admins/user/\*\*=perms[user:method] ,其中method为post，get，delete等。

port：例子/admins/user/\*\*=port[8081],当请求的url的端口不是8081是跳转到schemal://serverName:8081?queryString,其中schmal是协议http或https等，serverName是你访问的host,8081是url配置里port的端口，queryString

是你访问的url里的？后面的参数。

authcBasic：例如/admins/user/\*\*=authcBasic没有参数表示httpBasic认证

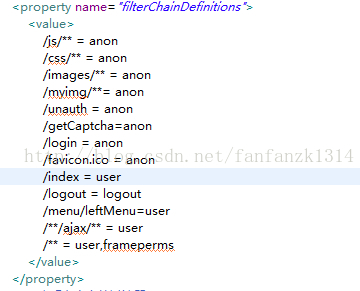
ssl:例子/admins/user/\*\*=ssl没有参数，表示安全的url请求，协议为https

user:例如/admins/user/\*\*=user没有参数表示必须存在用户，当登入操作时不做检查，user：登录用户才可以访问，包含remember me。

注：anon，authcBasic，auchc，user是认证过滤器，

perms，roles，ssl，rest，port是授权过滤器

例如（https://blog.csdn.net/fanfanzk1314/article/details/72780923）：



拦截器的优先级：从上到下，从左到右，如果有匹配的拦截器就会阻断并返回，例如：访问js/a.js,第一个拦截器anon符合，就返回true了，不在往下进行匹配了。

注意最后一个拦截最后一句是 /\*\*=user,frameperms 意思就是除了上面的那些,其他的所有都要经过 。user和frameperms.如果没有登陆 user就会阻断,不会执行到frameperms.frameperms 就是我们自定义实现的过滤器,从数据库中查询用户的权限,判断当前用户是否有权限访问拦截的url。

#### 角色判断方法重写，由并且的关系转换成或者的判断

以角色为例重写对应方法，将角色的判断由并且的关系转换成或者的关系，如下：

1. 重写类

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import org.apache.shiro.subject.Subject;

import org.apache.shiro.web.filter.authz.AuthorizationFilter;

public class CustomRolesAuthorizationFilter extends AuthorizationFilter {

@Override

protected boolean isAccessAllowed(ServletRequest req,ServletResponse resp, Object mappedValue) throws Exception {

Subject subject = getSubject(req, resp);

String[] rolesArray = (String[]) mappedValue;

if (rolesArray == null || rolesArray.length == 0) { //没有角色限制，有权限访问

return true;

}

for (int i = 0; i < rolesArray.length; i++) {

if (subject.hasRole(rolesArray[i])) { //若当前用户是rolesArray中的任何一个，则有权限访问

return true;

}

}

return false;

}

1. 配置重新的类

方法重写以后，通过IOC容器托管创建该类，如：

<bean id="anyRoles" class="com.ssxt.ycfxf.host.filter.CustomRolesAuthorizationFilter" />

1. 将类配置到shiro的过滤器中，如：

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

<property name="filterChainDefinitions">

<value>

<!-- 权限拦截 -->

/userRole/\*\*=anyRoles[admin,isShopMan]

<!-- 域名访问，未登录跳转登陆页面，已登陆根据角色跳转对应主页面 -->

/=examine

</value>

</property>

</bean>

### shiro与cas与spring的集成

https://www.cnblogs.com/nightThink/p/5169845.html

### 缓存

shiro的缓存和会话session的区别：<http://syq891015.iteye.com/blog/2357769>

#### shiro缓存的实现

[Shiro缓存使用Redis、Ehcache、自带的MpCache实现的三种方式实例](https://www.cnblogs.com/zfding/p/8536480.html)：https://www.cnblogs.com/zfding/p/8536480.html

##### ehcache

将数据存放在磁盘或内存中，使用时需要配置xml文件

##### redis

将数据存放在redis数据库，在利用IOC创建对象时，需指定数据库的连接信息

##### mpcache

此缓存为shiro内置缓存方式，数据存放在内容中，使用非常简单。

<!-- 配置shiro自带的缓存管理器 -->

<bean id = "shiroSpringCacheManager"class="org.apache.shiro.cache.MemoryConstrainedCacheManager"/>

### 会话session

## Dubbo

### 概念

<https://blog.csdn.net/abcde474524573/article/details/53026110>

### 服务提供者配置

<!-- 提供方应用信息，用于计算依赖关系 -->

<dubbo:application name="hello-world-app" />

<!-- 使用zookeeper注册中心暴露服务地址 -->

<dubbo:registry address="zookeeper://224.5.6.7:1234" />

<!-- 用dubbo协议在20880端口暴露服务 这个端口可用于点对点调试-->

<dubbo:protocol name="dubbo" port="20880" />

<!-- 声明需要暴露的服务接口 -->

<dubbo:service interface="com.alibaba.dubbo.demo.DemoService" ref="demoService" />

<!-- 和本地bean一样实现服务 -->

<bean id="demoService" class="com.alibaba.dubbo.demo.provider.DemoServiceImpl" />

### 服务消费者配置

<!-- 消费方应用名，用于计算依赖关系，不是匹配条件，不要与提供方一样 -->

<dubbo:application name="consumer-of-helloworld-app" />

<!-- 使用zookeeper注册中心暴露发现服务地址 -->

<dubbo:registry address="zookeeper://224.5.6.7:1234" />

<!-- 生成远程服务代理，可以和本地bean一样使用demoService -->

<dubbo:reference id="demoService" interface="com.alibaba.dubbo.demo.DemoService" />

## Webservice

### 概念

webService主要用于向其他系统提供接口以便调用，系统间可能开发语言等完全不同，根据约定的接口规范，调用者传递相关参数进行接口调用，服务方根据传入的条件进行业务处理并进行结果返回。

<http://www.cnblogs.com/xiaocai0923/archive/2011/12/30/2308154.html>

https://www.cnblogs.com/gissuifeng/p/3726900.html

### 服务器端的开发

<https://www.cnblogs.com/xiaochangwei/p/4969448.html>

<https://www.cnblogs.com/frankliiu-java/articles/1641949.html>

[https://blog.csdn.net/zl834205311/article/details/51612207 4](https://blog.csdn.net/zl834205311/article/details/51612207%204)种不同的外放方式

开发步骤：

1. 创建webproject
2. 创建class即准备外放的提供服务的webservice接口。该类需添加注解（@webService） 表示该类被发布成了一个ws。该注解由jdk进行提供。
3. 在ws类中创建方法。方法上添加注解(@webMethod(exclude=true))表示此方法不会被发布。其次使用static修饰的静态方法也不会被发布。final方法也不会被发布。
4. 调用main方法外放对应的webservice接口。需要外放的方法创建成功以后，使用Endpoint （Endpoint – 此类为端点服务类，该类也由jdk进行提供）将ws类中的指定方法进行发布。例如：

Endpoint.publish(“url”,new WSObject());

参数1：为服务器端ip，若设置的不是接口提供者的ip，而是其他服务器ip时，接口的外放正常进行。但是使用wsdl进行校验时，则由于访问的ip并不是接口提供者的ip则会因为访问不到服务而空白无法正常显示xml格式文件。Ip必须是服务提供的ip，服务名称可以随便写。

参数2：外放的类，该类为接口的实现类

1. 发布成功以后，在浏览器输入对应网址进行校验，即外放的url后面加 ?wsdl （wsdl一种网络服务描述语言）

### 客户端调用

<https://blog.csdn.net/woshishabiaaaaa/article/details/24868323>

## Restful

### 概念

REST是一种轻量级的Web Service架构风格。REST是一种轻量级的Web Service架构风格，其实现和操作比SOAP和XML-RPC更为简洁，可以完全通过HTTP协议实现，还可以利用缓存Cache来提高响应速度，性能、效率和易用性上都优于SOAP协议。REST架构对资源的操作包括获取、创建、修改和删除资源的操作正好对应HTTP协议提供的GET、POST、PUT和DELETE方法，这种针对网络应用的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。REST架构尤其适用于完全无状态的CRUD（Create、Read、Update、Delete，创建、读取、更新、删除）操作。

REST是一种架构风格，其核心是面向资源，REST专门针对网络应用设计和开发方式，以降低开发的复杂性，提高系统的可伸缩性。REST提出设计概念和准则为：

1. 网络上的所有事物都可以被抽象为资源(resource)
2. 每一个资源都有唯一的资源标识(resource identifier)，对资源的操作不会改变这些标识，其核心操作只有GET,PUT,POST,DELETE。
3. 所有的操作都是无状态的

由此通过资源统一标识符和操作符即可确定用户的请求。

## Soap

### 概念

基于xml的简单对象访问协议。它是轻型协议，用于分散的、分布式计算环境中交换信息。SOAP有助于以独立于平台的方式访问对象、服务和服务器。它借助于XML，提供了HTTP所需的扩展。

SOAP协议可以简单地理解为：SOAP=RPC+HTTP+XML，即采用HTTP作为通信协议，RPC（Remote Procedure Call Protocol - 远程过程调用协议）作为一致性的调用途径，XML作为数据传送的格式，从而允许服务提供者和服务客户经过防火墙在Internet上进行通信交互。

### SOAP协议规范

第一部分：SOAP封装（Envelop）定义了一个的框架（描述消息的内容多少、谁发送、谁应当接受、处理，以及如何处理它们）。

第二部分：SOAP编码规则（Encoding Rules）定义了可选数据编码规则，用于表示应用程序定义的数据类型和直接图表，以及一个用于序列化非语法数据模型统一标准。

第三部分：SOAP RPC表示（RPC Representation）定义一个远程调用风格（请求/响应）信息交换的模式。

第四部分：SOAP绑定（Binding）定义了SOAP和HTTP之间的绑定和使用底层协议的交换。

SOAP协议可以简单地理解为：SOAP=RPC+HTTP+XML，即采用HTTP作为通信协议，RPC（Remote Procedure Call Protocol - 远程过程调用协议）作为一致性的调用途径，XML作为数据传送的格式，从而允许服务提供者和服务客户经过防火墙在Internet上进行通信交互。

### Soap方式调用webservice接口

http://lbinzhang-126-com.iteye.com/blog/2215368

## http

### 概念

http:超文本传输协议。简单、快速、灵活、无状态、无连接。

http是标准超文本传输协议。使用对参数进行编码并将参数作为键值对传递，还使用关联的请求语义。每个协议都包含一系列HTTP请求标头及其他一些信息，定义客户端向服务器请求哪些内容，服务器用一系列HTTP响应标头和所请求的数据进行响应。HTTP-GET 使用 MIME 类型application/x-www-form-urlencoded（将追加到处理请求的服务器的 URL 中）以 URL 编码文本的形式传递其参数。 URL 编码是一种字符编码形式，可确保传递的参数中包含一致性文本，例如将空格编码为 %20，其它符号转换为%XX,其中XX为该符号以16进制表示的ASCII（或ISOLatin-1）值。 追加的参数也称为查询字符串;HTTP-POST参数也是 URL 编码的，但是，键/值对是在实际的 HTTP 请求消息内部传递的，而不是作为 URL 的一部分进行传递。

  HTTP请求是客户端往服务端发送请求动作，告知服务器自己的要求。HTTP请求由状态行、请求头、请求正文三部分组成：

1. 状态行：包括请求方式Method、资源路径URL、协议版本Version；
2. 请求头：包括一些访问的域名、用户代理、Cookie等信息；
3. 请求正文：就是HTTP请求的数据。

备注：请求方式Method一般有GET、POST、PUT、DELETE，含义分别是获取、修改、上传、删除，其中GET方式仅仅为获取服务器资源，方式较为简单，因此在请求方式为GET的HTTP请求数据中，请求正文部分可以省略，直接将想要获取的资源添加到URL中。下图所示就是GET的请求，没有请求正文。

### 调用

url:统一资源定位符。  
     组成部分：协议名://主机名(主机ip):端口号/项目资源地址?传递参数的键值对#锚点  
eg: http://192.168.5.151:8080/js/index.php?name=zhangsan#top  
     ① ip地址在同一网段是唯一的。如果是在公网中，整个公网的i是唯一的。  
     ② 端口号默认是:80，可以省略不写。常见的端口号: 80 Apache 8080 tomcat  
     ③ 本机ip：localhost 或者 127.0.0.1

### http响应状态码

 200---OK/请求已经正常处理完毕

 301---/请求永久重定向

 302---/请求临时重定向

 304---/请求被重定向到客户端本地缓存

 400---/客户端请求存在语法错误

  401---/客户端请求没有经过授权

  403---/客户端的请求被服务器拒绝，一般为客户端没有访问权限

  404---/客户端请求的URL在服务端不存在

  500---/服务端永久错误

  503---/服务端发生临时错误

### HTTP响应模型

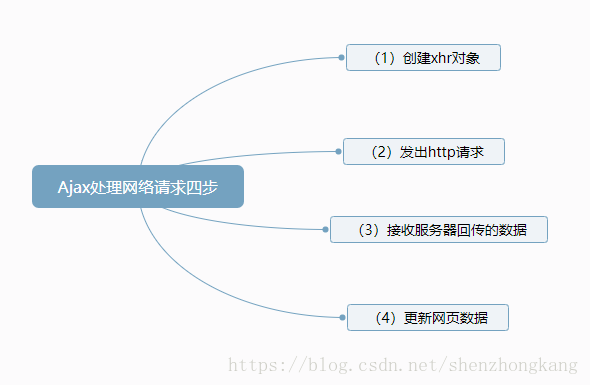
服务器收到HTTP请求之后，会有多种方法响应这个请求，下面是HTTP响应的四种模型：

1. 单进程I/O模型：服务端开启一个进程，一个进程仅能处理一个请求，并且对请求顺序处理；
2. 多进程I/O模型：服务端并行开启多个进程，同样的一个进程只能处理一个请求，这样服务端就可以同时处理多个请求；
3. 复用I/O模型：服务端开启一个进程，但是呢，同时开启多个线程，一个线程响应一个请求，同样可以达到同时处理多个请求，线程间并发执行；
4. 复用多线程I/O模型：服务端并行开启多个进程，同时每个进程开启多个线程，这样服务端可以同时处理进程数M\*每个进程的线程数N个请求。

## Ajax

### 概念

Ajax的请求方式为http协议。



## Mysql

### Varchar

Mysql中utf-8编码汉字一个汉字占三个字节，gbk编码汉字一个汉字占用两个字节。英文永远占用一个字节。存储长度可变。

4.0版本以下，varchar(20)，指的是20**字节**，如果存放UTF8汉字时，只能存6个（每个汉字3字节）   
5.0版本以上，varchar(20)，指的是20**字符**，无论存放的是数字、字母还是UTF8汉字（每个汉字3字节），都可以存放20个，最大大小是65532字节 。

### Char

存储长度固定。

### Varchar和char的区别

char与varchar的区别：

1. char (16)长度固定， 如'www.uphtm.com' 存储需要空间 14个字符。varchar(16) 可变长 如'www.uphtm.com' 需要存储空间 15字符。从上面可以看得出来char 长度是固定的，不管你存储的数据是多少他都会都固定的长度。而varchar则处可变长度但他要在总长度上加1字符，这个用来存储位置。所以实际应用中用户可以根据自己的数据类型来做。
2. 由于某种原因char 固定长度，所以在处理速度上要比varchar快速很多，但是对费存储空间，所以对存储不大，但在速度上有要求的可以使用char类型，反之可以用varchar类型来实例。