# 一、Spring基础

## 1.Spring介绍

### 1.1.面向对象编程

面向对象开发就是不断的创建对象，使用对象，指挥对象做事情

面向对象设计就是在管理和维护对象之间的关系

### 1.2.Spring框架解决什么问题

让程序员专注业务代码，高效编程

解决企业应用开发的复杂性而创建的

### 1.3.Spring框架核心

Spring Boot, Spring Cloud, Spring Cloud Data Flow

开发思路：reactive stack；servlet stack

## 2.Spring IOC

### 2.1.什么是IOC

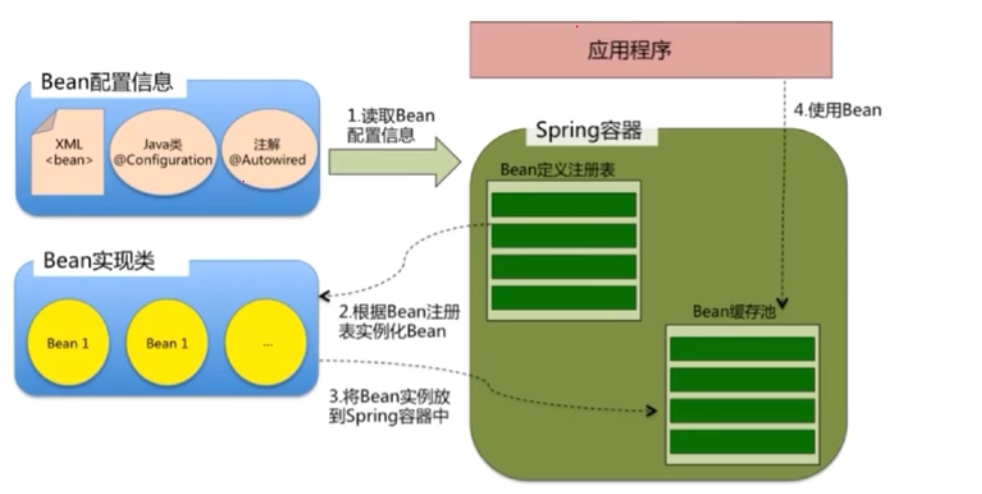
IOC-Inversion of Control，即“控制反转”，是一种设计思想。将设计好的对象交给容器控制，而不是传统的在对象内部直接（new）控制。

### 2.2.依赖注入（DI）

使用依赖注入实现控制反转，即上层控制下层

把底层类作为参数传递给上层类，实现上对下的控制

### 2.3.Spring IOC容器



## 3.Spring Bean

### 3.1.Spring Bean的使用方法

（1）定义Bean：

@Component Spring基础的注解，被Sring管理的组件或bean

@Repository 用于持久层，数据库访问层，是用来直接访问数据库的

@Service 用于服务层，处理业务逻辑

@Controller 用于呈现层，Spring-MVC，暴漏给前端的入口

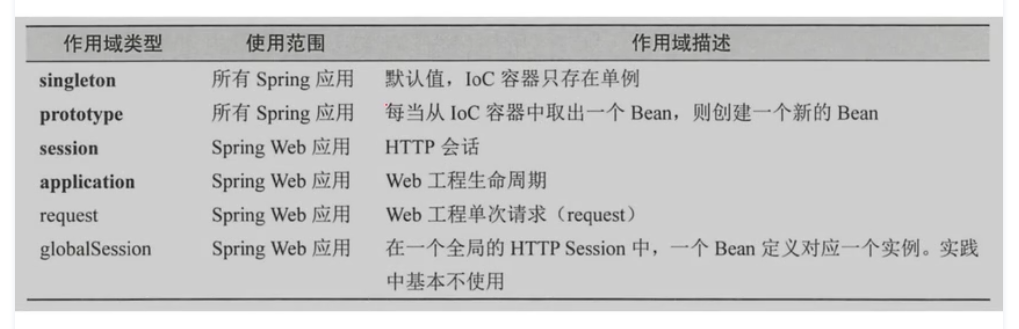
（2）注入Bean：

@Autowired 默认按照资源的类型（class）查找并注入的

@Resource 默认按照资源的名称（name）查找并注入的

@Qualifier 采用@Autwired如果存在多个实现类的情况下，@Qualifer注解通过资源名称确定唯一性

（3）Bean的作用域



### 3.2.Spring Bean的加载过程

AbstractBeanFactory

转换BeanName-从缓存中加载实例-实例化Bean-检查parentBeanFactory-初始化依赖的Bean-创建Bean

## 4.约定编程

约定编程：只要按照一定的约定规则编程之后，就可以把自己的代码织入约定的流程中。

（1）记住约定的流程是什么

（2）完成对应得编码任务

（3）不需要知道底层设计者是怎么样将约定的内容织入对应的流程

动态代理模式

调用者，代理对象（proxy），真实对象（target）

代理，可控制或者增加对目标对象的访问

通过动态代理对象织入切面

Spring AOP本质就是通过约定，把对应的方法通过动态代理织入到约定的流程中。

## 5.Spring AOP

### 5.1.Spring AOP的术语

目标对象：被代理的对象

连接点：被拦截的对象（方法）

切面：定义切点、各类通知和引入的内容。

切点：使用正则表达式和指示器规则去匹配多个类的不同方法

通知：按照约定的流程下的方法

前置通知、后置通知、环绕通知、事后返回通知、异常通知

织入：通过动态代理技术，为原有服务对象生成代理对象，然后将与切点定义匹配的连接点拦截，并按照约定将各类通知织入约定流程的过程。

### 5.2.Spring AOP的使用

@Aspect 切面

@Pointcut 切点

@Before、@After、@AfterReturning、@AfterThrowing 通知

# 二、Docker基础

## 1.Docker介绍

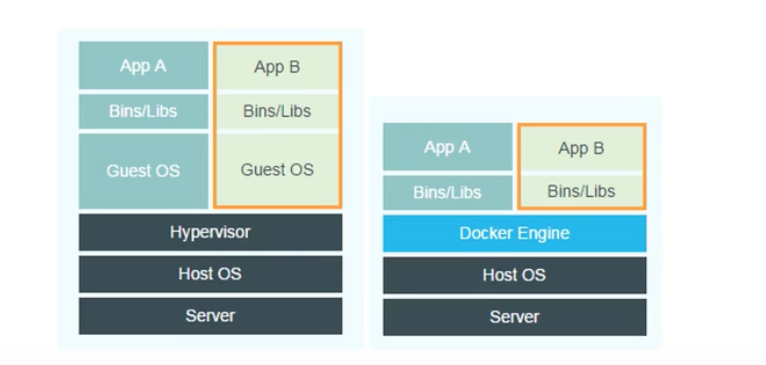
### 1.1.Docker的定义

Docker的思想来自于集装箱

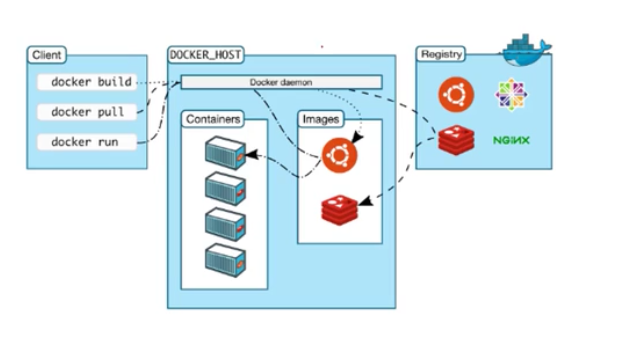
Docker就是操作系统级别的虚拟化方案

Docker是一个开源的引擎，可以轻松的为任何应用创建一个轻量级的、可移植的、自给自足的容器。

Docker和虚拟机的区别：



### 1.2.Docker的基本组成



Images：Docker镜像就是一个只读的模板，用来创建Docker容器。

Containers：Docker利用容器来运行应用，容器是从镜像创建的运行实例。

仓库是集中存放镜像文件的场所。

## 2.Docker安装

（1）安装Docker依赖的工具

yum install -y yum-utils device-mapper-persistent-data lvm2

（2）指定Docker安装库路径

yum-config-manager --add-repo https://download.dcoker.com/linux/centos/docker-ce.repo

yum-config-manager --add-repo https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/centos/docker-ce.repo

（3）安装

yum install docker-ce

（4）启动Docker

systemctl start docker

（5）测试

docker run hello-world

## 3.Docker常用命令

### 3.1.Docker镜像命令

查看本地镜像列表：docker images

查找远端仓库镜像：docker search <image\_name>

获取镜像：docker pull <域名>/<namespace>/<repo>:<tag>

查看镜像信息：docker inspect <image\_name>

删除镜像：docker rmi <image>:<tag>

创建镜像：docker commit <options> <container\_id> <repository:tag>

迁出镜像：docker save -o <image>.tar <image>:<tag>

载入镜像：docker load --input <image>.tar 或docker load <image>.tar

上传镜像：docker push <域名>/<namespace>/<repo>:<tag>

### 3.2.Docker容器命令

新建容器 docker create <image\_name>

新建并启动容器 docker run <image\_name>

守护态运行容器 docker run --name <container\_name> -d <image\_name>

终止容器：docker stop <container\_id>

查看终止的容器：docker ps -a

查看运行的容器：docker ps

重新启动容器：docker start <container\_id>

进入容器：

docker exec <options> <container\_id> <command>

docker exec -it <container\_id> bash

删除容器：

删除终止状态的容器：docker rm <container\_id>

删除正在运行的容器：docker rm -f <container\_id>

## 4.构建MySQL服务

（1）查找MySQL镜像

docker search mysql

（2）拉取MySQL镜像

docker pull mysql:5.6.35

（3）构建&启动MySQl容器

docker run --name mysqlavr -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=xxx -d mysql:5.6.35

（4）测试MySQl服务

使用navicat连接MySQL

## 5.构建Redis服务

（1）查找Redis镜像

docker search redis

（2）拉取Redis镜像

docker pull redis

（3）构建&启动Redis容器

docker run -p 6379:6379 -v $PWD/data:/data -d redis redis-server --appendonly yes

-v 将主机当前目录下data目录挂载到容器的data目录

--appendonly redis持久化配置

（4）测试Redis服务

使用Redis Destop Manager连接MySQL

# 三、Spring Boot基础

## 1.Spring Boot是什么

Spring Boot不是Spring Framwork的升级版；不是应用服务器；不是代码生成器；不是任何规范

Spring Boot是用来简化Spring项目的初始化搭建以及开发过程

### 1.1.Spring Boot的特性

创建独立运行的Sprig应用程序

直接内嵌Tomcat、Jetty等应用服务器

简化了项目的构建配置

自动配置Spring

提供生产级功能，如指标、健康检查

绝对没有代码生成并且对XML也没有配置要求

### 1.2.Spring Boot的核心

自动配置-Auto Configuration

起步依赖-Starter Denpendency

Actuator

命令行界面-Spring Boot CLI

## 2.Spring Boot启动器

### 2.1.Spring Boot版本仲裁中心

spring-boot-dependencies

### 2.2.场景启动器Starters

spring-boot-starter-web导入web模块运行所依赖的组件

Spring Boot将所有功能场景都抽取出来做成一个个Starter，只要在项目中引入这些启动器，即可具备这些功能。

## 3.Spring Boot自动配置

### 3.1.Spring Boot自动配置

基于添加的JAR依赖自动对Spring Boot应用程序进行配置

将项目包下的所有组件扫描到Spring容器；

将场景需要的所有组件注入到Spring容器，并配置好这些组件。

### 3.2.如何开启自动配置

@SpringBootApplication

@EnableAutoConfiguration

### 3.3.自动配置的实现原理

（1）将项目包下的所有组件扫描到Spring容器

@AutoConfigurationPackage注解 AutoConfigurationPackages.Registrar 自动配置包

（2）将场景需要的所有组件注入到Spring容器

@EnableAutoConfiguration注解 AutoConfigurationSelector 读取ETA-INF/spring.factories

## 4.定制动态Banner

（1）在src/main/resources路径下新建一个banner.txt文件

（2）借助第三方工具帮忙转化内容

http://www.network-science.de/ascii

（3）banner.txt文件中填好需要打印的字符串内容

## 5.日志功能

### 5.1.日志功能需求

（1）将关键数据打印在控制台

（2）将关键数据打印在文件中

（3）日志分类，根据日志不同类型打印到不同文件

（4）不同的运行环境，打印不同的日志内容

（5）日志自动压缩、自动清理

### 5.2.Spring Boot的默认日志实现

（1）日志门面-SLF4J

commons logging、jboss logging

（2）日志实现-LogBack

Log4j、Log4J2、java.util.logging

private final Logger logger = LoggerFactory.getLogger(XXX.class);

logger.info("hello");

### 5.3.Lombok安装与使用

作用：

（1）Lombok能以简单的注解形式来简化Java代码，提高开发人员的开发效率。

（2）Lombok能通过注解的方式，在编译时自动为属性生成构造器、getter/setter、equals、hashCode、toString方法。

开发步骤：

（1）idea下安装Lombok插件

setting-plugins，搜索lombok，点击安装

（2）引入lombok jar包

（3）在需要日志功能的类上添加@Slf4j注解

（4）使用log打印日志

## 6.Logback日志配置

### 6.1.Logback介绍

Logback的特点：

（1）更快的实现

（2）非常充分测试

（3）非常充分的文档

（4）自动重新加载配置文件

（5）自动压缩

（6）自动去除旧的日志文件

### 6.2.Logback日志配置

开发步骤：

（1）/resources目录新建logback-spring.xml

（2）根据规则配置logback-spring.xml

（3）/resources目录新建application.yml

（4）配置项目环境和日志目录

spring:profiles:active:dev

logging:path:F:/log

日志级别：

TRACE<DEBUG<INFO<WARN<ERROR<FATAL

常规配置：

configuration

-appender 负责写日志的组件

-logger 用来设置某一个类的日志打印级别

-root 是根logger

# 四、Spring数据操作

## 1.Spring Data数据访问

### 1.1.Spring Data是什么

Spring Data旨在统一和简化对各类型持久化存储和访问。

数据访问技术：

关系型数据库

非关系型数据库

Map-reduce框架

基于云的数据访问

### 1.2.Spring Data子项目

commons 提供共享的基础框架，适合各个子项目使用，支持跨数据库持久化

Hadoop 基于Spring的Hadoop作业配置和一个POJO编程模型的MapReduce作业

Key-Value 集成Redis和Riak，提供多个常用场景下的简单封装

Document 集成文档数据库：CouchDB和MongoDB并提供基本的配置映射和资料库支持

Graph 集成Neo4j提供强大的基于POJO的编程模型

Graph Roo AddOn - Roo support for Neo4j

JDBC Extensions 支持Oracle RAD、高级队列和高级数据类型

JPA 简化创建JPA数据访问层和跨存储的持久层功能

Mapping 基于Grails的提供对象映射框架，支持不同的数据库

Examples 实例程序、文档和图数据库

Guidance 高级文档

## 2.Mybatis核心知识

### 2.1.Mybatis数据操作流程

JDBC操作流程：

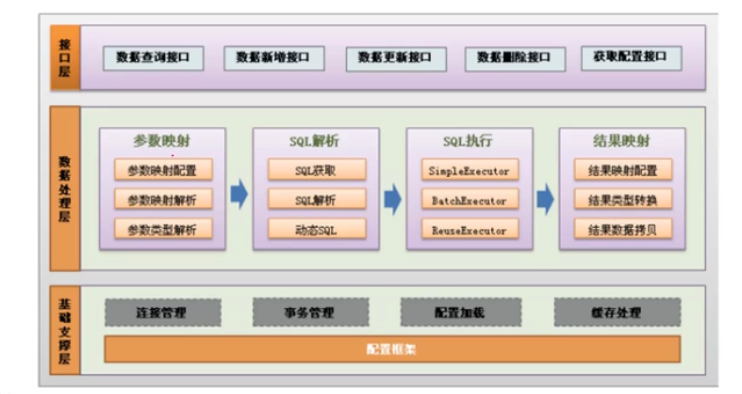
编写SQL--预编译--设置参数--执行SQL--封装结果



Mybatis是支持定制化SQL、存储过程以及高级映射的优秀的持久层框架。

Mybatis避免了几乎所有的JDBC代码和手动设置参数以及获取结果集。

### 2.2.Mybatis框架结构



## 3.Spring Boot整合Mybatis

构建步骤：

（1）引入是需要的启动器

引入Mysql和Mybatis能力

（2）配置数据源

（3）配置Mybatis

java bean、mapper

（4）编写实体类

（5）编写mapper映射接口

（6）编写测试代码

## 4.整合Druid连接池

连接池技术：

连接池技术预先建立多个数据库连接对象，然后将连接对象保存到连接池中，当客户请求到来时，从池中取出一个连接对象为客户服务，当请求完成后，客户程序调用close()方法，将连接对象放回池中。

HikariCP：速度比较快

Druid：功能比较全

## 5.使用注解方式操作数据

（1）@Mapper注解

把mapper接口交给Spring管理

不再写mapper映射文件

添加注解的接口生成一个实现类

（2）@Select注解

查询sql语句

@Insert注解

不需要返回主键

返回自增主键：@Option(useGenerateKeys=true,keyProperty="id")

返回非自增主键：@SelectKey(statement="SELECT LAST\_INSERT\_ID()",keyProperty="id",resultType=Long.class,before=false)

（3）@Delete注解

（4）@Update注解

表字段和Java属性字段映射方式

mapUnderscoreToCamelCase 配置驼峰命名

数据库字段别名 as

@Results/@ResultMap注解

动态SQL：#{}和${}

解析结果相同

预编译中处理的不一样，#是占位符，$是字符串拼接

## 6.使用配置方式操作数据

注解+xml的编码步骤：

（1）编写抽象方法

（2）配置xml扫描路径

（3）编写mapper.xml

配置Mapper自动提示：

Settings--Language&Framework--Schemas and DTDs添加：

http://mybatis.org/dtd/mybatis-3-mapper.dtd

## 7.JUnit单元测试

### 7.1.JUnit介绍

单元测试：

单元测试是测试应用程序的功能是否能够按需求正常运行

单元测试是一个对单一实体（类或方法）的测试

单元测试是软件公司提高产品质量、满足用户需求的重要环节

JUnit是一个Java语言的单元测试框架。

### 7.2.JUnit测试

单元测试用例：

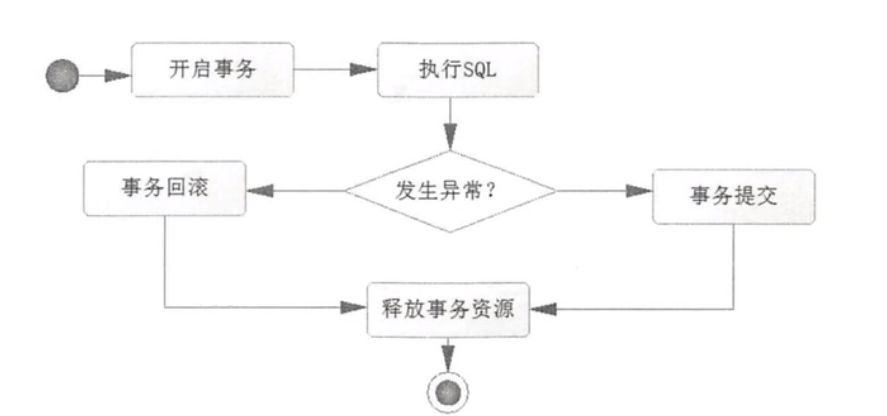
已知输入和预期输出，即在测试执行前就已知。

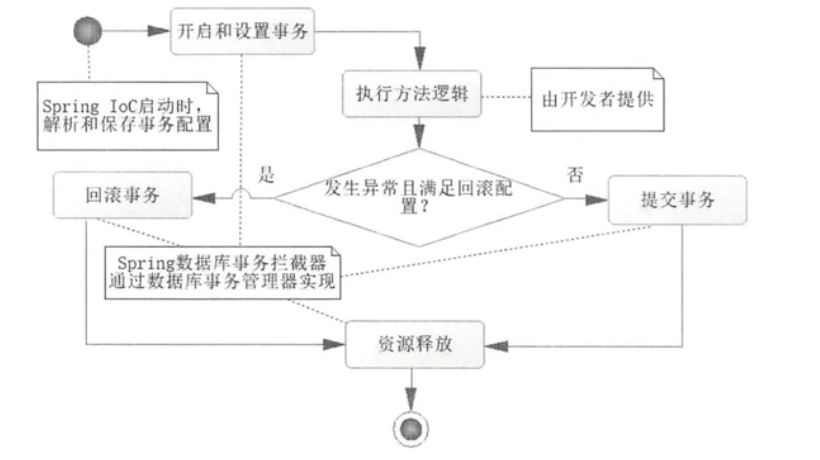
每一项需求至少需要两个单元测试用例：一个正检测，一个负检测。

## 8.Spring事务管理

### 8.1.事务流程

执行SQL的事务流程：



Spring事务约定：

### 8.2.声明式事务和编程式事务

声明式事务：@Transactional

编程式事务：TransactionTemplate

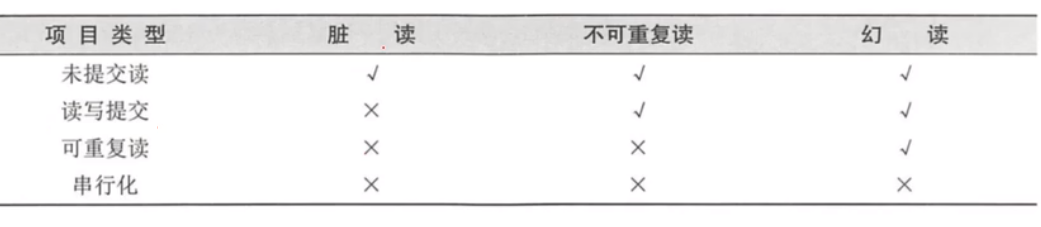
### 8.3.事务传播行为

传播行为是方法之间调用事务采取的策略问题。



PROPAGATION\_REQUIRED 默认的

### 8.4.事务隔离级别



（1）未提交读

允许一个事务读取另外一个事务没有提交的数据。

产生脏读

并发高，对数据一致型没有要求

（2）读写提交

一个事务只能读取另外一个事务已经提交的数据，不能读取未提交的数据

克服脏读，产生不可重复读

（3）可重复读

克服读写提交中出现的不可重复读现象。

产生幻读。

（4）串行化

要求所有的SQL都按照顺序执行，它能够完全保证数据一致性。

合理选择隔离级别，一般选择读写提交。

@Transactional(isolation = Isolation.READ\_COMMITTED)

### 8.5.Spring事务使用

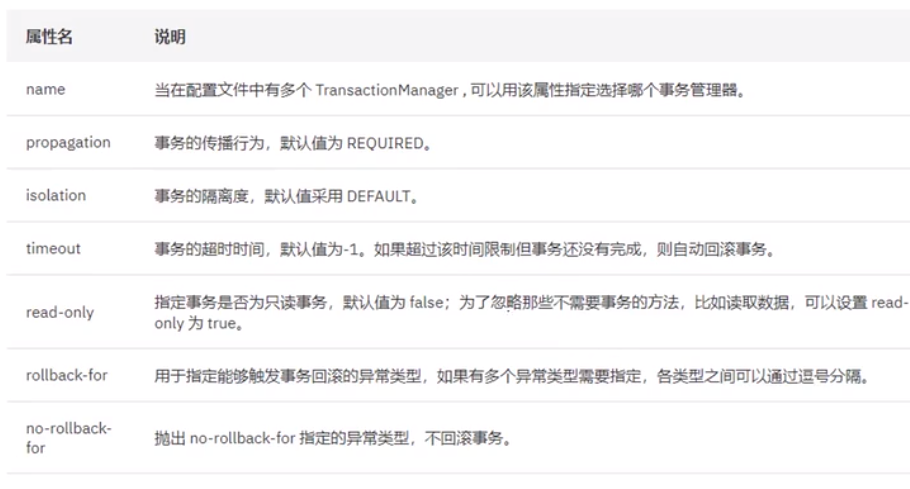
@Transactional用法

@Transactional可以用于接口、接口方法、类以及类方法上。

当作用于类上时，该类的所有public方法都具有该类型的事务属性。

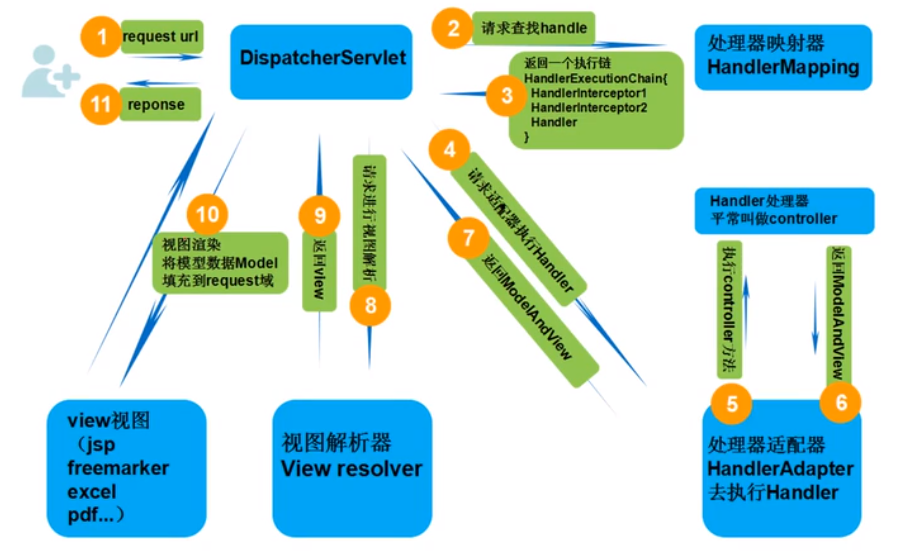
类内部方法调用本类内部的其他方法并不会引起事务行为。

@Transactional注解只能被应用到public方法上。



# 五、SpringMVC基础

## 1.SpringMVC工作流程



DispatcherServlet：中央调度器

HandlerMapping：处理器映射器

Handler：处理器

HandlerExecutionChain：执行链

HandlerAdapter：处理器适配器

ModelAndView对象

ViewResolver：视图解析器

## 2.常用注解命令

### 2.1.常用注解

@Controller/@RestController

@RequestMapping

@GetMapping/@PostMapping

@PutMapping/@DeleteMapping

@RequestBody/@ResponseBody/@ResponseStatus

### 2.2.定义映射关系

@RequestMapping

path/method 指定映射路径和方法

params/headers 参数和头

2.3.映射方法入参

（1）@PathVariable

（2）@RequestParam

（3）POJO对象绑定请求参数

### 2.4.处理模型数据

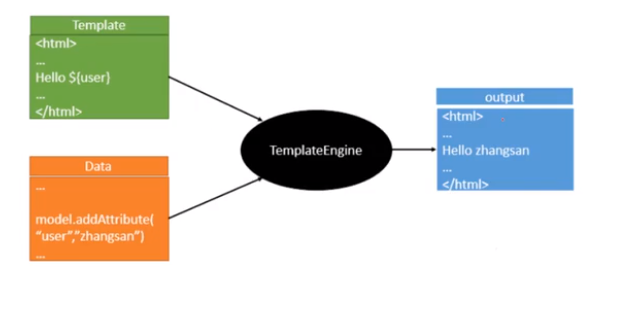
（1）ModelAndView

（2）Model和View

（3）@SessionAttributes

## 3.Thymeleaf模板引擎

### 3.1.模板引擎



### 3.2.Java模板引擎：

Thymeleaf、Freemarker、Velocity、JSP

### 3.3.SpringBoot中使用Thymeleaf

（1）引入Thymeleaf启动器

（2）渲染流程

ThymeleafProperties Thymeleaf配置

## 4.静态资源

静态资源问题

SpringBoot默认指定静态资源到static目录

修改：配置WebMvcConfigurerAdapter

# 六、开发项目

## 1.RBAC模型

### 1.1.权限管理系统定义

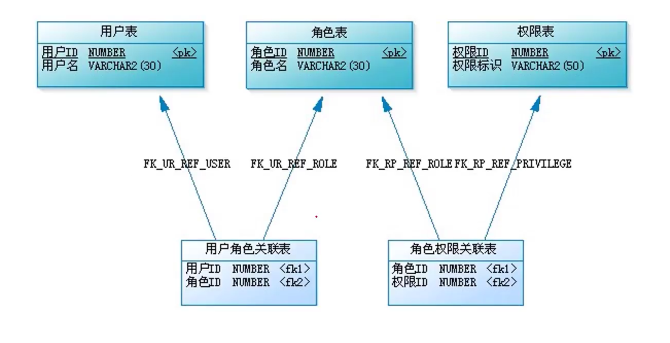
权限管理是一个几乎所有后台系统都会涉及的一个重要组成部分，主要目的是对整个后台管理系统进行权限的控制，而针对的对象是员工，避免因权限控制缺失或操作不当引发的风险问题，如操作错误，数据泄露等问题。

### 1.2.RBAC模型

RBAC（Role-Based Access Control）：用户--角色--权限

权限：页面权限、操作权限、数据权限

权限管理最终是为了风控，即哪个部门哪个用户具有哪个权限。



## 2.分页

（1）物理分页

实现原理：SELECT \* FROM xxx[WHERE...]LIMIT #{param1},#{param2}

优点：不会造成内存溢出

缺点：翻页的速度比较慢

（2）逻辑分页

实现原理：一次性将所有的数据查询出来放在内存之中，每次需要查询的时候就直接从内存之中去取出相应索引区间的数据

优点：翻页的速度比较快

缺点：可能造成内存溢出

## 3.Spring Boot热部署

提高开发者的开发效率，无需手动重启Spring Boot应用。

热部署：类的热部署、页面的热部署

（1）引入spring-boot-devtools的模块

Devtools的原理

两个ClassLoader：

ClassLoader加载不会改变的类（第三方Jar包）

ClassLoader加载会更改的类，称为restart ClassLoader

（2）idea设置

Settings--Build ... --Compiler--选中Build project automatically

Ctrl+Shift+Alt+/ -- 选择Registry--选中compiler.automake.allow.when.app.running

Run/Debug Configurations--Spring Boot--Running Application Update Policies下都选择“Update classes and resources”

（3）配置热部署目录

spring:devtools:restart:additional-paths:resources/\*\*,static/\*\*,templates/\*\*

# 七、Spring Security安全框架

主流框架：Shiro、Spring Security

## 1.Spring Security能做什么

认证（Authentication），确认用户可以访问当前系统

授权（Authorization），确定用户在当前系统下所拥有的功能权限。

## 2.Spring Security运行

（1）引入spring-boot-stater-security

（2）访问URL地址

（3）输入用户名：user，以及控制台打印的密码

## 3.Spring Security基本原理

DefaultSecurityFilterChain：

Filter顺序：

（1）ChannelProcessingFilter，如果访问的channel错了，那首先会在channel之间进行跳转，如http变为https

（2）SecurityContextPersistenceFilter，一开始运行request时就可以在SecurityContextHolder中建立一个SecurityContext，然后在请求结束时，任何SecurityContext的改变都可以被copy到HttpSession

（3）ConcurrentSessionFilter，因为它需要使用SecurityContextHolder的功能，而且更新对应session的最后更新时间，以及通过SessionRegistry获取当前的SessionInformation以检查当前的session省份已经过期，过期会调用LogoutHandler

（4）认证处理机制，如usernamePasswordAuthenticationFilter，CasAuthenticationFilter，BasicAuthenticationFilter等，以至于SecurityContextHolder可以被更新为包含一个有效的Authentication请求。

（5）SecurityContextHolderAwareRequestFilter，它将会把HttpServletRequest封装成一个继承自HttpServletRequestWrapper的SecurityContextHolderAwareRequestWrapper，同时使用SecurityContext实现了HttpServletRequest中与安全相关的方法

（6）JaasApilntegrationFilter，如果SecurityContextHolder中拥有的Authentication是一个JassAuthenticationToken，那么该Filter将使用包含在JaasAuthenticationToken中的Subject继续执行FilterChain

（7）RememberMeAuthenticationFilter，如果之前的认证处理机制没有更新SecurityContextHolder，并且用户请求包含了一个Remember-Me对应的cookie，那么一个对应的Authentication将会设给SecurityContextHolder

（8）AnonymousAuthenticationFilter，如果之前的认证机制都没有更新SecurityContextHolder拥有的Authentication，那么一个AnonymousAuthenticationToken将会设给SecurityContextHolder

（9）ExceptionTransactionFilter，用于处理在FilterChain范围内抛出的AccessDeniedException和AuthenticationException，并把它们转换为对应的Http错误码返回或者对应的页面

（10）FilterSecurityInterceptor，保护Web URI，并且在访问被拒绝时抛出异常。

## 4.基于注解的授权

开启授权，Application添加@EnableGlobalMethodSecurity(prePostEnabled = true)

使用授权，Controller方法上加上@PreAuthorize("hasAuthority('sys:user:add')")