# 使用注解加载容器

@RunWith(SpringJUnit4ClassRunner.**class**)

@ContextConfiguration(locations="classpath:beans.xml")

需要导包

## 

# 测试数据库是否连接成功代码

DataSource bean = context.getBean(DataSource.class);

Connection connection = bean.getConnection();

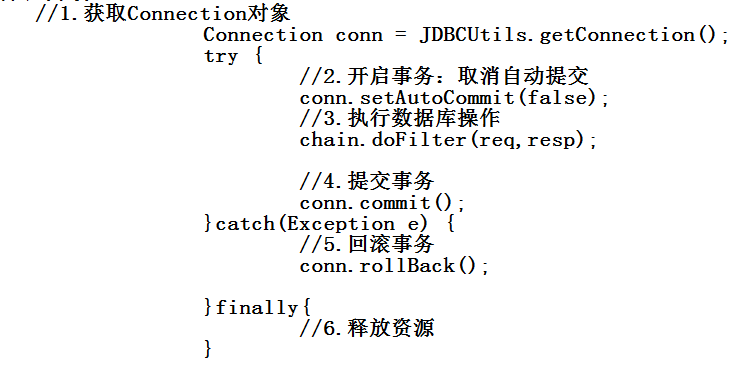
System.out.println(connection);

# 事务:

如果在一个功能单元【方法】执行过程中保持原子性，涉及到对数据库的多次操作，而且，我们希望这个功能单元可以看做一个最小执行单位，里面涉及到对数据库的多次操作要么同时成功，要么同时失败！

# 1）编程式事务

//1.获取Connection对象



# 声明式事务【切面】

1.导入jar包：

core模块

aop包

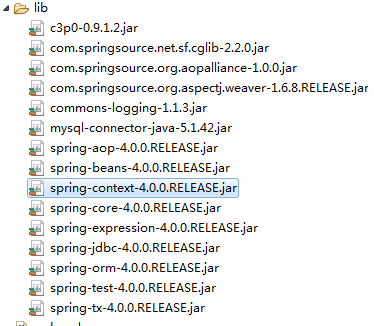
mysql驱动包和数据源的包

jdbc （连接数据库的包）

orm：对象关系模型

tx ：事务的属性（5个）

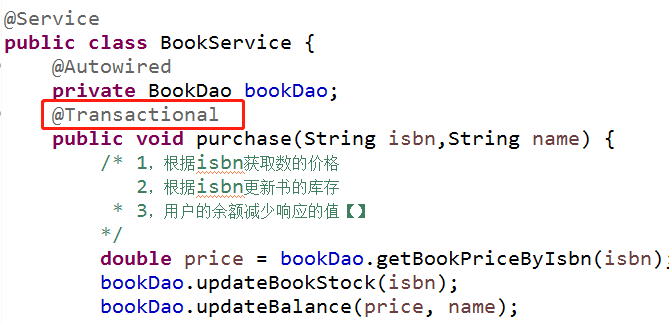
extra包另外3个



2.在spring的配置文件中配置数据源事务管理器，同时开启基于注解的事务支持

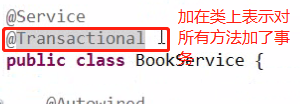


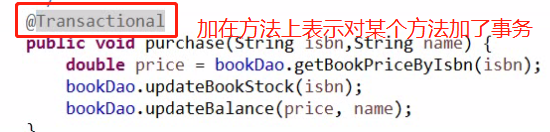
1. 在service层的方法上使用:@Transactional



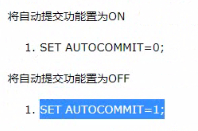


# 事务注解位置：





# 设置数据库自动提交



# 事务的五大属性：

## 1.传播机制[propagation]:

一个带有事务的方法A运行在另一个带有事务的方法B的内部的时候，内层方法A是使用自己的事务还是使用外层方法B的事务！

常用的值：

## required【默认值】:

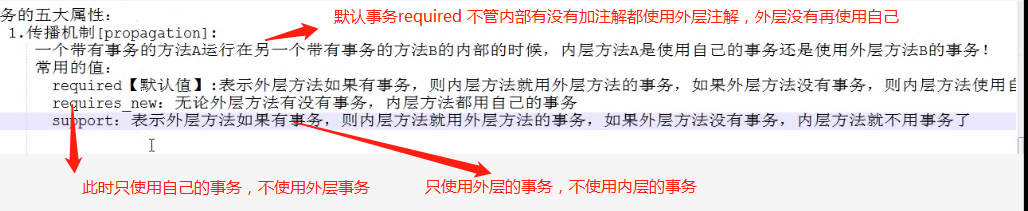
表示外层方法如果有事务，则内层方法就用外层方法的事务，如果外层方法没有事务，则内层方法使用自己的事务

## requires\_new：

无论外层方法有没有事务，内层方法都用自己的事务

## support：

表示外层方法如果有事务，则内层方法就用外层方法的事务，如果外层方法没有事务，内层方法就不用事务了



## 隔离级别[isolation]:

针对并发访问，设置不同的隔离级别，数据库有不同的解决方案

不同的数据库默认的隔离级别不同：

查看数据库默认的隔离级别：SELECT @@tx\_isolation

default:

mysql:可重复读

oracle:读已提交

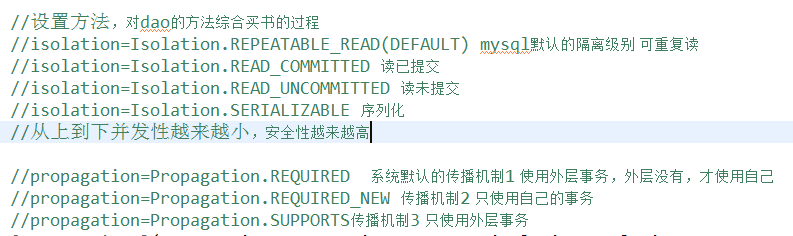
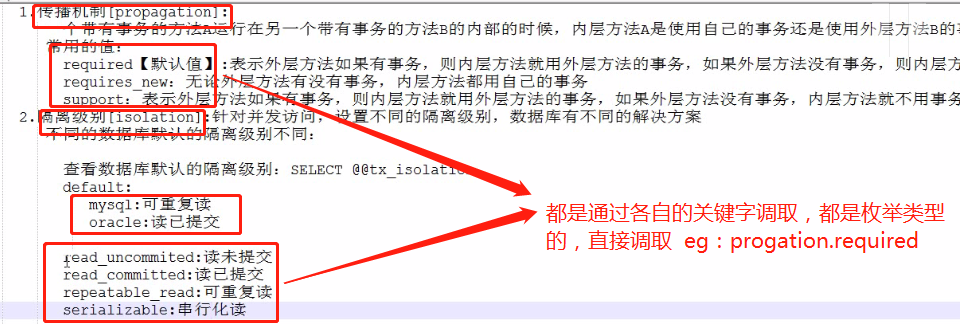
read\_uncommited:读未提交

read\_committed:读已提交

repeatable\_read:可重复读

serializable:串行化读

注意：以上四个隔离级别按着从上到下的顺序，他们的并发性依次降低，安全性越来越高。



## 3.回滚属性[rollback]:

事务不是遇到所有的异常都会回滚，默认只是在遇到RuntimeException的时候，才会回滚

### 编译时异常:

rollback

### 运行时异常：

norollback

5807c48cf039b45cc1dd0fc46b7e947

## 4，超时属性[timeout]：

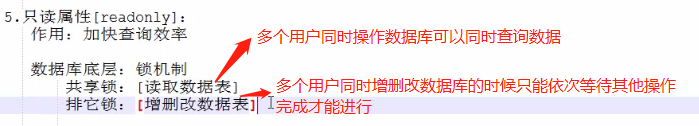
指的是多次操作数据库之间的时间不能超过指定时间！

数据库事务在执行过程中，会占用数据库资源，所以如果某一个事务执行的时间太长， 那么就会导致资源被长时间占用，影响其他事务执行。[死循环，网络问题]

long类型的为毫秒，int类型的为秒

## 5.只读属性[readonly]：

aa73ee7e10d0f35c111a6f45168997e

# Spring在Web项目中的应用：

域对象：

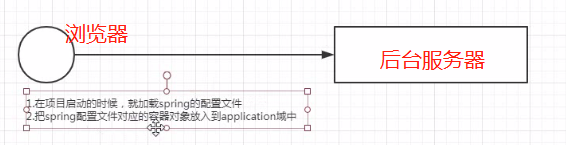
request：

session：

application:

# 设置监听器，监听域对象，一般使用application域对象





# SpringMVC:

SpringMVC是一个控制层框架，本质是Spring的一个小模块！springmvc位于软件三层架构中的web层

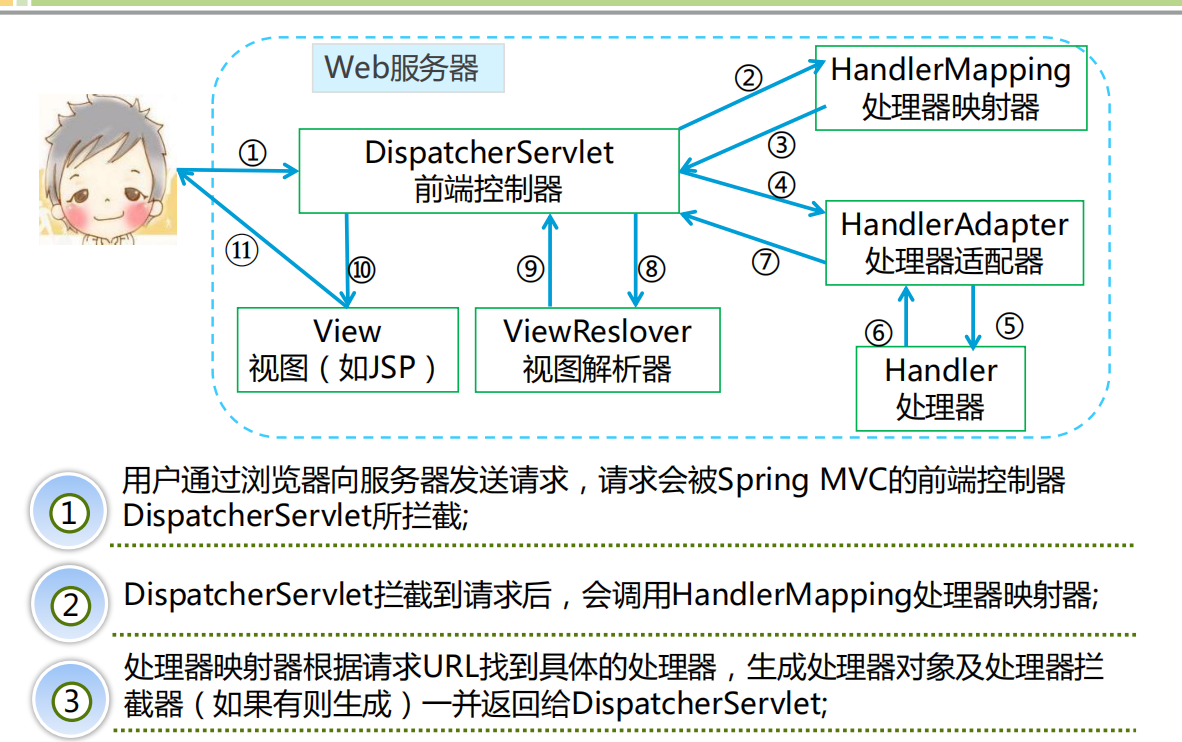
## web层：

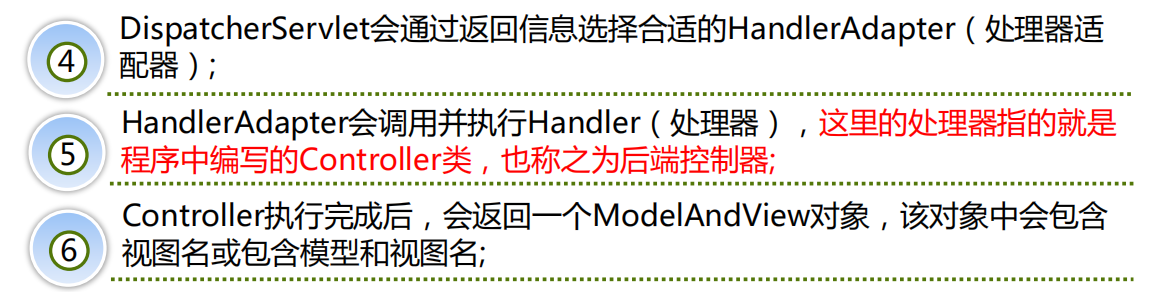
1. 接收浏览器端请求
2. 处理请求

3.返回响应数据

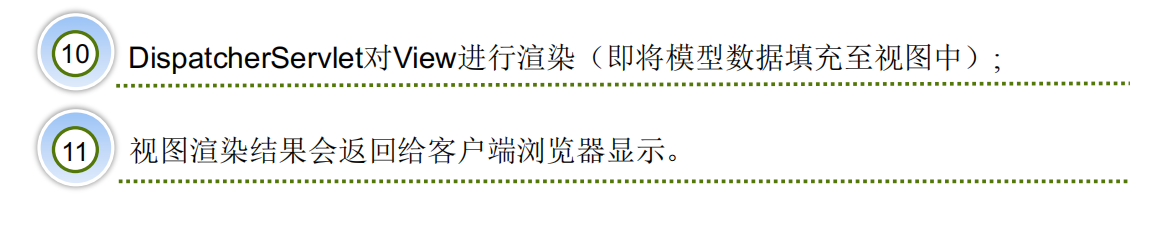
# SpringMVC的运行流程：

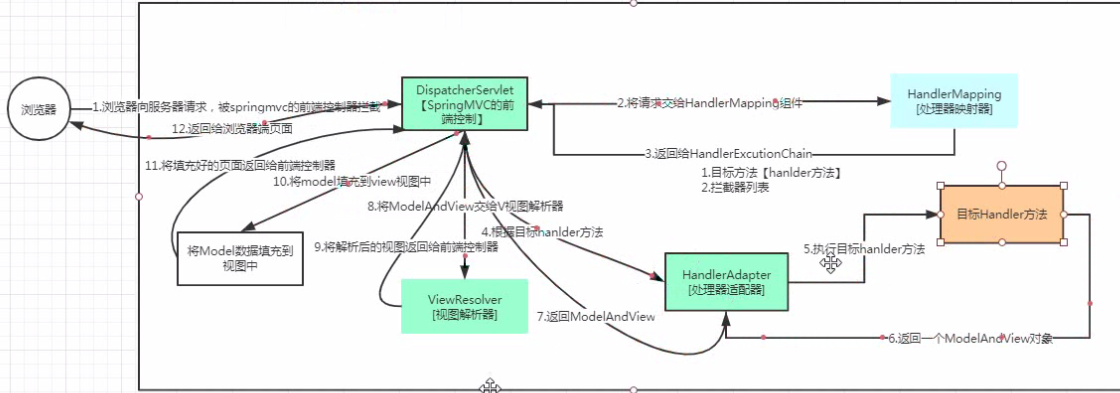
## handler方法就是web层方法











# SpringMVC的第一个案例：

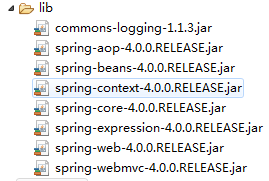
1.导入jar包：

core模块

aop:

web

Webmvc



|  |
| --- |
| （复杂版代码）  2.在web.xml文件中注册SpringMVC的前端控制器[DispatcherServlet]      3.创建SpringMVC的配置文件，并配置它    4.创建控制层： |
| 简洁版：        别的没变 |

# 5. InternalResourceViewResolver:内部资源视图解析器

## 作用：

1.将逻辑视图转换为物理视图,转化过程：

prefix+方法返回值+suffix

/WEB-INF/success.jsp

前缀+方法返回值+后缀

2.转发到对应的页面

## 6. @RequestMapping:专门用来映射请求url,可以标注在类上，也可以标注在方法上



## 属性：

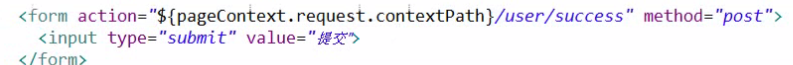
value属性：对应请求url,



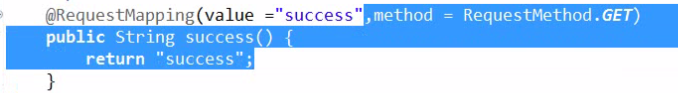
method:请求方式：GET/POST/PUT/DELETE，默认支持四种请求

@RequestMapping 注解的这些属性都是用来限制请求的，之间是“与”的关系

post



get

如何给后台发送PUT或者DELETE请求？

1.必须发送POST请求

2.必须在web.xml文件中配置Filter:HiddenHttpMethodFilter



3.必须携带一个\_method=put/delete

## 

# /和/\*的区别总结

**Url-pattern拦截模式：路径，后缀/path/\*或\*.jsp等**





