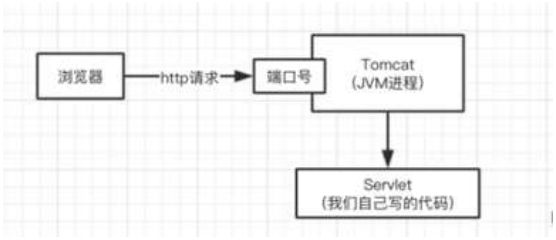
说说你对Spring的IoC机制的理解？

没有Spring之前：



写一套系统，web服务器，tomcat，一旦启动之后，他就可以监听一个端口号的http请求，然后可以把请求转交给你的servlet，jsp，配合起来使用的，servlet处理请求。

比如在我们的一个tomcat + servlet的这样的一个系统里，有几十个地方，都是直接用MyService myService = new MyServiceImpl()，直接创建、引用和依赖了一个MyServiceImpl这样的一个类的对象。

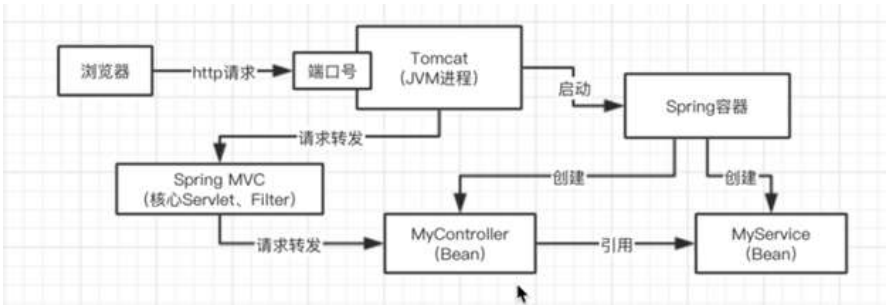
这个系统里，有几十个地方，都跟MyServiceImpl类直接耦合在一起了。

如果我现在不想要用MyServiceImpl了，我们希望用的是NewServiceManagerImpl，implements MyService这个接口的，所有的实现逻辑都不同了，此时我们很麻烦，我们需要在系统里，几十个地方，都去修改对应的MyServiceImpl这个类，切换为NewServiceManagerImpl这个类。

改动代码成本很大，改动完以后的测试的成本很大，改动的过程中可能很复杂，出现一些bug，此时就会很痛苦。

归根结底，代码里，**各种类之间完全耦合在一起，出现任何一丁点的变动，都需要改动大量的代码**，重新测试，可能还会有bug。

有Spring之后：



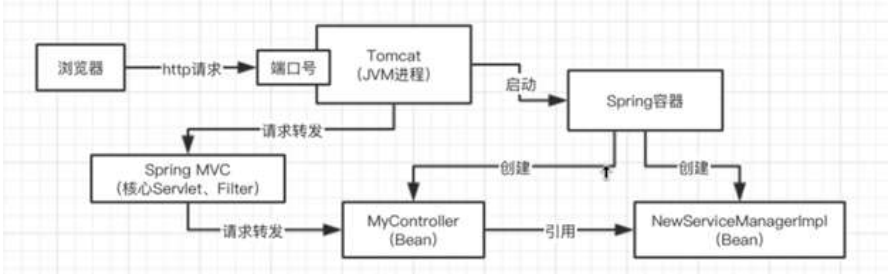
Spring IoC, Spring容器，根据XML配置或注解，去实例化你的一些bean对象，然后根据XML或注解，去对bean对象之间的引用关系，去进行依赖注入。

底层核心技术是反射。通过反射技术，直接根据你的类去自己构建对应的对象出来。

好处: 系统的类与类之间彻底解耦合。

比如原来是注解myService，几十个地方都依赖这个类，如果要升级或修改这个实现类为newServiceManager。只需要把myService头上的注解去掉，把@Service注解放在newServiceManager上面，

其他引用的地方不变。



说说你对Spring的AOP机制的理解？

Spring核心框架，两个机制，IoC和AOP。

Spring在运行时，AOP的核心技术，是动态代理。会给你的类生成动态代理类。

```
@Controller
public class MyController {

    @Resource
    private MyServiceA myServiceA; // 注入的是动态代理的对象实例，ProxyMyServiceA

    public void doRequest() {
        myServiceA.doServiceA(); // 直接调用到动态代理的对象实例的方法中去
    }
}

@Service
public class MyServiceAImpl implements MyServiceA {

    public void doServiceA() {
        // insert语句
        // update语句
        // update语句
        // delete语句
    }
}

@Service
public class MyServiceBImpl implements MyServiceB {

    public void doServiceB() {
        // update语句
        // update语句
        // insert语句
    }
}
```

.png

做一个切面，如何定义呢？MyServiceXXXX的这种类，在这些类的所有方法中，都去织入一些代码，在所有这些方法刚开始运行的时候，都先去开启一个事务，在所有这些方法运行完毕之后，去根据是否抛出异常来判断一下，如果抛出异常，就回滚事务，如果没有异常，就提交事务 => AOP。

```
public class ProxyMyServiceA implements MyServiceA {

    private MyServiceA myServiceA;

    public void doServiceA() {
        // 开启事务
        // 直接去调用我依赖的MyServiceA对象的方法
        myServiceA.doServiceA();
        // 根据他是否抛出异常，回滚事务，or，提交事务
    }
}
```

AOP总结：面向切面编程，通过**动态代理**的方式来生成其代理对象在方法执行前后加入自己的逻辑

代理方式：**jdk动态代理(接口) cglib动态代理(基于类)**

相关名词:

1. JoinPoint连接点：拦截的接口的方法
2. Pointcut切入点：对哪些连接点进行拦截
3. Advice通知：比如前置通知 后置通知 环绕通知
4. aspect切面：切入点和通知组成

切入点 execution表达式 1. execution 权限修饰符 返回值类型 包名.类名.方法名(参数)

通知类型 ：

1. 前置通知:方法执行之前
2. 后置通知:在方法正常执行完毕后(提交事务)
3. 最终通知:在方法正常执行完毕或者出现异常
4. 异常通知:执行过程中出现异常(事物回滚)
5. 环绕通知:方法执行前后,目标方法默认不执行需自己调用方法

了解Cglib动态代理？它跟jdk动态代理的区别？

优先是jdk动态代理，其次是cglib动态代理。

动态代理就是动态的创建一个代理类出来，创建这个代理类的实例对象，在这个里面引用你真正自己写的类，所有的方法的调用，都是先走代理类的对象，他负责做一些代码上的增强，再去调用你写的那个类。

Spring里使用aop，比如说你对一批类和他们的方法做了一个切面，定义好了要在这些类的方法里增强的代码，Spring必然要对那些类生成动态代理，在动态代理中去执行你定义的一些增强代码。

如果你的类是实现了某个**接口**的，spring aop会使用jdk动态代理，生成一个跟你实现同样接口的一个代理类，构造一个实例对象出来，jdk动态代理，他其实是在你的类有接口的时候，就会来使用。

很多时候我们可能某个类是没有实现接口的，spring aop会改用cglib来生成动态代理，他是生成你的类的一个**子类**，他可以动态生成字节码，覆盖你的一些方法，在方法里加入增强的代码。

Spring中的Bean是线程安全的吗？

线程不安全！

Spring容器中的bean可以分为5个范围：

（1）singleton：**默认**，每个容器中只有一个bean的实例

（2）prototype：为每一个bean请求提供一个实例

一般来说下面几种作用域，在开发的时候一般都不会用，99.99%的时候都是用singleton单例作用域

（3）request：为每一个网络请求创建一个实例，在请求完成以后，bean会失效并被垃圾回收器回收

（4）session：与request范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效

（5）global-session

```
@Service
public class MyServiceImpl implements MyService {
    public void doService() {
            // 访问数据库
    }
}

@Controller
public class MyController {

    @Resource
    private MyService myService;

    public void doRequest() {
        myService.doService();
    }
}
```

spring bean默认来说，singleton，都是线程不安全的，java web系统，一般来说很少在spring bean里放一些实例变量，一般来说他们都是多个组件互相调用，最终去访问数据库的。

Spring的事务实现原理是什么？能聊聊你对事务传播机制的理解吗？

主要考察1. 事务的实现原理，2. 事务的传播机制

1. 实现原理：加一个@Transactional注解，Spring会使用**AOP**，对这个方法在执行前，先开启事务，在执行完毕后，根据方法是否报错，来决定是回滚还是提交事务。

2. 传播机制：

1. **PROPAGATION_REQUIRED**(默认)： 如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。

2. PROPAGATION_SUPPORTS(少用)： 支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。‘

3. PROPAGATION_MANDATORY(很少用)： 支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。

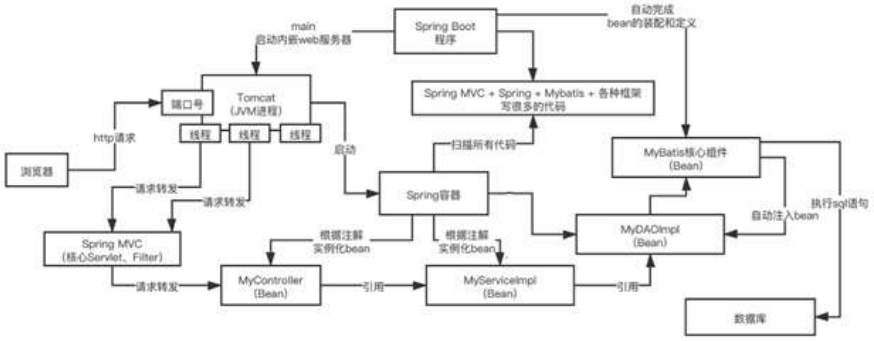
4. PROPAGATION_REQUIRES_NEW(常用)： 创建新事务，无论当前存不存在事务，都创建新事务。

5. PROPAGATION_NOT_SUPPORTED(很少用)：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

6. PROPAGATION_NEVER(很少用)： 以非事务方式执行，如果当前存在事务，则抛出异常。

7. PROPAGATION_NESTED(用)： 如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则按REQUIRED属性执行。

能画一张图说说Spring Boot的核心架构吗？



背景：**简化繁琐的Spring配置**

spring框架，mybatis，spring mvc，去做一些开发，打包部署到线上的tomcat里去，tomcat启动了，他就会接收http请求，转发给spring mvc框架，调用controller -> service -> dao -> mybatis（sql语句）。

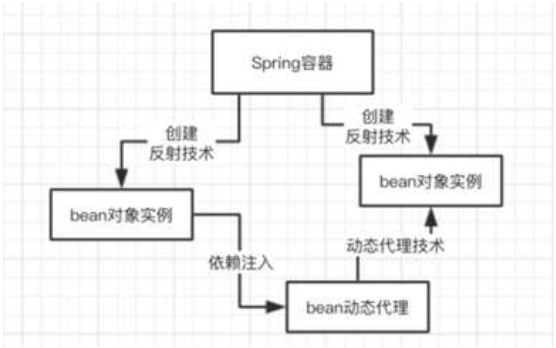
基于spring boot直接进行开发，里面还是使用spring + spring mvc + mybatis一些框架，我们可以一定程度上来简化我们之前的开发流程。

spring boot内嵌一个tomcat去直接让我们一下子就可以把写好的java web系统给启动起来，直接运行一个main方法，spring boot就直接把tomcat服务器给跑起来，把我们的代码运行起来了。

自动装配，比如说我们可以引入mybatis，我其实主要引入一个starter依赖，他会一定程度上个自动完成mybatis的一些配置和定义，不需要我们手工去做大量的配置了，一定程度上简化我们搭建一个工程的成本。

只要引入一个starter，他会自动给你引入需要的一些jar包，做非常简单的、必须的一些配置，比如数据库的地址，几乎就不用你做太多的其他额外的配置了，他会自动帮你去进行一些配置，定义和生成对应的bean生成的bean自动注入比如你的dao里去，让你免去一些手工配置+定义bean的一些工作。

能画一张图说说Spring的核心架构吗？



Spring生命周期： 创建 -> 使用 -> 销毁

首先用xml或注解，定义一堆bean.

1. 实例化bean

通过反射创建bean对象实例。

2. 设置对象属性（依赖注入）

实例化后的对象被封装在BeanWrapper对象中，Spring根据BeanDefinition中的信息以及通过BeanWrapper提供的设置属性接口完成依赖注入。

这个bean依赖了谁，把依赖的bean也创建出阿里，给你进行一个注入，比如通过构造函数或setter方法。

```
public class MyService {

    private MyDao myDao;

    public MyService(MyDao myDao) {
        this.myDao = myDao;
    }

    public void setMyDao(MyDao myDao) {
        this.myDao = myDao;
    }
```

3. 处理Aware接口

Spring检查该对象是否实现了xxxAware接口，并将相关的xxxAware实例注入给Bean。

3.1. 如果这个Bean实现了BeanNameAware接口，则调用它的实现setBeanName(String beanid)方法，传递就是Spring配置文件中的Bean的id值；

3.2. 如果这个Bean实现了BeanFactoryAware接口，则调用它实现的setBeanFactory()方法，传递的是Spring工厂自身；

3.3. 如果这个Bean实现了ApplicationContextAware接口，则调用setApplicationContext(ApplicationContext)方法，传入Spring上下文；

4. BeanPostProcessor

如果想在bean实例构建之后，在这个时间点对bean进行自定义处理，则可以让bean实现BeanPostProcessor接口，

会调用**postProcessBeforeInitialiazation**(Object obj, String s)方法。

5. InitalizingBean与init-method

如果bean在Spring配置文件中配置了**init-method**属性，则会自动调用其配置的初始化方法。

6. BeanPostProcessor

在bean初始化完成后，如果这个bean实现了BeanPostProcessor接口，会调用**postProcessAfterInitialization**(Object obj, String s)方法。

7. DisposableBean

当bean不再需要时，如果bean实现了DisposableBean接口，会调用其他实现的destroy()方法。

8. destroy-method

如果配置了**destroy-method**属性，会调用配置的销毁方法。

能说说Spring中都使用了哪些设计模式吗？

工厂模式、单例模式、代理模式

工厂模式：把一个对象的创建过程放入一个具体工厂类中。当需要使用时通过工厂把这个对象实例取出来。

Spring ioc核心的设计模式的思想提现，他自己就是一个大的工厂，把所有的bean实例都给放在了spring容器里（大工厂），如果你要使用bean，就找spring容器就可以了，你自己不用创建对象了。

```
public class MyController {
    private MyService myService = MyServiceFactory.getMyService();
}

public class MyServiceFactory {

    public static MyService getMyService() {
        return new MyServiceImpl();
    }
}
```

单例模式：

Spring默认来说，对每个bean走的都是一个单例模式，确保说你的一个类在系统运行期间只有一个实例对象，只有一个bean，用到了一个单例模式的思想，保证了每个bean都是单例的。

双重校验锁：

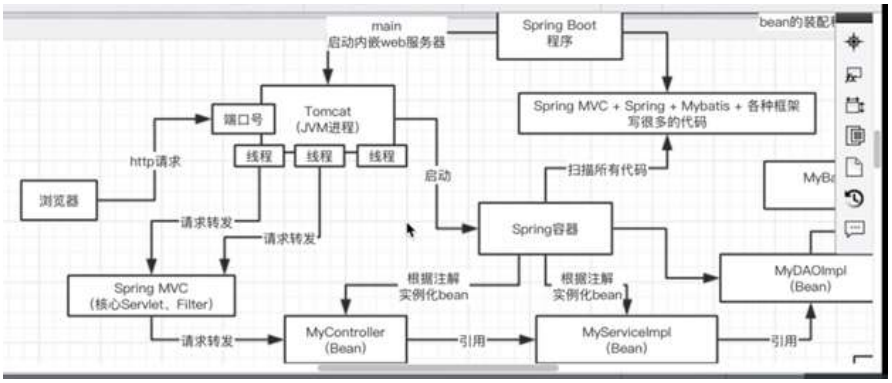
```
public class MyService {

    private static volatile MyService myService;

    public static MyService getInstance() {
        if(myService == null) {
            synchronized(MySerivce.class) {

                if(mySerivce == null) {
                    myService = new MyService();
                }
            }
        }
        return myService;

    }
}
```

代理模式:

如果说你要对一些类的方法切入一些增强的代码，会创建一些动态代理的对象，让你对那些目标对象的访问，先经过动态代理对象，动态代理对象先做一些增强的代码，调用你的目标对象。

能画一张图说说Spring Web MVC的核心架构吗？

Tomcat + Spring架构图：



1. Tomcat的工作线程将请求转交给spring mvc框架的DispatcherServlet

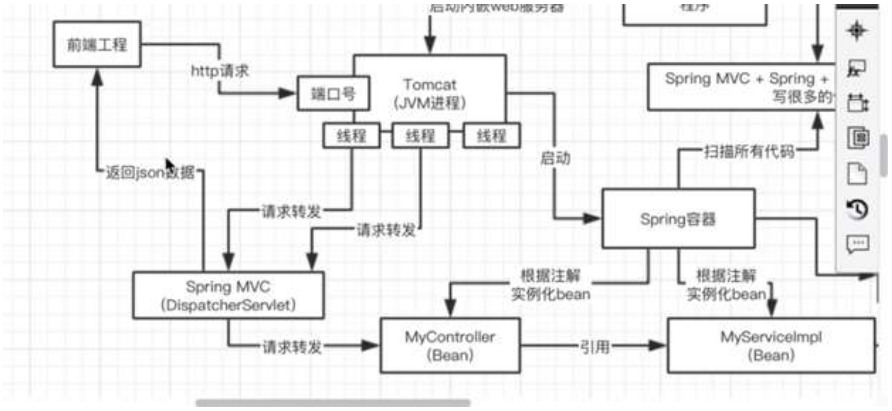
2. DispatcherServlet查找@Controller注解的controller，我们一般会给controller加上你@RequestMapping的注解，标注说哪些controller用来处理哪些请求，此时根据请求的uri，去定位到哪个controller来进行处理。

3. 根据@RequestMapping去查找，使用这个controller内的哪个方法来进行请求的处理，对每个方法一般也会加@RequestMapping的注解。

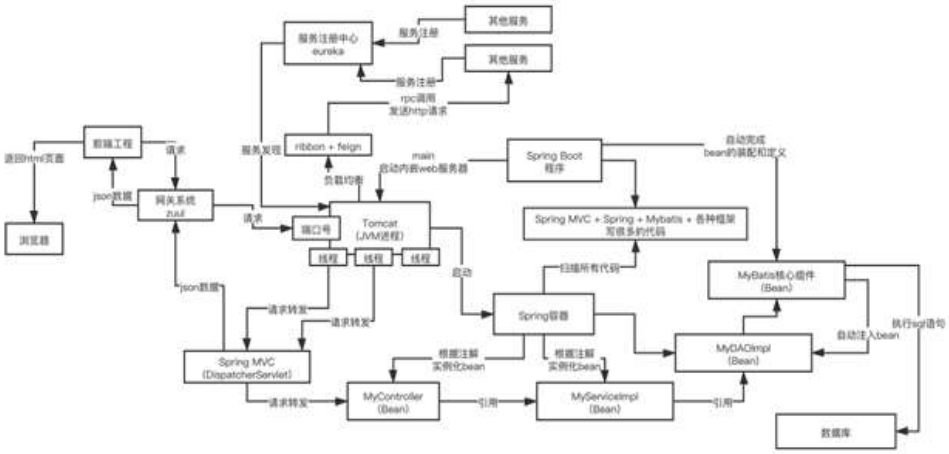
4. 会直接调用我们的controller里面的某个方法来进行请求的处理。

5. 我们的controller的方法会有一个返回值，以前的时候，一般来说还是走jsp、模板技术，我们会把前端页面放在后端的工程里面，返回一个页面模板的名字，spring mvc的框架使用模板技术，对html页面做一个渲染；

现在一般返回一个json串，前后端分离，可能前端发送一个请求过来，我们只要返回json数据。



能画一张图说说Spring Cloud的核心架构吗？



Spring Cloud主要由eureka、ribbon、feign、zuul、hystrix、链路追踪等组件组成。

参考资料：

互联网Java工程师面试突击（第三季）-- 中华石杉

[Java复习] 面试突击 - Spring

标签：对象 spring配置 alt 需要 连接点 war 其他 ribbon rop

原文地址：https://www.cnblogs.com/fyql/p/12195585.html