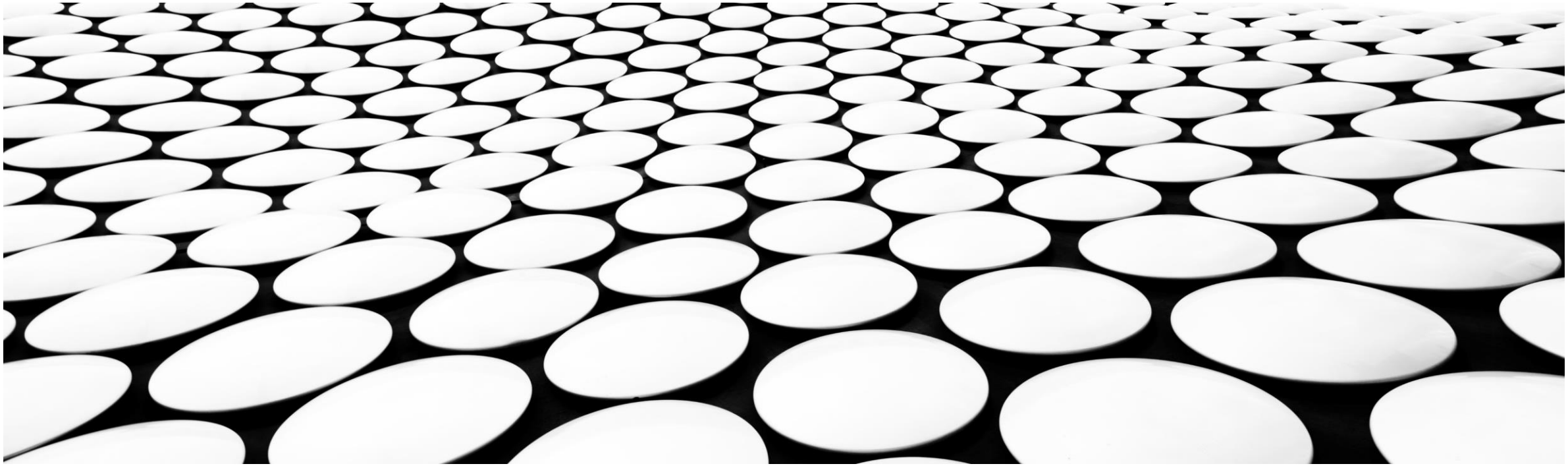
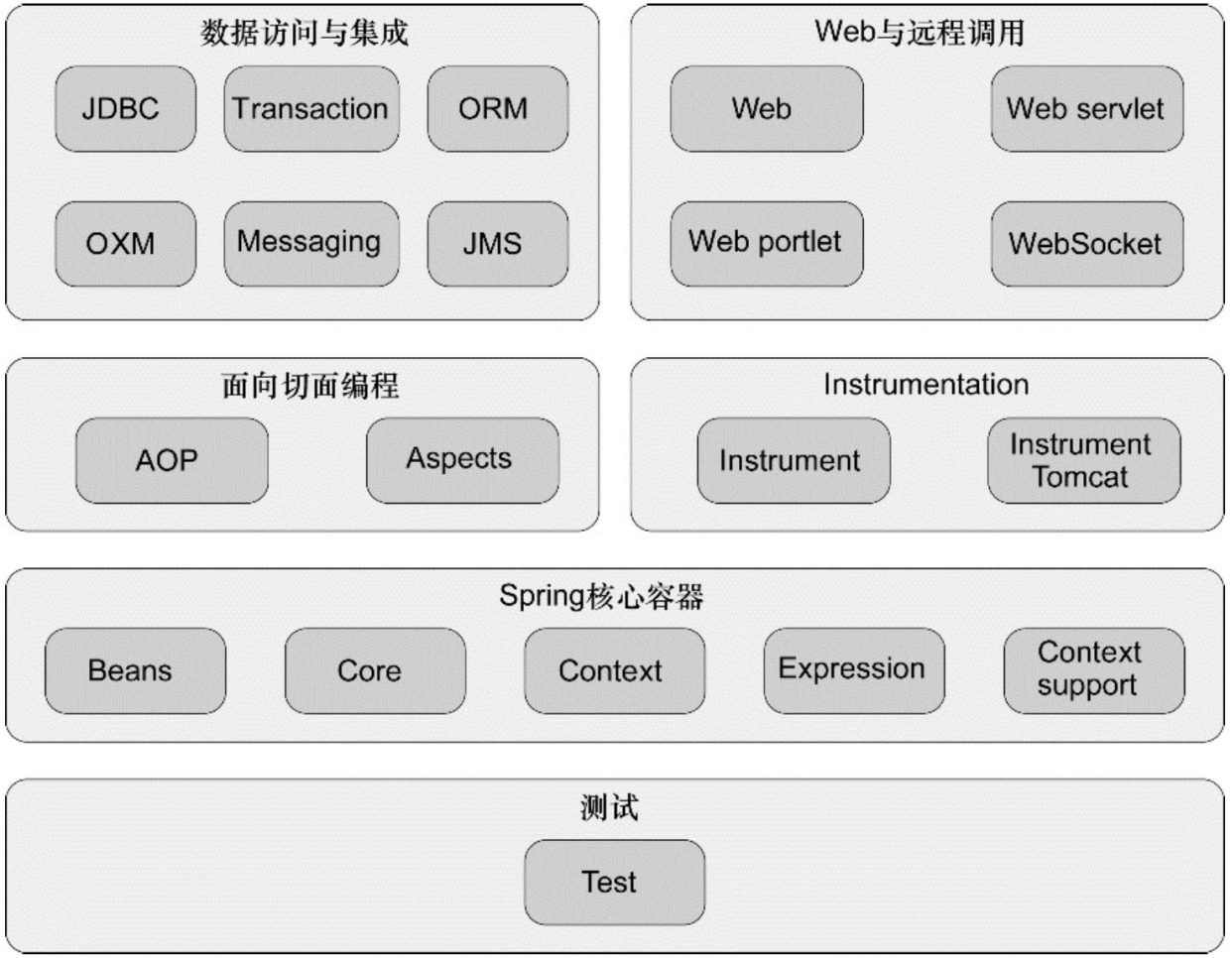


服务端开发-依赖注入（Dependency Injection）

陶召胜

仅用于南京大学软件学院

Spring的模块组成



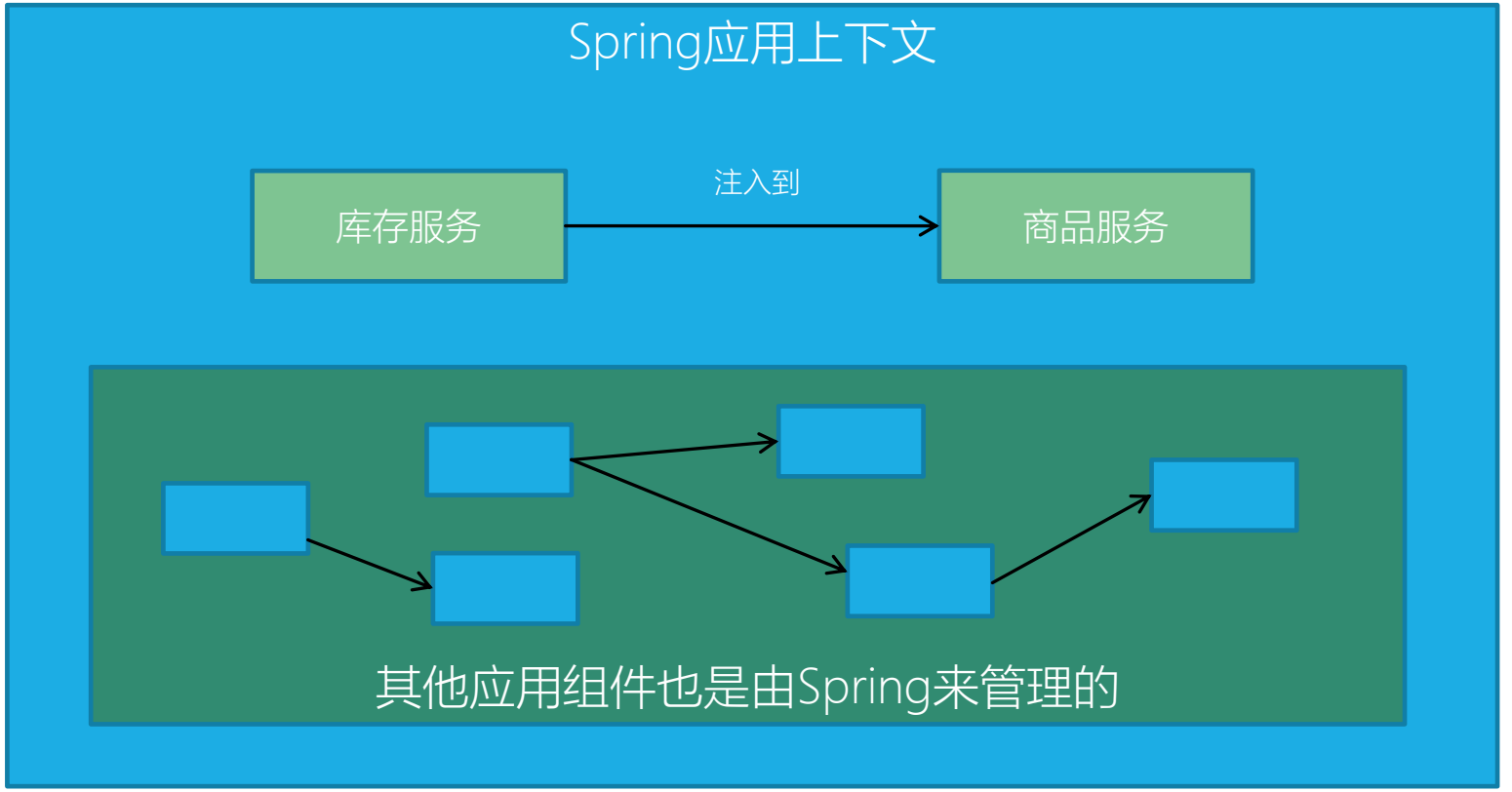
Spring的两个核心技术

- DI（Dependency Injection）
- 保留抽象接口，让组件（Component）依赖于抽象接口，当组件要与其他实际的对象发生依赖关系时，由抽象接口来注入依赖的实际对象

- AOP（Aspect Oriented Programming）
- 通过预编译方式和运行期间动态代理实现程序功能的统一维护的一种技术
- 利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发效率

仅用于南京大学软件学院

Spring的核心是提供了一个容器（container）



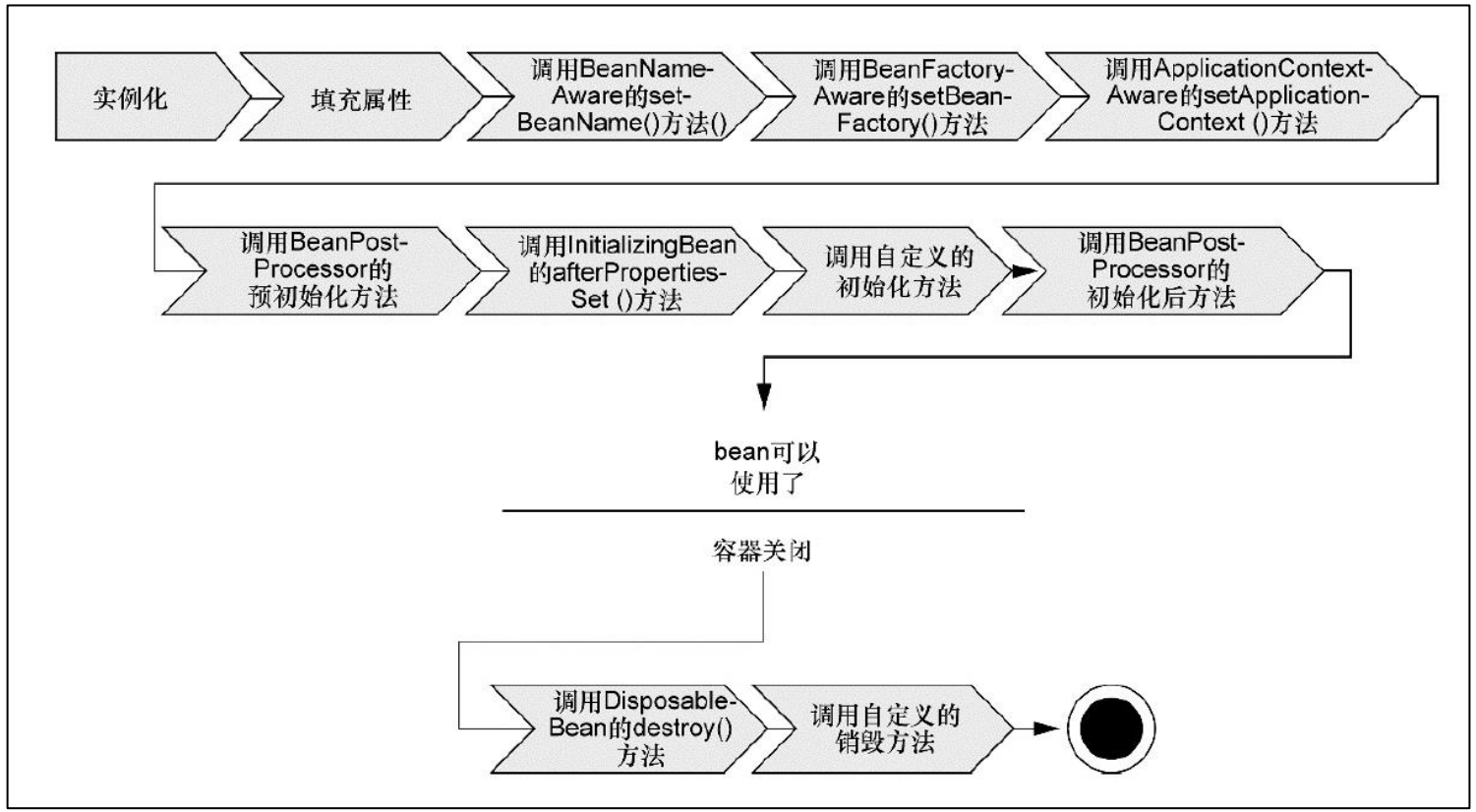
图摘自《Spring 实战》（第6版）

应用上下文

- AnnotationConfigApplicationContext
- AnnotationConfigWebApplicationContext
- ClassPathXmlApplicationContext
- FileSystemXmlApplicationContext
- XmlWebApplicationContext

```
7  public class MyXmlApp {
8      public static void main(String[] args) {
9          ApplicationContext ctx = new ClassPathXmlApplicationContext("classpath:META-INF/spring/soundsystem.xml");
10         CDPlayer player = ctx.getBean("CDPlayer", CDPlayer.class);
11         player.play();
12     }
13 }
```

bean的生命周期



图摘自《Spring 实战》（第4版）

Spring配置方案

1、自动化配置

```java
@Component
public class CDPlayer implements MediaPlayer {
    private CompactDisc cd;

    @Autowired
    public CDPlayer(CompactDisc cd) {
        this.cd = cd;
    }

    public void play() {
        cd.play();
    }
```

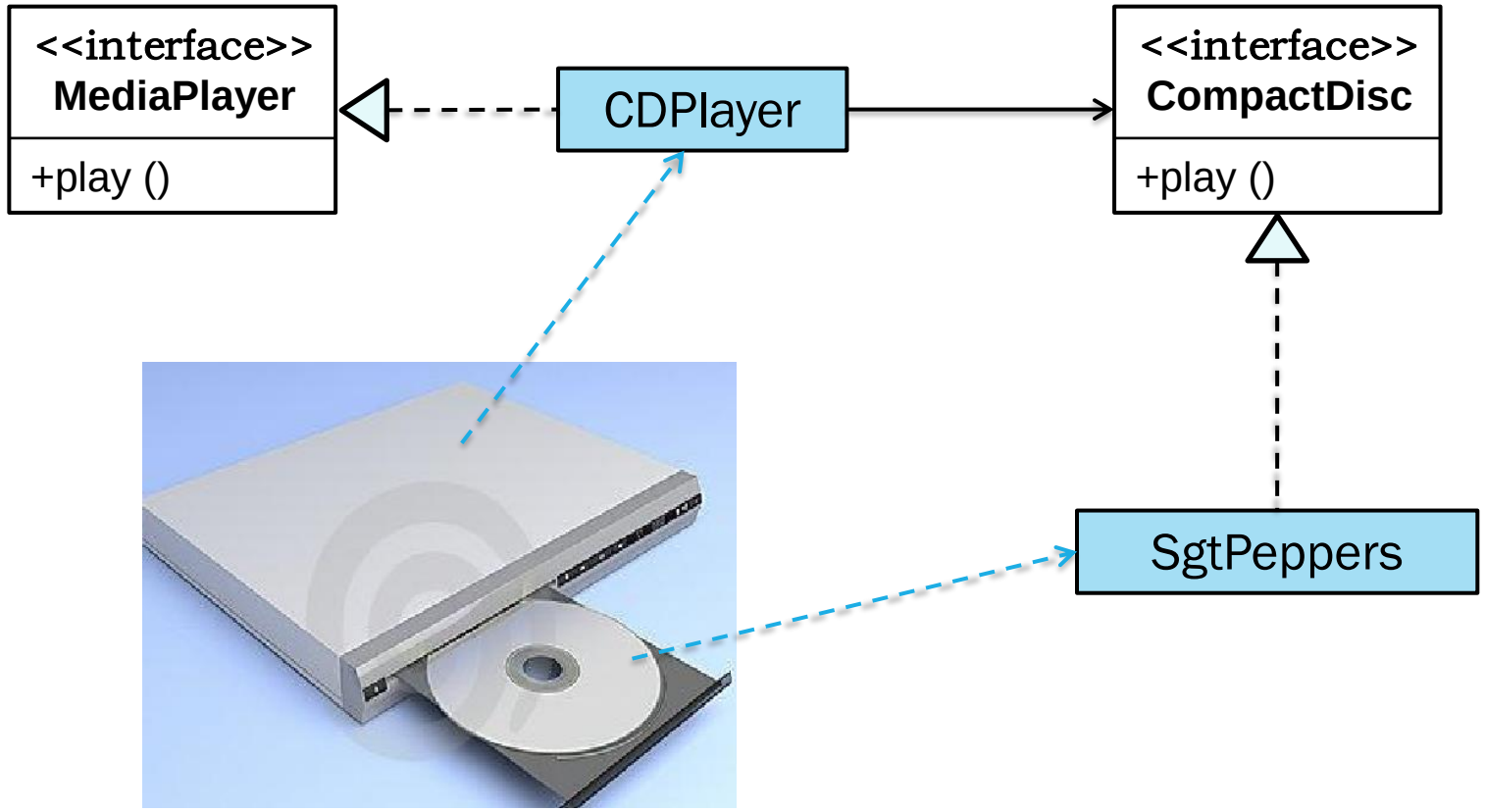
2、 JavaConfig

```java
@Configuration
public class CDPlayerConfig {

    @Bean
    public CompactDisc compactDisc() {
        return new SgtPeppers();
    }

    @Bean
    public CDPlayer cdPlayer(CompactDisc cd) {
        return new CDPlayer(cd);
    }
```

3、XML配置

```xml
<bean id="compactDisc" class="soundsystem.SgtPeppers" />

<bean id="cdPlayer" class="soundsystem.CDPlayer">
  <constructor-arg ref="compactDisc" />
</bean>
```

例子代码类图


<<interface>>
MediaPlayer
+play ()
CDPlayer
<<interface>>
CompactDisc
+play ()
SgtPeppers

自动化配置

- 组件扫描（component scanning）
- 自动装配（autowiring）

组件扫描

- @Configuration
- @ComponentScan
  - ✓ 等价 <context:component-scan base-package="..."/>
  - ✓ 基础包（basePackages={"...","..."})
  - ✓ 类型不安全（not type-safe）
  - ✓ basePackageClasses={.class,.class}
  - ✓ Marker interface

自动装配

- @Autowired
  - ✓ 用在构造器
  - ✓ 用在属性Setter方法
  - ✓ 用在（私有）属性
  - ✓ required=false

JavaConfig

- 自动化配置有时会行不通，如：第三方库
- @Configuration
- @Bean(name="..")
- 注入
  - ✓ 调用方法（）
  - ✓ 通过方法参数自动装配（其它配置类、其它方式创建的Bean）
- 注意与业务逻辑和领域代码分开

XML装配

- <beans><bean>
- 不能类型检查
- 构造器注入
  - ✓ <constructor-arg>
  - ✓ c-命名空间
  - ✓ 注入字面量值
  - ✓ 注入集合
- 属性注入
  - ✓ p-命名空间
  - ✓ util-命名空间
- 建议：强依赖使用构造器注入

仅用于南京大学软件学院

混合配置

- JavaConfig中的导入
  - ✓ @Import(配置类.class,...)
  - ✓ @ImportResource(xml文件)
- XML中的导入
  - ✓ <import resource="xml文件"/>
  - ✓ <bean class="配置类"/>

仅用于南京大学软件学院

根配置

@Configeration

@ComponetScan

@Import(其它配置类...)

@ImportResource(其它xml文件)

Class BootConfig(){

}

@Profile

@Configuration

@Profile(“dev”)

类

@Bean

@Profile(“prov”)

方法

激活

spring.profiles.default

spring.profiles.active

@ActiveProfiles(“dev”)

@Conditional

@Bean或@Component

@Conditional(**.class)

接口

Condition{

boolean matches(...)

}

自动装配的歧义性

@Component 或 @Bean

@Primary

定义时

@Componet或@Bean

@Qualifier("...")自定义限定符

使用时

@Autowired

@Qualifier("...") bean名称或自定义限定符，默认Bean名是限定符

也可以自定义注解，这些注解本身也加了@Qualifier注解

@Cold

@Creamy

Bean的作用域

@Scope可以与@Component和@Bean一起使用，指定作用域

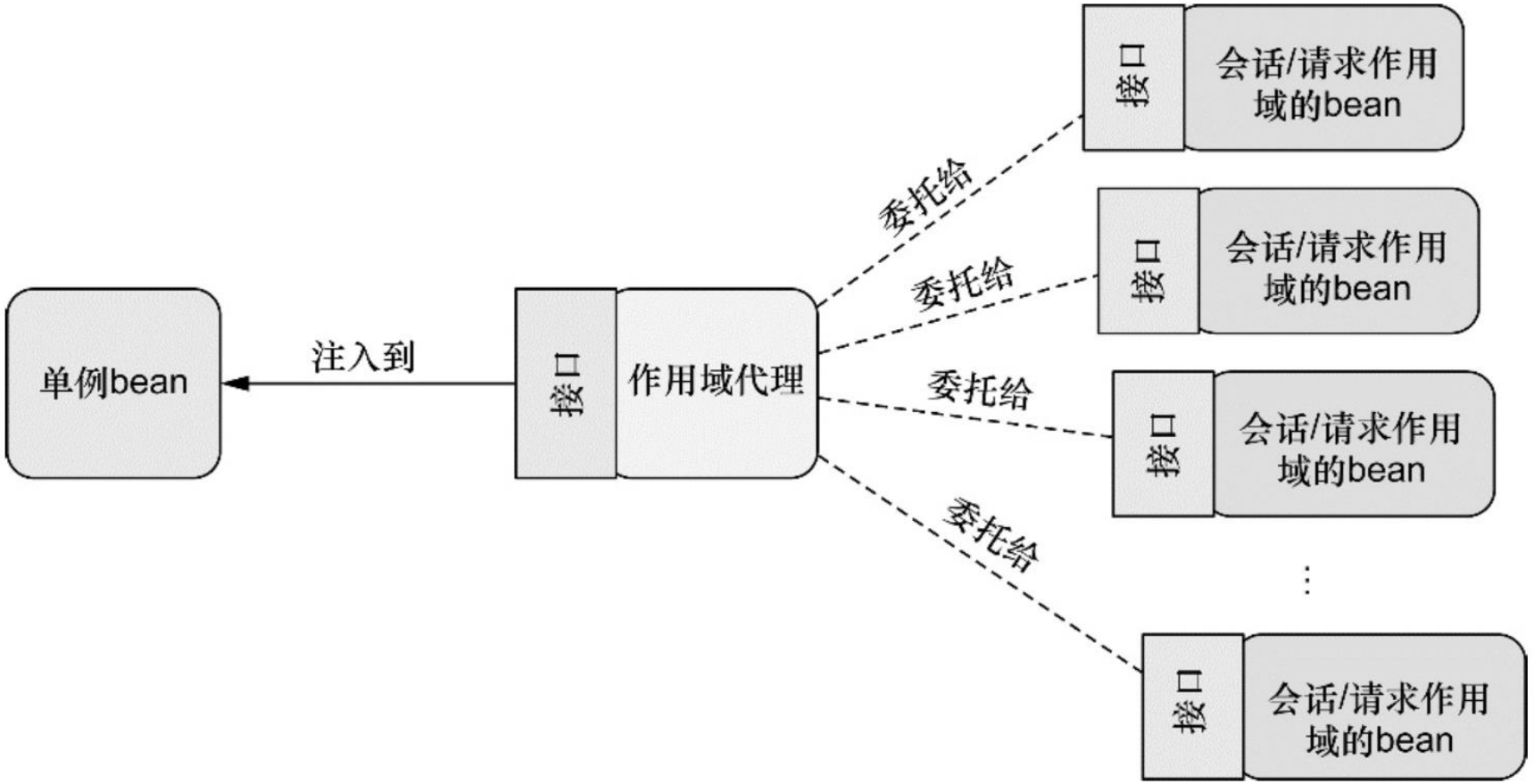
- Singleton，单例，在整个应用中，只创建bean的一个实例
- Prototype，原型，每次注入或者通过Spring应用上下文获取的时候，都会创建一个新bean实例
- Session，会话，在Web应用中，为每个会话创建一个bean实例
- Request，请求，在Web应用中，为每个请求创建一个bean实例

使用会话和请求作用域

```
@Component
@Scope(value=WebApplicationContext.SCOPE_SESSION, proxyMode=ScopedProxyMode.INTERFACES)
public ShoppingCart cart(){....}
```

仅用于南京大学软件学院

通过代理注入给单例对象



运行时注入外部值

- 使用Environment检索属性
- 使用@PropertySource指定属性文件

Environment.getProperty的几种形式

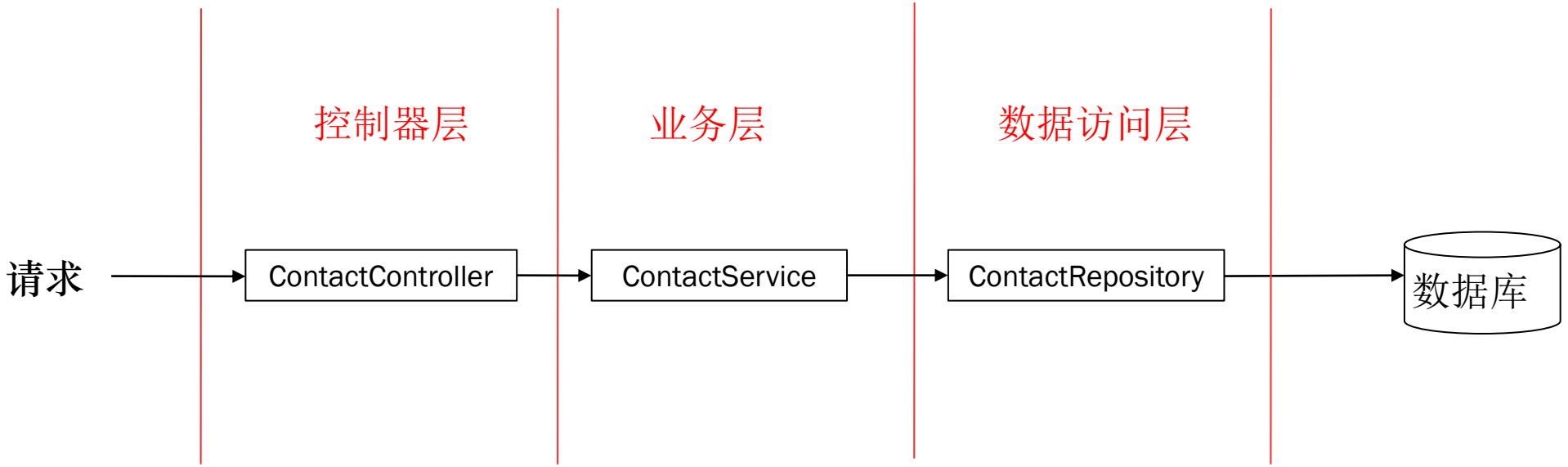
- String getProperty（String key）
- String getProperty（String key, String defaultValue）
- T getProperty（String key, Class<T> type）
- T getProperty（String key, Class<T> type, T defaultValue ）

仅用于南京大学软件学院

使用占位符

- ${ … }

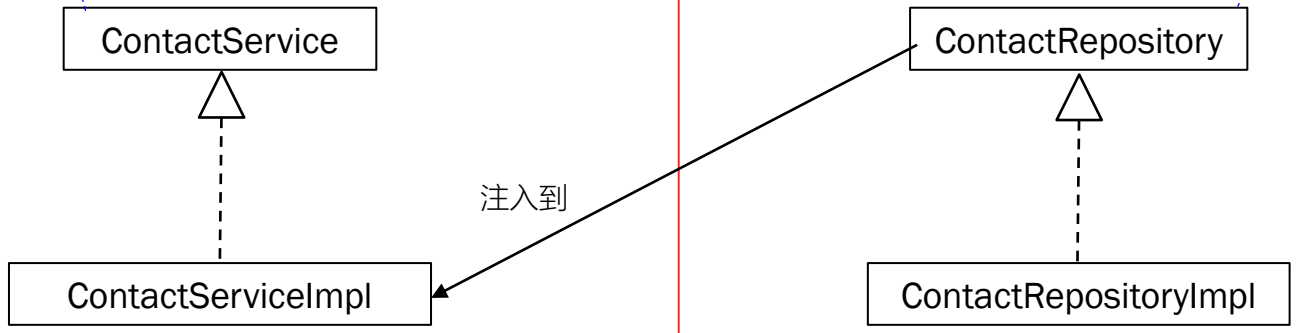
回顾：Spring Web开发框架的分层



本次作业实现范围：业务层、数据访问层

```
public interface ContactService {
    List<Contact> getAll();

    void add(Contact contact);
}
```

业务层



数据访问层

```
public interface ContactRepository {
    List<Contact> findAll();

    void save(Contact contact);

    void clear();
}
```

已提供的文件

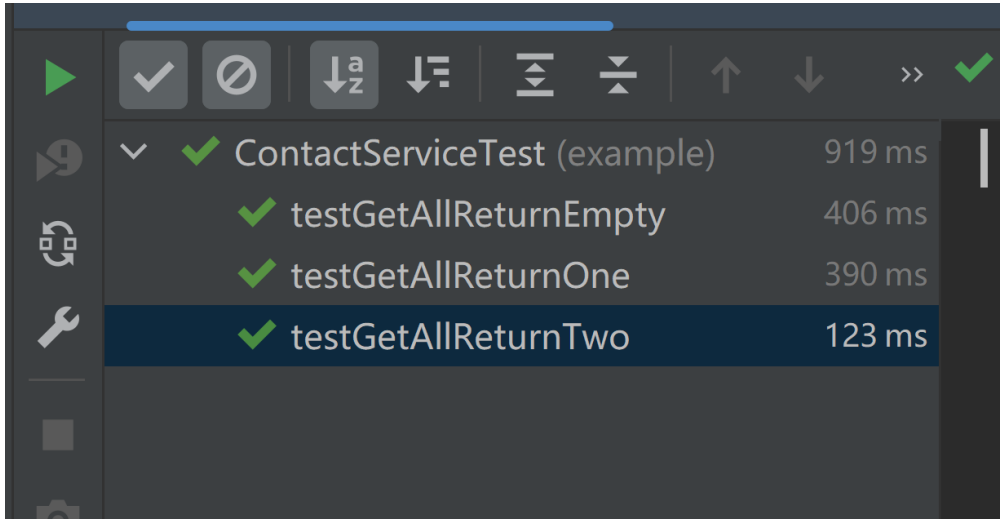
- Contact.java，领域类，表示一个人的联系信息
- ContactService.java，业务类接口
- ContactRepository.java，数据访问层接口
- ContactServiceTest.java，测试代码

仅用于南京大学软件学院

提交

- 所有源代码，压缩成一个文件。不包含编译后的class文件（删除target目录）

- 测试成功的截图



工具java-faker

- https://github.com/DiUS/java-faker

谢谢观看！