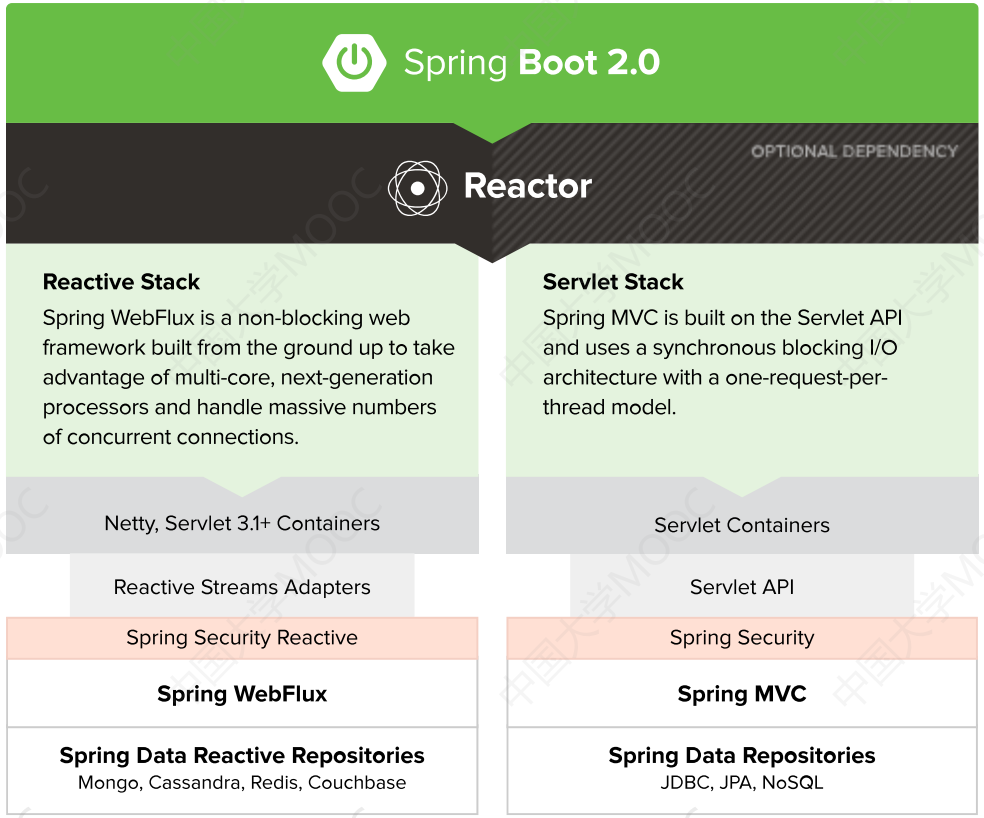
JavaEE平台技术
Spring与SpringBoot

邱明 博士
厦门大学信息学院
mingqiu@xmu.edu.cn

1. Spring

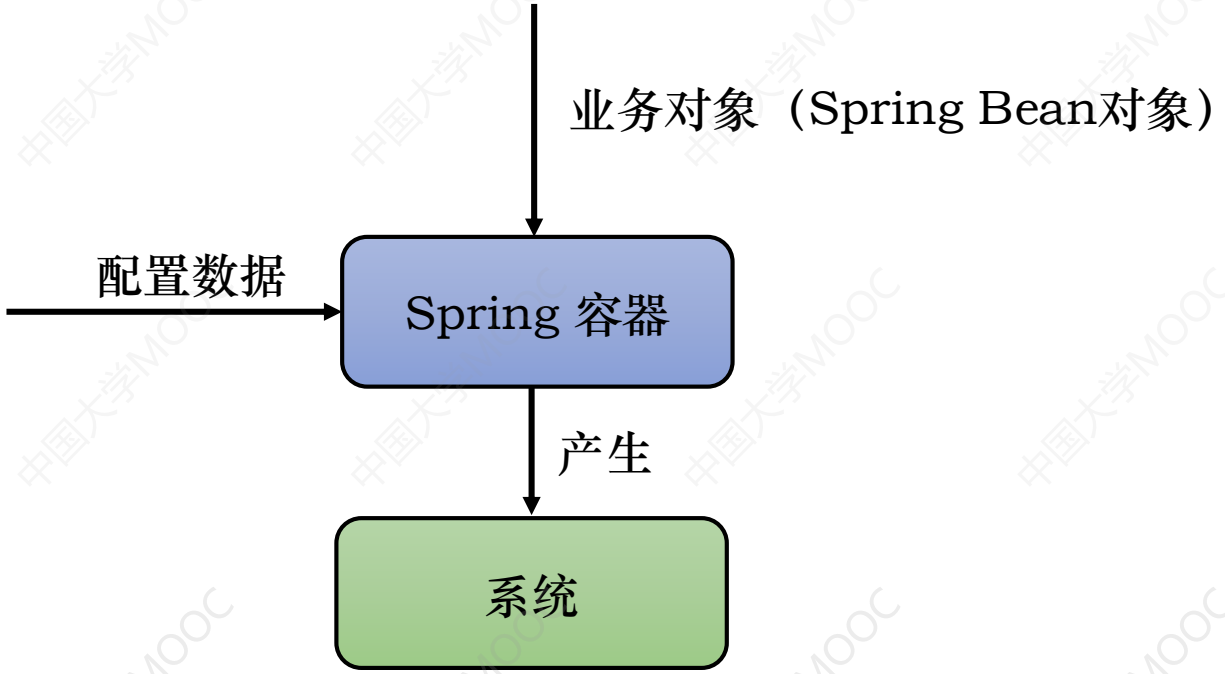

Spring Boot 2.0
Reactor
OPTIONAL DEPENDENCY
Reactive Stack
Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.
Servlet Stack
Spring MVC is built on the Servlet API and uses a synchronous blocking I/O architecture with a one-request-per-thread model.
Netty, Servlet 3.1+ Containers
Servlet Containers
Reactive Streams Adapters
Servlet API
Spring Security Reactive
Spring Security
Spring WebFlux
Spring MVC
Spring Data Reactive Repositories
Mongo, Cassandra, Redis, Couchbase
Spring Data Repositories
JDBC, JPA, NoSQL

2. Spring IoC容器

- 控制反转 （ Inversion of Control，IoC）
  - 等同于依赖注入 （Dependency Injection， DI）
  - 对象之间的依赖只能在对象创建时，通过构造方法，工厂方法或者属性注入的方式设定
  - 容器在创建对象时注入对象的依赖
  - IoC的最大收益是-松耦合，使得切片测试成为可能

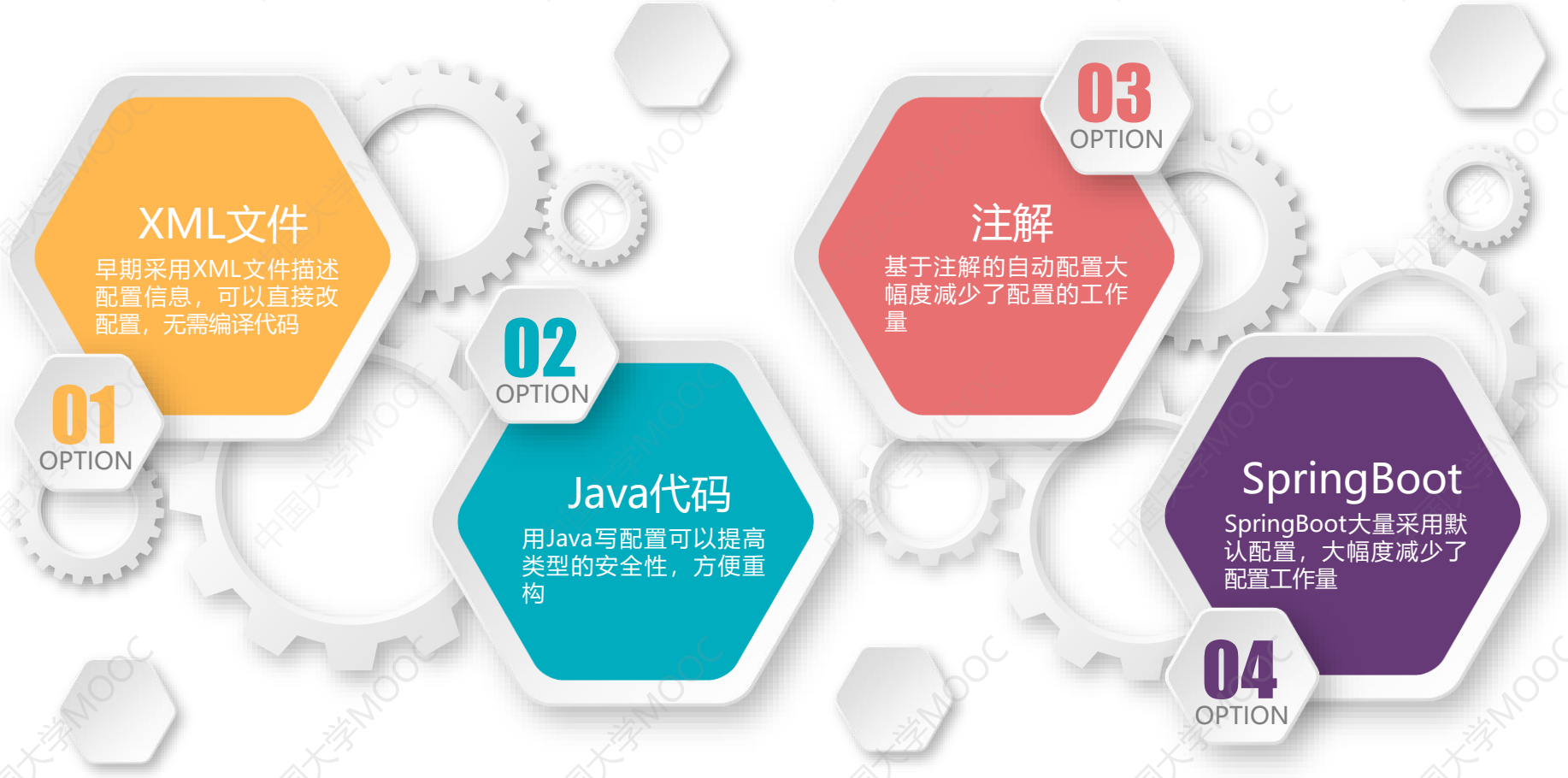
2. Spring IoC容器

- Spring IoC容器来负责创建、配置对象，并把对象关联起来提供服务。
-



2. Spring IoC容器

- Spring框架的配置



3. Spring Bean对象

- Bean对象命名
  - Bean对象的命名在Spring容器中必须唯一，默认为类名的首字母小写
  - Bean对象的Scope
    - Singleton：容器中只存在一个Bean对象。
    - Prototype：每次使用的时候，都会生成一个新的Bean对象
    - Request：为每个HTTP请求创建一个新的Bean对象
    - Session：为每个独立的session创建一个Bean对象
    - Application：类似于Singleton，但是实在ServletContext范围内唯一，而不仅限于Spring的容器范围内

-

3. Spring Bean对象

- 用注解定义Spring Bean对象
  - @Component
    - id - Bean对象的名称，id属性名称理论上可以任意命名，默认为类名且首字母小写
    - scope - singleton（默认值），prototype，request，session, application
  - @Controller, @Service， @Repository与@Component含义相同，分别用于标识Controller层，Service层，DAO层的Bean对象

-
```
@Component
@Scope("prototype")
public class Toyota implements Car{
}
```

3. Spring Bean对象

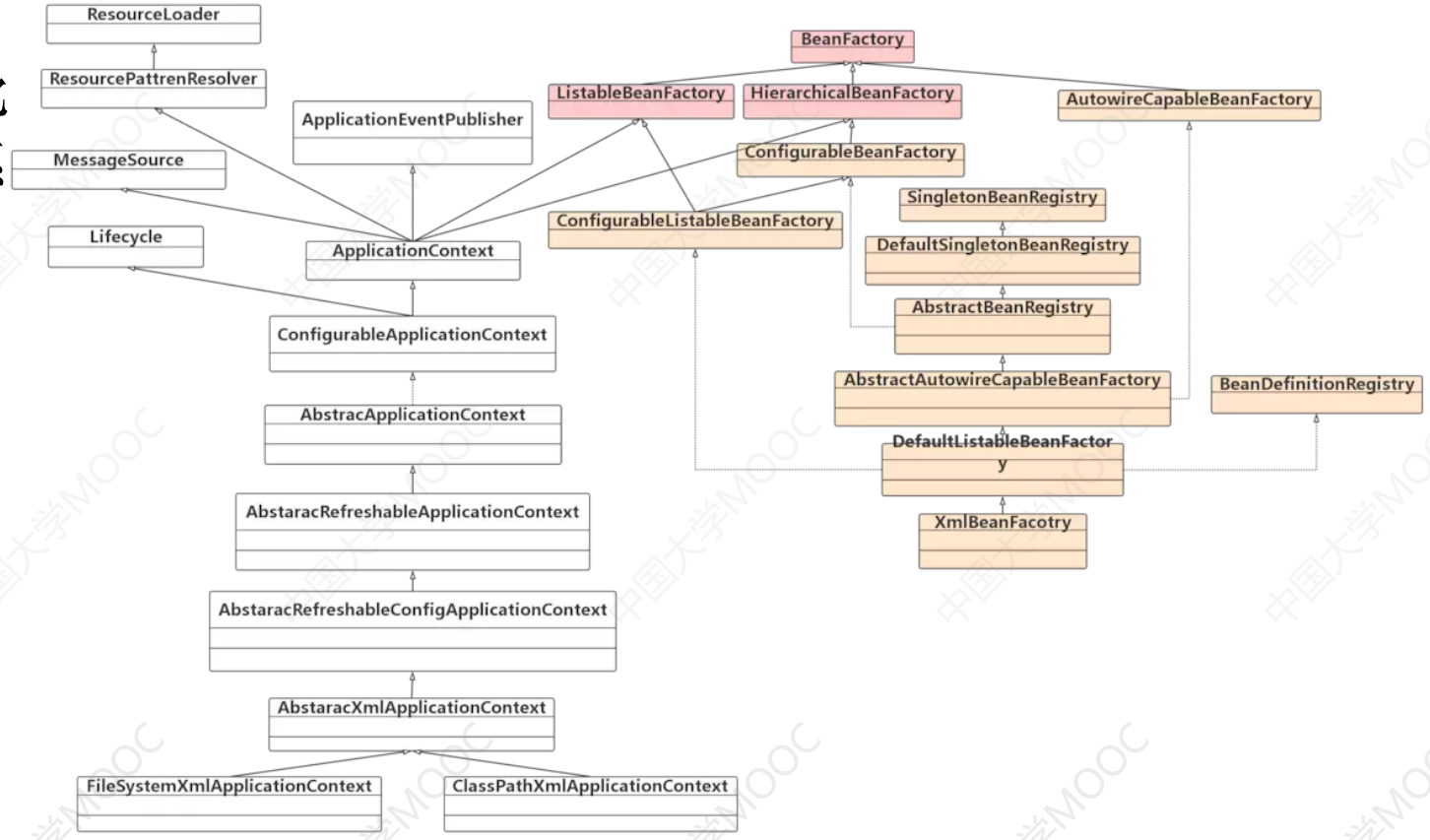
- 用Java代码定义Spring Bean对象

```
@Configuration
public class AppConfig {
```

- ```
    @Bean
    @Scope("singleton")
    public Car haval(){
        return new Haval();
    }
}.
```
```

4. BeanFactory和ApplicationContext

- Spring容器依靠BeanFactory来创建和管理Bean对象
  - 负责读取bean配置信息
  - 管理bean的加载和实例化
  - 维护bean之间的依赖关系
  - 负责bean的生命周期
-



4. BeanFactory和ApplicationContext

- ApplicationContext是BeanFactory的子接口
  - MessageSource，支持国际化的消息
  - ResourceLoader，支持URLs和文件的访问接口）
  - ApplicationEventPublisher，支持应用事件发布接口
  - 提供了多层次的上下文（Context），支持每一层专注于特定功能
-

5. 控制反转

- 控制反转是指Bean对象之间的依赖不由它们自己管理，而是由Spring容器负责管理对象之间的依赖
- Spring容器采用依赖注入（DI）的方式实现控制反转

5. 控制反转

- @Autowired时，首先在容器中查询对应类型的Bean对象
  - 如果查询结果刚好为一个，就将该bean装配给@Autowired指定的对象
  - 如果查询的结果不止一个，那么@Autowired会根据变量的名称来查找。

-

5. 控制反转

- @Autowired 标注在构造方法上

```
@Component
public class Boss_constructor {
    private Car car;
    private Office office;

    @Autowired
    public Boss_constructor(Car toyota, Office office){
        this.car = toyota;
        this.office = office;
    }
}
```

5. 控制反转

- @Autowired 标注在 Setter 方法上

```
@Component("boss_Setter")
public class Boss_Setter {

    private Car car;
    private Office office;

    @Autowired
    public void setCar(Car toyota){
        this.car = toyota;
    }

    @Autowired
    public void setOffice(Office office){
        this.office = office;
    }

}
```

5. 控制反转

- @Autowired 标注在@Bean 方法上

```
@Bean
@Autowired
public Boss boss(Car car, Office office){
    return new Boss(car, office);
}
```

5. 控制反转

- 依赖注入的过程（Singleton Scope）
  - ApplicationContext创建后即用配置数据初始化所有的Bean对象
  - 先根据Bean的构造方法参数的依赖注入对象，构造Bean对象
  - 再根据Set方法的依赖注入对象
  - 构造方法的参数注入不解决循环依赖问题

- https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-setter-injection

6. Bean对象的生命周期

- Spring Bean对象的生命周期从创建容器开始，到容器销毁Bean为止。

Bean级生命周期接口

- BeanNameAware
- BeanFactoryAware
- ApplicationContextAware
- IntializingBean
- DisposableBean

只影响一个Bean的接口

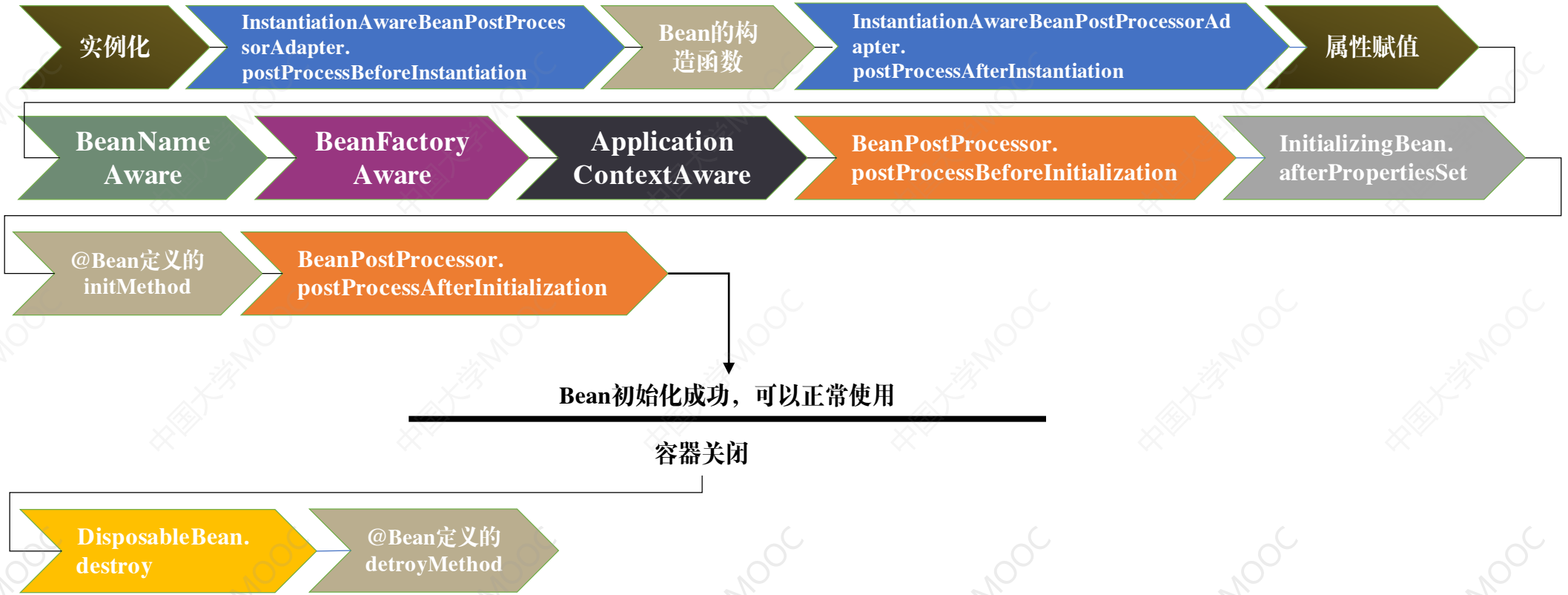
容器级生命周期接口

- InstantiationAwareBeanPostProcessorAdapter
- BeanPostProcessor

影响多个Bean的接口

6. Bean对象的生命周期

- Bean的生命周期



6. Bean对象的生命周期

- 无所不知的Aware

BeanNameAware	BeanFactoryAware	ApplicationContextAware
• void setBeanName(String beanName)	• viod setBeanFactory (BeanFactory var1) throws BeansException	• void setApplicationContext (ApplicationContext context) throws BeansException
待对象实例化并设置属性之后调用该方法设置BeanName	待调用setBeanName之后调用该方法设置BeanFactory	获得ApplicationContext

6. Bean对象的生命周期

- Bean级生命周期接口

6. Bean对象的生命周期

- 容器级生命周期接口

InstantiationAware BeanPost

- Object postProcessBeforeInstantiation (Class<?> beanClass, String beanName) throws BeansException;

在Bean对象实例化前调用

- boolean postProcessAfterInstantiation(Object bean, String beanName) throws BeansException;

在Bean对象实例化后调用

BeanPostProcessor

- Object postProcessBeforeInitialization(Object o, String s) throws BeansException;

实例化完成前

- Object postProcessAfterInitialization(Object o, String s) throws BeansException;

全部实例化完成以后调用该方法

7. SpringBoot

- SpringBoot的作用的是方便开发独立的Spring应用程序
  - 内嵌Tomcat、Jetty或Undertow
  - 采用Starter POM简化Maven的配置
  - 大量采用约定简化Spring的配置
  - 提供产品级的运行监控功能

7. SpringBoot

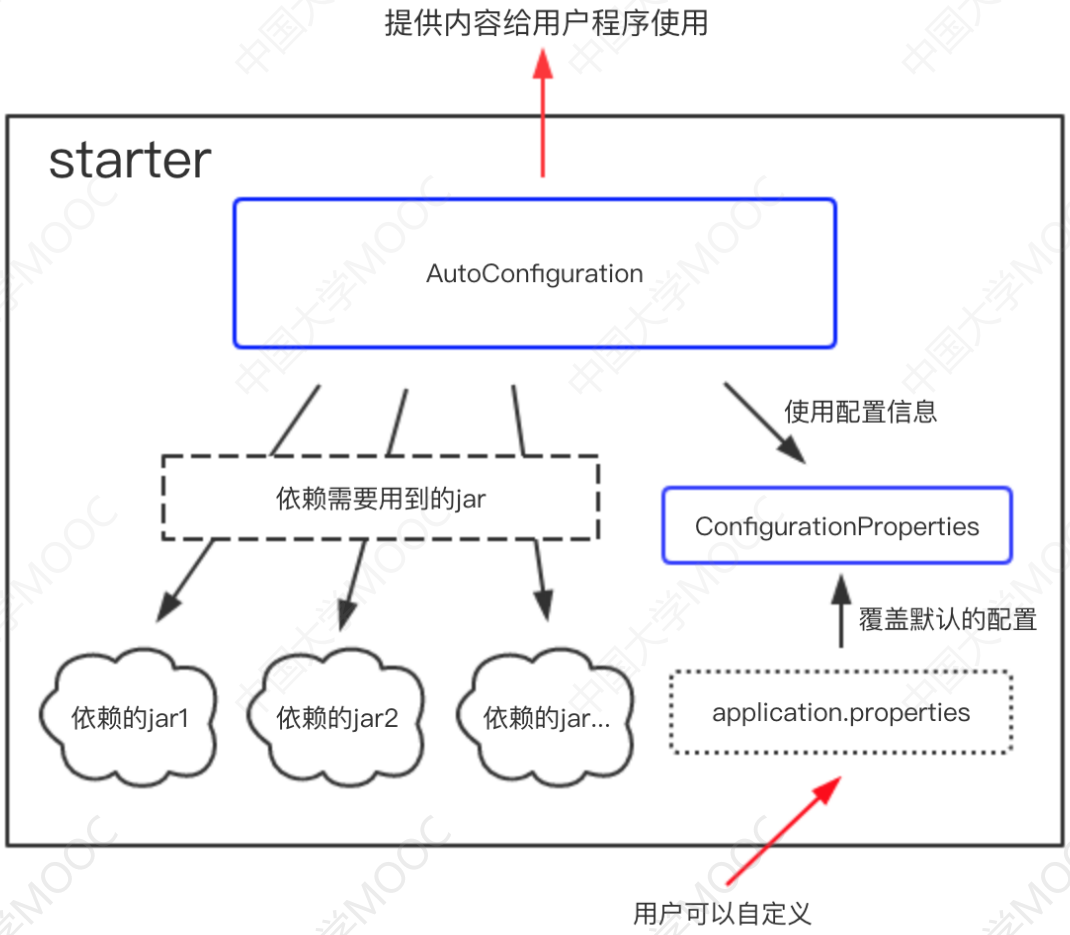
- Spring Boot 2.7.4 要求

环境	版本
Java	8+
Spring Framework	5.3.23+
Maven	3.5+
Tomcat	9.0 (Servlet 4.0)
Jetty	9.4 (Servlet 3.1)
Jetty	10.0 (Servlet 4.0)
Undertow2.0	1.3+ (Servlet 4.0)

7. SpringBoot

- Starter
  - 没有Starter之前
    - 在Maven中引入使用的库
    - 引入使用的库所依赖的库
    - 在xxx.xml中配置一些属性信息
    - 反复的调试直到可以正常运行
  - 有了Starter
    - 只需要引入一个Starter
    - starter会把所有用到的依赖都给包含进来，避免了开发者自己去引入依赖所带来的麻烦

7. SpringBoot



7. SpringBoot

- 在POM文件中定义继承Spring-boot-starter-parent

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.6.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

7. SpringBoot

- 在依赖中使用Starter来简化依赖配置

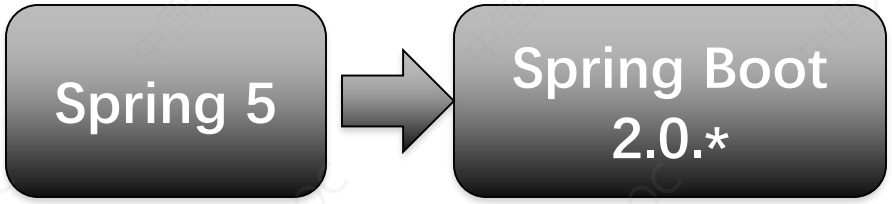
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

7. SpringBoot

- 在插件中采用SpringBoot的插件来编译打包应用

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

7. SpringBoot



spring-boot-starter	Core starter, including auto-configuration support, logging and YAML
spring-boot-starter-aop	Starter for aspect-oriented programming with Spring AOP and AspectJ
spring-boot-starter-cache	Starter for using Spring Framework's caching support
spring-boot-starter-data-redis	Starter for using Redis key-value data store with Spring Data Redis and the Lettuce client
spring-boot-starter-freemarker	Starter for building MVC web applications using FreeMarker views
spring-boot-starter-quartz	Spring Boot Quartz Starter
spring-boot-starter-security	Starter for using Spring Security
spring-boot-starter-test	Starter for testing Spring Boot applications with libraries including JUnit, Hamcrest and Mockito
spring-boot-starter-web	Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container

8. SpringBoot的配置

- Spring Boot应用的Main函数

```
@SpringBootApplication(scanBasePackages = {"cn.edu.xmu.javaee.autowiredemo.autowiredbean",
                                           "cn.edu.xmu.javaee.autowiredemo.cirualbean.set"})
public class AutowiredemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(AutowiredemoApplication.class, args);
    }

}
```

8. SpringBoot的配置

- @SpringBootApplication是一个复合注解

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@Configuration
@EnableAutoConfiguration
@ComponentScan
@interface
```

8. SpringBoot的配置

- SpringBoot使用一个全局的配置文件application.properties或者application.yml(或者是yaml)
  - 作用是修改SpringBoot自动配置的默认值；

https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

8. SpringBoot的配置

- 例如修改Servlet容器的监听端口：
  - 在application.yml中定义

```
server:
  port: 9090
```

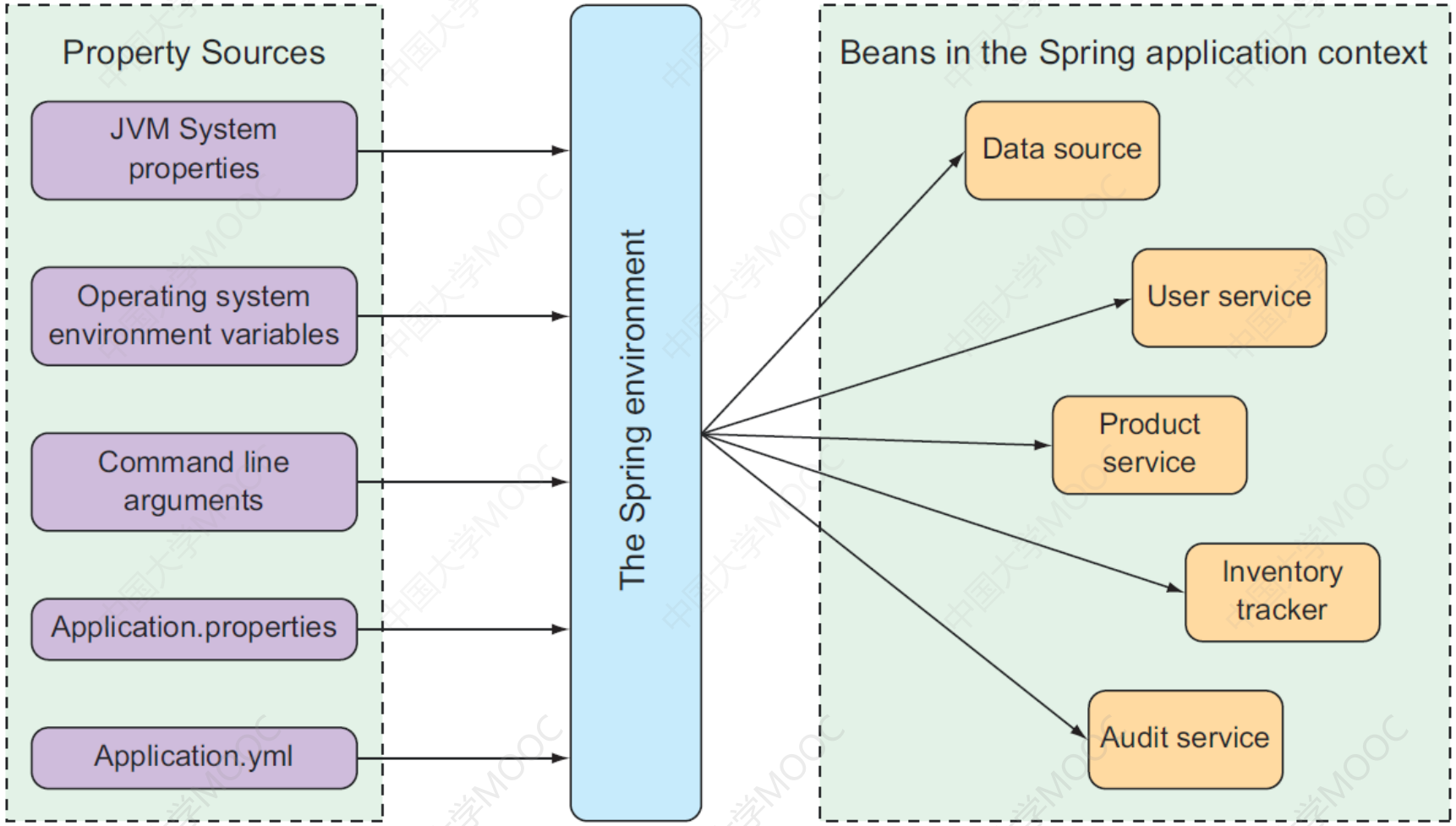
  - 在Java的命令行中定义

```
$ java -jar RestfulDemo-0.0.1-SNAPSHOT.jar --server.port=9090
```

  - 在环境变量中定义

```
$ export SERVER_PORT=9090
```

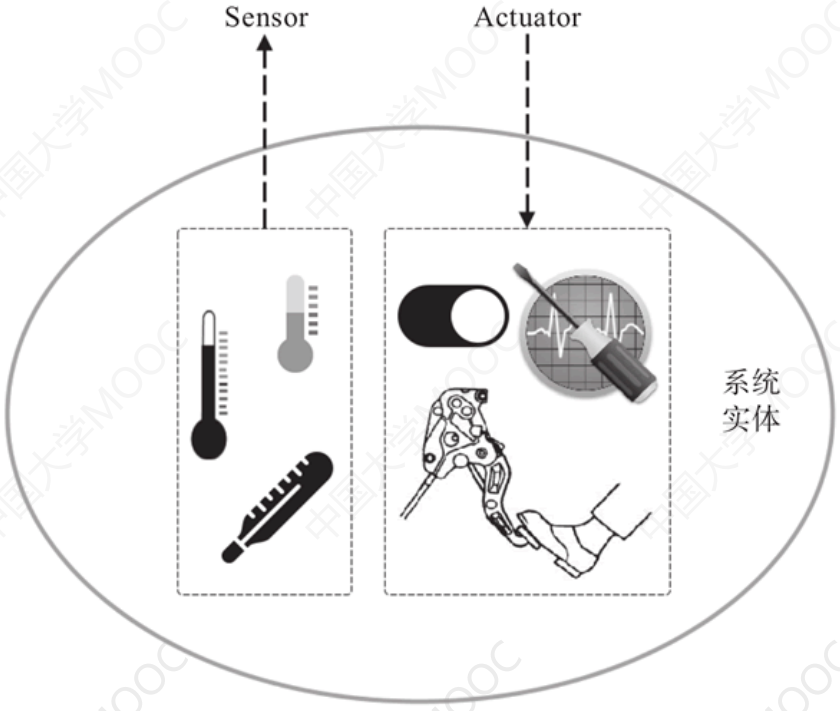
8. SpringBoot的配置



@Value

9. SpringBoot Actuator

- Spring Boot Actuator
  - 健康检查
  - 审计
  - 统计
  - 监控



9. SpringBoot Actuator

- Sensors类Endpoints

名称	说明
autoconfig	这个 endpoint 会为我们提供一份 SpringBoot 的自动配置报告，告诉我们哪些自动配置模块生效了，以及哪些没有生效，原因是什么。
beans	给出当前应用的容器中所有 bean 的信息。
configprops	对现有容器中的 ConfigurationProperties 提供的信息进行“消毒”处理后给出汇总信息。
info	提供当前 SpringBoot 应用的任意信息，我们可以通过 Environment 或者 application.properties 等形式提供以 info. 为前缀的任何配置项，然后 info 这个 endpoint 就会将这些配置项的值作为信息的一部分展示出来。
health	针对当前 SpringBoot 应用的健康检查用的 endpoint。
env	关于当前 SpringBoot 应用对应的 Environment 信息。
metrics	当前 SprinBoot 应用的 metrics 信息。
trace	当前 SpringBoot 应用的 trace 信息。
mapping	如果是基于 SpringMVC 的 Web 应用，mapping 这个 endpoint 将给出 @RequestMapping 相关信息。

9. SpringBoot Actuator

- Actuator类Endpoints
  - shutdown：用于关闭当前 SpringBoot 应用的 endpoint。
  - dump：用于执行线程的 dump 操作。