

Spring

Fernando Camargo

12 de junho de 2017

ZG Soluções

O que é Spring?

O que é Spring?

- Mais popular framework para desenvolvimento Java

O que é Spring?

- Mais popular framework para desenvolvimento ~~Java~~ na JVM

O que é Spring?

- Mais popular framework para desenvolvimento Java na JVM
- Objetivos
 - Simplificar o desenvolvimento em JEE

O que é Spring?

- Mais popular framework para desenvolvimento ~~Java~~ na JVM
- Objetivos
 - Simplificar o desenvolvimento em JEE
 - Facilitar as tarefas comuns (inúmeros módulos disponíveis)

O que é Spring?

- Mais popular framework para desenvolvimento Java na JVM
- Objetivos
 - Simplificar o desenvolvimento em JEE
 - Facilitar as tarefas comuns (inúmeros módulos disponíveis)
 - Promover boas práticas de desenvolvimento

O que é Spring?

- Mais popular framework para desenvolvimento Java na JVM
- Objetivos
 - Simplificar o desenvolvimento em JEE
 - Facilitar as tarefas comuns (inúmeros módulos disponíveis)
 - Promover boas práticas de desenvolvimento
 - Código fácil de testar e de manter

O que é Spring?

- Mais popular framework para desenvolvimento Java na JVM
- Objetivos
 - Simplificar o desenvolvimento em JEE
 - Facilitar as tarefas comuns (inúmeros módulos disponíveis)
 - Promover boas práticas de desenvolvimento
 - Código fácil de testar e de manter
- Spring Boot para começar rápido um projeto

Estratégias chaves

- Uso de POJOs
 - Leve
 - Minimamente invasivo

Estratégias chaves

- Uso de POJOs
 - Leve
 - Minimamente invasivo
- Baixo acoplamento
 - Injeção de dependências
 - Interfaces

Estratégias chaves

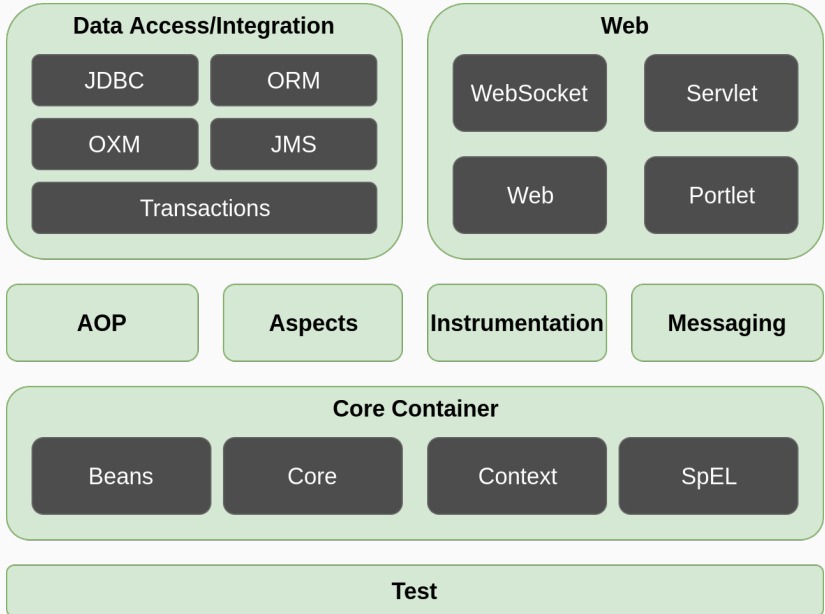
- Uso de POJOs
 - Leve
 - Minimamente invasivo
- Baixo acoplamento
 - Injeção de dependências
 - Interfaces
- Programação declarativa
 - Aspectos
 - Convenções comuns

Estratégias chaves

- Uso de POJOs
 - Leve
 - Minimamente invasivo
- Baixo acoplamento
 - Injeção de dependências
 - Interfaces
- Programação declarativa
 - Aspectos
 - Convenções comuns
- Eliminação de código "chiclê"
 - Aspectos
 - Templates

Arquitetura

Arquitectura



- **Core** (spring-core) e **Bean** (spring-beans)
 - Parte fundamental do framework
 - Inversão de Controle (IoC) e Injeção de Dependências (DI)

- **Core** (spring-core) e **Bean** (spring-beans)
 - Parte fundamental do framework
 - Inversão de Controle (IoC) e Injeção de Dependências (DI)
- **Context** (spring-context) → acesso a objetos definidos e configurados

- **Core** (spring-core) e **Bean** (spring-beans)
 - Parte fundamental do framework
 - Inversão de Controle (IoC) e Injeção de Dependências (DI)
- **Context** (spring-context) → acesso a objetos definidos e configurados
- **SpEL** (spring-expression)
 - Linguagem de Expressão (EL)
 - Busca e manipulação de um objeto

- **JDBC** (spring-jdbc) → camada de abstração sob JDBC

- **JDBC** (spring-jdbc) → camada de abstração sob JDBC
- **Transaction** (spring-tx) → gerenciamento de transações
 - Programática
 - Declarativa

- **JDBC** (spring-jdbc) → camada de abstração sob JDBC
- **Transaction** (spring-tx) → gerenciamento de transações
 - Programática
 - Declarativa
- **ORM** (spring-orm)
 - Camada de abstração para populares ORMs (JPA, Hibernate, etc.)
 - Permite integrar com outras funcionalidades do Spring, como **Transaction**

- **JDBC** (spring-jdbc) → camada de abstração sob JDBC
- **Transaction** (spring-tx) → gerenciamento de transações
 - Programática
 - Declarativa
- **ORM** (spring-orm)
 - Camada de abstração para populares ORMs (JPA, Hibernate, etc.)
 - Permite integrar com outras funcionalidades do Spring, como **Transaction**
- **OXM** (spring-oxm)
 - Camada de abstração para Mapeamento Objeto/XML
 - Suporta implementações como JAXB, Castor, XMLBeans, etc.
- **JMS** (spring-jms) → serviço de mensagens

- **Web** (spring-web)
 - Inicialização do IoC para Web
 - Utilitários Web

- **Web** (spring-web)
 - Inicialização do IoC para Web
 - Utilitários Web
- **Servlet** (spring-webmvc)
 - Spring MVC → framework web
 - REST Web Services
- **WebSocket** (spring-websocket) → suporte a WebSocket

- **Web** (spring-web)
 - Inicialização do IoC para Web
 - Utilitários Web
- **Servlet** (spring-webmvc)
 - Spring MVC → framework web
 - REST Web Services
- **Portlet** (spring-webmvc-portlet) → Spring MVC para Portlets
- **WebSocket** (spring-websocket) → suporte a WebSocket

- **AOP** (spring-aop) → implementação de Programação Orientada a Aspectos

- **AOP** (spring-aop) → implementação de Programação Orientada a Aspectos
- **Aspects** (spring-aspects) → integração com AspectJ

- **AOP** (spring-aop) → implementação de Programação Orientada a Aspectos
- **Aspects** (spring-aspects) → integração com AspectJ
- **Instrumentation** (spring-instrument)
 - Suporte a instrumentação de classes
 - Implementação de Class Loaders para certos servidores

- **AOP** (spring-aop) → implementação de Programação Orientada a Aspectos
- **Aspects** (spring-aspects) → integração com AspectJ
- **Instrumentation** (spring-instrument)
 - Suporte a instrumentação de classes
 - Implementação de Class Loaders para certo servidores
- **Test** (spring-test) → suporte a testes com JUnit e TestNG para aplicações Spring

Conceitos de base

Injeção de Dependências (DI)

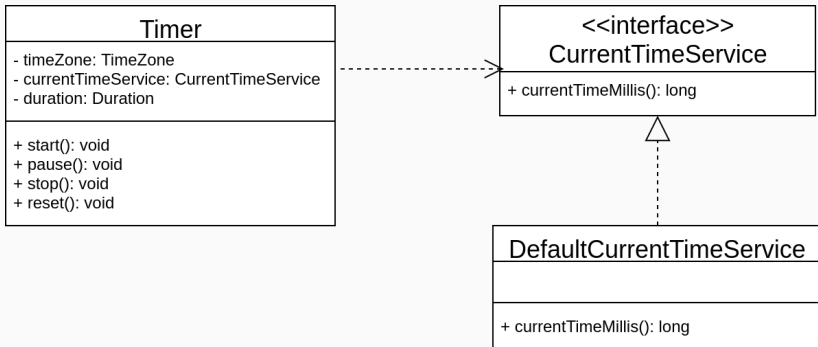
- Separa a criação de dependências de uma classe de seu comportamento

Injeção de Dependências (DI)

- Separa a criação de dependências de uma classe de seu comportamento
- Reduz o acoplamento e aumenta coesão

Injeção de Dependências (DI)

- Separa a criação de dependências de uma classe de seu comportamento
- Reduz o acoplamento e aumenta coesão



Exemplo sem injeção de dependências

```
public class Timer {

    private CurrentTimeService currentTimeService;
    private TimeZone timeZone;

    public Timer(){
        currentTimeService = new DefaultCurrentTimeService();
        timeZone = TimeZone.getDefault();
    }

    public void start(){
        // Faz uso de CurrentTimeService e TimeZone
    }

    // ...

}

// Uso do Timer
Timer timer = new Timer();
timer.start();
```

Exemplo com injeção de dependências

```
public class Timer {

    private CurrentTimeService currentTimeService;
    private TimeZone timeZone;

    public Timer(CurrentTimeService currentTimeService, TimeZone timeZone){
        currentTimeService = currentTimeService;
        timeZone = timeZone;
    }

    public void start(){
        // Faz uso de CurrentTimeService e TimeZone
    }

    // ...

}

// Uso do Timer
CurrentTimeService currentTimeService = new DefaultCurrentTimeService();
TimeZone timeZone = TimeZone.getDefault();

Timer timer = new Timer(currentTimeService, timeZone);
timer.start();
```

Exemplo de teste com injeção de dependências

```
public class Timer {

    private CurrentTimeService currentTimeService;
    private TimeZone timeZone;

    public Timer(CurrentTimeService currentTimeService, TimeZone timeZone){
        currentTimeService = currentTimeService;
        timeZone = timeZone;
    }

    public void start(){
        // Faz uso de CurrentTimeService e TimeZone
    }

    // ...

}

// Teste do Timer
StubCurrentTimeService currentTimeService = new StubCurrentTimeService();
TimeZone timeZone = TimeZone.getTimeZone("America/Sao_Paulo");

Timer timer = new Timer(currentTimeService, timeZone);
timer.start();

// Ajuste de tempo simulado
currentTimeService.tick(5, TimeUnit.SECONDS);
```

Injeção de Dependências (DI)

- Mesmo a injeção de dependências mais básica (sem frameworks) já apresenta benefícios

Injeção de Dependências (DI)

- Mesmo a injeção de dependências mais básica (sem frameworks) já apresenta benefícios
- Quem vai criar toda estrutura de objetos?

Injeção de Dependências (DI)

- Mesmo a injeção de dependências mais básica (sem frameworks) já apresenta benefícios
- Quem vai criar toda estrutura de objetos?
- Existem muitos frameworks de injeção de dependência
 - Spring
 - CDI/EJB
 - Guice
 - Dagger
 - etc.

Injeção de Dependências (DI)

- Mesmo a injeção de dependências mais básica (sem frameworks) já apresenta benefícios
- Quem vai criar toda estrutura de objetos?
- Existem muitos frameworks de injeção de dependência
 - Spring
 - CDI/EJB
 - Guice
 - Dagger
 - etc.
- Um container é responsável pela construção dos objetos (e suas dependências)
 - Main: "Me dê um Timer"
 - Container: "Só um momento, o Timer tem duas dependências, vou criá-las como configurado e te retornar um Timer totalmente funcional."

- Princípio de Hollywood: "Não nos chame, nós o chamaremos"

Inversão de Controle (IoC)

- Princípio de Hollywood: "Não nos chame, nós o chamaremos"
- Tradicional: código criado possui todas responsabilidades

Inversão de Controle (IoC)

- Princípio de Hollywood: "Não nos chame, nós o chamaremos"
- Tradicional: código criado possui todas responsabilidades
 - Constrói suas dependências

Inversão de Controle (IoC)

- Princípio de Hollywood: "Não nos chame, nós o chamaremos"
- Tradicional: código criado possui todas responsabilidades
 - Constrói suas dependências
 - Gerencia seu próprio ciclo de vida

Inversão de Controle (IoC)

- Princípio de Hollywood: "Não nos chame, nós o chamaremos"
- Tradicional: código criado possui todas responsabilidades
 - Constrói suas dependências
 - Gerencia seu próprio ciclo de vida
 - Executa lógica de negócios

Inversão de Controle (IoC)

- Princípio de Hollywood: "Não nos chame, nós o chamaremos"
- Tradicional: código criado possui todas responsabilidades
 - Constrói suas dependências
 - Gerencia seu próprio ciclo de vida
 - Executa lógica de negócios
- IoC: separação de responsabilidades
 - Container gerencia ciclo de vida, além de contruir e injetar dependências
 - Código criado executa lógica de negócios

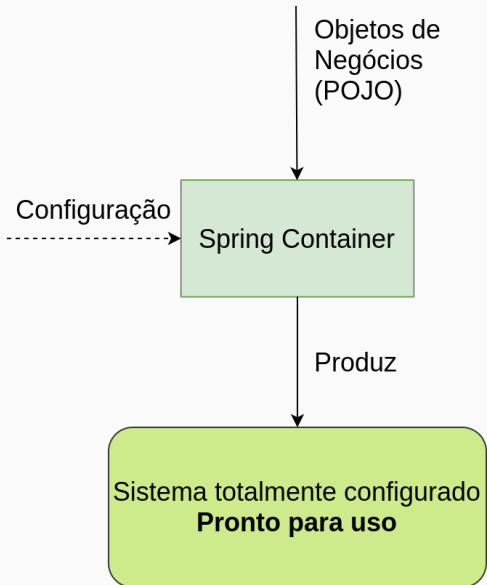
Todo objeto da aplicação, gerenciado pelo Container do Spring, é chamado **bean**.



Todo objeto da aplicação, gerenciado pelo Container do Spring, é chamado **bean**.



Spring IoC



- **BeanFactory**

- Avançado mecanismo de configuração capaz de gerenciar qualquer tipo de objeto

- **BeanFactory**

- Avançado mecanismo de configuração capaz de gerenciar qualquer tipo de objeto
- Responsável pela criação e injeção de beans

- **BeanFactory**

- Avançado mecanismo de configuração capaz de gerenciar qualquer tipo de objeto
- Responsável pela criação e injeção de beans

- **ApplicationContext**

- Sub-interface de **BeanFactory**, adicionando:
 - Integração mais fácil com funcionalidades do Spring AOP

- **BeanFactory**

- Avançado mecanismo de configuração capaz de gerenciar qualquer tipo de objeto
- Responsável pela criação e injeção de beans

- **ApplicationContext**

- Sub-interface de **BeanFactory**, adicionando:
 - Integração mais fácil com funcionalidades do Spring AOP
 - Internacionalização de mensagens

- **BeanFactory**

- Avançado mecanismo de configuração capaz de gerenciar qualquer tipo de objeto
- Responsável pela criação e injeção de beans

- **ApplicationContext**

- Sub-interface de **BeanFactory**, adicionando:
 - Integração mais fácil com funcionalidades do Spring AOP
 - Internacionalização de mensagens
 - Publicação de eventos

- **BeanFactory**

- Avançado mecanismo de configuração capaz de gerenciar qualquer tipo de objeto
- Responsável pela criação e injeção de beans

- **ApplicationContext**

- Sub-interface de **BeanFactory**, adicionando:
 - Integração mais fácil com funcionalidades do Spring AOP
 - Internacionalização de mensagens
 - Publicação de eventos
 - Contextos específicos para a camada de aplicação (**WebApplicationContext** para Web, por exemplo)

- **BeanFactory**

- Avançado mecanismo de configuração capaz de gerenciar qualquer tipo de objeto
- Responsável pela criação e injeção de beans

- **ApplicationContext**

- Sub-interface de **BeanFactory**, adicionando:
 - Integração mais fácil com funcionalidades do Spring AOP
 - Internacionalização de mensagens
 - Publicação de eventos
 - Contextos específicos para a camada de aplicação (**WebApplicationContext** para Web, por exemplo)
- Representa o Container

- **BeanFactory**

- Avançado mecanismo de configuração capaz de gerenciar qualquer tipo de objeto
- Responsável pela criação e injeção de beans

- **ApplicationContext**

- Sub-interface de **BeanFactory**, adicionando:
 - Integração mais fácil com funcionalidades do Spring AOP
 - Internacionalização de mensagens
 - Publicação de eventos
 - Contextos específicos para a camada de aplicação (**WebApplicationContext** para Web, por exemplo)
- Representa o Container
- Obtém configurações via metadados: XML, Annotations ou Código de configuração

Implementações do ApplicationContext

- ClassPathXmlApplicationContext
- FileSystemXmlApplicationContext
- XmlWebApplicationContext
- AnnotationConfigApplicationContext
- AnnotationConfigWebApplicationContext

- Configuração explícita via XML

- Configuração explícita via XML
- Configuração explícita via Java

- Configuração explícita via XML
- Configuração explícita via Java
- Configuração implícita via descoberta e conexão automática de beans

- **Component scanning:** automaticamente descobre beans a serem criados no `ApplicationContext`

- **Component scanning:** automaticamente descobre beans a serem criados no `ApplicationContext`
- **Autowiring:** automaticamente satisfaz dependências dos beans

Exemplo Bean auto descoberto e injetado

```
@Service
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Autowired
    public SimpleMovieLister(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...

}
```

Habilitando auto descoberta via Java

```
@Configuration
@ComponentScan("org.example")
public class AppConfig {
    // ...
}
```

Habilitando auto descoberta via XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd">

    <context:component-scan base-package="org.example"/>

</beans>
```

Definindo Beans via Annotations

- `@Component("beanName")`

Definindo Beans via Annotations

- `@Component("beanName")`
 - `@Service("beanName")`

Definindo Beans via Annotations

- `@Component(" beanName")`
 - `@Service(" beanName")`
 - `@Repository(" beanName")`

Definindo Beans via Annotations

- `@Component(" beanName")`
 - `@Service(" beanName")`
 - `@Repository(" beanName")`
 - `@Controller(" beanName")`

Definindo Beans via Annotations

- `@Component(" beanName")`
 - `@Service(" beanName")`
 - `@Repository(" beanName")`
 - `@Controller(" beanName")`
- `@Named(" beanName")` → anotação definida pelo CDI e suportada pelo Spring
- Bean detectado possuirá um nome que o identifique

Definindo Beans via Annotations

- `@Component(" beanName")`
 - `@Service(" beanName")`
 - `@Repository(" beanName")`
 - `@Controller(" beanName")`
- `@Named(" beanName")` → anotação definida pelo CDI e suportada pelo Spring
- Bean detectado possuirá um nome que o identifique
- Se nome não especificado, utiliza-se nome da classe (primeira letra minúscula)

Definindo dependências via Annotations

- @Autowired

Definindo dependências via Annotations

- **@Autowired**
 - Injeção via Construtor: todos parâmetros são injetados

Definindo dependências via Annotations

- @Autowired
 - Injeção via Construtor: todos parâmetros são injetados
 - Injeção via Setter: setters anotados usados na injeção **após** **construção**

Definindo dependências via Annotations

- @Autowired

- Injeção via Construtor: todos parâmetros são injetados
- Injeção via Setter: setters anotados usados na injeção **após construção**
- Injeção via Propriedade: propriedades anotadas são injetadas **após construção**

Definindo dependências via Annotations

- @Autowired

- Injeção via Construtor: todos parâmetros são injetados
- Injeção via Setter: setters anotados usados na injeção **após construção**
- Injeção via Propriedade: propriedades anotadas são injetadas **após construção**

- @Inject

- Anotação definida pelo CDI e suportada pelo Spring

Definindo dependências via Annotations

- @Autowired

- Injeção via Construtor: todos parâmetros são injetados
- Injeção via Setter: setters anotados usados na injeção **após construção**
- Injeção via Propriedade: propriedades anotadas são injetadas **após construção**

- @Inject

- Anotação definida pelo CDI e suportada pelo Spring
- Mesmo funcionamento de @Autowired, mas com limitações

Definindo dependências via Annotations

- **@Autowired**
 - Injeção via Construtor: todos parâmetros são injetados
 - Injeção via Setter: setters anotados usados na injeção **após construção**
 - Injeção via Propriedade: propriedades anotadas são injetadas **após construção**
- **@Inject**
 - Anotação definida pelo CDI e suportada pelo Spring
 - Mesmo funcionamento de **@Autowired**, mas com limitações
- **@Value("property.name")**
 - Injeta valores carregador de arquivos .properties

Definindo dependências via Annotations

- **@Autowired**
 - Injeção via Construtor: todos parâmetros são injetados
 - Injeção via Setter: setters anotados usados na injeção **após construção**
 - Injeção via Propriedade: propriedades anotadas são injetadas **após construção**
- **@Inject**
 - Anotação definida pelo CDI e suportada pelo Spring
 - Mesmo funcionamento de **@Autowired**, mas com limitações
- **@Value("property.name")**
 - Injeta valores carregador de arquivos .properties
 - Mesmos métodos de injeção do **@Autowired**

Injeção via Construtor

```
@Service
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Autowired
    public SimpleMovieLister(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...

}
```

Injeção via Setter

```
@Service
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Autowired
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...

}
```

Injeção via Propriedade

```
@Service
public class SimpleMovieLister {

    @Autowired
    private MovieFinder movieFinder;

    // ...

}
```

- @Autowired de uma interface/classe abstrata:
 - Se existir uma única implementação, ela é utilizada
 - Se existirem múltiplas implementações, é necessário uma especificação: @Qualifier

@Qualifier com beanName

```
@Service
public class MainMovieCatalog implements MovieCatalog {
    // ...
}
```

```
@Service
public class MovieRecommender {

    @Autowired
    @Qualifier("mainMovieCatalog")
    private MovieCatalog movieCatalog;

    // ...

}
```

@Qualifier com característica

```
@Service
@Qualifier("main")
public class MainMovieCatalog implements MovieCatalog {
    // ...
}
```

```
@Service
public class MovieRecommender {

    @Autowired
    @Qualifier("main")
    private MovieCatalog movieCatalog;

    // ...

}
```

@Qualifier com Custom Annotation

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Main {

}

@Service
@Main
public class MainMovieCatalog implements MovieCatalog {
    // ...
}

@Service
public class MovieRecommender {

    @Autowired
    @Main
    private MovieCatalog movieCatalog;

    // ...
}
```


@Qualifier com Custom Annotation e valor

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}

@Service
@Genre("action")
public class ActionMovieCatalog implements MovieCatalog {
    // ...
}

@Service
public class MovieRecommender {

    @Autowired
    @Genre("action")
    private MovieCatalog movieCatalog;

    // ...
}
```

@Qualifier com Custom Annotation e valores

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface MovieQualifier {
    String genre();
    String format();
}

@Service
@MovieQualifier(genre="action", format="dvd")
public class ActionMovieCatalog implements MovieCatalog {
    // ...
}

@Service
public class MovieRecommender {

    @Autowired
    @MovieQualifier(genre="action", format="dvd")
    private MovieCatalog movieCatalog;

    // ...
}
```

- Além de auto detectados, beans também podem ser declarados
 - Via XML
 - Via Java (@Bean)

Registrando Beans diretamente

- Além de auto detectados, beans também podem ser declarados
 - Via XML
 - Via Java (@Bean)
- Essas declarações também incluem injeção de dependência, referenciando outros beans

Registrando Beans via XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">

  <bean id="accountRepository" class="com.acme.AccountRepositoryImpl" />

  <bean id="transferService" class="com.acme.TransferServiceImpl">
    <constructor-arg ref="accountRepository" />
  </bean>

</beans>
```

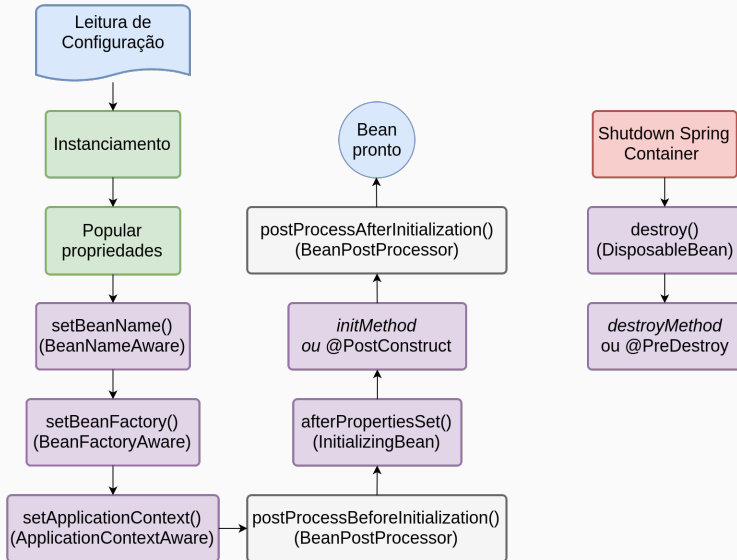
Registrando Beans via Java

```
@Configuration
public class AppConfig {

    @Bean
    public AccountRepository accountRepository() {
        new AccountRepositoryImpl();
    }

    @Bean
    public TransferService transferService(AccountRepository accountRepository) {
        return new TransferServiceImpl(accountRepository);
    }
}
```

Ciclo de Vida



Escopo	Descrição
Singleton (Padrão)	Única instância criada e mantida pelo Container
Prototype	Múltiplas instâncias criadas
Request	Única instância criada por requisição
Session	Única instância criada por sessão


```
@Service
@Scope("prototype")
public class MovieRecommender {

    @Autowired
    private MovieCatalog movieCatalog;

    // ...

}
```

Conclusões

- Spring IoC disponibiliza um Container com Inversão de Controle/Injeção de Dependências

- **Spring IoC** disponibiliza um Container com Inversão de Controle/Injeção de Dependências
- **Spring Boot** agiliza a configuração de aplicações Spring, além de trazer funcionalidades úteis

- **Spring IoC** disponibiliza um Container com Inversão de Controle/Injeção de Dependências
- **Spring Boot** agiliza a configuração de aplicações Spring, além de trazer funcionalidades úteis
- **Spring MVC** permite criação de Controllers que simplificam o mapeamento de requisições

- **Spring IoC** disponibiliza um Container com Inversão de Controle/Injeção de Dependências
- **Spring Boot** agiliza a configuração de aplicações Spring, além de trazer funcionalidades úteis
- **Spring MVC** permite criação de Controllers que simplificam o mapeamento de requisições
- Spring possui uma variedade de projetos para resolver diversos problemas comuns