Spring Boot:

Este artigo é útil para entendermos o que é uma arquitetura baseada em micro serviços e aprender como implementar uma aplicação baseada neste padrão com a ferramenta que será apresentada, o Spring Boot. Além disso, também será apresentado um novo modo de desenvolvimento voltado para a plataforma Java EE que possivelmente influenciará toda a comunidade de desenvolvedores Java.

Apesar de ser uma ferramenta poderosa e que aumenta significativamente a produtividade na escrita de aplicações corporativas, o Spring Framework ainda é alvo de críticas a respeito do tempo necessário para se iniciar o desenvolvimento de novos projetos. Pensando nisso, foi introduzido no Spring 4.0 um novo projeto que tem, dentre seus objetivos, responder a estas críticas e, como veremos neste artigo, também mudar bastante nossa percepção acerca do desenvolvimento de aplicações para a plataforma Java EE. Este projeto é o Spring Boot.

A principal crítica feita ao Spring é sobre o modo como configuramos o seu container de injeção de dependências e inversão de controle usando arquivos de configuração no formato XML. Artefatos estes que, conforme aumentam de tamanho, se tornam cada vez mais difíceis de serem mantidos, muitas vezes se transformando em um gargalo para a equipe de desenvolvimento. No decorrer da história do framework vimos que este problema foi sendo tratado a partir de uma série de melhorias no modo como declaramos nossos beans a cada novo release: namespaces na versão 1.2, anotações na versão 2.0 e, finalmente, passamos a poder tratar arquivos XML como um artefato opcional no lançamento da versão 3.0, que nos trouxe a possibilidade de declarar nossos beans usando apenas código Java e anotações.

Outra crítica relevante diz respeito à complexidade na gestão de dependências. Conforme nossos projetos precisam interagir com outras bibliotecas e frameworks como, por exemplo, JPA, mensageria, frameworks de segurança e tantos outros, garantir que todas as bibliotecas estejam presentes no classpath da aplicação acaba se tornando um pesadelo. Não é raro encontrarmos em projetos baseados em Maven arquivos POM nos quais 90% do seu conteúdo sejam apenas para gestão de dependências. E esta é apenas a primeira parte do problema. O grande desafio surge quando precisamos integrar todos estes componentes.

O projeto Spring Boot (ou simplesmente Boot) resolve estas questões e ainda nos apresenta um novo modelo de desenvolvimento, mais simples e direto, sem propor novas soluções para problemas já resolvidos, mas sim alavancando as tecnologias existentes presentes no ecossistema Spring de modo a aumentar significativamente a produtividade do desenvolvedor.

O Que É O Spring Boot?

Trata-se de mais um framework, mas talvez a melhor denominação seja micro framework. Como mencionado na introdução deste artigo, seu objetivo não é trazer novas soluções para problemas que já foram resolvidos, mas sim reaproveitar estas tecnologias e aumentar a produtividade do desenvolvedor. Como veremos mais à frente, trata-se também de uma excelente ferramenta que podemos adotar na escrita de aplicações que fazem uso da arquitetura de micro serviços.

Se pudéssemos desenhar um diagrama arquitetural do Spring Boot, este seria muito similar ao que vemos no Grails: uma fina camada sobre tecnologias já consagradas pelo mercado, tal como podemos verificar na **Figura 1**. A grande mudança está no modo como agora empacotamos e acessamos estas soluções.

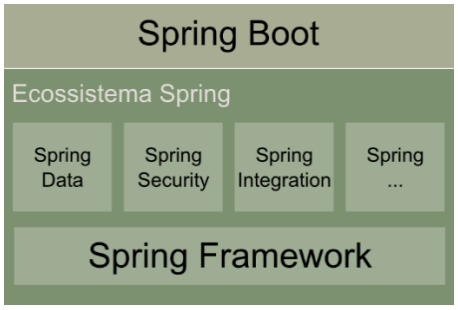

Figura 1. Posicionamento do Spring Boot no ecossistema Spring.

O desenvolvedor não precisa se preocupar em aprender novas tecnologias, pois todo o conhecimento adquirido sobre o ecossistema Spring é reaproveitado. A principal diferença se dá no modo como configuramos, organizamos nosso código e executamos a aplicação, tal como veremos neste artigo.

Como sabemos, todo framework se baseia em alguns princípios. No caso do Boot, são quatro:

1. Prover uma experiência de início de projeto (getting started experience) extremamente rápida e direta;

2. Apresentar uma visão bastante opinativa (opinionated) sobre o modo como devemos configurar nossos projetos Spring, mas ao mesmo tempo flexível o suficiente para que possa ser facilmente substituída de acordo com os requisitos do projeto;

3. Fornecer uma série de requisitos não funcionais já pré-configurados para o desenvolvedor como, por exemplo, métricas, segurança, acesso a base de dados, servidor de aplicações/servlet embarcado, etc.;

4. Não prover nenhuma geração de código e minimizar a zero a necessidade de arquivos XML.

Uma Visão Opinativa Sobre A Configuração?

Na documentação oficial do projeto Boot, assim como em posts a seu respeito, encontraremos muitas vezes o termo opinionated view (visão opinativa). Ao citá-lo, os responsáveis pelo desenvolvimento da ferramenta na realidade estão se referindo ao conceito de convenção sobre configuração, porém levemente modificado.

O conceito de convenção sobre configuração é o grande motor por trás do ganho de produtividade do Spring Boot, porém não foi algo introduzido por ele. Frameworks como Ruby on Rails e Grails já o aplicam há bastante tempo. A ideia é bastante simples: dado que a maior parte das configurações que o desenvolvedor precisa escrever no início de um projeto são sempre as mesmas, por que já não iniciar um novo projeto com todas estas configurações já definidas?

Pense no modo como estamos habituados a trabalhar, por exemplo, em um projeto baseado em Spring MVC. Nossos primeiros passos serão incluir as dependências necessárias no projeto, adequar o arquivo web.xml e organizar a estrutura do nosso código fonte para que possamos iniciar o desenvolvimento. E isto é feito no início de todo projeto. Sendo assim, por que não já começar com isto pronto?

Voltando nossa atenção para esse termo, é importante prestar atenção na palavra **sobre** em “convenção **sobre**configuração”. Note que não é “convenção **ao invés** de configuração”. Não temos uma imposição aqui, mas sim sugestões. Deste modo, se seu projeto requer um aspecto diferente daquele definido pelas convenções do framework, o programador precisa alterar apenas aqueles locais nos quais a customização se aplica.

Indo além na análise dos termos mencionados, percebemos que no Spring Boot não foi usado o termo “convenção sobre configuração”, mas sim “visão opinada sobre configuração”. Dito isso, você pode estar se perguntando: Qual a diferença entre eles? A diferença está no fato da equipe de desenvolvimento do Spring Boot assumir que algumas escolhas tomadas baseiam-se em aspectos subjetivos e não estritamente técnicos. Um exemplo é a escolha da biblioteca de log. Foi adotado o Log4J, no entanto o Commons Logging seria uma opção igualmente competente. Sendo assim, por que um ao invés do outro? Em grande parte, preferências pessoais.

Há outra grande vantagem na adoção deste princípio. A partir do momento em que a equipe conhece as convenções, torna-se menor o tempo necessário para que novos membros se adaptem ao projeto e a manutenção passa a ser mais simples.

Spring Scripts

O projeto Spring Boot nos permite criar dois tipos de aplicações: as tradicionais, escritas primariamente em Java; e uma segunda forma, chamada pela equipe de desenvolvimento do Boot de “Spring Scripts”, que nada mais são do que códigos escritos em Groovy – o que não é de se estranhar, dado

que desde o lançamento da versão 4.0 do framework há uma forte tendência da Pivotal em abraçar esta linguagem de programação e torná-la cada vez mais presente no dia a dia do programador Java.

Para ilustrar esse tipo de aplicação, vamos escrever um “Olá mundo!” bastante simples que nos conduzirá na apresentação de alguns conceitos fundamentais por trás do Spring Boot. Todo o código fonte do nosso projeto pode ser visto na **Listagem 1**.

Listagem 1. “Olá mundo” com Spring Scripts – OlaMundo.groovy.

```
  @RestController
class OlaMundo {
   @RequestMapping("/")
   String home() {
       "Olá mundo!"
   }
}
```

Conheçendo o Spring Boot!

O que é Spring Boot

Se você programa em JAVA provavelmente já usou ou ouviu falar sobre o Spring Framework. Atualmente na versão 4.x é uma ferramenta poderosa, madura e com uma comunidade gigantesca por trás. Mas e o Spring Boot ? Bom, já vou chegar lá.

Conforme o Spring foi evoluindo, suas configurações necessárias foram crescendo, o que antes era um simples XML começou a parecer a jornada de Frodo(Senhor dos Anéis) para ser configurado. Outro problema comum era a quantidade de dependências que era preciso gerenciar, aquele arquivo pom.xml cada vez maior não era nada legal.

A própria equipe do Spring junto com feedbacks da comunidade perceberam a necessidade de tornar o seu poderoso framework mais ágil, e então surgiu o Spring Boot!

Em versões anteriores do Spring Framework, mesmo sem o “Boot” já era possível criar suas Beans e até mesmo sua aplicação inteira sem a a necessidade de XML’s. Tudo poderia ser feito por anotações e código Java.

O que o Spring Boot fez foi juntar todas essa facilidades que encontraram ao longo de sua evolução em um só lugar. Podemos dizer que o Spring juntou suas próprias tecnologias de forma a reaproveita-la e tornar a sua vida mais produtiva na hora de desenvolver.

Rápido E Leve

Me lembro que à alguns anos atrás, mesmo desenvolvendo com Spring MVC, algo que muito me incomodava era a hora da subir meu serviço para testar(mesmo localmente). Jboss, Glassfish, Weblogic e tantos outros Server’s JAVA EE. Muitas vezes era aquela tortura para configurar, cada um com seu “jeito” de configuração, XML’s gigantescos, e etc… Bom, esses tempos acabaram depois que conheci o Spring Boot! Um dos princípais pontos que gosto do Spring Boot é o fato dele ser “Embedded”, ou seja, o nosso server application está dentro da nossa aplicação, e não se assuste, esse server pode ser um Tomcat embedded ou até mesmo um Jetty. Sendo assim fica tão leve e rápido como uma Robbit!

Não tenho nada contra Server’s JAVA EE, mas a complexidade desses server’s pode acabar atrapalhando a agilidade de desenvolvimento, e o seu uso pode não ser necessário dependendo da arquitetura que iremos trabalhar(será que sempre vou precisar de um server EE ? Acho que não rsrs).

Talvez você se pergunte, “mas meu Jboss eu configuro do jeito que eu quero!”, “No Weblogic eu parametrizo o que eu quiser” e muitas outras justificativas. Agora, será que precisa configura tudo isso pra começar a desenvolver? Será que o único modo de escalar bem seus serviços é usando Servers EE? Tantas configurações ajudam ou atrapalham? É de se penssar…

Claro que existe uma curva de aprendizado para utilizar o Spring Boot, e o que quero mostrar aqui é que essa curva não é tão curva assim rsrs, vai valer apena acredite! E alias, essas configurações que fazemos em outros servers também conseguimos fazer no Jetty ou Tomcat(lembra que são embedded, então não é magica, existe um server rodando ali). A diferença é que essas configurações são muito mais enxutas e sem tantas complexidades.

Outro ponto bem legal, como podemos usar nosso server embedded, basta dar um start em nosso JAR e sua aplicação estará de pé. Sim, JAR e não WAR, podemos gerar um WAR e colocar ele dentro de um server qualquer se quisermos, mas porque faríamos isso se podemos simplesmente dar o comando java -jar gandalf.jar e tudo funciona, pois nosso server está junto com nossa aplicação.

E Se Eu Precisar Usar Outras Bibliotecas, Acessar Dados, Exportar Relatórios E Etc ?

É legal frisar que por traz do Spring Boot existe uma comunidade chamada “SPRING” e esses caras são poderosos. Existem inúmeros projetos da Spring.io e todos se conversam muito bem entre sí, seria super tranquilo fazer uma aplicação com Spring Boot usando o Spring Data para acessar um MongoDB por exemplo (O Alexandre Queiroz escreveu sobre Spring Data e MongoDB aqui no blog recentemente, vale conferir). Não gosta do Spring Data? Sem problemas, você pode usar as bibliotecas que quiser dentro do seu projeto, alias, muitas vezes você vai precisar.

Mão Na Massa

Agora que já falei bastante sobre Spring Boot, que tal construirmos algo? (E sem XML, a não ser nosso pom é claro)

Vamos montar uma calculadora Rest, teremos os serviços para somar, subtrair, multiplicar e dividir. A logica é bem simples não é? Agora, se precisasse configurar seu container, gerar seu arquivo WAR para fazer deploy, configurar seus XML’s, properties e tudo mais… Acredito que toda essa configuração levaria muito mais tempo do que implementar a própria calculadora. Então vamos ver como fica com Spring Boot.

Vou usar o eclipse para construir o projeto, mas podem usar a IDE que melhor preferirem.

Vamos começar criando um projeto Maven simples, e então vamos mexer no arquivo pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLoc
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.lucas.spring.calculadora</groupId>
  <artifactId>CalcRest</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
      <java.version>1.8</java.version>
  </properties>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.1.RELEASE</version>
  </parent>

  <dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>

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

vejam que adicionei o parent para o spring Boot:

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.4.1.RELEASE</version>
</parent>
```

E a dependência spring-boot-starter-web do Spring Boot

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

Essa é a dependência que contem tudo o que precisamos para iniciar nosso projeto Web.

OBS: Coloquei a versão 8 do Java para nosso projeto e a tag build com o plugin do Spring Boot para o maven.

Vamos criar 2 pacotes em nosso projeto, um para conter noss Main Application e outro para conter nosso Controller onde iremos mapear os serviços da nossa calculadora. No meu caso criei o pacote com.lucas.spring.calculadora e com.lucas.spring.calculadora.controller

Dentro do pacote com.lucas.spring.calculadora vamos criar a classe Application

```
package com.lucas.spring.calculadora;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ApplicationContext ctx = SpringApplication.run(Application.class, args)

    }

}
```

Repare que é uma simples classe main Java, e não precisamos mais do que isso para disponibilizar nossa aplicação.

A anotação @SpringBootApplication configura algumas coisas defaults do Spring para nós, ela inclui automaticamente as anotações:

@Configuration – classifica a classe como uma bean de configurações para a aplicação.

@EnableAutoConfiguration – habilita a auto configuração do Spring baseado em suas convenções.

@ComponentScan – Busca componentes, serviços, configurações que estão em nosso pacote(ou sub-pacote em nosso caso)

@EnableWebMvc – seta nossa aplicação como uma Web Application para o Spring

Por fim com uma única linha de código damos um “RUN” em nossa própria classe. Basta rodar nossa classe Main como uma aplicação JAVA comum, e por default o spring Boot vai iniciar o Tomcat Embedded na porta 8080 (é muito rápido, acredite!).

No pacote com.lucas.spring.calculadora.controller vamos criar a classe CalcController

```
package com.lucas.spring.calculadora.controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/calculadora")
public class CalcController {

    @RequestMapping(value ="/soma", method = RequestMethod.GET)
    public Double soma(@RequestParam("valA") Double valA, @RequestParam("valB") Double valB) {

        return valA + valB;
    }

    @RequestMapping(value ="/subtrai", method = RequestMethod.GET)
    public Double subtrai(@RequestParam("valA") Double valA, @RequestParam("valB") Double valB) {

        return valA - valB;
    }

    @RequestMapping(value ="/multiplica", method = RequestMethod.GET)
    public Double multiplica(@RequestParam("valA") Double valA, @RequestParam("valB") Double valB) {

        return valA * valB;
    }

    @RequestMapping(value ="/divide", method = RequestMethod.GET)
    public Double divide(@RequestParam("valA") Double valA, @RequestParam("valB") Double valB) {

        return valA / valB;
    }

}
```

Aqui implementamos a logica da nossa calculadora e mapeamos as requisições, poderiamos dividir entre regras de negócio e mapeamento do nosso resource mas não é nosso foco.

A anotação @RestController habilita minha classe como controller e faz com que cada método seja anotado como @ResponseBody (Podemos falar melhor dessas anotações em outro artigo mais detalhado, por enquanto o importante é saber que essa anotação é responsável por habilitar o mapeamento de nossos métodos).

A anotação @RequestMapping mapeia literalmente nosso método, passamos o nome com que ele sera chamado e qual método de request ele é (GET, POST…), no nosso caso mapeei todos os métodos como GET.

Por fim a anotação @RequestParam apenas mapeia os parâmetros que quero receber na requisição(valor A e valor B) para efetuar minhas operações(Podemos detalhar melhor essa anotação em outro artigo também).

Pronto, Calculadora Construída! Vamos Subir E Testar ?

Para testar vamos executar por linha de comando(no meu caso estou usando linux). Para isso, precisamos estar com o JAVA 8 e o Maven instalado.

Na pasta onde se encontra o projeto vamos executar o comando: mvn clean install

Se tudo ocorrer com sucesso será gerado o arquivo CalcRest-0.0.1-SNAPSHOT.jar na pasta target, é esse jar que iremos usar. Ainda na pasta do seu projeto entre com o comando java -jar target/CalcRest-0.0.1-SNAPSHOT.jar

O Spring Boot vai iniciar nossa aplicação com o Tomcat Embedded na porta 8080, feito isso teremos nossos calculadora mapeada, basta acessar pelo browser mesmo e nossas operações serão executadas, exemplo:

- http://localhost:8080/calculadora/soma?valA=2&valB=3
- http://localhost:8080/calculadora/subtrai?valA=2&valB=3
- http://localhost:8080/calculadora/multiplica?valA=2&valB=3
- http://localhost:8080/calculadora/divide?valA=2&valB=3

Começando Com A Magia Do Spring Boot

O primeiro passo, como não poderia ser diferente, é a criação do projeto. Você pode realizar este passo no próprio SetupMyProject. O zip vai vir com o mínimo configurado, mas é aí que você já vai perceber a diferença. Para conseguir subir o servidor e acessar a primeira rota, só precisamos de uma classe configurada no projeto.

```
@SpringBootApplication
@Controller
public class CasadocodigoSpringbootApplication {

    @RequestMapping("/")
    @ResponseBody
    public String index(){
        return "funciona?";
    }

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

Aqui já temos um pouco de mágica. Você vai rodar sua aplicação web a partir de um bom e velho método main. A classe SpringApplication vai ler as configurações da classe passada como argumento e pronto, você tem a sua aplicação no ar. A outra parte da mágica acontece por conta da annotation @SpringBootApplication. Ela é um Sterotype do Spring Boot que já encapsula algumas outras

annotations, como a @EnableAutoConfiguration. Essa última, por sua vez, carrega a AutoConfigurationPackages que é responsável por configurar os pacotes que devem ser escaneados, baseados na localização da sua classe.

Um outro ponto bem impressionante é o pom.xml criado, você vai perceber que ele possui pouquíssimas dependências! Vamos dar uma rápida olhada.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.0.BUILD-SNAPSHOT</version>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

Perceba que você adiciona algumas dependências que eles chamam de starters. A grande sacada é fazer com esses artefatos já baixem tudo que você precisa. E aqui, eu preciso dizer que, pelo menos para mim, eles tomaram uma decisão bastante acertada. Ao invés de te dar a opção de escolher entre vários frameworks, servidores web, libs específicas e etc, eles tomaram algumas decisões e empurraram pra gente. Para a maioria dos projetos isso não muda e, se for preciso, você pode sobreescrever tudo que eles fizeram. A motivação deles foi a mesma que tivemos ao construir o SetupMyProject. Já estamos cheios de frameworks, precisamos é de uma maneira mais fácil de usá-los.

Por mais que já tenhamos o projeto configurado, provavelmente esse não é bem o modelo que vamos seguir. Os nossos controllers não vão ser declarados dentro da mesma classe de configuração. O normal é criar uma outra classe que representa este seu controller.

```
    @Controller
@Transactional
@RequestMapping("/produtos")
public class ProductsController {

    @Autowired
    private ProductDAO products;
    @Autowired
    private FileSaver fileSaver;
    @Autowired
    private ServletContext ctx;
    @Autowired
    private InternalResourceViewResolver resolver;

    @RequestMapping(method=RequestMethod.GET)
    @Cacheable(value="lastProducts")
    public ModelAndView list() throws ServletException, IOException{
        System.out.println("ola");
        ModelAndView modelAndView = new ModelAndView("products/list");
        modelAndView.addObject("products", products.findAll());
        return modelAndView;
    }

    ...

}
```

E agora é só pedir para escanear o pacote… Opa, não precisa mais! Como eu criei esse classe em um pacote abaixo do pacote da classe de configuração, todas as classes já vão ser escaneadas de maneira automática. Agora é necessário configurar o acesso a JPA e, nesse momento, nossos corações vão ficar felizes novamente. O nosso único trabalho é adicionar um starter no pom.xml.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Quase tudo que você acha que deveria fazer para configurar o acesso ao banco com a JPA, vai ser feito automagicamente. Não será necessário configurar persistence.xml, EntityManagerFactory, EntityManager, PlatformTransactionManager nem nada, tudo vai ser feito pra você. A única coisa que você precisa entregar a ele é o DataSource com os dados de acesso ao banco. Podemos criar um método com essa configuração na própria classe que contém nosso main.

```
@SpringBootApplication
    public class CasadocodigoSpringbootApplication {
        @Bean
        public DataSource dataSource(Environment environment) {
            DriverManagerDataSource dataSource = new DriverManagerDataSource();
            dataSource.setDriverClassName("com.mysql.jdbc.Driver");
            dataSource.setUrl("jdbc:mysql://localhost:3306/casadocodigo");
            dataSource.setUsername("root");
            dataSource.setPassword("");
            return dataSource;
        }

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

Além disso, ele também já inclui as dependências para a Spring Data JPA, para facilitar a criação dos seus DAOs. Para completar essa primeira parte da migração do projeto, foi necessário fazer uma pequena configuração extra para os jsps. O Spring Boot sugere que você use outra tecnologia de view, mas como o jsp era a utilizada no projeto do livro, eu resolvi ir com ela mesmo. Como o tomcat está sendo executado de forma embedded, é necessário que adicionemos uma nova dependência no pom, que é a do compilador de jsps para este tipo de cenário.

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope>
</dependency>
```

Além disso, como é procedimento normal, precisamos configurar um prefixo e um sufixo para busca da nossa view. Podemos fazer isso do jeito tradicional, adicionando um método que retorna um InternalResourceViewResolver. Só que podemos aproveitar a oportunidade para já apresentar um outro detalhe que você pode tirar proveito, que são os arquivos de configuração externos. Você pode criar um arquivo chamado application.properties na raiz do seu classpath, geralmente em src/main/resources. Lá você pode colocar configurações específicas das tecnologias utilizadas no projeto, abaixo segue o meu exemplo.

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp

#hibernate
spring.jpa.hibernate.ddl-auto=update
```

Perceba que ele é bem diferente de um spring-context.xml da vida. Você não tem que declarar um bean para ser usado, simplesmente configura as propriedades do bean já escolhido.

Para fechar com chave de ouro, pelo menos para mim :), eles deram mais um passo pensando na vida fácil do usuário. Nessa semana eles lançaram um novo starter chamado de DevTools. Uma facilidade adicionada foi a de recarregamento do projeto para cada mudança que você faz no código. Só que, como eles mesmos disseram, esse recarregamento é mais esperto do que um simples hot deploy de um servidor web tradicional. Além disso, ainda adicionaram o suporte para o Live Reload, que possibilita o recarregamento automático das páginas no seu navegador sempre que você alterar uma classe sua no servidor!

Ufa! Falei bastante hoje :). Acho o Spring Boot um projeto muito útil e que, com certeza, você devia dar uma olhada. É um approach que por muito eu esperava no mundo Java e espero que influencie outros projetos a fazer o mesmo. Por fim, acho importante lembrar que você deve continuar procurando o motivo do acontecimento das coisas, para não ficar um “programador de Spring Boot” ao invés de um programador Java.