Francisco das Chagas Torres dos Santos

Setembro de 2022

Neste material, meu objetivo é ir além do gabarito da questões. Nos comentários, eu aprofundo mais no assunto, sem me distanciar do que a questão está pedindo. Isso me dá mais conhecimento sobre o assunto e me faz ver a questão abordada por diferentes ângulos.

Quando há bastantes questões sobre o assunto, opto pro questões de provas do ano corrente e das bancas CESPE, FCC e FGV. No entanto, em alguns caso, é necessário pegar provas de anos anteriores e de outras bancas.

Devemos esquecer questões muito antigas, pois TI atualiza a todo instante e as bancas também podem mudar a pegada, a forma de cobrar o conteúdo.

Frameworks Java

Map Struct - Questões

---

1. Ano: 2022 Banca: FCC Órgão: TRT-23 MT Prova: Analista Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

Para definir um mapeador de bean com MapStruct deve-se definir uma interface Java com os métodos de mapeamento necessários e anotá-la com a anotação

a) @MapStruct
b) @EntityMapper
c) @GeneratedMapper
d) @Mapper
e) @BeanMapping

Resposta: Letra D

Comentários

Com @Mapper se mapea a interface e não precisa criar a classe de implementação , pois o Mapstruct cria.

```
@Mapper
public interface SimpleSourceDestinationMapper {
   SimpleDestination sourceToDestination(SimpleSource source);
   SimpleSource destinationToSource(SimpleDestination destination);
}
```

Para mais, consulte:

- https://www.baeldung.com/mapstruct

---

2. Ano: 2022 Banca: FCC Órgão: TRT-22 PI Prova: Analista Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

Em uma aplicação, considere a existência de uma classe chamada Advogado com o atributo *nome* e outra classe *AdvogadoDto* com o atributo *nomeCompleto*. Usando MapStruct, uma Analista criou a interface abaixo.

```
___I___
public interface AdvogadoMapper {
    ___II___ (target = "nomeCompleto", source = "nome")
    AdvogadoDto advogadoToAdvogadoDto(Advogado advogado);
}
```

Para que o mapeamento seja realizado corretamente, as lacunas I e II devem ser preenchidas, correta e respectivamente, por

a) @MapStruct e @Map
b) @Mapper e @Equity
c) @Mapper e @Mapping
d) @Mapping e @Equals
e) @MapStruct e @Mappings

Resposta: Letra C

Comentários

O @Mapper mapea a interface, logo não precisa criar a classe de implementação, pois o Mapstruct cria. Já o @Mapping mapea nomes de campos diferente.

```
@Mapper
public interface EmployeeMapper {

   @Mapping(target="employeeId", source="entity.id")
   @Mapping(target="employeeName", source="entity.name")
   EmployeeDTO employeeToEmployeeDTO(Employee entity);

   @Mapping(target="id", source="dto.employeeId")
   @Mapping(target="name", source="dto.employeeName")
   Employee employeeDTOtoEmployee(EmployeeDTO dto);
}
```

Spring, Spring Boot, Spring Cloud, Spring Eureka, Spring Zuul – Questões

---

1. Ano: 2022 Banca: FCC Órgão: TRT-22 PI Prova: Analista Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

As versões mais recentes do Spring Boot permitem a configuração de inicialização lenta (lazy initialization), que faz com que os beans sejam criados à medida que são necessários e não durante a inicialização do aplicativo. Uma das maneiras de habilitar a inicialização lenta é colocar

a) lazy-initialization=true no arquivo spring-application-builder.xml
b) @SpringMainLazyInitialization na classe SpringApplicationBuilder
c) @Lazy(true) no arquivo application.properties
d) spring.main.lazy-initialization=true no arquivo application.properties
e) set-lazy-initialization=true no arquivo spring-boot.xml

Resposta: Letra D

Spring Boot é para fugir de XML e não encontrei os arquivos spring-application-builder.xml e spring-boot.xml. Existe o arquivo pom.xml que é o "coração" de um projeto Maven, onde se gerencia as dependências e centraliza documentação sobre o projeto. Portanto, estão fora as letras A e E.

A anotação @SpringMainLazyInitialization não existe, portanto, fora letra B. Podemos dizer que é inicialização lenta com a anotação @Lazy na classe, no construtor, no método ou no campo @Autowired. Ou seja, o @Lazy fica no arquivo Java e não em application.properties. Portanto, fora letra C.

Dessa forma, a resposta é a letra D que afirma que devemos setar como true a propriedade spring.main.lazy-initialization no arquivo application.properties. Isso significa que todos os beans usarão a inicialização lenta.

Para mais sobre o assunto: https://www.baeldung.com/spring-boot-lazy-initialization

---

2. Ano: 2022 Banca: FCC Órgão: TRT-22 PI Prova: Técnico Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

Um Técnico deseja incluir as configurações abaixo para serem executadas pelo Spring Boot quando a aplicação for iniciada.

spring.application.name = spring-cloud-config-server
server.port=8888
spring.cloud.config.server.git.uri = file:///c:/Users/test/config-files

Estas configurações devem ser inseridas no arquivo

a) spring-boot.xml
b) spring-boot.properties
c) spring-application.xml
d) spring-server.properties
e) application.properties

Resposta: Letra E

No arquivo application.properties colocamos as configurações do banco de dados:

1. spring.datasource.url = jdbc:postgresql://localhost:5432/parking-control-db
2. spring.datasource.username = postgres
3. spring.datasource.password = banco123
4. spring.jpa.hibernate.ddl-auto = update
5. spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation = true

Também colocamos as configurações do Eureka Server:

1. server.port = 8761
2. eureka.client.registrer-with-eureka=false
3. eureka.client.fetch-registry=false

As configurações de rotas do Zuul:

1. zuul.prefix
2. zuul.ignored-services
3. zuul.routes.customers.path
4. zuul.routes.customers.serviceId
5. zuul.routes.products.strip-prefix

Para mais: https://www.tutorialspoint.com/spring_boot/spring_boot_application_properties.htm

---

3. Ano: 2022 Banca: FCC Órgão: TRT-22 PI Prova: Técnico Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

À classe principal da aplicação Spring Boot, um Técnico adicionou a anotação @EnableEurekaServer para fazer com que a aplicação atue como um servidor Eureka (Discovery Server). Em seguida, adicionou ao arquivo de configuração Maven pom.xml uma dependência, como mostrado abaixo.

<dependency>
__I__
</dependency>

Para que a dependência adicionada seja do servidor Spring Cloud Eureka, a lacuna I deve ser corretamente preenchida por

a) <packageName>org.springframework.cloud</packageName>

<resourceName>spring-cloud-eureka-server</resourceName>
b) <packageName>org.springframework.cloud</packageName>
<resourceName>spring-cloud-dependencies</resourceName>
c) <groupId>spring.cloud.framework</groupId>
<artifactId>eureka-discovery-server</artifactId>
d) compile('org.springframework.cloud:spring-cloud-starter-eureka-server')
e) <groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka-server</artifactId>

Resposta: Letra E

A única dependência a ser adicionada para configurar o Eureka é a spring-cloud-netflix-eureka-server, que já carrega outras dependências necessárias para inicializar o Spring Boot no projeto (spring-bootstarter), portanto, letra E.

---

4. Ano: 2022 Banca: FCC Órgão: TRT-23 MT Prova: Analista Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

A base do contêiner Inversion of Control (IoC), também conhecido como Dependency Injection (DI), do Spring Framework, é formada pelos pacotes

a) org.springframework.beans e org.springframework.context
b) org.springframework.orm e org.springframework.jdbc
c) org.springframework.web e org.springframework.webmvc
d) org.springframework.core e org.springframework.expression
e) org.springframework.webmvc e org.springframework.websocket

Resposta: Letra A

Os pacotes org.springframework.beans e org.springframework.context são a base para o contêiner IoC do Spring Framework. A interface BeanFactory oferece um mecanismo de configuração avançado capaz de gerenciar qualquer tipo de objeto e ApplicationContext é uma subinterface de BeanFactory. Acrescenta:

- Integração mais fácil com os recursos AOP do Spring
- Manipulação de recursos de mensagens (para uso em internacionalização)
- Publicação do evento
- Contextos específicos da camada de aplicativo, como o WebApplicationContext para uso em aplicativos da Web.

O BeanFactory fornece a estrutura de configuração e a funcionalidade básica e ApplicationContext adiciona mais funcionalidades específicas da empresa.

Para mais: https://docs.spring.io/spring-framework/docs/current/reference/html/core.html

Pacotes do Spring Framework:

- **org.springframework.beans** - contém interfaces e classes para manipulação de Java beans.

- **org.springframework.context** - se baseia no pacote beans para incluir suporte para origens de mensagens e para o padrão de design Observer, e a capacidade de objetos de aplicativo para obter recursos usando uma API consistente.
- **org.springframework.core** - Fornece classes básicas para tratamento de exceção e detecção de versão e outros auxiliares principais que não são específicos de nenhuma parte da estrutura.
- **org.springframework.orm** - Pacote raiz para as classes de integração de mapeamento O/R do Spring.
- **org.springframework.jdbc** - As classes neste pacote tornam o JDBC mais fácil de usar e reduzem a probabilidade de erros comuns.
- **org.springframework.expression** - As principais abstrações por trás da Spring Expression Language.
- **org.springframework.web** - Interfaces comuns e genéricas que definem pontos de limite mínimos entre a infraestrutura web do Spring e outros módulos de estrutura.
- **org.springframework.beans.factory** - O pacote principal que implementa o contêiner leve de Inversão de Controle (IoC) do Spring.

Os pacotes org.springframework.webmvc e org.springframework.websocket não fazem parte do Spring

Para mais: https://docs.spring.io/spring-framework/docs/current/javadoc-api/overview-summary.html

GitHub: https://github.com/spring-projects/spring-framework

---

5. Ano: 2022 Banca: FCC Órgão: TRT-23 MT Prova: Analista Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

Considere a classe abaixo, criada utilizando Spring Boot, em condições ideais.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@EnableAutoConfiguration
public class Trt {
	@RequestMapping("/")
	String home() {
		return " TRIBUNAL REGIONAL DO TRABALHO DA 23ª REGIÃO";
	}
	public static void main(String[] args) {
		SpringApplication.run( __I__ , args);
	}
}
```

Para dizer à classe SpringApplication do Spring Boot qual é o componente primário do Spring, a lacuna I deve ser preenchida corretamente com

a) SpringBoot.class
b) Trt.class

c) SpringBoot.start[Trt.class]
d) Trt.app
e) SpringBoot.main

Resposta: Letra B

No método main, em StringApplication.run é passado junto com args a própria classe, ficando portanto SpringApplication.run(Trt.class, args) já que estamos na classe Trt.

---

6. Ano: 2022 Banca: FCC Órgão: TRT-23 MT Prova: Analista Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

Considere o código fonte a seguir.

```
public class Application {
	public static void main(String[] args) {
		new SpringApplicationBuilder(Application.class).web(true).run(args);
	}
}
```

Em condições ideais, para este código ser de um servidor Eureka mínimo, antes da declaração da classe devem ser colocadas as anotações

a) @EnableNetflixEureka e @EnableDiscoveryClient
b) @EnableEurekaConfiguration e @EnableRegistryServer
c) @SpringBootServer e @SpringBootEureka
d) @SpringBootApplication e @EnableEurekaServer
e) @EurekaDiscoveryClient e @EurekaServiceRegistry

Resposta: Letra D

Para tornar a aplicação um Eureka Server basta adicionar a anotação @EnableEurekaServer em conjunto com a anotação do Spring Boot (@SpringBootApplication) que no startup da aplicação será inicializado o Eureka.

A classe de configuração é igual para qualquer aplicação Spring Boot, apenas adicionando a anotação @EnableDiscoveryClient para fazer o registro da aplicação no Eureka.

---

7. Ano: 2022 Banca: FCC Órgão: TRT-23 MT Prova: Analista Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

Para fazer um aplicativo criado com Spring Boot, em condições ideais, funcionar como um servidor Zuul Proxy deve-se anotar a classe principal com

(A) @EnableZuulProxy
(B) @ZuulServerApplication
(C) @EnabledZuulEdgeServer

(D) @ZuulEdgeServerOn
(E) @ZuulProxyApplication

Resposta: Letra A

A anotação @EnableZuulProxy ativa o Zuul na classe de configuração do spring Boot, logo o Zuul é inicializado no startup da aplicação.

---

8. Ano: 2022 Banca: FCC Órgão: TRT-23 MT Prova: Analista Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

O mecanismo de configuração automática baseado em convenção do Spring Boot pode ser iniciado anotando a classe que contém o método principal estático, com a anotação

a) @SpringMainClass
b) @SpringBootComponent
c) @SpringBootApplication
d) @SpringRootApplication
e) @SpringBootEntity

Resposta: Letra C

A questão fala um pouco confuso, mas vamos lá: o método principal estático é o public static void main() e este método fica na classe que vai ser iniciada, logo estamos falando de @SpringbootApplication que é a anotação que diz que a classe é o ponto de entrada de uma aplicação Spring Boot. Esta anotação marca a classe principal da aplicação Spring Boot e encapsula as seguintes anotações: @Configuration, @EnableAutoConfiguration e @ComponentScan com seus atributos padrão.

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

---

9. Ano: 2022 Banca: FCC Órgão: TRT-23 MT Prova: Analista Judiciário Área Apoio Especializado Especialidade Tecnologia da Informação

Para gerenciar a configuração de um panorama do sistema de microsserviços, o Spring Cloud contém o Spring Cloud Config, que fornece o gerenciamento centralizado de arquivos de configuração. O Spring Cloud Config

a) oferece suporte à criptografia de informações confidenciais na configuração, exceto de credenciais.
b) suporta o armazenamento de arquivos de configuração em repositório Git, por exemplo, no GitHub.
c) não permite separar as partes comuns da configuração das partes específicas de microsserviços.
d) não suporta o armazenamento de arquivos de configuração em uma base de dados banco de dados JDBC.
e) não suporta o armazenamento de arquivos de configuração em um sistema de arquivos local (local filesystem).

Resposta: Letra B

As demais questões tem exceto, não suporta, não permite, restando como correta apenas a letra B.

O Spring Cloud Config fornece suporte do lado do servidor e do lado do cliente para configuração externa em um sistema distribuído. Com o Config Server, você tem um local central para gerenciar propriedades externas para aplicativos em todos os ambientes.

O Spring Cloud Config Server fornece uma API baseada em recursos HTTP para configuração externa (pares nome-valor ou conteúdo YAML equivalente). O servidor pode ser incorporado em um aplicativo Spring Boot, usando a anotação @EnableConfigServer. Consequentemente, o seguinte aplicativo é um servidor de configuração:

Como todos os aplicativos Spring Boot, ele é executado na porta 8080 por padrão, mas você pode alterná-lo para a porta 8888 mais convencional de várias maneiras. O mais fácil, que também define um repositório de configuração padrão, é iniciá-lo com spring.config.name=configserver (há um configserver.yml no jar do Config Server). Outra é usar seu próprio application.properties:

application.properties

```
server.port: 8888
spring.cloud.config.server.git.uri: file://${user.home}/config-repo
```

Onde ${user.home}/config-repo é um repositório git contendo YAML e arquivos de propriedades.

Para mais: https://cloud.spring.io/spring-cloud-config/reference/html/
