Spring Framework

com Spring Boot

O Spring é um framework de integração que disponibiliza serviços corporativos para aplicações baseadas em POJOs

As principais características:

* não intrusivo

* container flexível

* integração com os frameworks mais populares do mercado

Injeção de Dependência

Declarando um bean no container

```
@Component //singleton
public class EmployeeServiceImpl

                    implements EmployeeService {

    public void create (Employee employee) {...}

}
```

Declarando um bean no container

```
@Service //singleton
public class EmployeeServiceImpl

                    implements EmployeeService {

    public void create (Employee employee) {...}

}
```

Declarando um bean no container

```
@Repository //singleton
public class DepartmentRepositoryImpl

            implements DepartmentRepository {

   public Department findById (Long id) {...}

}
```

Declarando as dependências

```
@Service
public class EmployeeServiceImpl

                        implements EmployeeService {


   @Autowired
   private EntityManager entityManager;

}
```

Declarando as dependências

```
@Service
public class EmployeeServiceImpl

                    implements EmployeeService {


   @Autowired
   public EmployeeServiceImpl (EntityManager em){

      ...

   }
```

Declarando as dependências

```
@Service
public class EmployeeServiceImpl

                    implements EmployeeService {

    @Autowired
    @Qualifier ("employeeJdbcRepository")
    private EmployeeRepository repository;

}
```

Callbacks de ciclo de vida

```
@Service
public class EmployeeServiceImpl

                        implements EmployeeService {



    @PostConstruct
    public void init () { ... }

}
```

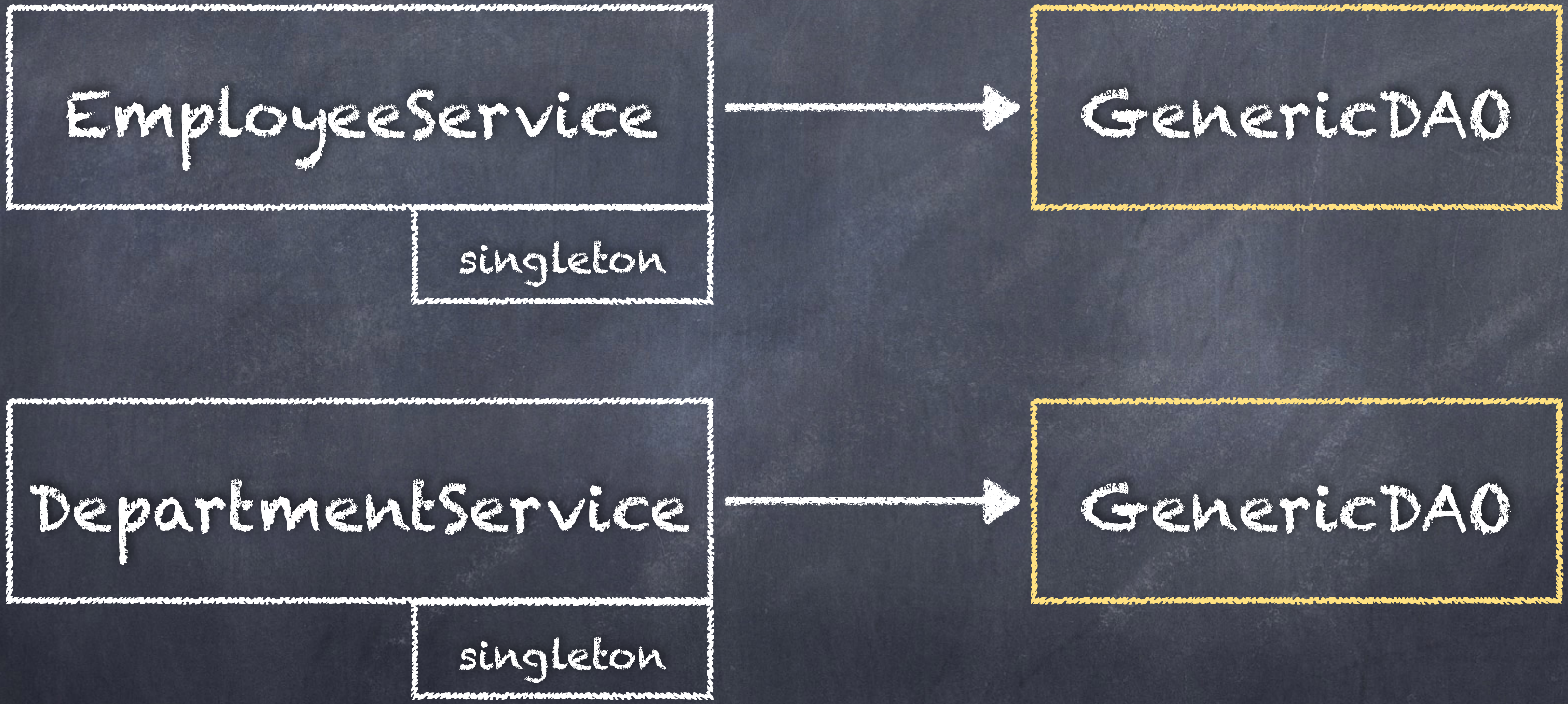
Callbacks de ciclo de vida

```
@Service
public class EmployeeServiceImpl

                    implements EmployeeService {



    @PreDestroy
    public void cleanUp () { ... }

}
```

Definindo o escopo de um bean

```
@Repository
@Scope (name="prototype")
public class GenericDAOImpl

                implements GenericDAO {

    public <T> void create (T obj) {...}

}
```

Prototype Scope



Definindo o escopo de um bean

```
@Component
@Scope ("request")  //somente na WEB
public class ManagedBean {

    public String getName () {...}

    public void setName (String name) {...}

}
```

Definindo o escopo de um bean

```
@Component
@Scope ("session") //somente na WEB
public class ManagedBean {

    public String getName () {...}

    public void setName (String name) {...}

}
```

Definindo o escopo de um bean

```
@Component
@Scope ("application")   //somente na WEB
public class ManagedBean {

    public String getName () {...}

    public void setName (String name) {...}

}
```

```
<html>

    ...

    <h:outputText value="#{managedBean.name}"/>

    ...

</html>
```

Gerenciamento de Transações

```
@Service
public class EmployeeServiceImpl
                    implements EmployeeService {

    @Autowired
    private EntityManager entityManager;

    @Transactional
    public void save (Employee emp) {
        entityManager.persist(emp);  ✗
    }
}
```

RESTful Web Services

REST é um estilo arquitetural para projetar aplicações de rede. Ao invés de usar protocolos complexos, usa-se HTTP para comunicação entre aplicações

REST - Representational State Transfer

Usa-se todos os métodos do HTTP:

* GET	* Ler
* POST	* Criar
* PUT/PATCH	* Alterar
* DELETE	* Remover

Formatos mais comuns para troca de dados:

* JSON
* XML

Em REST, os recursos são representados por URLs:

```
GET /employees
```

```
status: 200 ok
[{"id":1,"name":"employee 1"},
 {"id":2,"name":"employee 2"}]
```

```
POST /employees
{"name":"employee 2"}
```

```
status: 200 ok
{"id":2,"name":"employee 2"}
```

```
PUT /employees/2
{"name":"employee 2.1"}
```

```
status: 200 ok
{"id":2,"name":"employee 2.1"}
```

```
status: 404 not found
```

DELETE /employees/2

status: 204 no content
status: 200 ok
status: 404 not found

GET /employees GET /employees/:id

POST /employees

PUT /employees/{id}

DELETE /employees/{id}

```
GET /employees/1/addresses

POST /employees/1/addresses

PUT /employees/1/phones/2

DELETE /employees/1/adresses/2
```

JAX-RS

Java API for REST WebServices

```
@Path ("/employees")
@Produces (MediaType.APPLICATION_JSON)
@Consumes (MediaType.APPLICATION_JSON)
public class EmployeesResource {


    @GET
    public List<Employee> findAll() {...}


    @POST
    public Employee create (Employee emp) {...}


}
```

```
@Path ("/employees")
public class EmployeesResource {

    @DELETE
    @Path("{id}")
    public void delete ( @PathParam ("id")
                         Long id ) {...}

}
```

```
@Path ("/employees")
public class EmployeesResource {

    @PUT
    @Path("{id}")
    public Employee update (

                        @PathParam ("id") Long id,

                        Employee employee ) {...}
}
```

Controlando a resposta

```
@Path ("/employees")
public class EmployeesResource {


  @GET
  @Path("{id}")
  public Response get (@PathParam ("id") Long id) {

      Employee e = find(id);

      if(e != null) { return Response.entity(e).build(); }

      else { return Response.status(Status.NOT_FOUND).build(); }

  }
}
```

Configurações Adicionais

Spring

Spring

```
@Component
@Path ("/employees")
public class EmployeesResource {

   ...

   ...

   ...

}
```

Spring

```
@Configuration
public class JerseyConfig extends ResourceConfig {

    public JerseyConfig() {
        register(EmployeesResource.class);
    }

}
```

Configurações Adicionais

JBoss 6.4 EAP

1. As classes de recursos ficam em um Módulo Web

2. Habilitar CDI (WEB-INF/beans.xml)

3. Registrar os recursos em uma herança de javax.ws.rs.core.Application

Crie um arquivo beans.xml vazio em WEB-INF

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
      http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
</beans>
```

JBoss

```
@Path ("/employees")
public class EmployeesResource {

   ...

   @Inject

   private EmployeeService empService;

   ...

}
```

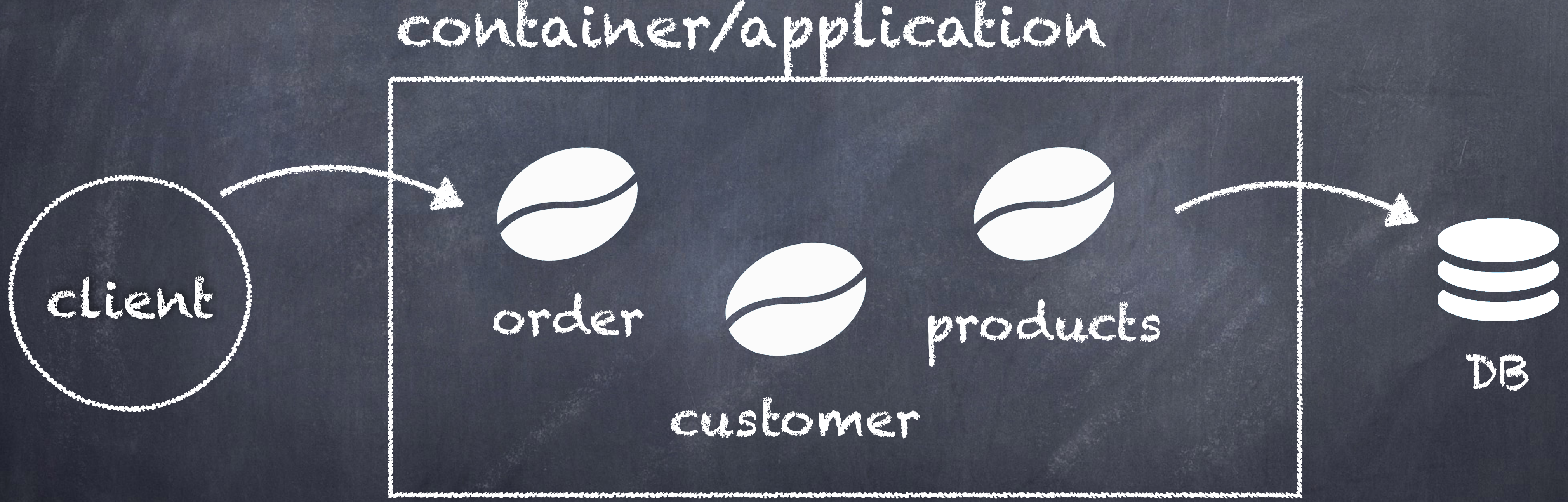
JBoss

```
@ApplicationPath ("/rs")
public class RestApplication extends Application {

  ...

  public Set<Class<?>> getClasses(){

   return new

          HashSet(Arrays.asList(EmployeeResource.class));

  }

}
```

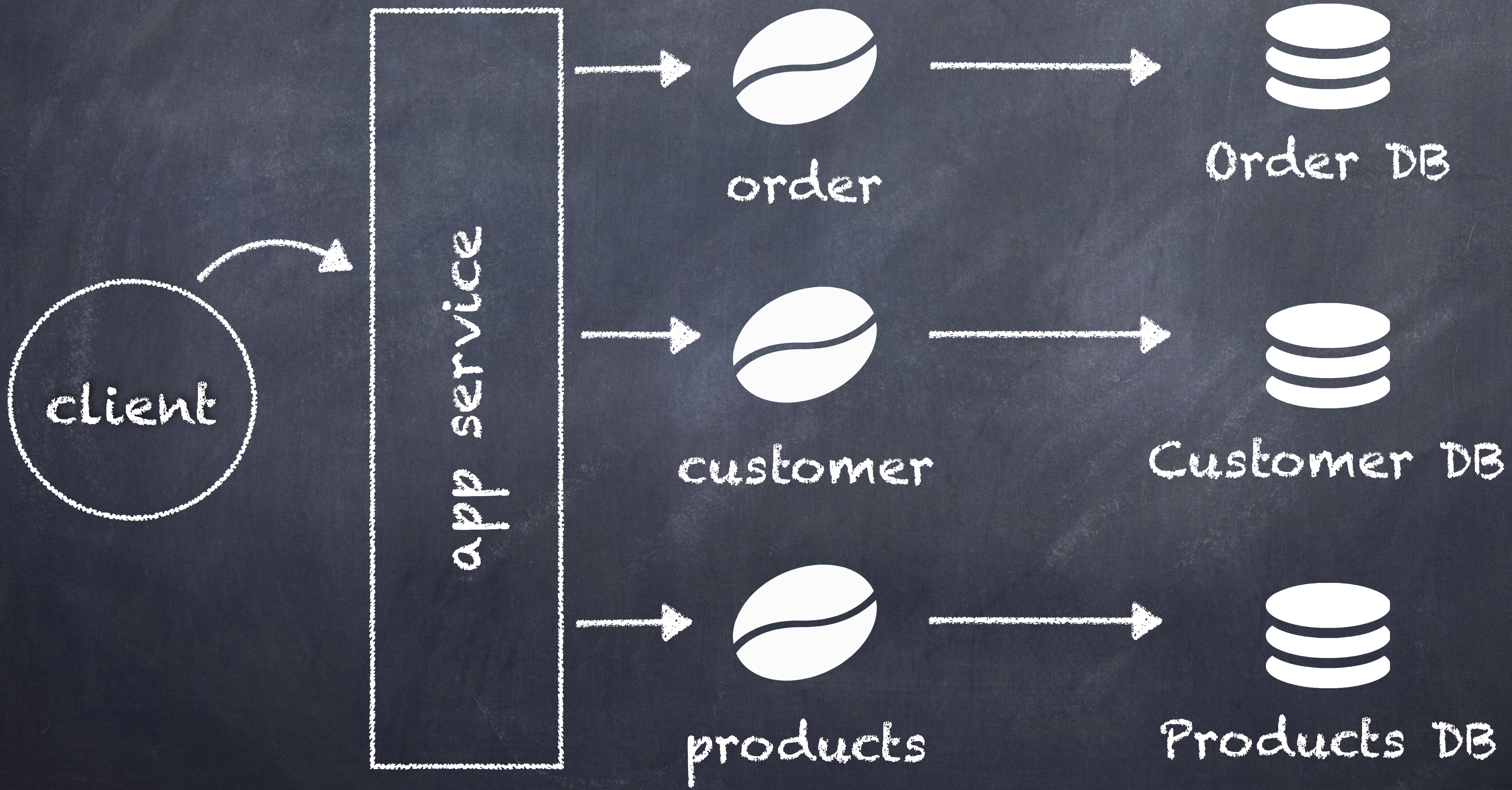
Spring Boot

e Arquitetura de Micro-serviços

Aplicação Monolítica



Micro-Serviços



O Spring Boot fornece API para se criar aplicações standalone baseadas em Spring que podem ser executadas independentes

Principais características:

* O mínimo de configuração
* Gerenciamento automático de dependências
* Embedded runtimes

Pode ser com:

* Maven
* Gradle
* Ant (não recomendado)

Como iniciar uma aplicação em Spring Boot

```
package br.fa7.spring.exercicios;

@SpringBootApplication

public class Main {

    public static void main(String[] args) {

        SpringApplication.run(Main.class, args);

    }

}
```

@SpringBootApplication

* Auto configura o ambiente
* Registra os componentes do Spring

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <!-- Inherit defaults from Spring Boot -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.3.RELEASE</version>
  </parent>

  <groupId>br.gov.fa7.spring</groupId>
  <artifactId>exercicios</artifactId>
  <version>0.0.1-SNAPSHOT</version>

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

</project>
```

Configurando o ambiente com Maven

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

Controlando o 'restart' de forma automática

```
...
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <optional>true</optional>
</dependency>
...
```

Criando um jar executável

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

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.3.3.RELEASE)
```

Exercícios

- Crie uma aplicação em Maven e configure com Spring Boot
- Crie um CRUD (GET, POST, PUT, DELETE) com a API de JAX-RS para Employee
  - id
  - nome
- O mapeamento da URL: /employees
- Separe em camadas
  - resource
  - service
  - entity
- Teste com o plugin do Chrome Advanced REST Client