



Frameworks

ARQUITETURAS APLICACIONAIS

JOÃO MARQUES

JOSÉ PEREIRA

RICARDO PETRONILHO

4 ° ANO – MIEI/MEI

Estrutura da Apresentação

2

- ▶ Groils
- ▶ Spring
- ▶ Arquitetura Tipo

- ▶ Características:
 - ▶ multiplataforma (Máquina Virtual Java)
 - ▶ redução da complexidade das frameworks modernas
 - ▶ open-source
 - ▶ alta produtividade (sem necessidade de muitas configurações)
 - ▶ MVC (Model-View-Controller)
 - ▶ utiliza a linguagem Groovy (Plataforma Java)
 - ▶ ferramenta grails (interpretador interativo)
 - ▶ server-side & client-side
 - ▶ grande comunidade e documentação

Grails

4

```
Desktop$ grails create-app app
| Application created at /Users/josepereira/Desktop/app
Desktop$ cd app/
| Resolving Dependencies. Please wait...

| Starting interactive mode...
| Enter a command name to run. Use TAB for completion:
grails> █
```

```
app$ ls
build                gradle.properties  grails-app          grailsw.bat
build.gradle         gradlew            grails-wrapper.jar  src
gradle              gradlew.bat        grailsw
app$ █
```

Grails

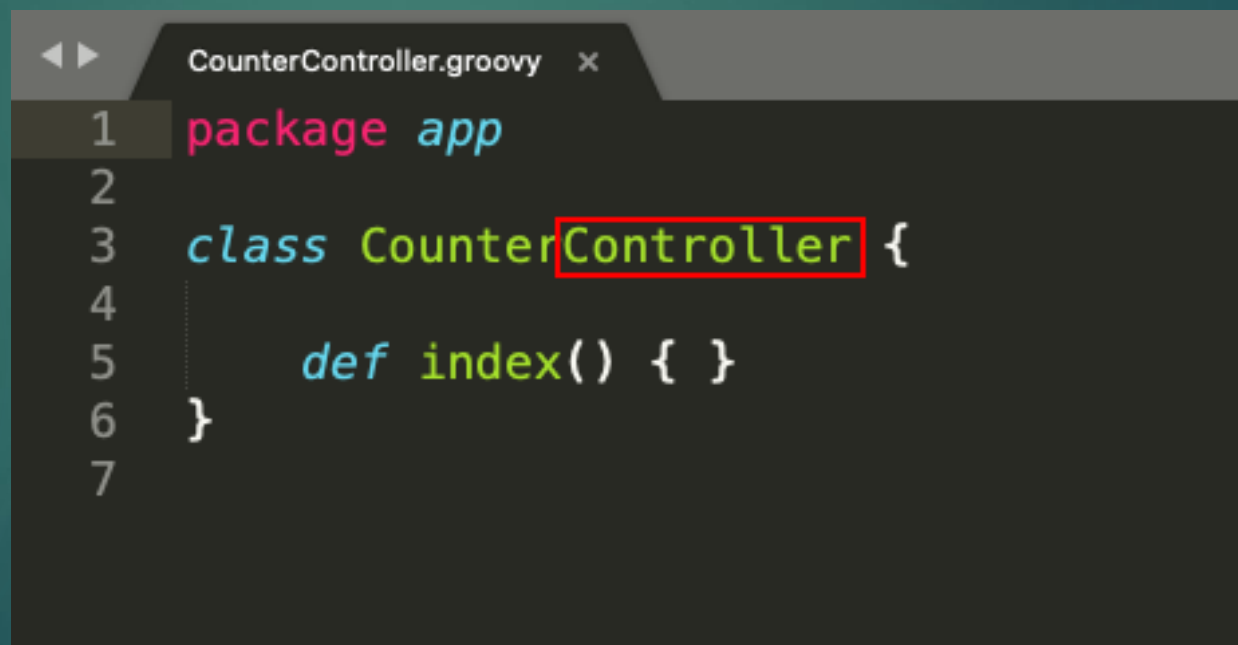
5

- `grails-app` - top level directory for Groovy sources
 - `conf` - [Configuration sources](#)
 - `controllers` - [Web controllers](#) - The C in MVC.
 - `domain` - The [application domain](#). - The M in MVC
 - `i18n` - Support for [internationalization \(i18n\)](#).
 - `services` - The [service layer](#).
 - `taglib` - [Tag libraries](#).
 - `utils` - Grails specific utilities.
 - `views` - [Groovy Server Pages](#) or [JSON Views](#) - The V in MVC.
- `src/main/scripts` - [Code generation scripts](#).
- `src/main/groovy` - Supporting sources
- `src/test/groovy` - [Unit and integration tests](#).

Grails

6

```
grails> create-controller counter
| Created grails-app/controllers/app/CounterController.groovy
| Created src/test/groovy/app/CounterControllerSpec.groovy
grails> 
```



```
CounterController.groovy x
1 package app
2
3 class CounterController {
4
5     def index() { }
6 }
7
```

Grails

← → ↻ ⓘ localhost:8080/counter/index
Standard method!

← → ↻ ⓘ localhost:8080/counter/increment
value = 1

← → ↻ ⓘ localhost:8080/counter/set?value=5000
value = 5000

```
CounterController.groovy x
package app

class CounterController {

    int value = 0

    /* standard method */
    def index() {
        render "Standard method!"
    }

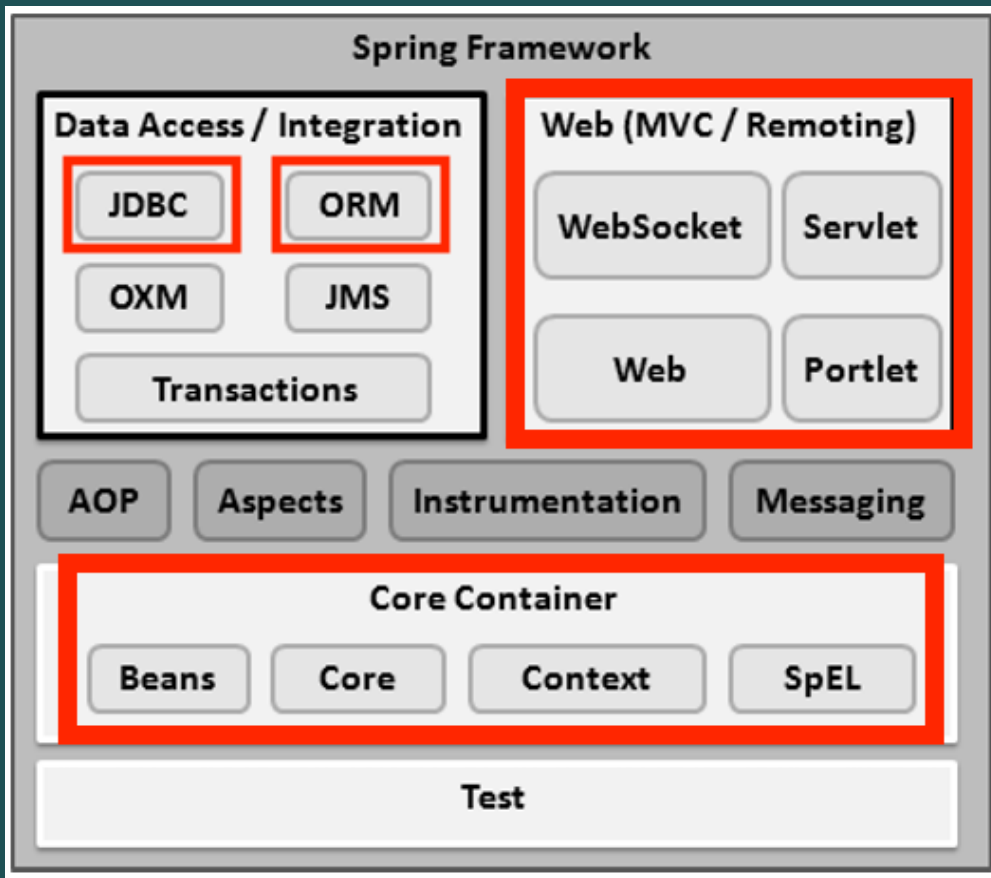
    /* increment value */
    def increment(){
        this.value++
        display()
    }

    /* decrement value */
    def decrement(){
        this.value--
        display()
    }

    /* display value */
    def display(){
        render "value = " + this.value
    }

    /* set value */
    def set(int value){
        this.value = value
        display()
    }
}
```





Package Data

- ✓ **JDBC** - Java Database Connectivity
Estabelece a conexão e permite executar queries aos SGBDs.
- ✓ **ORM** - Object-Relational Mapping
Automaticamente (através de anotações e interfaces) implementa o mapeamento de relações e definição do modelo lógico da base de dados na camada aplicacional.

Pakcage Core

- ✓ **Beans e Core (Núcleo)**
Implementam a arquitetura base da Spring através dos padrões arquiteturais IoC e DI.
- ✓ **Context (Contexto do projeto em questão)**
Utiliza os módulos nucleares (Core e Bean) para gerar automaticamente os objetos configurados pela implementação específica do programador (maioritariamente usado pelo Spring Boot).

Pakcage Web

Implementa o web server (ex: utilizando apache Tomcat) e o mapeamento dos pedidos HTTP (ou outros protocolos) para o servidor aplicacional (processado pelo respetivo Controller).

@Entity

```
public class Person {
```

@Id**@GeneratedValue(strategy=GenerationType.AUTO)**

```
private String id;
```

@Min(8)**@Max(64)****@NotNull**

```
private String name;
```

@Positive**@NotNull**

```
private int age;
```

@Email**@NotEmpty**

```
private String email;
```

- ✓ Na própria classe do modelo de dados (Model) é descrito o modelo lógico referente à base de dados através de anotações. (@Entity, @Id, @Table, e outras para representar relações 1-1, 1-N, N-N)
- ✓ Ainda no Model podem ser definidas anotações de validação que o spring boot verifica quando evocada a anotação @Valid.

```
@RestController
@RequestMapping(path="/person")
public class PersonController {

    @Autowired
    public PersonRepository personRepository;

    @GetMapping("/all")
    public ResponseEntity<Iterable<Person>> getPerson() {
        return new ResponseEntity<>(personRepository.findAll(), HttpStatus.OK);
    }

    @GetMapping("/get")
    public ResponseEntity<Person> getPerson @RequestParam(value = "id", defaultValue = "") String id) {
        if (personRepository.existsById(id) == false) return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        else return new ResponseEntity<>(personRepository.findById(id).get(), HttpStatus.OK);
    }

    @PostMapping("/put")
    public ResponseEntity<Person> putPerson @Valid @RequestBody Person person {
        if (personRepository.existsById(person.getName())) return new ResponseEntity<>(HttpStatus.CONFLICT);
        else {
            personRepository.save(person);
            return new ResponseEntity<>(HttpStatus.CREATED);
        }
    }
}
```

```
@Repository  
public interface PersonRepository extends CrudRepository<Person, String> {}
```

Pontos fortes

- ✓ Utiliza Java (conceitos POO, preparada para projetos de elevada complexidade, comunidade forte, etc)
- ✓ Comunidade da própria framework (consistente com elevado suporte e prestável)
- ✓ Utiliza bibliotecas de terceiros de forma confiável (hibernate, etc) tornando a framework segura
- ✓ Spring boot (com anotações e implementação automática dos padrões IoC e DI) tornando o desenvolvimento simples e intuitivo
- ✓ Equipas de software conceituadas confiam na framework (Microsoft, Google, Amazon, etc)

Arquitetura Tipo

14

Frontend



GRAILS

Backend



ORM



HIBERNATE



Frameworks

ARQUITETURAS APLICACIONAIS

JOÃO MARQUES

JOSÉ PEREIRA

RICARDO PETRONILHO

4 ° ANO – MIEI/MEI