

INE5646 Programação para Web

- Tópico :
Aplicações Baseadas na Java Virtual Machine
(JVM)
- Subtópico:
Play (Parte 1)

(estes slides fazem parte do material didático da disciplina
INE5646 Programação para Web)

Sumário

- Parte 1:
 - Instalação, configuração e administração de servidores.
 - Processamento síncrono e assíncrono de requisições
 - Template engine.
- Parte 2:
 - Acesso a bases de dados.
 - Serviços web.
 - Processamento de stream de dados.

Play - Instalação

- O framework Play (<http://www.playframework.com>), como característica geral, preza pela agilidade e simplicidade.
- Pré-requisito para instalação:
 - Ter instalado a JVM JDK 6 ou superior.
- Instalação em 3 passos:
 - 1/3: Fazer o download do arquivo zipado (.zip)
 - 2/3: Extrair o conteúdo do arquivo zipado em algum diretório em que o usuário tenha permissão de leitura e escrita.
 - 3/3: Adicionar o caminho (path) do script play à variável PATH.

Play – Instalação - Exemplo

- Exemplo de instalação do Play em ambiente Linux (Ubuntu).
- Antes de instalar o Play:
 - verifique se os programas utilitários **wget** e **unzip** estão instalados. Se não estiverem:
 - `sudo apt-get install wget unzip <enter>`
 - Verifique se a plataforma **Java** está instalada (executando `javac -version`). Se não estiver:
 - `sudo apt-get install openjdk-7-jdk <enter>`

Play – Instalação - Exemplo

- 1/3 : Fazer download do arquivo play-2.1.0.zip
 - A figura abaixo mostra o arquivo sendo baixado e salvo em /home/leandro/apps.



```
File Edit View Terminal Help
leandro@leandro-desktop:~/apps$ pwd
/home/leandro/apps
leandro@leandro-desktop:~/apps$ wget http://downloads.typesafe.com/play/2.1.0/play-2.1.0.zip
--2013-03-06 15:58:56-- http://downloads.typesafe.com/play/2.1.0/play-2.1.0.zip
Resolving downloads.typesafe.com... 205.251.223.25, 205.251.223.41, 205.251.223.109, ...
Connecting to downloads.typesafe.com|205.251.223.25|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 152041721 (145M) [binary/octet-stream]
Saving to: `play-2.1.0.zip'

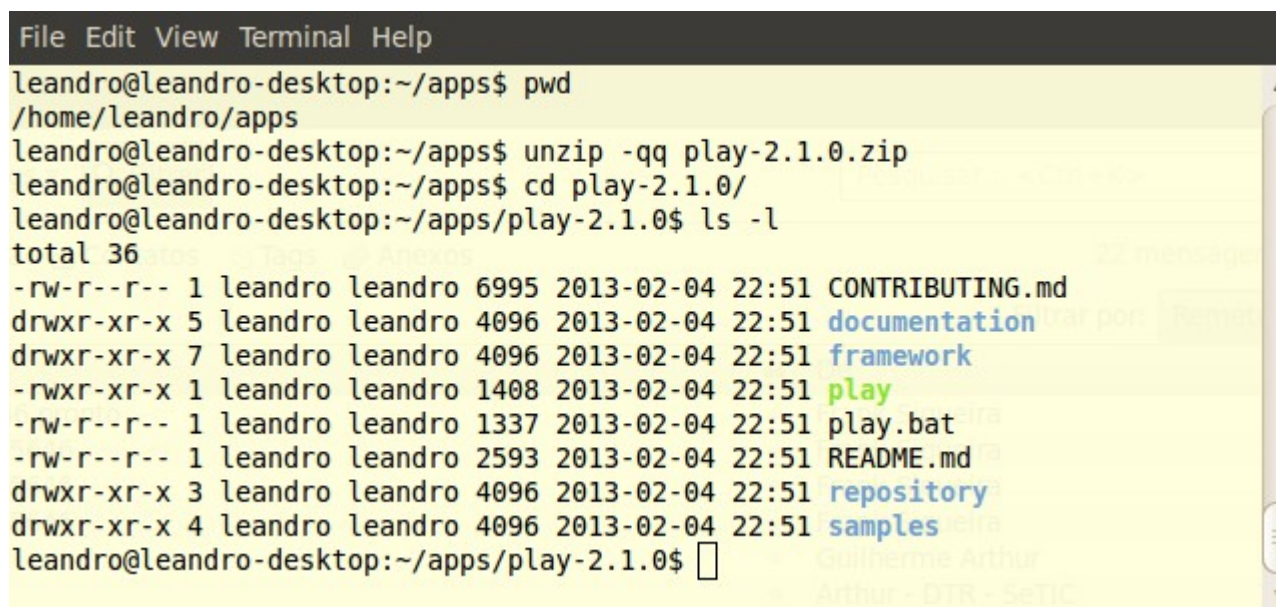
100%[=====>] 152,041,721 38.6M/s  in 6.1s

2013-03-06 15:59:03 (23.7 MB/s) - `play-2.1.0.zip' saved [152041721/152041721]

leandro@leandro-desktop:~/apps$
```

Play – Instalação - Exemplo

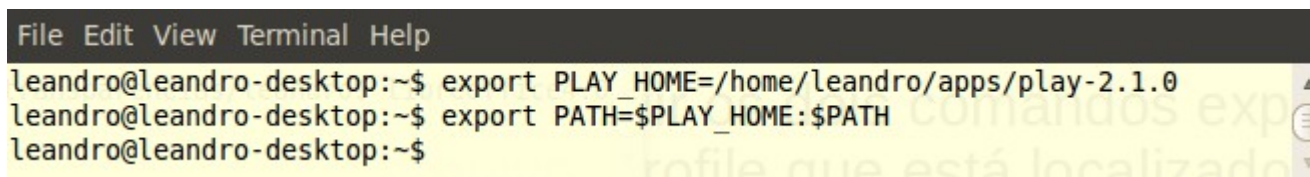
- 2/3 : Descompactar o arquivo play-2.1.0.zip
 - A figura abaixo mostra o arquivo zipado sendo descompactado. Os arquivos que compõem o Play estão armazenados no diretório /home/leandro/apps/play-2.1.0



```
File Edit View Terminal Help
leandro@leandro-desktop:~/apps$ pwd
/home/leandro/apps
leandro@leandro-desktop:~/apps$ unzip -qq play-2.1.0.zip
leandro@leandro-desktop:~/apps$ cd play-2.1.0/
leandro@leandro-desktop:~/apps/play-2.1.0$ ls -l
total 36
-rw-r--r-- 1 leandro leandro 6995 2013-02-04 22:51 CONTRIBUTING.md
drwxr-xr-x 5 leandro leandro 4096 2013-02-04 22:51 documentation
drwxr-xr-x 7 leandro leandro 4096 2013-02-04 22:51 framework
-rwxr-xr-x 1 leandro leandro 1408 2013-02-04 22:51 play
-rw-r--r-- 1 leandro leandro 1337 2013-02-04 22:51 play.bat
-rw-r--r-- 1 leandro leandro 2593 2013-02-04 22:51 README.md
drwxr-xr-x 3 leandro leandro 4096 2013-02-04 22:51 repository
drwxr-xr-x 4 leandro leandro 4096 2013-02-04 22:51 samples
leandro@leandro-desktop:~/apps/play-2.1.0$
```

Play – Instalação - Exemplo

- 3/3 : Adicionar o caminho do script play na variável de ambiente PATH
 - Dica: incluir os dois comandos export dentro do arquivo .profile que está localizado no diretório home (no exemplo, em /home/leandro)



A terminal window with a dark title bar containing 'File Edit View Terminal Help'. The terminal text shows three lines of commands and their prompts: 'leandro@leandro-desktop:~\$ export PLAY_HOME=/home/leandro/apps/play-2.1.0', 'leandro@leandro-desktop:~\$ export PATH=\$PLAY_HOME:\$PATH', and 'leandro@leandro-desktop:~\$'. The text is highlighted in yellow. On the right side of the terminal, there are navigation icons: an up arrow, a menu icon (three horizontal lines), and a down arrow.

```
File Edit View Terminal Help
leandro@leandro-desktop:~$ export PLAY_HOME=/home/leandro/apps/play-2.1.0
leandro@leandro-desktop:~$ export PATH=$PLAY_HOME:$PATH
leandro@leandro-desktop:~$
```


Play – Instalação - Exemplo

- Para testar se a instalação foi feita corretamente basta executar o script play

```
File Edit View Terminal Help
leandro@leandro-desktop:~$ play
Getting net.java.dev.jna jna 3.2.3 ...
:: retrieving :: org.scala-sbt#boot-jna
   confs: [default]
   1 artifacts copied, 0 already retrieved (838kB/13ms)
Getting play console_2.9.2 2.1.0 ...
:: retrieving :: org.scala-sbt#boot-app
   confs: [default]
   29 artifacts copied, 0 already retrieved (5976kB/34ms)
Getting Scala 2.9.2 (for console)...
:: retrieving :: org.scala-sbt#boot-scala
   confs: [default]
   4 artifacts copied, 0 already retrieved (20090kB/77ms)

play! 2.1.0 (using Java 1.7.0_17 and Scala 2.10.0), http://www.playframework.org

This is not a play application!

Use `play new` to create a new Play application in the current directory,
or go to an existing application and launch the development console using `play`.

You can also browse the complete documentation at http://www.playframework.org.

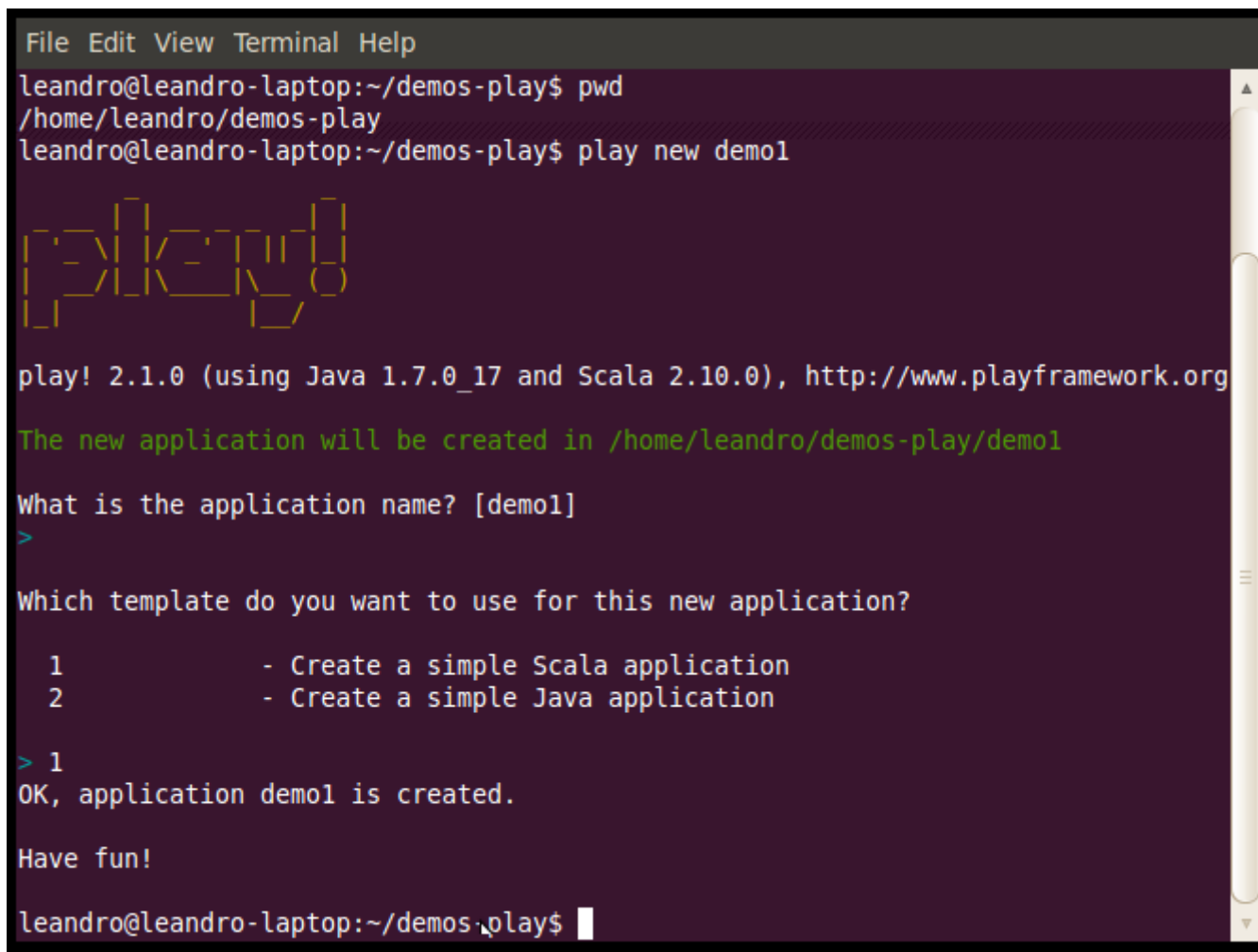
leandro@leandro-desktop:~$
```


Play - Configuração

- O servidor Play não segue a especificação JEE:
 - Não há um “servlet container” ou um “EJB container”.
 - Não existe a noção de servlet.
- A filosofia do Play é: um servidor para cada aplicação.
- Portanto: não existe nada para ser configurado no servidor.
- As configurações possíveis estão diretamente relacionadas às necessidades específicas de cada aplicação.

Play - Configuração

- Criando a aplicação demo1.



```
File Edit View Terminal Help
leandro@leandro-laptop:~/demos-play$ pwd
/home/leandro/demos-play
leandro@leandro-laptop:~/demos-play$ play new demo1

play! 2.1.0 (using Java 1.7.0_17 and Scala 2.10.0), http://www.playframework.org

The new application will be created in /home/leandro/demos-play/demo1

What is the application name? [demo1]
>

Which template do you want to use for this new application?

  1 - Create a simple Scala application
  2 - Create a simple Java application

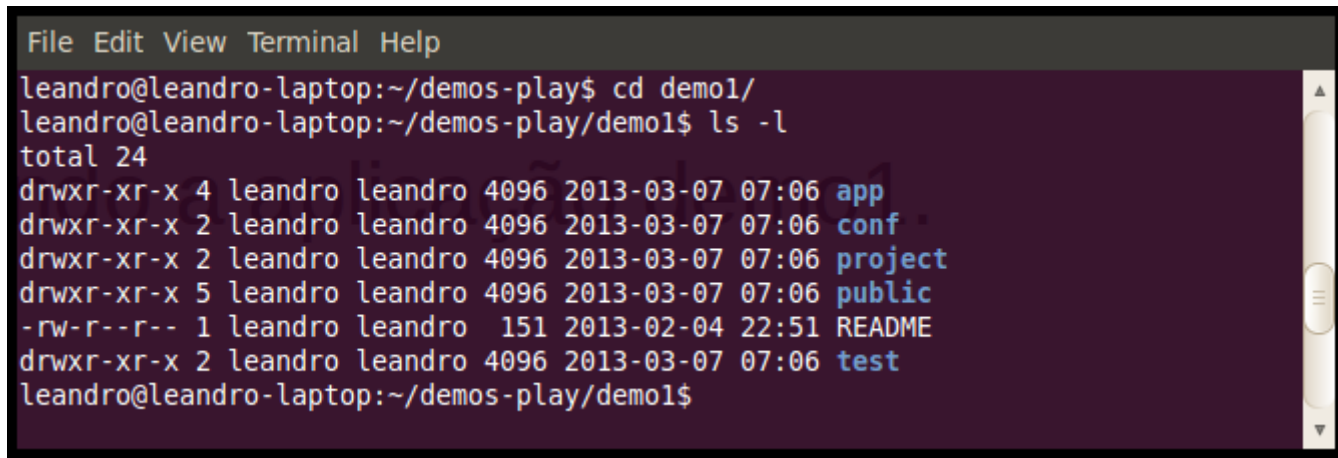
> 1
OK, application demo1 is created.

Have fun!

leandro@leandro-laptop:~/demos-play$
```

Play - Configuração

- Estrutura (inicial) de uma aplicação Play:
 - **app** : contém o código da aplicação.
 - **conf** : arquivos de configuração e roteamento.
 - **project** : arquivos pelo Play para gerenciar o processo de compilação.
 - **public** : recursos (assets) JavaScript, CSS e imagens.
 - **test** : conterá as classes para testes.



```
File Edit View Terminal Help
leandro@leandro-laptop:~/demos-play$ cd demo1/
leandro@leandro-laptop:~/demos-play/demo1$ ls -l
total 24
drwxr-xr-x 4 leandro leandro 4096 2013-03-07 07:06 app
drwxr-xr-x 2 leandro leandro 4096 2013-03-07 07:06 conf
drwxr-xr-x 2 leandro leandro 4096 2013-03-07 07:06 project
drwxr-xr-x 5 leandro leandro 4096 2013-03-07 07:06 public
-rw-r--r-- 1 leandro leandro 151 2013-02-04 22:51 README
drwxr-xr-x 2 leandro leandro 4096 2013-03-07 07:06 test
leandro@leandro-laptop:~/demos-play/demo1$
```

Play - Configuração

- Estrutura completa de uma aplicação Play:
The standard application layout

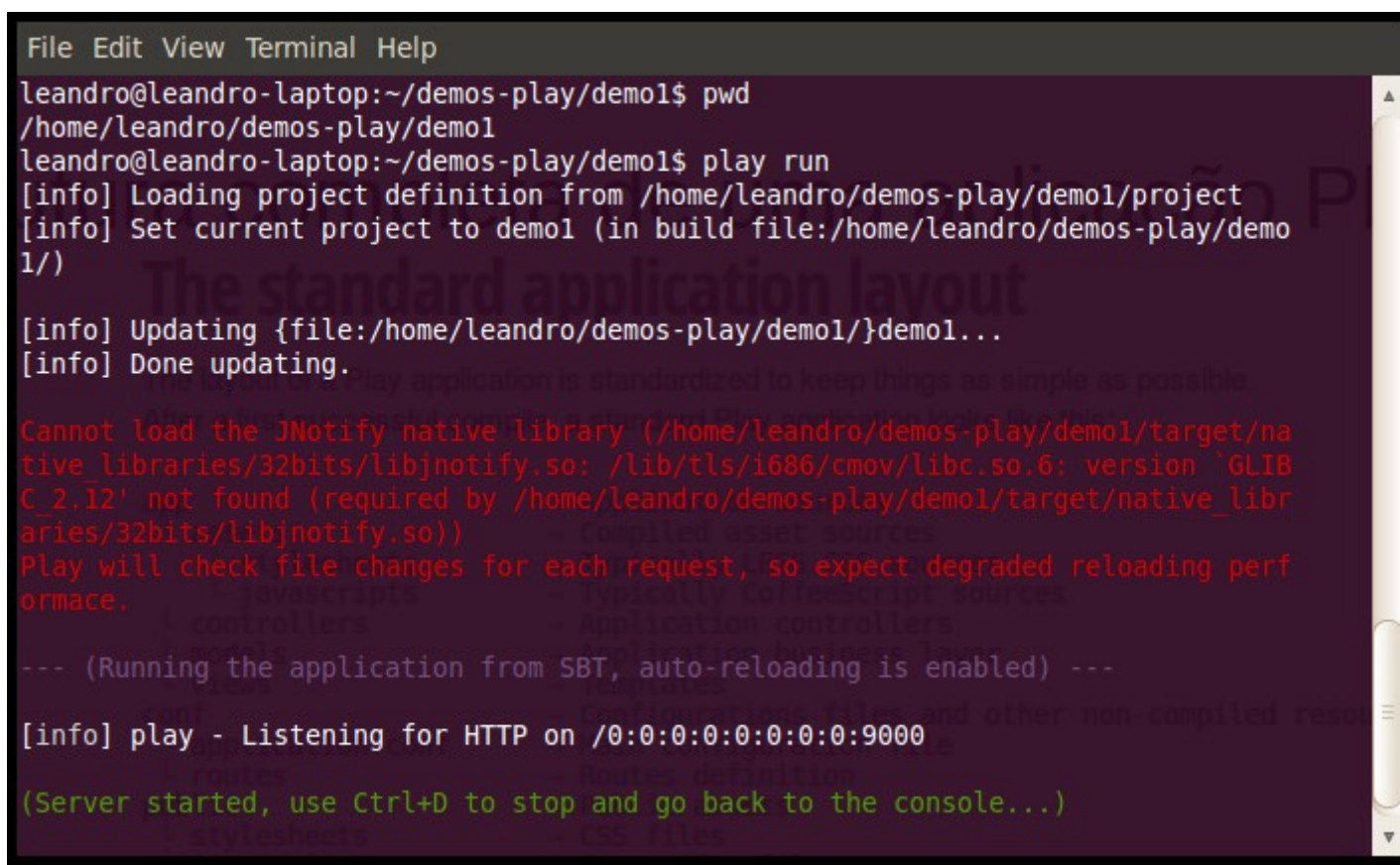
The layout of a Play application is standardized to keep things as simple as possible.
After a first successful compile, a standard Play application looks like this:

app	→ Application sources
└ assets	→ Compiled asset sources
└ stylesheets	→ Typically LESS CSS sources
└ javascripts	→ Typically CoffeeScript sources
└ controllers	→ Application controllers
└ models	→ Application business layer
└ views	→ Templates
conf	→ Configurations files and other non-compiled resources
└ application.conf	→ Main configuration file
└ routes	→ Routes definition
public	→ Public assets
└ stylesheets	→ CSS files
└ javascripts	→ Javascript files
└ images	→ Image files
project	→ sbt configuration files
└ build.properties	→ Marker for sbt project
└ Build.scala	→ Application build script
└ plugins.sbt	→ sbt plugins
lib	→ Unmanaged libraries dependencies
logs	→ Standard logs folder
└ application.log	→ Default log file
target	→ Generated stuff
└ scala-2.10.0	
└ cache	
└ classes	→ Compiled class files
└ classes_managed	→ Managed class files (templates, ...)
└ resource_managed	→ Managed resources (less, ...)
└ src_managed	→ Generated sources (templates, ...)
test	→ source folder for unit or functional tests

Fonte: <http://www.playframework.com/documentation/2.1.0/Anatomy>

Play - Configuração

- “play run”: inicia a execução do servidor/aplicação.



```
File Edit View Terminal Help
leandro@leandro-laptop:~/demos-play/demo1$ pwd
/home/leandro/demos-play/demo1
leandro@leandro-laptop:~/demos-play/demo1$ play run
[info] Loading project definition from /home/leandro/demos-play/demo1/project
[info] Set current project to demo1 (in build file:/home/leandro/demos-play/demo1/)

[info] Updating {file:/home/leandro/demos-play/demo1/}demo1...
[info] Done updating.

Cannot load the JNotify native library (/home/leandro/demos-play/demo1/target/native_libraries/32bits/libjnotify.so: /lib/tls/i686/cmov/libc.so.6: version `GLIBC 2.12' not found (required by /home/leandro/demos-play/demo1/target/native_libraries/32bits/libjnotify.so))
Play will check file changes for each request, so expect degraded reloading performance.

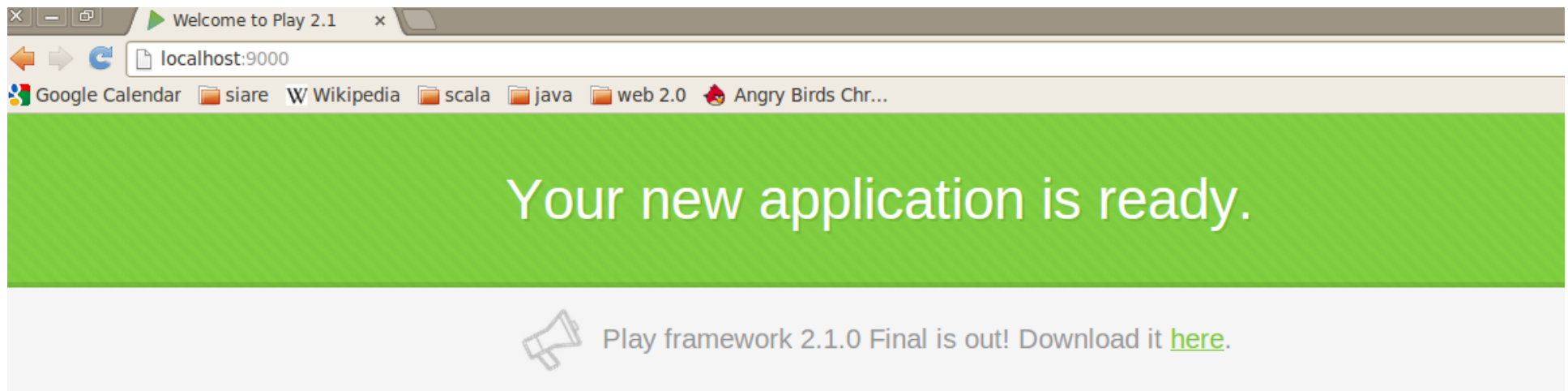
--- (Running the application from SBT, auto-reloading is enabled) ---

[info] play - Listening for HTTP on /0:0:0:0:0:0:0:0:9000

(Server started, use Ctrl+D to stop and go back to the console...)
```

Play - Configuração

- Acessando a aplicação:



Welcome to Play 2.1

Congratulations, you've just created a new Play application. This page will help you with the few next steps.

You're using Play 2.1.0

Play - Configuração

- As possibilidades de configuração do servidor de uma aplicação são registradas no arquivo **conf/application.conf**
- O que pode ser configurado (principais itens):
 - Acesso a banco de dados.
 - Thread pools (para aumentar o desempenho de aplicações caso necessário)
 - Nível de log (INFO, DEBUG, ERROR, etc)

Play - Configuração

- Exemplo de arquivo de configuração para acesso a banco de dados e nível de log da aplicação:

```
application.conf x
1
2 # Configuração da base de dados
3 # ~~~~
4 # O nome (alias) da base de dados default chama-se "default"
5 #
6 # O SGBD usado é o H2 (poderia ser MySQL, Postgres, etc)
7 db.default.driver=org.h2.Driver
8
9 # A base de dados é armazenada em memória e chama-se "clientes"
10 db.default.url="jdbc:h2:mem:clientes"
11
12 # A aplicação acessará a base de dados via usuário "leandro"
13 db.default.user=leandro
14
15 # A senha do usuário leandro é "secreta"
16 db.default.password="secreta"
17
18
19 # Apenas mensagens de log até o nível DEBUG serão registradas.
20 logger.application=DEBUG
```

Play - Administração

- O mesmo raciocínio da configuração aplica-se à administração: administra-se o servidor da aplicação e não o servidor isoladamente.
- A administração é simples e consiste basicamente em decidir:
 - Se a aplicação será executada em modo de produção (priorizando a eficiência) ou em modo de desenvolvimento (priorizando a produtividade)
 - Qual porta será usada (por default, 9000).

Play – Administração - Exemplos

- Atenção: executar os comandos abaixo sempre dentro do diretório da aplicação.
- Para executar a aplicação no modo desenvolvimento usando a porta 9000:
 - **play** <enter>
 - **run** <enter>
- Para executar a aplicação no modo desenvolvimento usando a porta 8080:
 - **play** <enter>
 - **run 8080** <enter>
- Pode-se usar, naturalmente, qualquer porta desejada (como a 80)
- Para executar a aplicação no **modo produção** basta **substituir run por start**. Neste caso, é iniciada uma nova instância da JVM.

Play – Administração - Exemplos

- Administração do ciclo de desenvolvimento “editar-salvar-compilar-executar-visualizar”:
 - É realizado **automaticamente**.
 - Basta “editar-salvar-visualizar”. O desenvolvedor sempre está vendo a versão atualizada da aplicação (sempre que algum arquivo é alterado).
 - É um processo relativamente lento pois todos os recursos são recompilados visando detectar o máximo de erros em tempo de compilação.
 - As mensagens de erro aparecem no browser e costumam ser muito mais claras que o “printStackTrace” de Java.

Processamento síncrono e assíncrono de requisições

- **O processamento síncrono de requisições** obedece ao seguinte algoritmo:
 1. O usuário, via navegador (camada 1), envia uma requisição HTTP ao servidor.
 2. O servidor (camada 2) recebe a requisição e aloca uma thread (retira de algum pool de threads) para processá-la.
 3. **A thread processa a requisição.**
 4. A thread monta a resposta HTTP e a envia ao usuário.
 5. O servidor libera a thread (devolve a thread para o pool) para atender outra requisição.

Processamento síncrono e assíncrono de requisições

- Considerações sobre o passo 3:
 - O número de threads para atender requisições sempre é limitado (não é escalável) pois consome muitos recursos do servidor (memória e CPU).
 - Se o passo 3 for executado muito rapidamente, a thread logo é liberada (passo 5) e está tudo certo.
 - Se o passo 3 for demorado podemos ter problema:
 - A **thread pode estar parada** esperando que uma operação de E/S bloqueante se complete (ex: resposta de uma consulta ao banco de dados ou resposta de um serviço em outro site).

Processamento síncrono e assíncrono de requisições

- Considerações sobre o passo 3 (cont.):
 - Outra requisição não pode ser atendida pois a thread, que não está fazendo nada (aguardando algum processamento), só será liberada depois que a resposta tiver sido enviada.
 - Em termos relativos, comparando-se o tempo necessário para atender uma requisição e o tempo necessário para receber uma resposta de uma solicitação de serviço em outro site, por exemplo, a **thread pode ficar a maior parte do tempo sem fazer nada.**

Processamento síncrono e assíncrono de requisições

- Considerações sobre o passo 3 (cont.):
 - Nas aplicações web 1.0 o problema da falta de escalabilidade das threads não é, normalmente, um grande problema. As requisições são processadas muito rapidamente no servidor.
 - Resumindo: **requisições síncronas não são um problema desde que o tempo de processamento no servidor seja muito pequeno.**
 - **Observação:** a tecnologia de servlets (Java) só definia requisições síncronas até a versão 2.5 (de 2005). Isto mudou com a especificação 3.0 em dezembro de 2009.

Processamento síncrono e assíncrono de requisições

- **Processamento assíncrono de requisições** obedece ao seguinte algoritmo:
 1. O usuário, via navegador (camada 1), envia uma requisição HTTP ao servidor.
 2. O servidor (camada 2) recebe a requisição e aloca uma thread do pool para processá-la.
 3. **A thread inicia o processamento da requisição. Se houver alguma operação de E/S não bloqueante então o processamento é suspenso (aguardando a operação ser concluída) e a thread é liberada para atender outra requisição. Quando a operação de E/S não bloqueante é completada, alguma thread (possivelmente outra) é alocada e o processamento da requisição continua.**
 4. A thread monta a resposta HTTP e a envia ao usuário.
 5. O servidor libera (devolve ao pool) a thread para atender outra requisição.

Processamento síncrono e assíncrono de requisições

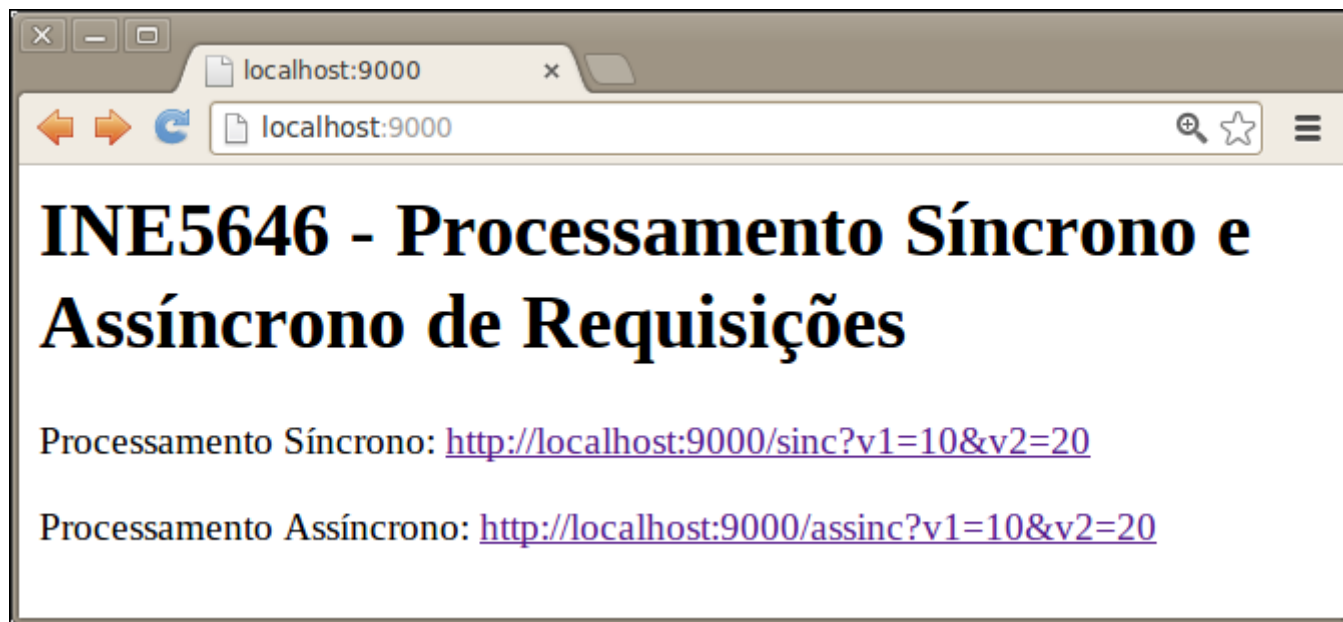
- O framework Play, por ter sido criado no contexto web 2.0, incorpora a ideia de requisições assíncronas.
- O passo 3 de uma requisição assíncrona utiliza o conceito de **promessa de resposta**.
- **Promessa de resposta:** no lugar de gerar uma resposta a thread retorna uma promessa de resposta. Assim a thread fica livre para atender uma nova requisição.
- Quando a promessa de resposta se realizar então (possivelmente) outra thread irá gerar a resposta a ser enviada de volta ao browser.

Proc. síncrono e assíncrono de requisições - Exemplo

- No exemplo, o problema consiste em calcular a media de dois números.
- Esta tarefa, de forma simulada, leva muito tempo para ser concluída.
- A requisição pode ser processada sincronamente ou assincronamente.

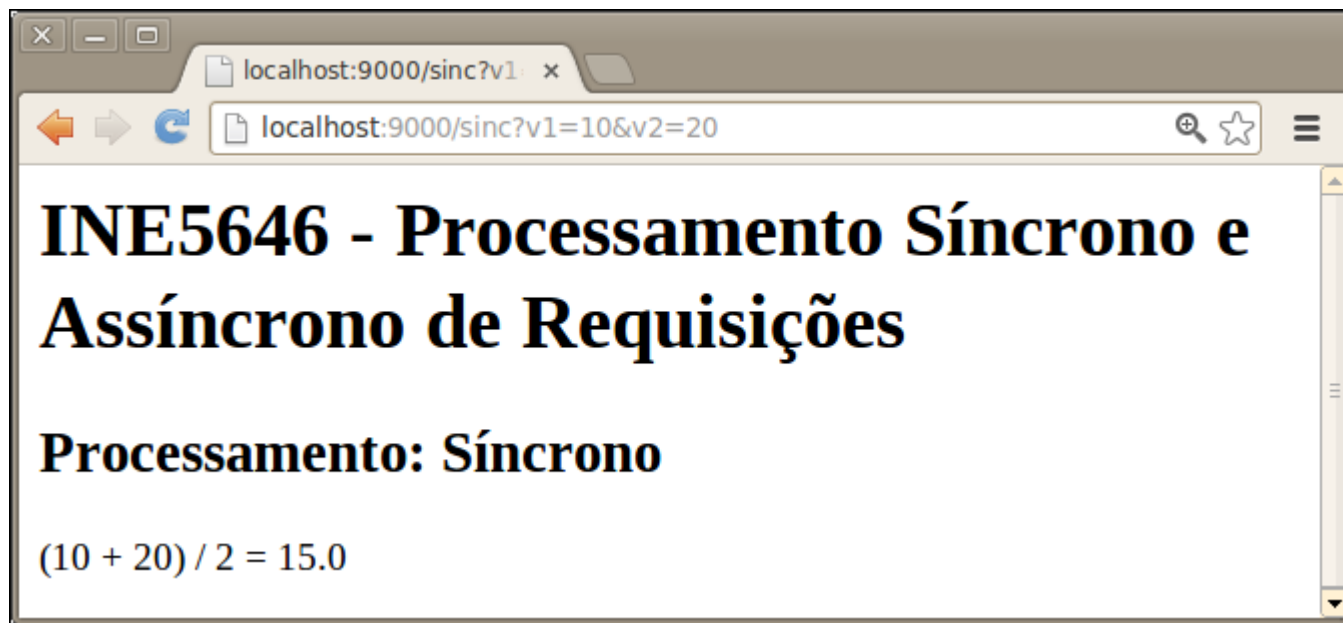
Proc. síncrono e assíncrono de requisições - Exemplo

- Página inicial:



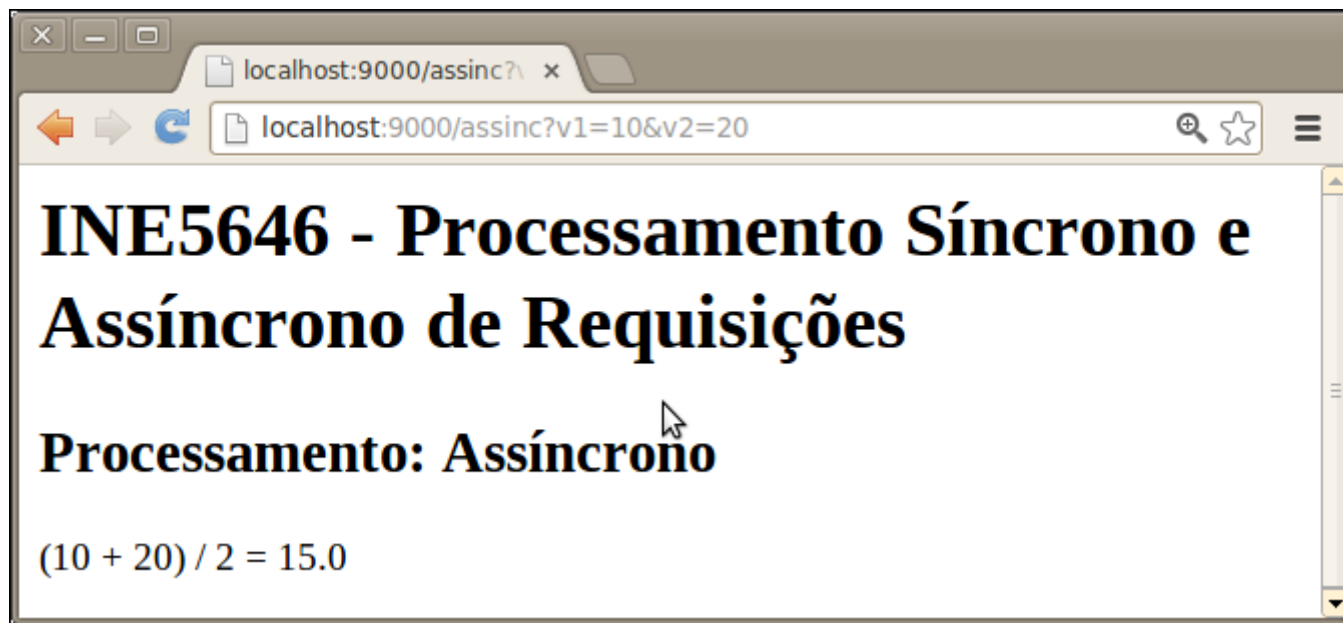
Proc. síncrono e assíncrono de requisições - Exemplo

- Página resultante de processamento síncrono:



Proc. síncrono e assíncrono de requisições - Exemplo

- Página resultante de processamento assíncrono:



Proc. síncrono e assíncrono de requisições - Exemplo

- O Play utiliza o padrão MVC na camada 2:
 - Model:
 - Classe TarefaDemorada que calcula a média dos números.
 - View:
 - Página inicial
 - Página com o resultado
 - Control:
 - Objeto Application (subclasse de Controller)

Proc. síncrono e assíncrono de requisições - Exemplo

- **Model:**
 - Arquivo TarefaDemorada.scala:
 - O método calcule (linha 8) demora 3 segundos (linha 9) para calcular e retornar a média (linha 11).

```
1 package models
2
3 class TarefaDemorada {
4
5     val espera = 3000    // 3 segundos
6
7     // simula a execução de uma tarefa demorada
8     def calcule(v1: Int, v2: Int) = {
9         Thread.sleep(espera)
10
11         (v1 + v2) / 2.0    // retorna a media
12     }
13 }
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **View:** página inicial
 - Arquivo: index.scala.html
 - Este arquivo é compilado gerando uma função sem parâmetros.

```
1 <h1>INE5646 - Processamento Síncrono e Assíncrono de Requisições</h1>
2 <p>Processamento Síncrono:
3   <a href="sinc?v1=10&v2=20">http://localhost:9000/sinc?v1=10&v2=20</a>
4 </p>
5
6 <p>Processamento Assíncrono:
7   <a href="assinc?v1=10&v2=20">http://localhost:9000/assinc?v1=10&v2=20</a>
8 </p>
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **View:** página resposta
 - Arquivo: media.scala.html
 - Este arquivo é compilado gerando uma função com 4 parâmetros (tipo, v1, v2 e media).

```
1 @(tipo: String, v1: Int, v2: Int, media: Double)
2
3 <h1>INE5646 - Processamento Síncrono e Assíncrono de Requisições</h1>
4 <h2>Processamento: @tipo</h2>
5 <p>(@v1 + @v2) / 2 = @media</p>
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **Control**: roteamento
 - O arquivo conf/routes indica, para cada padrão de requisição, qual método do controlador Application deve ser executado.

```
1 # requisição inicial
2 GET      /                controllers.Application.index
3
4 # requisição síncrona
5 GET      /sync            controllers.Application.calculaSincrono(v1: Int, v2: Int)
6
7 # requisição assíncrona
8 GET      /assinc          controllers.Application.calculaAssincrono(v1: Int, v2: Int)
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **Control**: roteamento (cont)
 - A linha 2, por exemplo, diz que requisições HTTP do tipo GET para a URL `http://localhost:9000` devem ser tratadas pelo método `index` do objeto `Application`.

```
1 # requisição inicial
2 GET      /                controllers.Application.index
3
4 # requisição síncrona
5 GET      /sync            controllers.Application.calculaSincrono(v1: Int, v2: Int)
6
7 # requisição assíncrona
8 GET      /assinc          controllers.Application.calculaAssincrono(v1: Int, v2: Int)
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **Control**: roteamento (cont)
 - A linha 5, por exemplo, diz que requisições HTTP do tipo GET para a URL `http://localhost:9000/sinc` devem ser tratadas pelo método `calculeSincrono` do objeto `Application`:
 - Como este método exige dois parâmetros (`v1` e `v2`) então a requisição esperada é `http://localhost:9000/sinc?v1=10&v2=20`

```
1 # requisição inicial
2 GET      /                controllers.Application.index
3
4 # requisição síncrona
5 GET      /sinc            controllers.Application.calculeSincrono(v1: Int, v2: Int)
6
7 # requisição assíncrona
8 GET      /assinc          controllers.Application.calculeAssincrono(v1: Int, v2: Int)
```


Proc. síncrono e assíncrono de requisições - Exemplo

- **Control**: objeto controlador.
 - Arquivo Application.scala (1/6):
 - O objeto Application, como qualquer controlador, tem o objetivo de processar as requisições e definir qual página deverá ser gerada e enviada como resposta ao browser.

```
1 package controllers
2
3 import play.api.mvc.{Controller, Action}
4 import play.api.libs.concurrent.Execution.Implicits.defaultContext
5 import scala.concurrent.Future
6
7 import models.TarefaDemorada
8
9 object Application extends Controller {
10
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **Control**: objeto controlador.
 - Arquivo Application.scala (2/6):
 - O método index retornará uma página HTML gerada a partir da execução da função index.

```
11  def index = Action {  
12    Ok(views.html.index())  
13  }  
14
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **Control**: objeto controlador.
 - Arquivo Application.scala (3/6):
 - O método calculeSincrono:
 - Instancia um objeto da classe TarefaDemorada (linha 16)
 - Determina o valor da media (linha 17)
 - Após a media ter sido definida, retorna uma página HTML gerada a partir da execução da função media.
 - Obs: por ser síncrono o tempo de execução do método é pouco mais de 3 segundos (por causa da linha 17).

```
15  def calculeSincrono(v1: Int, v2: Int) = Action {  
16      val tarefa = new TarefaDemorada  
17      val media = tarefa.calcule(v1,v2)  
18  
19      Ok(views.html.media("Síncrono", v1, v2, media))  
20  }  
21
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **Control**: objeto controlador.
 - Arquivo Application.scala (4/6):
 - O método **calculeAssincrono**:
 - O cálculo da média (linhas 24 a 26) será realizado em outra thread. Estas 3 linhas são executadas em poucos milésimos de segundo.
 - **Todo o método é executado em poucos milésimos de segundo.**

```
22  def calculeAssincrono(v1: Int, v2: Int) = Action {
23    val tarefa = new TarefaDemorada
24    val futureMedia = Future {
25      tarefa.calcule(v1,v2)
26    }
27
28    Async {
29      futureMedia.map (media => Ok(views.html.media("Assíncrono", v1, v2, media)))
30    }
31  }
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **Control**: objeto controlador.
 - Arquivo Application.scala (5/6):
 - O método calculeAssincrono:
 - A variável **futureMedia** é um container que, em algum momento no futuro, receberá e armazenará o valor resultante da execução do método calcule do objeto tarefa.

```
22  def calculeAssincrono(v1: Int, v2: Int) = Action {
23    val tarefa = new TarefaDemorada
24    val futureMedia = Future {
25      tarefa.calcule(v1,v2)
26    }
27
28    Async {
29      futureMedia.map (media => Ok(views.html.media("Assíncrono", v1, v2, media)))
30    }
31  }
```

Proc. síncrono e assíncrono de requisições - Exemplo

- **Control**: objeto controlador.
 - Arquivo Application.scala (6/6):
 - O método calculeAssincrono:
 - As linhas 28 a 30 determinam: quando a variável futureMedia contiver a media (variável media) então execute a função views.html.media (com seus 4 parâmetros) para gerar a página HTML a ser enviada como resposta ao browser.

```
22  def calculeAssincrono(v1: Int, v2: Int) = Action {
23    val tarefa = new TarefaDemorada
24    val futureMedia = Future {
25      tarefa.calcule(v1,v2)
26    }
27
28    Async {
29      futureMedia.map (media => Ok(views.html.media("Assíncrono", v1, v2, media)))
30    }
31  }
```

Template Engine

- **Template engine** é o nome da tecnologia usada pelo Play para definir páginas dinâmicas geradas no servidor (camada 2).
- A estratégia usada é a seguinte:
 1. O desenvolvedor edita páginas HTML e insere expressões válidas na linguagem Scala. Estas páginas são chamadas de templates.
 2. As páginas são compiladas e é gerada uma função (código Scala ou Java) para cada página.
 3. O controlador (MVC) executa a função apropriada para gerar o código HTML da página a ser enviada como resposta ao browser.
- As funções, quando executadas, geram páginas HTML idênticas aos templates e, no lugar das expressões Scala, são inseridos os valores resultantes da avaliação das expressões.

Template Engine

- A página abaixo é um exemplo (já visto) de template:
 - **Expressões Scala** são precedidas por @.
 - A linha 1 define os parâmetros da função. Os parâmetros podem ser objetos de qualquer classe.
 - As linhas 4 e 5 mostram como os parâmetros podem ser usados.

```
1 @(tipo: String, v1: Int, v2: Int, media: Double)
2
3 <h1>INE5646 - Processamento Síncrono e Assíncrono de Requisições</h1>
4 <h2>Processamento: @tipo</h2>
5 <p>(@v1 + @v2) / 2 = @media</p>
```


Template Engine - Exemplo

- Na aplicação exemplo mostrada a seguir o usuário pode:
 - Cadastrar contatos (nome e telefone) em uma agenda telefônica.
 - Pesquisar contatos a partir do código de área do telefone (ex. Todos os contatos cujo código é 48)
 - A agenda, com seus contatos, é armazenada em memória, simulando um banco de dados.
- A aplicação possui 3 páginas:
 - A inicial (que mostra todo o conteúdo da agenda)
 - A de cadastro de contato
 - A de pesquisa por código de área
- Obs: a aplicação não contempla questões de validação dos dados informados pelo usuário.

Template Engine - Exemplo

- Página inicial:



Template Engine - Exemplo

- Página para adicionar contatos:



The screenshot shows a web browser window with the address bar displaying 'localhost:9000/adicionar'. The page title is 'INE5646 - Template Engine'. Below the title, the heading 'Agenda de Telefones - Adiciona Contato' is visible. The form contains three input fields: 'Nome:' (a long text box), 'Cód. de Área:' (a small text box), and 'Número:' (a medium text box). Below these fields is a button labeled 'Adicione'. At the bottom left of the form area, there is a link labeled '[Home]'.

Template Engine - Exemplo

- Página para pesquisar contatos:



Template Engine - Exemplo

- Cadastrando “João da Silva”:



The screenshot shows a web browser window with the address bar displaying 'localhost:9000/adicionar'. The page title is 'INE5646 - Template Engine' and the main heading is 'Agenda de Telefones - Adiciona Contato'. The form includes the following fields and elements:

- Nome:
- Cód. de Área:
- Número:
-
- [\[Home\]](#)

Template Engine - Exemplo

- Página inicial, após “João da Silva” ter sido cadastrado:



Template Engine - Exemplo

- Página inicial, após alguns contatos cadastrados:



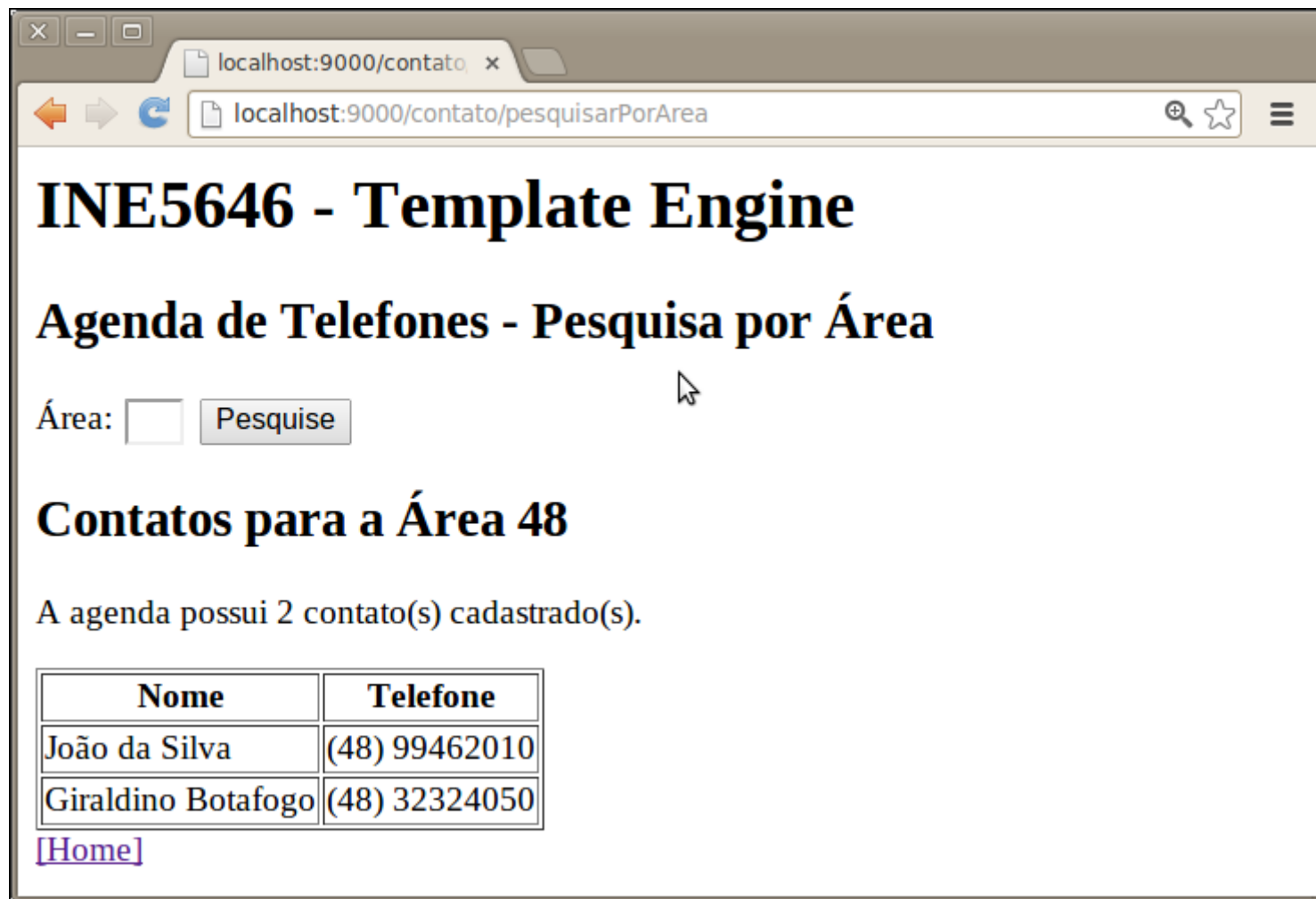
Template Engine - Exemplo

- Pesquisando contatos com código de área 48:



Template Engine - Exemplo

- Resultado: 2 contatos com código de área 48:



The screenshot shows a web browser window with the address bar displaying 'localhost:9000/contato/pesquisarPorArea'. The page title is 'INE5646 - Template Engine'. The main heading is 'Agenda de Telefones - Pesquisa por Área'. Below this, there is a search form with a label 'Área:', a text input field, and a 'Pesquise' button. The search results are displayed under the heading 'Contatos para a Área 48'. A message states 'A agenda possui 2 contato(s) cadastrado(s)'. Below this, a table lists two contacts: João da Silva with phone number (48) 99462010, and Giralдино Botafogo with phone number (48) 32324050. At the bottom, there is a link labeled '[Home]'.

INE5646 - Template Engine

Agenda de Telefones - Pesquisa por Área

Área:

Contatos para a Área 48

A agenda possui 2 contato(s) cadastrado(s).

Nome	Telefone
João da Silva	(48) 99462010
Giralдино Botafogo	(48) 32324050

[\[Home\]](#)

Template Engine - Exemplo

- Pesquisando contatos com código de área 49:
 - (não há nenhum contato com código de área 49)



Template Engine - Exemplo

- Resultado: nenhum contato com código de área 49:



Template Engine - Exemplo

- MVC – **Model**: Telefone.scala
 - Um telefone é caracterizado por seu código de área e pelo seu número.
 - Obs: Scala: a linha 3 define a classe, o método construtor e os métodos “getters” para os dois atributos. Não há “setters” → objetos imutáveis.

```
Telefone.scala x
1 package models.dados
2
3 case class Telefone(area: Int, numero: Long)
```

Template Engine - Exemplo

- MVC – **Model**: Contato.scala
 - Um contato é caracterizado por seu nome e pelo seu telefone.

```
Contato.scala x
1 package models.dados
2
3 case class Contato(nome: String, telefone: Telefone)
```

Template Engine - Exemplo

- MVC – **Model**: Agenda.scala
 - Uma agenda armazena seus contatos em um mapeamento (tabela de hash) onde a chave é o nome da pessoa (do contato).

```
Agenda.scala x
1 package models.dados
2
3 case class Agenda(contatos: Map[String, Contato] = Map()) {
4
5     def numContatos = contatos.size
6
7     def adicione(contato: Contato) = {
13 }
14
15     def pesquisePorArea(area : Int) = {
19 }
20 }
```

Template Engine - Exemplo

- MVC – **Model**: Agenda.scala
 - Em Java o atributo contatos seria representado como *Map<String, Contato> contatos;*
 - Linha 3: Scala: a expressão “= Map()” define o valor default (no caso, um mapeamento vazio) caso não seja passado um mapeamento na instanciação da agenda.

```
Agenda.scala x
1 package models.dados
2
3 case class Agenda(contatos: Map[String, Contato] = Map()) {
4
```

Template Engine - Exemplo

- MVC – **Model**: Agenda.scala
 - O método numContatos retorna o número de contatos da agenda, ou seja, o tamanho do mapeamento.

```
5  def numContatos = contatos.size  
6
```


Template Engine - Exemplo

- MVC – **Model**: Agenda.scala
 - O método `adicione` adiciona um novo contato na agenda.
 - Como o a agenda é um objeto imutável, o método retorna a própria agenda caso já exista um contato armazenado com o mesmo nome do contato a ser cadastrado (linhas 8 e 9) ou uma nova agenda (linha 11).

```
7      def adicione(contato: Contato) = {  
8          if (contatos.contains(contato.nome))  
9              this  
10         else {  
11             Agenda(contatos + (contato.nome -> contato))  
12         }  
13     }  
14
```

Template Engine - Exemplo

- MVC – **Model**: Agenda.scala
 - O método `pesquisePorArea` retorna uma (nova) agenda contendo apenas os contatos cujo telefone tem código de área igual ao parâmetro `area`.
 - A variável `contadosDaArea` é um mapeamento que contém apenas os contatos cuja expressão “`contato.telefone.area == area`” for verdadeira.

```
15     def pesquisePorArea(area : Int) = {  
16         val contadosDaArea =  
17             contatos filter {case (nome, contato) => contato.telefone.area == area}  
18         Agenda(contadosDaArea)  
19     }
```

Template Engine - Exemplo

- MVC – **Model**: BD.scala
 - O objeto BD representa, simuladamente, um banco de dados que armazena uma agenda.
 - O atributo agenda armazena uma agenda (inicialmente vazia – linha 6).

```
BD.scala
1 package models.persistencia
2
3 import models.dados.Agenda
4
5 object BD {
6     var agenda = Agenda()
7
8     def salve(agenda: Agenda) {this.agenda = agenda}
9
10    def leia = agenda
11 }
```

Template Engine - Exemplo

- MVC – **Model**: BD.scala
 - O método `salve` “armazena” uma agenda.
 - O método `leia` retorna a agenda.
 - Em Java, a linha 10 seria escrita assim:
 - `public Agenda leia() {return agenda;}`

```
BD.scala
1 package models.persistencia
2
3 import models.dados.Agenda
4
5 object BD {
6     var agenda = Agenda()
7
8     def salve(agenda: Agenda) {this.agenda = agenda}
9
10    def leia = agenda
11 }
```

Template Engine - Exemplo

- MVC – **Model**: CRUD.scala

```
CRUD.scala
1 package models
2
3 import models.persistencia.BD
4 import models.dados.Contato
5
6 class CRUD {
7     val agenda = BD.leia
8
9     def pesquiseTodos = agenda
10
11     def pesquisePorArea(area: Int) = {
12
13     }
14
15     def adicione(contato: Contato) = {
16
17     }
18
19     }
20 }
```

- A classe CRUD é responsável pelas operações de adição de contatos na agenda e de pesquisa.
- Obs: só estão implementadas as operações C e R.

Template Engine - Exemplo

- MVC – **Model**: CRUD.scala
 - O método **pesquiseTodos** retorna a agenda.
 - O método **pesquisePorArea** delega para a agenda a tarefa de retornar uma agenda cujos contatos tem telefone cujo código de área é igual ao passado como parâmetro.

```
6 class CRUD {  
7     val agenda = BD.leia  
8  
9     def pesquisaTodos = agenda  
10  
11     def pesquisaPorArea(area: Int) = {  
12         agenda.pesquisaPorArea area  
13     }  
14 }
```

Template Engine - Exemplo

- MVC – **Model**: CRUD.scala
 - O método **adicione** adiciona um contato na agenda.
 - Somente se a agenda foi modificada ela deve ser armazenada no banco (linha 19).
 - Obs: lembrar que agenda é objeto imutável.

```
15  def adicione(contato: Contato) = {  
16      val agendaAdicionada = agenda.adicione(contato)  
17  
18      if (agenda.numContatos != agendaAdicionada.numContatos)  
19          BD salve agendaAdicionada  
20  
21      agendaAdicionada  
22  }
```

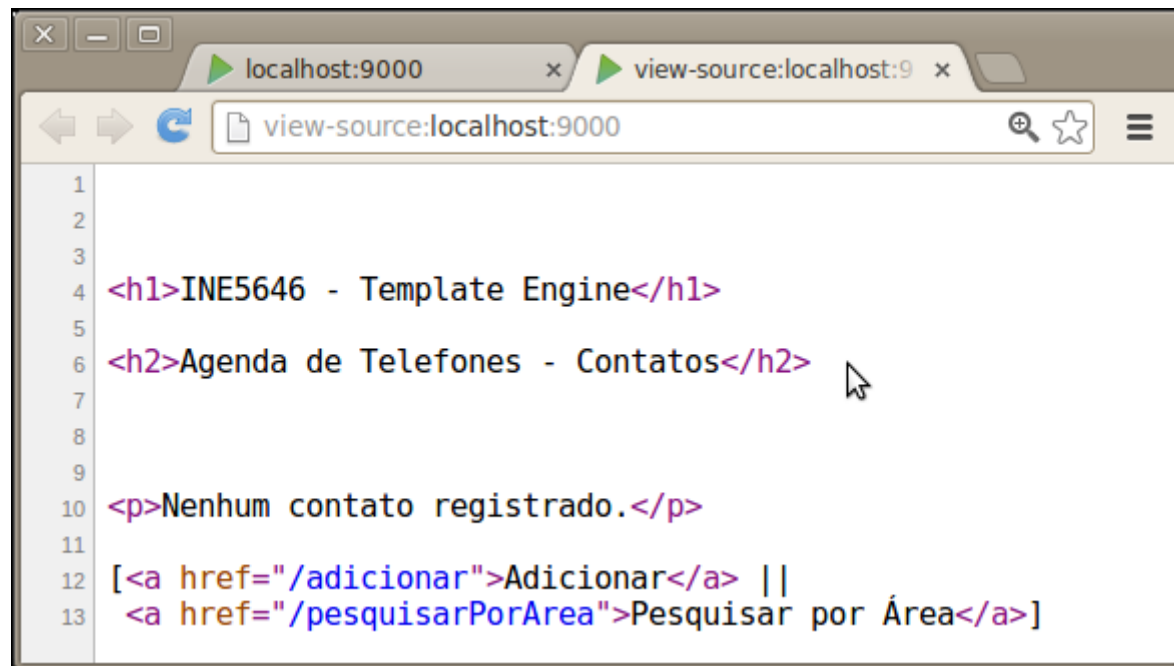

Template Engine - Exemplo

- MVC – **View**: index.scala.html:
 - Representa a página inicial.
 - No Play, **cada página será compilada e transformada em uma função** que, quando executada, retorna código HTML. No exemplo, a função possui um argumento do tipo Agenda (linha 1).

```
index.scala.html x
1 @(agenda: models.dados.Agenda)
2
3 @tags.cabecalho("Contatos")
4
5 @tags.mostraAgenda(agenda)
6
7 [<a href="@routes.Application.adicionar">Adicionar</a> | |
8  <a href="@routes.Application.pesquisarPorArea">Pesquisar por Área</a>]
```


Template Engine - Exemplo

- MVC – **View**: index.scala.html:
 - Quando a função é executada, gera o seguinte código HTML (quando a agenda não possui contatos):



```
1
2
3
4 <h1>INE5646 - Template Engine</h1>
5
6 <h2>Agenda de Telefones - Contatos</h2>
7
8
9
10 <p>Nenhum contato registrado.</p>
11
12 [<a href="/adicionar">Adicionar</a> ||
13  <a href="/pesquisarPorArea">Pesquisar por Área</a>]
```

Template Engine - Exemplo

- MVC – **View**: index.scala.html:
 - Linha 3: a invocação da função **tags.cabecalho** gerará o cabeçalho da página.
 - Linha 5: a invocação da função **tags.mostraAgenda** mostrará o conteúdo da agenda passada como parâmetro.

```
index.scala.html x
1 @(agenda: models.dados.Agenda)
2
3 @tags.cabecalho("Contatos")
4
5 @tags.mostraAgenda(agenda)
6
7 [<a href="@routes.Application.adicionar">Adicionar</a> ||
8  <a href="@routes.Application.pesquisarPorArea">Pesquisar por Área</a>]
```

Template Engine - Exemplo

- MVC – **View**: index.scala.html:
 - Linhas 7 e 8: os links (valores dos atributos href) são determinados pelo conteúdo do **arquivo conf/routes**.

```
index.scala.html x
1 @(agenda: models.dados.Agenda)
2
3 @tags.cabecalho("Contatos")
4
5 @tags.mostraAgenda(agenda)
6
7 [<a href="@routes.Application.adicionar">Adicionar</a> | |
8  <a href="@routes.Application.pesquisarPorArea">Pesquisar por Área</a>]
```

Template Engine - Exemplo

- MVC – **View**: index.scala.html:
 - Segundo o arquivo **conf/routes** (abaixo) o método `controllers.Application.adicionar` está associado com a URL `/adicionar`. Assim, o link gerado para a tag HTML “A” será:
 - **`http://localhost:9000/adicionar`**.
 - O mesmo raciocínio se aplica ao link para pesquisar por área.

```
5 GET      /adicionar      controllers.Application.adicionar
6
7 GET      /pesquisarPorArea controllers.Application.pesquisarPorArea
```

Template Engine - Exemplo

- MVC – **View**: adiciona.scala.html:
 - Página para adicionar um contato na agenda.

```
adiciona.scala.html x
1 @tags.cabecalho("Adiciona Contato")
2
3 <form action="@routes.Application.adicione" method="post">
4   <table>
5     <tr><td>Nome:</td>
6       <td><input type="text" name="nome" size="60"></td></tr>
7     <tr><td>Cód. de Área:</td>
8       <td><input type="text" name="area" size="2"></td></tr>
9     <tr><td>Número:</td>
10      <td><input type="text" name="numero" size="10"></td></tr>
11   </table>
12   <input type="submit" value="Adicione">
13 </form>
14
15 <a href="@routes.Application.index">[Home]</a>
```

Template Engine - Exemplo

- MVC – **View**: adiciona.scala.html:
 - Quando a função é executada gera o seguinte código HTML:



```
1 <h1>INE5646 - Template Engine</h1>
2
3
4 <h2>Agenda de Telefones - Adiciona Contato</h2>
5
6 <form action="/contato/adicione" method="post">
7   <table>
8     <tr><td>Nome:</td>
9       <td><input type="text" name="nome" size="60"></td></tr>
10    <tr><td>Cód. de Área:</td>
11      <td><input type="text" name="area" size="2"></td></tr>
12    <tr><td>Número:</td>
13      <td><input type="text" name="numero"
14        size="10"></td></tr>
15    </table>
16    <input type="submit" value="Adicione">
17  </form>
18  <a href="/">[Home]</a>
```

Template Engine - Exemplo

- MVC – **View**: pesquisaPorArea.scala.html:
 - Página para pesquisar por área.

```
pesquisaPorArea.scala.html x
1  @(area: Option[Int] = None, agenda: Option[models.dados.Agenda] = None)
2
3  @tags.cabecalho("Pesquisa por Área")
4
5  <form action="@routes.Application.pesquisePorArea" method="post">
6      Área: <input type="text" name="area" size="2">
7      <input type="submit" value="Pesquise">
8  </form>
9
10 @area match {
11     case None => {<p>Digite a área</p>}
12     case Some(a) => {@tags.agendaPorArea(a, agenda.get)}
13 }
14
15 <a href="@routes.Application.index">[Home]</a>
```

Template Engine - Exemplo

- MVC – **View**: pesquisaPorArea.scala.html:
 - Esta página é usada tanto para solicitar que o usuário digite o código de área a ser pesquisado como para mostrar o resultado da pesquisa. Em cada caso, o código HTML gerado é bem diferente.
 - Linha 1: Scala: uma variável do tipo Option “é uma collection” que contém ou um dado (Some(dado)) ou não contém nada (None).
 - Exemplo: o valor da variável **area** pode ser **None** (representando nenhum dado) ou então **Some(48)** representando o número 48.

```
1 @(area: Option[Int] = None, agenda: Option[models.dados.Agenda] = None)  
2
```


Template Engine - Exemplo

- MVC – **View**: pesquisaPorArea.scala.html:
 - Caso o valor de **area** seja **None** então a página está sendo exibida para que o usuário digite a área a ser pesquisada.
 - Caso o valor de **area** seja **Some(a)** então a variável “a” representa o código de área pesquisado e, neste caso, a variável **agenda** (que também é um Option) contém a agenda resultante da pesquisa. O método **get** retorna o objeto da classe Agenda contida no Option agenda.

```
10 @area match {  
11     case None => {<p>Digite a área</p>}  
12     case Some(a) => {@tags.agendaPorArea(a, agenda.get)}  
13 }
```

Template Engine - Exemplo

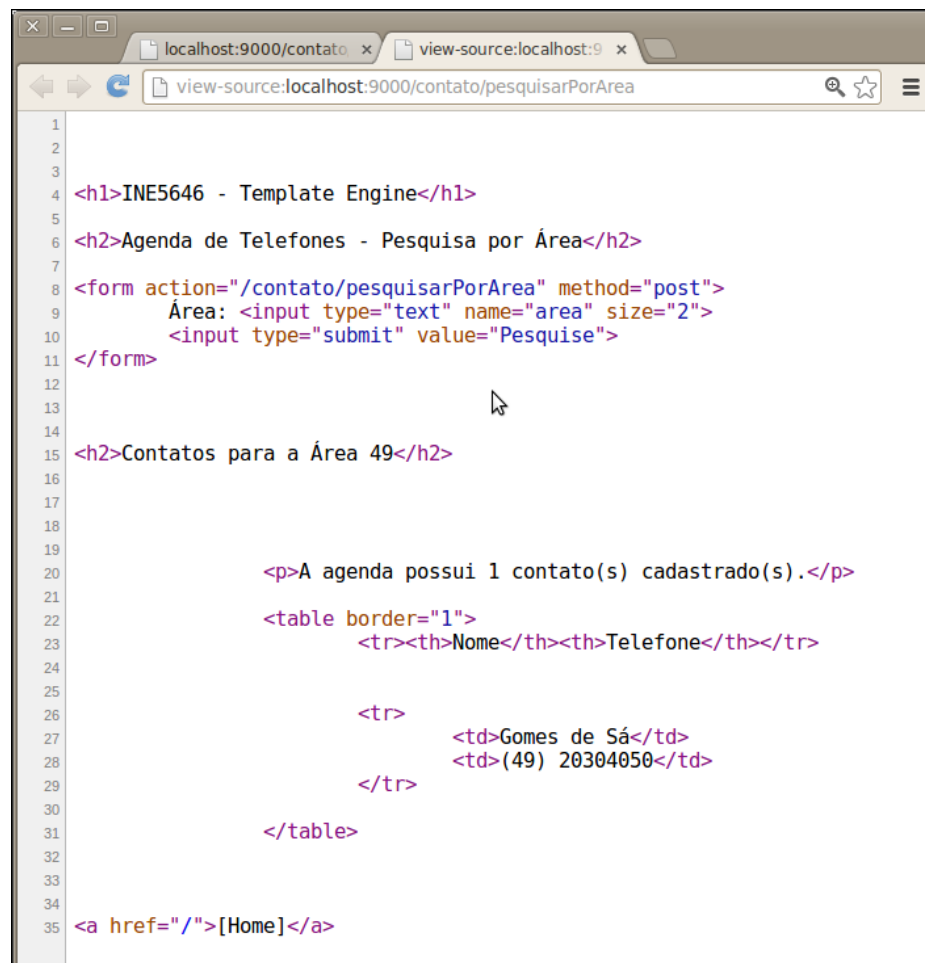
- MVC – **View**: pesquisaPorArea.scala.html:
 - Código HTML gerado para realizar pesquisa:



```
1
2
3
4 <h1>INE5646 - Template Engine</h1>
5
6 <h2>Agenda de Telefones - Pesquisa por Área</h2>
7
8 <form action="/contato/pesquisarPorArea" method="post">
9     Área: <input type="text" name="area" size="2">
10     <input type="submit" value="Pesquise">
11 </form>
12
13 <p>Digite a área</p>
14
15 <a href="/">[Home]</a>
```

Template Engine - Exemplo

- MVC – **View**: pesquisaPorArea.scala.html:
 - Código HTML gerado para resultado de pesquisa:



```
1
2
3
4 <h1>INE5646 - Template Engine</h1>
5
6 <h2>Agenda de Telefones - Pesquisa por Área</h2>
7
8 <form action="/contato/pesquisarPorArea" method="post">
9   Área: <input type="text" name="area" size="2">
10   <input type="submit" value="Pesquise">
11 </form>
12
13
14
15 <h2>Contatos para a Área 49</h2>
16
17
18
19
20 <p>A agenda possui 1 contato(s) cadastrado(s).</p>
21
22 <table border="1">
23   <tr><th>Nome</th><th>Telefone</th></tr>
24
25
26   <tr>
27     <td>Gomes de Sá</td>
28     <td>(49) 20304050</td>
29   </tr>
30
31 </table>
32
33
34 <a href="/">[Home]</a>
35
```

Template Engine - Exemplo

- MVC – **View**: `cabecalho.scala.html`:
 - A função **`cabecalho.scala.html`** gera um fragmento de código HTML (e não uma página).
 - Note que esta função é usada nas três páginas da aplicação exemplo.
 - Obs: sua função é equivalente às taglibs de JSP.

```
cabecalho.scala.html x
1 @(titulo: String)
2 <h1>INE5646 - Template Engine</h1>
3
4 <h2>Agenda de Telefones - @titulo</h2>
```

Template Engine - Exemplo

- MVC – **View**: agendaPorArea.scala.html:
 - A função agendaPorArea.scala.html gera um fragmento de código HTML (e não uma página).
 - Seu objetivo é mostrar o resultado de uma pesquisa.

```
agendaPorArea.scala.html x
1 @(area: Int, agenda: models.dados.Agenda)
2
3 <h2>Contatos para a Área @area</h2>
4
5 @tags.mostraAgenda(agenda)
```

Template Engine - Exemplo

- MVC – **View**: mostraAgenda.scala.html:
 - O objetivo é mostrar o conteúdo de uma agenda. Note (linhas 4 e 5) que a tabela (tag table) só será gerada caso a agenda tenha pelo menos um contato.

```
mostraAgenda.scala.html ✕
1  @(agenda: models.dados.Agenda)
2
3  @agenda.numContatos match {
4    case 0 => {<p>Nenhum contato registrado.</p>}
5    case _ => {
6      <p>A agenda possui @agenda.numContatos contato(s) cadastrado(s).</p>
7
8      <table border="1">
9        <tr><th>Nome</th><th>Telefone</th></tr>
10
11        @for((nome, contato) <- agenda.contatos) {
12          <tr>
13            <td>@nome</td>
14            <td>(@contato.telefone.area) @contato.telefone.numero</td>
15          </tr>
16        }
17      </table>
18    }
19  }
```

Template Engine - Exemplo

- MVC – **Control**: conf/routes:
 - Para cada requisição (padrão de URL) que chegar no servidor haverá um método a ser executado.

routes			
1	#	Página inicial	
2	GET	/	controllers.Application.index
3			
4	GET	/adicionar	controllers.Application.adicionar
5			
6	GET	/pesquisarPorArea	controllers.Application.pesquisarPorArea
7			
8	POST	/contato/adicione	controllers.Application.adicione
9			
10	POST	/contato/pesquisarPorArea	controllers.Application.pesquisePorArea

Template Engine - Exemplo

- MVC – **Control**: conf/routes:
 - Exemplo: quando o browser (camada 1) enviar a requisição `http://localhost:9000/adicionar` então o objeto `Application` (camada 2) irá executar o método **“adicionar”**.

routes			
1	#	Página inicial	
2	GET	/	controllers.Application.index
3			
4	GET	/adicionar	controllers.Application.adicionar
5			
6	GET	/pesquisarPorArea	controllers.Application.pesquisarPorArea
7			
8	POST	/contato/adicione	controllers.Application.adicione
9			
10	POST	/contato/pesquisarPorArea	controllers.Application.pesquisePorArea

Template Engine - Exemplo

- MVC – **Control**: Application.scala:

```
Application.scala x
1 package controllers
2
3 import play.api.mvc.{Controller, Action}
4 import play.api.data.Form
5 import play.api.data.Forms.{tuple, text, number}
6
7 import models.dados.{Agenda, Contato, Telefone}
8 import models.CRUD
9
10 object Application extends Controller {
11
12     def index = Action { ...
13     }
14
15     def adicionar = Action { ...
16     }
17
18     def pesquisarPorArea = Action { ...
19     }
20
21     def adicione = Action { implicit request => ...
22     }
23
24     def pesquisePorArea = Action { implicit request => ...
25     }
26
27 }
```

- A função do objeto Application é ser o controlador.
- Ele deve:
 - receber as requisições;
 - decidir como tratá-las (com o auxílio dos objetos do modelo);
 - definir qual página será enviada como resposta ao browser.

Template Engine - Exemplo

- MVC – **Control**: Application.scala:
 - O método **index** retornará a página inicial (linha 16) contendo a agenda com todos os seus contatos (linha 14).
 - A agenda é encontrada por meio do objeto **crud** (linhas 13 e 14).

```
12  def index = Action {  
13      val crud = new CRUD  
14      val agenda = crud pesquiseTodos  
15  
16      Ok(views.html.index(agenda))  
17  }
```

Template Engine - Exemplo

- MVC – **Control**: Application.scala:
 - Quando o controlador executar o método **adicionar** ele enviará como resposta ao browser a página HTML gerada pela função `views.html.adiciona`.
 - O mesmo raciocínio acontece com o método **pesquisarPorArea**.

```
19     def adicionar = Action {  
20         Ok(views.html.adiciona())  
21     }  
22  
23     def pesquisarPorArea = Action {  
24         Ok(views.html.pesquisaPorArea())  
25     }
```

Template Engine - Exemplo

- MVC – **Control**: Application.scala:
 - O objetivo do método **adicione** é adicionar o novo contato na agenda.
 - Linha 28 : define os campos do formulário enviado.
 - Linha 29 : obtém os valores dos campos.

```
27 def adicione = Action { implicit request =>
28   val form = Form(tuple("nome" -> text, "area" -> number, "numero" -> number))
29   val (nome, area, numero) = form.bindFromRequest.get
30   val crud = new CRUD
31
32   crud adicione(Contato(nome, Telefone(area, numero)))
33
34   Redirect(routes.Application.index)
35 }
```

Template Engine - Exemplo

- MVC – **Control**: Application.scala:
 - Linha 32 : objeto **crud** adiciona o novo contato.
 - Linha 34 : a resposta ao browser é: acesse a página inicial cuja URL está associada ao método **index** do objeto Application no arquivo conf/routes.

```
27 def adicione = Action { implicit request =>
28   val form = Form(tuple("nome" -> text, "area" -> number, "numero" -> number))
29   val (nome, area, numero) = form.bindFromRequest.get
30   val crud = new CRUD
31
32   crud adicione(Contato(nome, Telefone(area, numero)))
33
34   Redirect(routes.Application.index)
35 }
```

Template Engine - Exemplo

- MVC – **Control**: Application.scala:
 - O objetivo do método **pesquisePorArea** é obter uma agenda contendo apenas os contatos cujo número de telefone tiver código de área igual ao valor informado pelo usuário (linha 39).

```
37  def pesquisePorArea = Action { implicit request =>
38    val form = Form("area" -> number)
39    val area = form.bindFromRequest.get
40    val crud = new CRUD
41    val agendaPorArea = crud.pesquisePorArea(area)
42
43    Ok(views.html.pesquisaPorArea(Some(area), Some(agendaPorArea)))
44  }
```

Template Engine - Exemplo

- MVC – **Control**: Application.scala:
 - Linha 43: uma vez feita a pesquisa (linha 41), o resultado (**area** e **agendaPorArea**) deve ser usado para montar a página de resposta, gerada pela função **views.html.pesquisaPorArea**.

```
37  def pesquisaPorArea = Action { implicit request =>
38    val form = Form("area" -> number)
39    val area = form.bindFromRequest.get
40    val crud = new CRUD
41    val agendaPorArea = crud.pesquisaPorArea(area)
42
43    Ok(views.html.pesquisaPorArea(Some(area), Some(agendaPorArea)))
44  }
```