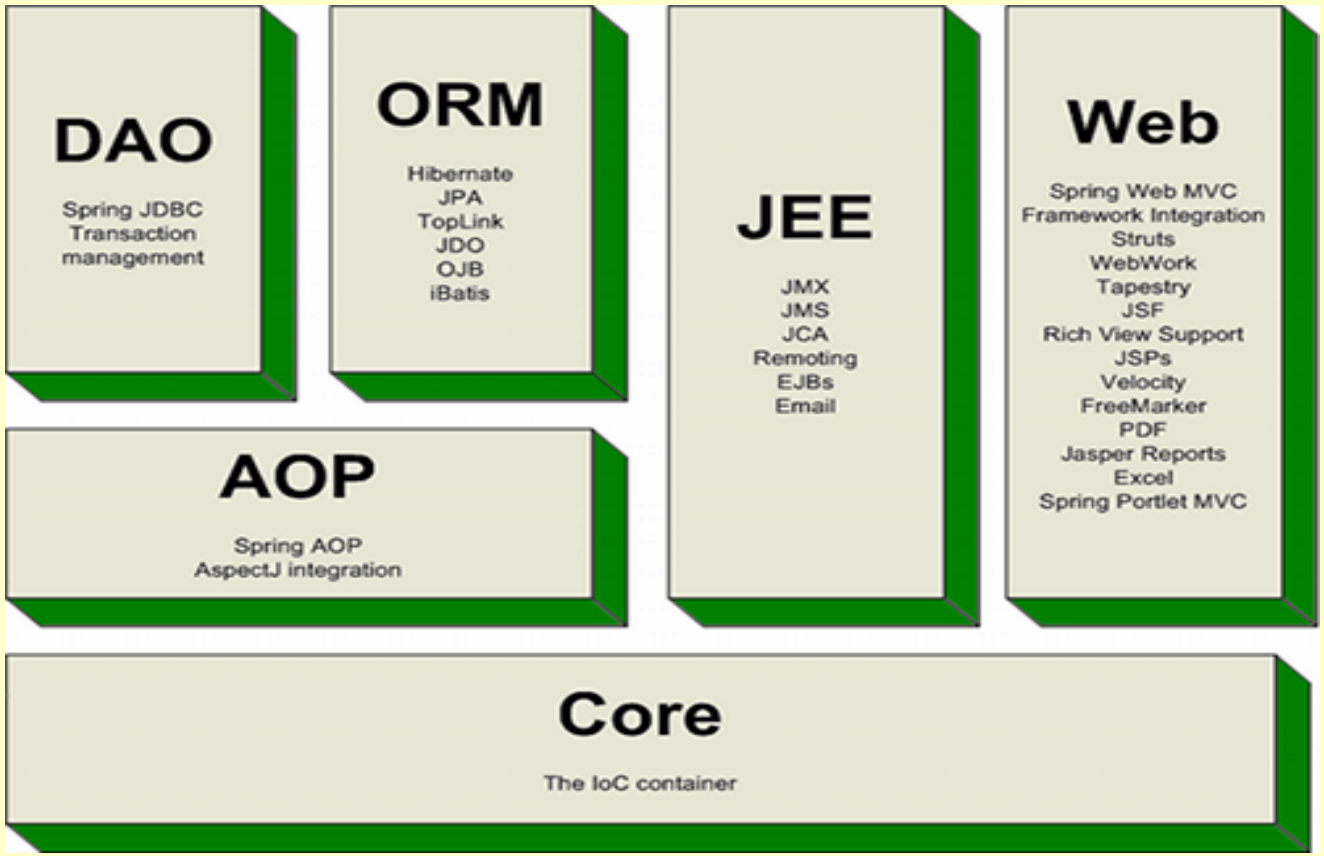
SPRING avanzado

SPRING

- SPRING es un framework amplísimo, que gracias a su simplicidad de diseño basada en beans, alberga y hace compatibles multitud de tecnologías Java estándar -Servlets- y no estándar- Hibernate-

SPRING


DAO
Spring JDBC
Transaction
management
ORM
Hibernate
JPA
TopLink
JDO
OJB
iBatis
JEE
JMX
JMS
JCA
Remoting
EJBs
Email
Web
Spring Web MVC
Framework Integration
Struts
WebWork
Tapestry
JSF
Rich View Support
JSPs
Velocity
FreeMarker
PDF
Jasper Reports
Excel
Spring Portlet MVC
AOP
Spring AOP
AspectJ integration
Core
The IoC container

XML vs JConfig

La configuración de una app Spring así como la iniciaización del contexto y dependencias, puede hacerse por medio de anotaciones y clases de configuración o por medio de xml auxiliares, detallados a parte.

Hay gran debate. Nosotros, usaremos XML

POA

- La programación orientada a aspectos es una de las características clave de Spring
- Aplicándola, puedo conseguir diseños más optimizados: menos código y mejor encapsulación de mi sistema
- Es un gran desconocido y está infrautilizada

POA CLAVES

- ASPECTO: La clase que estará relacionada con muchas otras y que representa un “aspecto” de las demás

- JOINPOINT: El punto de nuestro código fuente donde es posible la ejecución del aspecto

POA CLAVES

- POINTCUT: Define la intersección del Aspecto con un JOINPOINT.

- ADVICE: Acciones que se ejecutan alrededor de un POINTCUT y que amplían la lógica de negocio

POA Anotaciones

@Aspect

@PointCut

@Before

@After

@AfterThrowing

@AfterReturning

@Around

POA MAVEN dependency

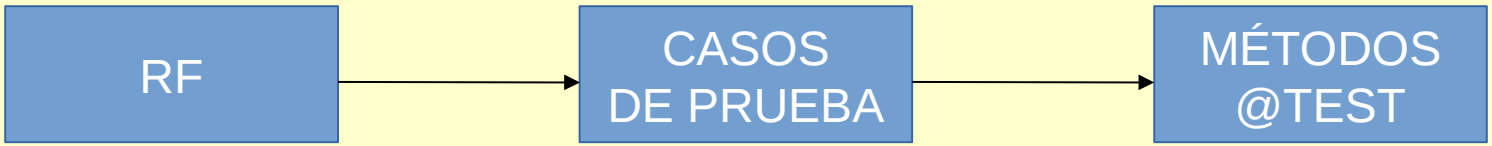
```
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-aop</artifactId>
<version>${versionSpring}</version>
</dependency>
```

TESTING Unitario

- El desarrollo dirigido por tests también conocido por su acrónimo inglés TDD se ha convertido casi en un mantra metodológico en nuestros días
- Repasemos los principios y las ventas que aporta Spring en esta fase de pruebas

TESTING Unitario

Requisitos → Casos de Prueba → Test



SRC/MAIN/JAVA → SRC/TEST/JAVA

MOCKS

- Los MOCKS son objetos sustitutivos de un entorno real
- Unas veces por seguridad otras por no disponibilidad, podemos simular el comportamiento de ciertos objetos en nuestro entorno de pruebas (servidor, base de datos, etc..)

MOCKS

MockHttpServletRequest

MockHttpServletResponse

Ambas clases son del paquete

org.springframework.mock.web

Y nos valen para imitar la petición y la respuesta a un HttpServlet

Mockito

- Es una bilblioteca complementaria a Junit, ampliamente usada en el testing unitario.

- Puedo definir *Mocks* de objetos de negocio o persistentes y simular y testar su comportamiento gracias

Anotaciones y métodos

@Mock

MockitoAnnotations.initMocks(this);

@RunWith(MockitoJUnitRunner.class)

@InjectMocks

(Mockito.when(method)).thenReturn(obj)

(Mockito.verify(mock, n*).method)

Anotaciones Junit

@Test

@Before

@Test(expected = RuntimeException.class)

Tareas Programadas

- Spring ofrece un API para la ejecución planificada de tareas, de forma que desde un contexto de dado, se realicen periódicamente determinadas o se lancen determinados servicios
- Es parecido al cron de Linux o al progrmador de tareas de Windows

Tareas Programadas

- La clase que contiene las tareas debe contener la anotación @Configuration

- Con @EnableScheduling se asegura la creación del servicio que en segundo plano irá tirando las tareas en se

Tareas Programadas

- Cada método que representa la tarea debe tener la siguiente cabecera

```
@Scheduled //anotación
public void task1() //& void
{
… cuerpo de la tarea
}
```

Tareas Programadas

Atributos posibles de @Scheduled

initialDelay milisegundos desde la instanciación del bean hasta la primera ejecución.

fixedDelay: milisegundos de intervalo entre ejecuciones finalizadas tras el fin de la última ejecución del método que deben pasar hasta una nueva ejecución del método…

Tareas Programadas

Atributos posibles de @Scheduled

...

fixedDelay: Lo usaremos para métodos que se deban ejecutar con una frecuencia muy alta y que sean susceptibles de durar bastante.

fixedRate: milisegundos de intervalo entre inicios de ejecución, (sin tenerse en cuenta si la última ejecución finalizó).

Tareas Programadas

cron: Permite definir una regla al estilo unix

```
  –--- segundos(0-59)
|    ---------------------- minuto (0-59)
| |  .------------ hora (0-23)
| | |  .--------- día del mes (1-31)
| | | |  .------ mes (1-12) o jan,feb,mar,(meses en inglés)
| | | | |  .--- día de la semana (0-6)
| | | | | |  -(domingo=0 ó 7) o sun,mon, (días en inglés)
| | | | | | |
* * * * * *
```

Tareas Programadas

EJEMPLOS estilo CRON

1.- 30 10 * * 1 --> Los lunes, a las 10:30

2.- 0,30 * * * 1 --> Los lunes, cada media hora

3.- 0,15,30,45 * * * * --> Cada 15 minutos

4.- 0 0/30 8-10 * * * 8, 8:30, 9, 9:30 y 10 diario

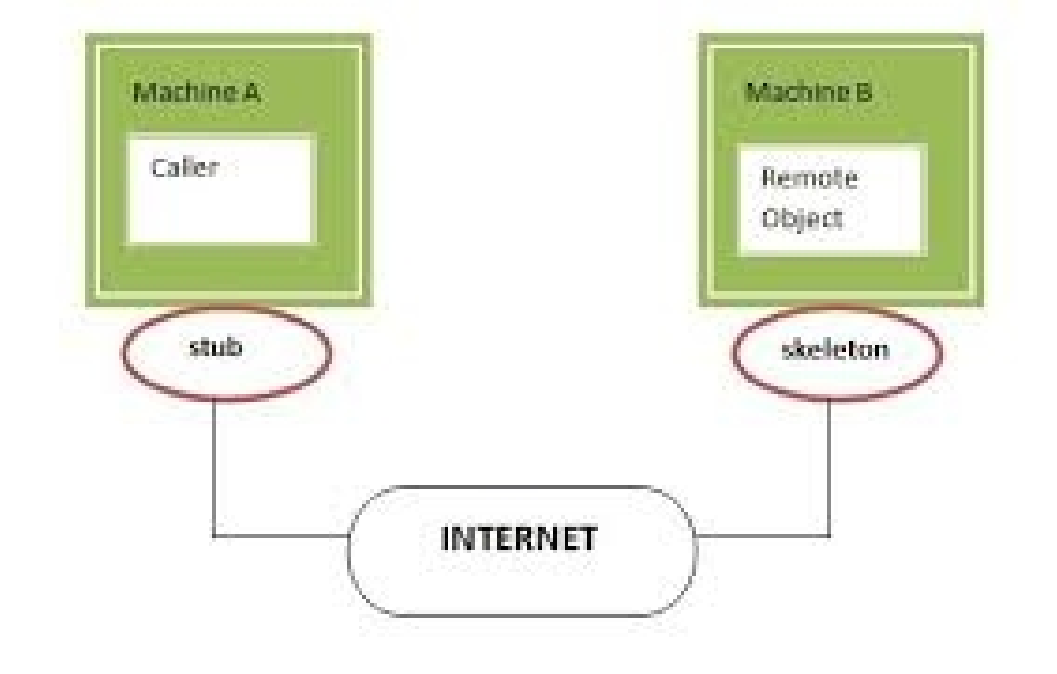
5.- 0 0 9-17 * * MON-FRI En diario,cada hora de 9 a 17

6.- 0 0 0 25 12 ? A las 24, todas las navidades

WEB REMOTING RMI

- Spring incorpora la posiblidad de definir y consumir servicios RMI entre servidores web
- Remote Method Invocation permite llamar a un método remoto, como si estuviera en local
- Es una forma de servicio web primario

WEB REMOTING RMI



WEB REMOTING RMI

- En el servidor donde se aloja la implementación, hace falta configurar el objeto e indicar qué clase lo implementa y en qué puerto

org.springframework.remoting.rmi.RmiServiceExporter

WEB REMOTING RMI

- En el cliente, hace falta “apuntar” al servicio mediante la URL, configurando para ello la clase

org.springframework.remoting.rmi.RmiProxyFactoryBean

SERVICIOS WEB REST

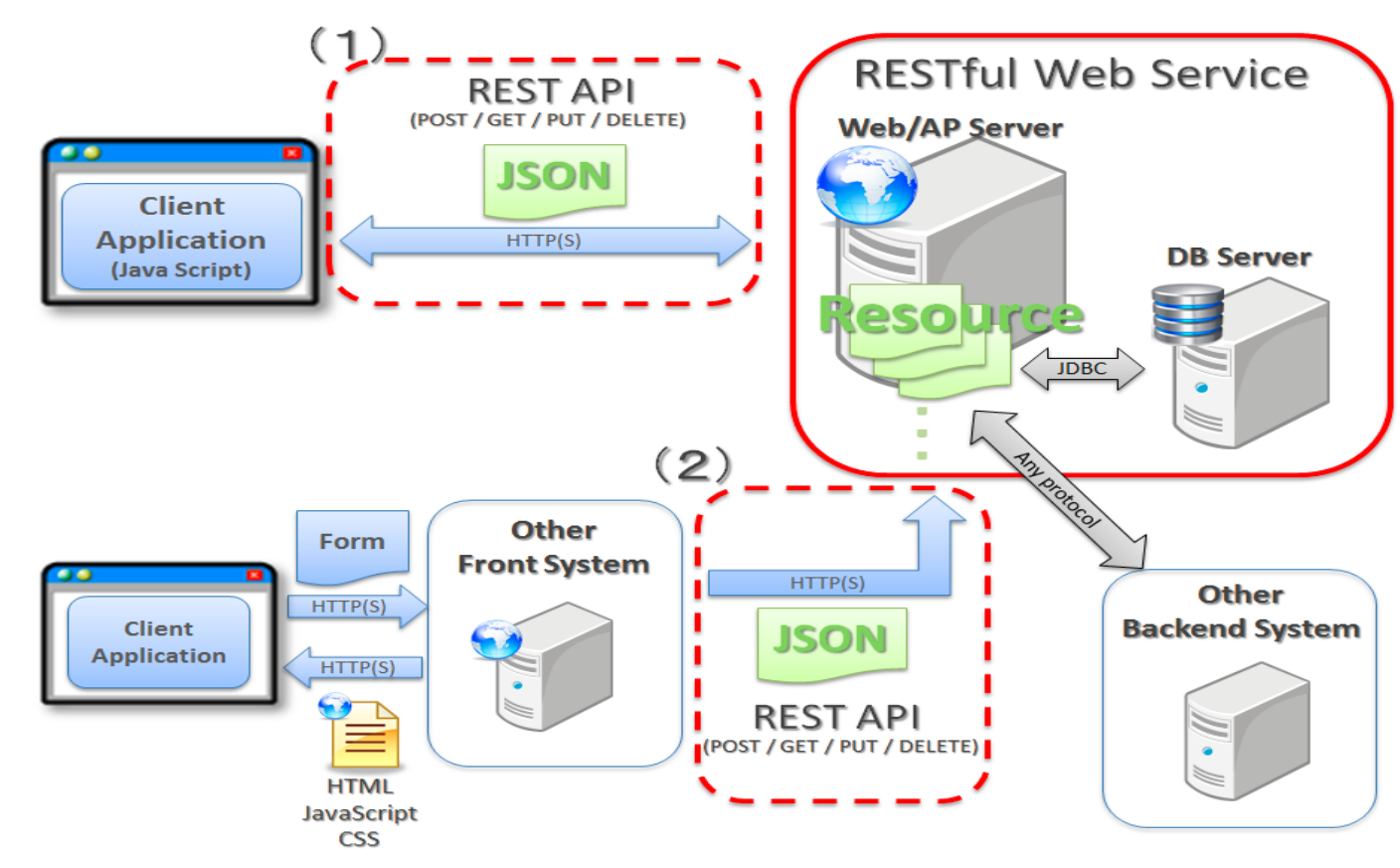
- Spring ofrece soporte a la construcción de una capa de Servicios REST mediante unas pocas anotaciones (JAX-RS)

- Los servicios REST por su economía y sencillez, han ganado terreno en los últimos años en las arquitecturas SOA

SERVICIOS WEB REST

@RestController con esta anotación en la cabecera de la clase, indico que representa una capa de servicios REST. Con ella me ahorro declarar @Responsebody en la cabecera de cada método

SERVICIOS WEB REST



SERVICIOS WEB REST

VENTAJAS

Independencia del cliente

Operadores conocidos

Protocolo HTTP

Estándar

JPA

- Spring presenta fácil integración con el Api estándar Java Persistence API

- Para ellos es necesario definir:
  - Beans de configuración
  - Beans de persistencia
  - Persistence.xml

JPA BeansConfiguración

Para ello es necesario configurar los siguientes beans

org.springframework.jdbc.datasource.DriverManagerDataSource

org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean

org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter

org.springframework.orm.jpa.JpaTransactionManager

JPA BeansPersistencia

Estos beans son “Entidades” (clases que representan una tabla de la base de datos) que generaremos por ingeniería inversa con la herramienta hbm2java

JPA Persistence.xml

ParaFichero alojado en src/main/resources que aúna los dos apartados anteriores: datos y configuración.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence-unit ..>
<class>entity.Animales</class>
</persistence-unit>
</persistence>
```

SESIONES SSecurity

SPRING Security es un framework que facilita la autenticación y la autorización en entornos Java, dando además respuesta a ataques y vulnerabilidades comunes y permitiendo la integración en entornos distribuidos (MVC)

SESIONES SSecurity

Ejemplos para descargar

Una vez descargado, importar el proyecto maven dentro de samples/xml/servletapi

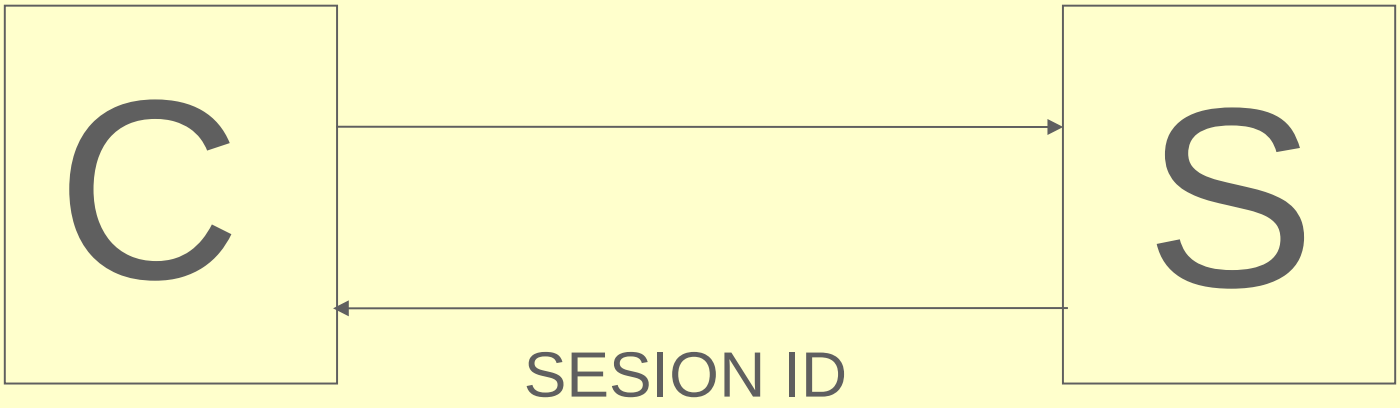
SESIONES HTTP

La sesión puede usarse como elemento de control para saber si:

- Una petición proviene de un usuario autenticado y autorizado para ello
- La petición se produce en un momento de validez (sesión no caducada)

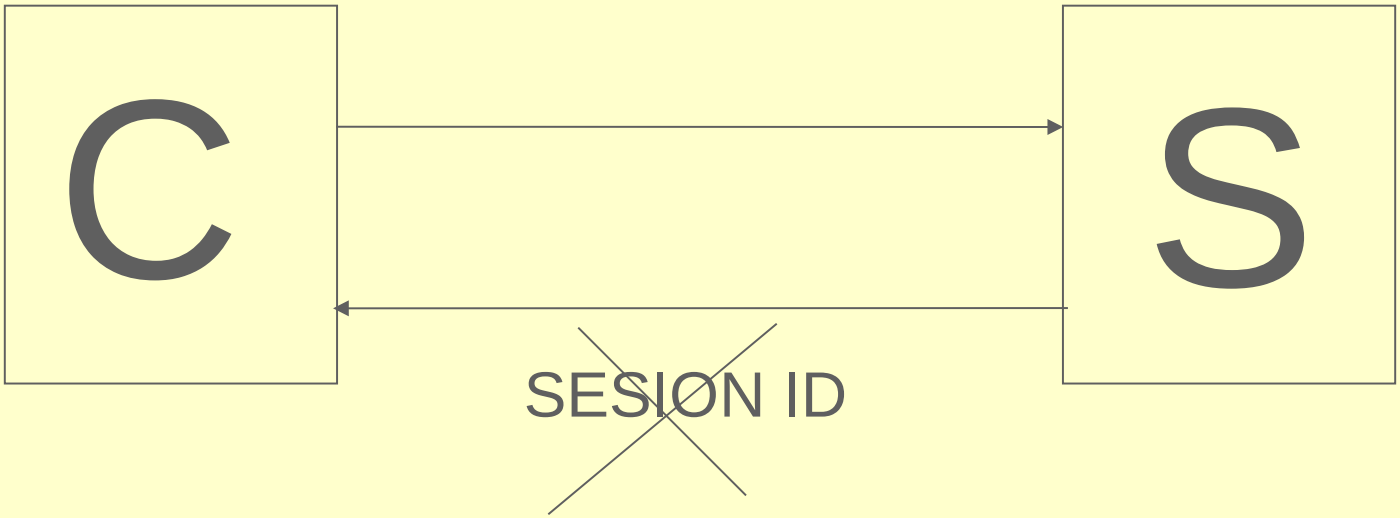
Es por tanto otro elemento básico para la gestión de la seguridad en entornos web

getSession () - (true)



getSession () - (true)

getSession (false)



getSession (false)

Fin de una Sesión

En algún momento, el usuario finalizará la sesión con el servidor.

El método invalidate() será invocado y con ello, el objeto HttpSession y los atributos asociados a él, desaparecen

Fin de una Sesión

- Podemos llamar a invalidate()
- Explícitamente (el usuario pulsa salir-logout)
- Implícitamente, por ausencia de peticiones
- Por defecto, una sesión muere a los 30’
- Fijado por setMaxInactiveInterval(secs)
- Parametrizado web.xml (minutos)

```
<session-config>
    <session-timeout>2</session-timeout>
</session-config>
```

Gestión de Sesión Spring

```
 <http create-session= *>
<session-management
session-fixation-protection = *
invalid-session-url = *>
</session-management>
</http>
```

Atributo invalid-session-url

```
<session-management invalid-session-
   url="/invalidSession.htm" />
</http>
```

Redirijo a la web/servicio deseado

Atributo create-session

always – SIEMPRE se crea una sesión

Nueva (si no existe)

ifRequired – Opción por defecto

never – Spring no crea (sí reutiliza)

stateless – No se usa ninguna sesión

Atributo
session-fixation-protection

¿QUÉ PASA SI ME LOGUEO DE NUEVAS?

none – No hace nada. Se mantiene la sesión
newSession – Se crea una nueva sesión, nueva, sin datos de la anterior
migrateSession – Se crea una nueva sesión, manteniendo los valores anteiores
ChangeSessionId – Sólo se cambia el ID (3.1)

MANEJO DE ERRORES

Lo mejor, es que los posibles errores o excepciones de un servicio se registren internamente, pero al usuario se le muestre un mensaje genérico, ocultando así las posibles pistas sobre los fallos que se han provocado el sistema

MANEJO DE ERRORES

Con la anotación @ControllerAdvice puedo indicar en un ecosistema Spring que la clase anotada así, se encarga de recibir excepciones

Además con @ExceptionHandler puedo indicar los métodos que hace “match” con cada tipo de excepción prevista

MANEJO DE ERRORES

```
@ControllerAdvice(basePackages =
{"org.springframework."} )
public class GestionaErrores {
    @ExceptionHandler(Throwable.class)
    public String errores (Exception e)
    {   return "error";
    }
```

Tipos Devueltos

- El controlador, puede devolver muchos tipos diferentes.
- Pese a su aparente complejidad, es realmente intuitivo manejarse con los diferentes tipos de datos devueltos
- Repasamos los principales tipos y su utilidad en las siguientes páginas

Tipos Devueltos

ModelAndView

Model

Tipo + @ModelAttribute

String

ResponseEntity + @ResponseBody

Recibiendo GET params

```
@RequestMapping("/bienvenido2")
    public String
    holaMundo2(@RequestParam(value="name",
    required=false, defaultValue="Mundo") String
    name, Model model) {
        model.addAttribute("name", name);
        return "bienvenido2";
    }
```

Recibiendo POST params

```
@RequestMapping(path = "/personaction", method =
    RequestMethod.POST)
public String formularioPersona (@ModelAttribute
    Persona persona,
Model model) {

System.out.println("Edad = "+ persona.getEdad());
System.out.println("Nombre = "+ persona.getNombre());
System.out.println("Descripcion = "+
    persona.getDescripcion());
model.addAttribute("personita", persona);
return "datospersona";
}
```

ViewResolver

Spring me permite asociar un string a una vista. Esa traducción se hace mediante una instancia de ViewResolver.

Puedo tener distintos ViewResolver con distinta prioridad.

ViewResolver

Spring por defecto está preparado para devolver JSP, pero puedo devolver cualquier otro tipo MIME.

Para ello debo definir una clase que

implemente ViewResolver y una Vista (que herede de View)

ViewResolver

```
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView" />
        <property name="prefix" value="/" />
        <property name="suffix" value=".jsp" />
        <property name="order" value="1" />
    </bean>

    <bean id="myviewresolver" class="springbasics4.ExeJsViewResolver">
        <property name="order" value="0" />
    </bean>
```

@Valid Annotation

Bean Validation API 1.1 JSR 349

Anotaciones

Hibernate Validator

@Valid Annotation

```
@Controller
public class LoginController {

    @RequestMapping(value = "/login", method = RequestMethod.POST)
   public String doLogin(@Valid @ModelAttribute("userForm") User
      userForm,
         BindingResult result, Map<String, Object> model) {
String salida = “menu”;

      if (result.hasErrors()) {
        salida = “formulario”
      }
       return salida;
   }
```

Validaciones

EJEMPLO **VALIDACIÓN JS**

```
const E_R_TELEFONO = /^\+\d{7,15}$/;
Var patron = new RegExp
(E_R_TELEFONO);

Patron.test (cadena) == true || false
```

Validaciones

EJEMPLO **VALIDACION JAVA**

final String PATRON_MAIL = "\\w ([.-]?\\w+)*@\\w+([.-]?\\w+)*(.\\w{2,4})+";

Validaciones

```
public static boolean emailValido (String email)
    {
        boolean bdev = false;

            Pattern p = Pattern.compile(PATRON_MAIL);
            Matcher m = p.matcher(email);
            bdev = m.matches();

        return bdev;
    }
```

i18n

La internacionalización se consigue en Spring mendiante el empleo de ficheros de propiedades en el directorio raíz. Sólo es necesario indicar el prefijo común de los distintos idiomas

i18n

```
<beanid="messageSource"
    class="org.springframework.context.support.ResourceBundleMes
    sageSource">
<property name="basename">
<value>mensajes</value>
</property>
</bean>

@Autowired
 private ResourceBundleMessageSource mensajes;
```

L10n

La localización la consigo pasando un parámetro en la request que después permite obtener el mensaje en un determinado idioma

L10n

ResourceBundleMessageSource.getMessage (BindingResult.getFieldError(), new Locale (lang));

Donde lang es el parámetro ISO de la lengua, que coincide con el sufijo del fichero de propiedades

Recursos estáticos

- Para acceder a recursos estáticos, debemos definir rutas explícitamente en el fichero de configuración (e indicarle así al Dispatcher, dónde encaminarse)

```
<mvc:resources mapping="/htm/**" location="/htm/" />
<mvc:resources mapping="/css/**" location="/css/" />
```

Spring WebFlow

Spring ofrece el mecanismo de “flows” (flujo o corriente) para definir recorridos entre estados (vistas) de forma declarativa.

Entre esas vistas, puedo definir “Acciones” que condicionan el flujo a la vista siguiente hasta alcanzar un estado final. Todo mediante un XML.

WebFlow Elementos

<flow> Elemento raíz

<view-state> Representa una vista

Id: nombre identificativo

View: nombre de la vista (puede omitirse)

<transition>

On: se corresponse con el valor del atributo recibido (tipo swicth)

To: nombre del siguiente estado

WebFlow Elementos

<action-state> Elemento de acción anida elementos evaluate y transition

<evaluate>

Expression: invoca a un método de algún bean, que realiza un cálculo y devuelve un resultado, que recibe el siguiente elemento transition

WebFlow Elementos

<end-state> Elemento final del flujo

Id: nombre identificativo

View: nombre la vista

Spring WebFlow

1 FLOWEXECUTOR Y N FLOWREGISTRY
PUEDO FORMAR FLOWS en BUCLES
PUEDO FORMAR SUBFLOWS
PUEDO USAR EL FLOWSCOPE

Ejemplos avanzados GitHub

Spring JAVAMAIL

Integrar JavaMail con nuestro sencillo
A través del api que ofrece la biblioteca
Podemos enviar correos desde nuestra cuenta de gmail u otra

Sólo hace falta activar el acceso en el caso de gmail a aplicaciones menos seguras
Acceso menos seguro

Dependencia JAVAMAIL

```
<dependency>
        <groupId>javax.mail</groupId>
        <artifactId>mail</artifactId>
        <version>1.4</version>
</dependency>
```

Subir Ficheros

```
@RequestMapping(value = "/subirUnFichero", method
 = RequestMethod.POST)
 public String
guardarFichero(@RequestParam("name") String[]
name, @RequestParam("file") MultipartFile file) {
```

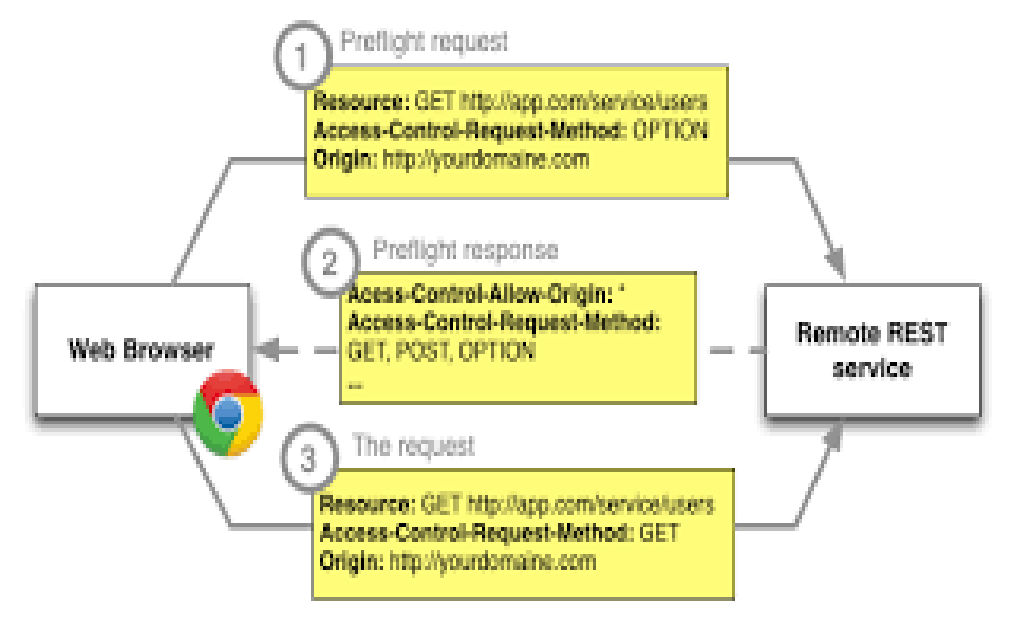
Simplemente indicando nombre y el tipo MultipartFile como parámetros de entrada, podemos subir ficheros al servidor

ATAQUES CORS

Los ataques Cross-origin Resource Sharing o CORS consisten en realizar peticiones por JS a un dominio distinto del que se realizó la petición original

Las peticiones realizadas por XMLHttpRequest son en principio denegadas pot defecto, según recomendación del W3C

ATAQUES CORS



ATAQUES CORS

Para configurar una respuesta apropiada desde el servidor que ofrece sus servicios a terceros, podemos definir:

Una política global (SERVIDOR- TOMCAT)
O Local (APP-SPRING)