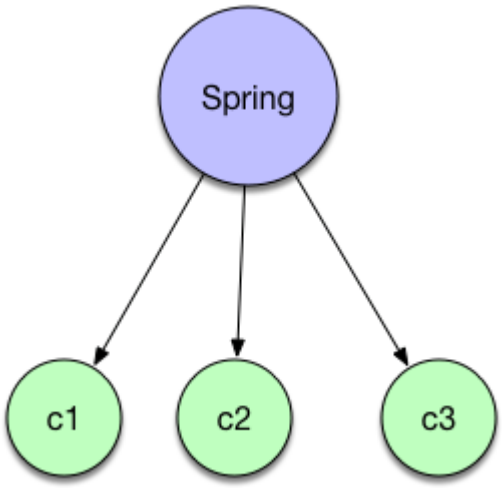
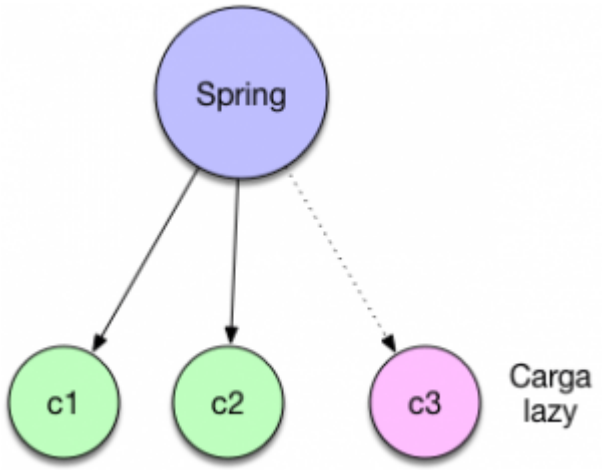
El uso de Spring Boot @Lazy , nos puede ser útil cuando construimos aplicaciones de gran tamaño y tenemos componentes en Spring Framework que por su funcionalidad tardan en cargar. Cuando Spring Framework se inicializa, por defecto genera una instancia de cada uno de sus componentes en memoria.



Spring Boot y componentes

Sin embargo puede haber situaciones en las que deseemos una carga vaga (lazy) de alguno de estos componentes.



Vamos a ver un ejemplo sencillo de como utilizar este enfoque. Supongamos que tenemos los siguientes interfaces y clases definidos con una aplicación de Spring Boot.

```
package com.arquitecturajava.lazy;

public interface ServicioA {

	String mensajeA();

}
```

```
package com.arquitecturajava.lazy;

import org.springframework.stereotype.Service;

@Service
public class ServicioAImpl implements ServicioA {

	public ServicioAImpl() {
```

```
                System.out.println("instancia ServicioA");
        }

        @Override
        public String mensajeA() {

                return "hola servicioA";
        }
}
```

```
package com.arquitecturajava.lazy;

public interface ServicioB {

        String mensajeB();

}
```

```
package com.arquitecturajava.lazy;

import org.springframework.stereotype.Service;

@Service
public class ServicioBImpl implements ServicioB {

        public ServicioBImpl() {

                System.out.println("instancia ServicioB");
        }

        @Override
        public String mensajeB() {
```

```
                return "hola servicioB";
        }
}
```

```
package com.arquitecturajava.lazy;

public interface ServicioC {

        String mensajeC();

}
```

```
package com.arquitecturajava.lazy;

import org.springframework.stereotype.Service;

@Service
public class ServicioCImpl implements ServicioC {

        public ServicioCImpl() {

                System.out.println(&amp;quot;instancia ServicioC&amp;quot;);
        }

        @Override
        public String mensajeC() {

                return &amp;quot;hola servicioC&amp;quot;;
        }
}
```

Spring Boot estandar

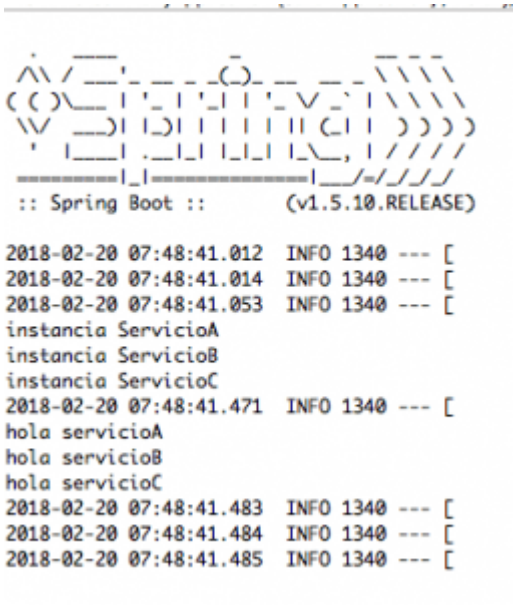
En estos momentos tenemos tres Beans que se cargarán con Spring Boot al arrancar vamos a ver como queda el programa principal de la consola para ejecutar los métodos de estos beans.

```
package com.arquitecturajava.lazy;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class LazyApplication implements CommandLineRunner {

	@Autowired
	ServicioA miservicioA;
	@Autowired
	ServicioB miservicioB;

	@Autowired
	ServicioC miservicioC;

	public static void main(String[] args) {
		SpringApplication.run(LazyApplication.class, args);
	}

	@Override
	public void run(String... arg0) throws Exception {
```

```
                System.out.println(miservicioA.mensajeA());
                System.out.println(miservicioB.mensajeB());
                System.out.println(miservicioC.mensajeC());
        }

}
```

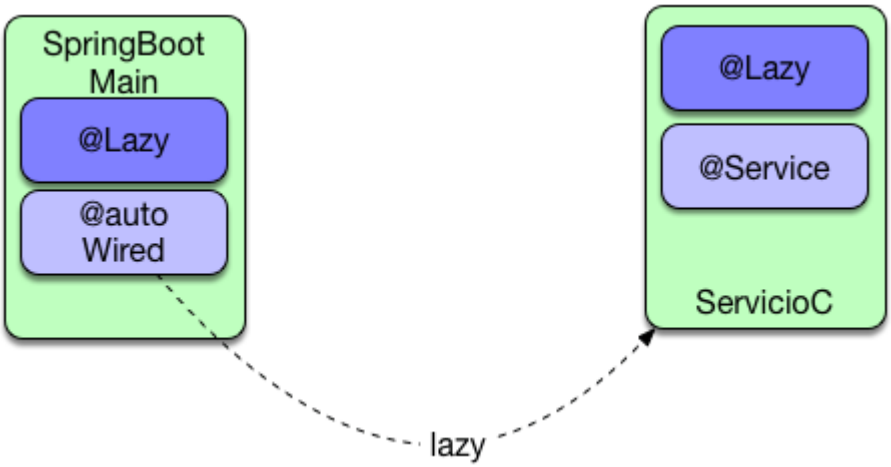
Spring Boot y la consola

Si ejecutamos el programa veremos que los servicios se instancian todos a la vez y luego se ejecutan las invocaciones a cada uno de sus métodos.



Spring Boot @Lazy

Cómo podemos modificar el comportamiento del programa para que el servicioC se cargue de forma vaga. Es bastante sencillo bastará con marcar que el servicio se puede instanciar de forma vaga y cuando hagamos el @Autowired confirmar que la carga es de esta forma.



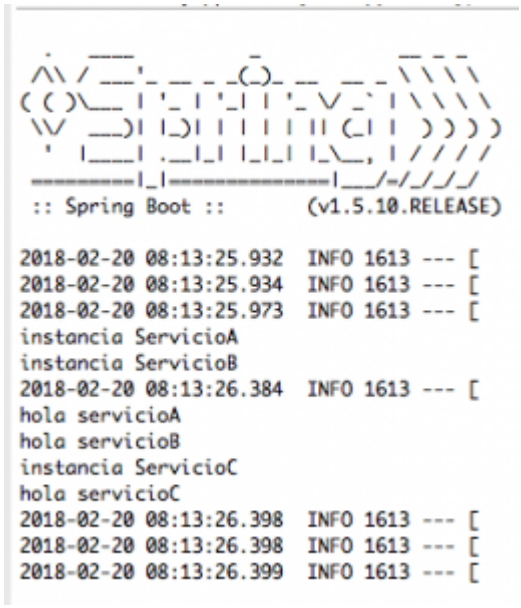
Vamos a ver las modificaciones a nivel de código:

```
@Lazy
@Service
public class ServicioCImpl implements ServicioC {

.....
}
```

En primer lugar anotamos el servicio, nos queda modificar el programa main y solicitar que el @Autowired incluya @Lazy:

```
@Lazy
@Autowired
ServicioC miservicioC;
```

Ya solo nos queda volver a ejecutar el programa y ver como el ServicioC se carga de forma lazy:



Acabamos de configurar Spring Boot @Lazy, permitiendo cargar parte de los beans de forma vaga cuando realmente se requieran. Recordemos que hoy por hoy en vez de autowired es muy habitual usar constructor inyection. En este caso hemos puesto @Autowired para ver la diferencia entre dos tipos de anotaciones muy relacionadas

Otros artículos relacionados:

- Spring @Autowired y la inyección de dependencias
- Angular Lazy Loading Modules y sus opciones
- Spring @Service , usando el patrón Servicio
- Curso Spring Boot y MicroServicios

Enlaces Externos:

1. Spring Boot