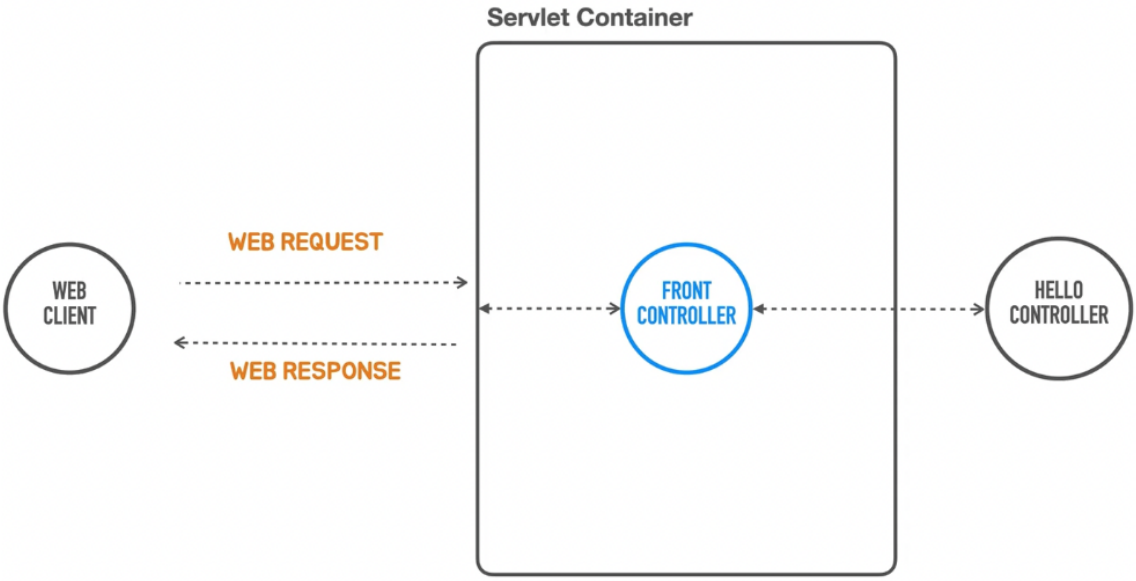
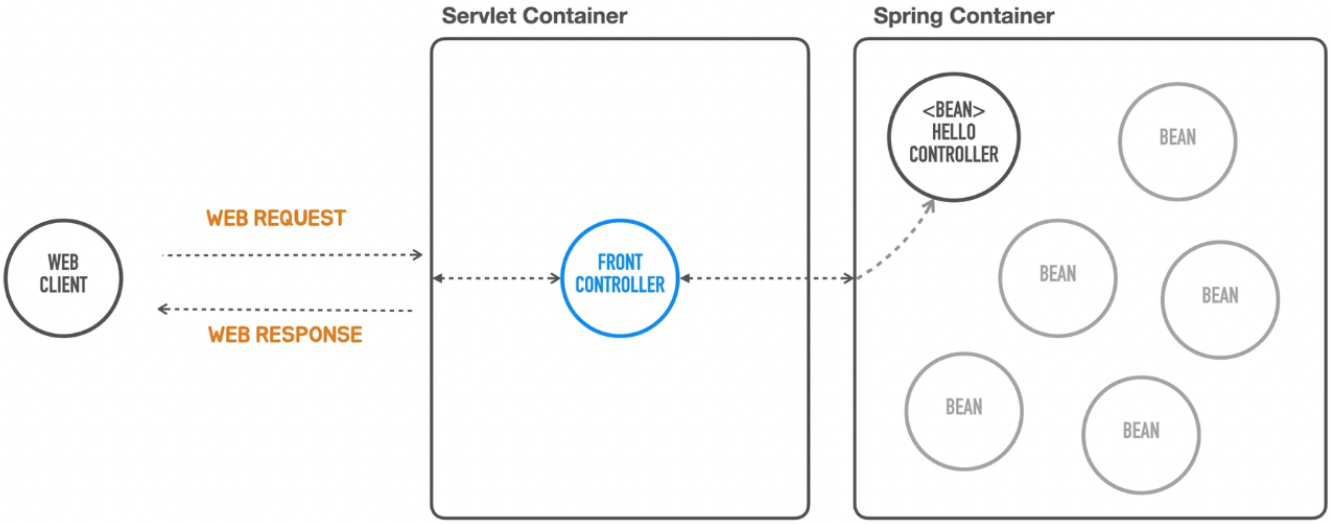
04. 독립 실행형 스프링 애플리케이션



(이전)



(목표)

서블릿 관련 코드들을 어떻게 개발중에 신경쓰지 않고 개발하게 되는지 보자

Hello Controller를 spring container 안에 넣어보자

container는 여러 개의 오브젝트를 가지고 있다가 필요할때 사용되도록 관리를 해주는 건데

프론트 컨트롤러를 직접 생성해서 서블릿 컨테이너에 넣어주었음

스프링 컨테이너는 살짝 다름

두가지가 필요한데 (spring container 안에)

1. 비지니스 로직을 담고 있는 비지니스 오브젝트(POJO: 어떤 클래스를 상속하지 않은) == 우리가 만든 애플리케이션 코드
2. 만든 코드들의 어떤식으로 구성할지에 대한 구성 정보가 들어있는 configuration metadata

두 개를 조합해서 빈 구성 -> 사용 가능한 시스템을 만들어 냄(서버 애플리케이션)

실습

1. spring container 생성
2. hello controller를 spring container에 집어 넣고
3. hello controller를 직접 생성해서 사용하는 대신에 spring container한테 요청해서 가져와서 사용

```
package tobyspring.helloboot;

import org.apache.catalina.startup.Tomcat;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.WebServer;
import org.springframework.boot.web.servlet.ServletContextInitializer;
import org.springframework.boot.web.servlet.server.ServletWebServerFactory;
import org.springframework.context.support.GenericApplicationContext;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;



public class HellobootApplication {

        public static void main(String[] args) {
                //    (   )     == application context == spring container
                // container
                GenericApplicationContext genericApplicationContext = new GenericApplicationContext();

                //
                genericApplicationContext.registerBean(HelloController.class);

                //       -> application context bean object
                genericApplicationContext.refresh();

                //Tomcat servlet webserver
                //ServletWebServerFactory serverFactory = new TomcatServletWebServerFactory(); ->
                TomcatServletWebServerFactory serverFactory = new TomcatServletWebServerFactory();

                //servlet container
                //servlet container      object
                WebServer webServer = serverFactory.getWebServer(servletContext -> {

                        //servlet container
                        servletContext.addServlet("frontController", new HttpServlet() {
                                @Override
                                protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
                                        //    ,

                                        // front controller servlet     ( servlet container )
                                        if(req.getRequestURI().equals("/hello") && req.getMethod().equals
(HttpMethod.GET.name())){ // mapping
                                                // http request
                                                String name = req.getParameter("name");

                                                //
                                                //             .
```

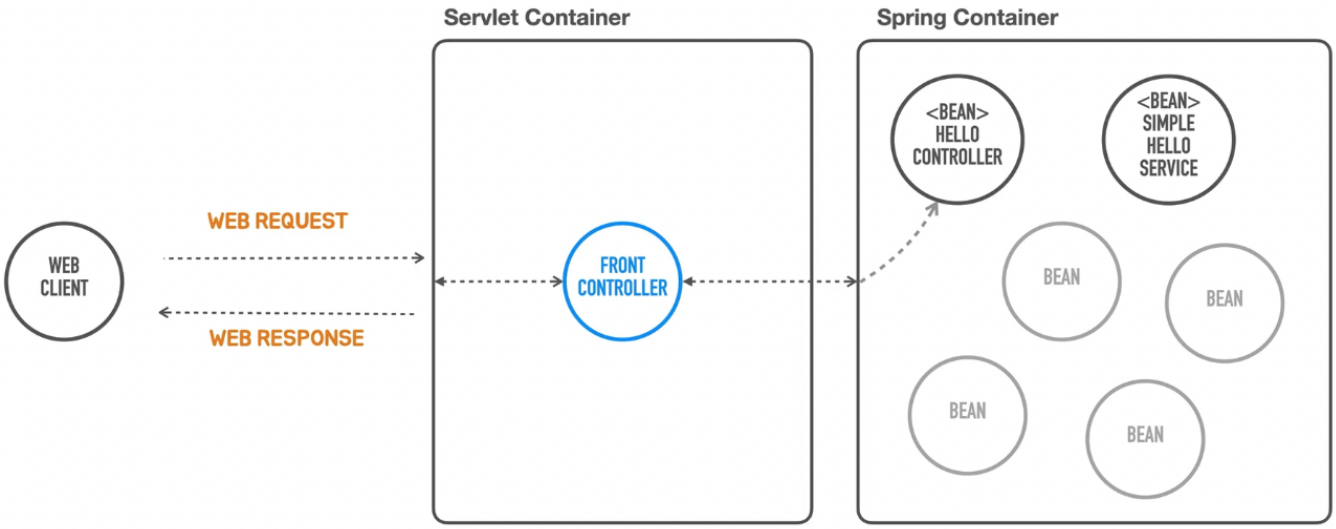
```
                                                // servlet container hello controller     controller
                                                HelloController helloController = genericApplicationContext.
getBean(HelloController.class);
                                                //  (  )
                                                String ret = helloController.hello(name);

                                                //resp.setStatus(HttpStatus.OK.value()); // 200
                                                resp.setContentType(MediaType.TEXT_PLAIN_VALUE);
                                                resp.getWriter().println(ret); //getWriter.print: object to
string .
                                        }
                                        else{
                                                resp.setStatus(HttpStatus.NOT_FOUND.value());
                                        }
                                }
                        }).addMapping("/*"); // /        -> front controller
                });

                // Tomcat servlet container
                webServer.start();
        }

}
```

의존 오브젝트 추가



지금까지 한 작업들이 spring container가 할수있는 여러 중요한 일들을 적용할 수 있는 기본 구조를 짜놨다는 것에 의의가 있음

spring container는 기본적으로 안에 어떤 오브젝트를 만들때 딱 한번만 만듬

spring container가 가지고 있는 오브젝트를 필요로 하는 여러 오브젝트(서블릿)들이 있을텐데, 그들이 요청할때마다 새로운 오브젝트를 만들어서 주는게 아니라 처음에 만들어둔 동일, 유일한 특정 오브젝트를 리턴해줌 (== 싱글톤 패턴)

spring container == singletone registry = 싱글톤 패턴을 사용하지 않고도 마치 싱글톤 패턴을 쓰는 것처럼 오브젝트를 재사용함

controller는 기본적으로 web controller이기 때문에 웹을 통해 들어온 어떤 요청사항을 검증하고 이를 비즈니스 로직을 수행하는 다른 오브젝트에게 요청을 보내서 결과를 돌려받고 웹 클라이언트에게 어떤 형식으로 돌려줄 것인가 정도 결정하면 됨 → 책임이 많이 줄어드는 것

HelloController

```
package tobyspring.helloboot;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Objects;

//    spring boot annotation

/**
 * controller   :
 * ex) name parameter  null  check
 */
public class HelloController {
    public String hello(String name){
        SimpleHelloService helloService = new SimpleHelloService();
        //if(name == null) throw NullPointerException;

        // controller
        //  service

        // Objects.requireNonNull(obj) : obj null
        // null
        return helloService.sayHello(Objects.requireNonNull(name));
    }
}

// spring container
```

SimpleHelloService

```
package tobyspring.helloboot;


//
public class SimpleHelloService {
    String sayHello(String name){
        return "Hello " + name;
    }
}
```

Dependency Injection

Spring DI container

의존관계

helloController는 SimpleHelloService가 변경되면 영향을 받음

의존관계에 있으면 변경사항이 있을때마다 코드를 수정해야함

ex) hello controller는 service 클래스가 바뀌면 클래스 선언부를 바꿔줘야함

그래서 java에서는 인터페이스를 사용해서 인터페이스에 의존하도록 함, 그리고 인터페이스를 구현한 클래스를 만듦

hello controller는 클래스 대신 인터페이스로 서비스를 의존하면 인터페이스를 구현한 서비스 클래스를 아무리 많이 만들어도 특정 클래스에 의존하고 있지 않기 때문에 hello controller 고치지 않아도 됨

하지만 런타임에서는 hello controller는 hello service 인터페이스의 구현체인 특정 클래스의 오브젝트에 의존해야함

그때는 어느 클래스의 오브젝트를 사용할 것인가 결정해야되야함 == dependency injection

DI에는 제 3의 존재가 필요함 -> 어셈블러

hello controller는 헬로 서비스 인터페이를 구현한 어떠한 클래스에 의존을 하는데 소스 코드 레벨에서는 의존하고 싶지 않은 것

SimpleHelloService에서 ComplexHelloService로 바꾸었다고 해서 소스코드는 고치고 싶지 않단 말이지

하지만 런타임에 필요하다면 simple hello service의 오브젝트 대신 ComplexHelloService의 오브젝트를 사용하겠다 등 결정이 있을텐데, 누군가가 가능하도록 만들어줘야겠지?

hello controller가 사용하는 오브젝트를 직접 new 키워드를 사용해서 만드는 대신에 외부에서 그 오브젝트를 만들어서 hello Controller가 사용할 수 있도록 주입을 해주는 것! -> assembler가 해줌!

Assembler가 마치 오브젝트를 가져다가 하나의 커다란 레고를 만드는 것처럼 원래는 직접 의존관계가 없는 클래스들의 오브젝트를 가져다가 서로 관계를 연결시켜주고 사용할 수 있도록 만들어주는 그런 역할을 함

이 assembler가 Spring Container다!

Spring container가 하는 역할

1. 우리가 메타정보를 주면 그걸 가지고 클래스의 싱글톤 오브젝트를 만들고
2. 이 오브젝트가 사용할 다른 의존 오브젝트가 있다면 그것을 주입해줌

주입을 해준다는 것은???

다른 오브젝트의 레퍼런스를 넘겨주는 것이다

어디서??

1. Hello Controller를 만들때 생성자 파라미터로 simple hello service의 오브젝트를 넣어줌 (물론 이때 파라미터 타입은 클래스가 구현하고 있는 hello service 인터페이스 타입으로 되어있음)
2. factory method로 bean 만들어서 넘기기
3. hello controller 클래스에 프로퍼티를 정의해서 setter method를 통해 simpleHelloService를 주입하는 방식

의존 오브젝트 DI 적용

기존 방식: Hello Controller가 Simple Hello Service라는 클래스의 오브젝트를 직접 생성해서 사용

이젠 스프링 빈으로 등록하고 container 가 어셈블러로써 Hello Controller가 simple hello service를 사용할 수 있도록 주입(DI) 하는 작업 수행

실습

spring container에 두개의 bean을 띄우고 둘 사이의 의존관계 만들기 (container가 제공하는 DI로)

```
package tobyspring.helloboot;

import org.apache.catalina.startup.Tomcat;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.WebServer;
```

```
import org.springframework.boot.web.servlet.ServletContextInitializer;
import org.springframework.boot.web.servlet.server.ServletWebServerFactory;
import org.springframework.context.support.GenericApplicationContext;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;



public class HellobootApplication {

        public static void main(String[] args) {
                //    (   )     == application context == spring container
                // container
                GenericApplicationContext genericApplicationContext = new GenericApplicationContext();

                //
                genericApplicationContext.registerBean(HelloController.class);
                genericApplicationContext.registerBean(SimpleHelloService.class); // controller       ?

                //       -> application context bean object
                genericApplicationContext.refresh();

                //Tomcat servlet webserver
                //ServletWebServerFactory serverFactory = new TomcatServletWebServerFactory(); ->
                TomcatServletWebServerFactory serverFactory = new TomcatServletWebServerFactory();

                //servlet container
                //servlet container      object
                WebServer webServer = serverFactory.getWebServer(servletContext -> {

                        //servlet container
                        servletContext.addServlet("frontController", new HttpServlet() {
                                @Override
                                protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
                                        //    ,

                                        // front controller servlet     ( servlet container )
                                        if(req.getRequestURI().equals("/hello") && req.getMethod().equals
(HttpMethod.GET.name())){ // mapping
                                                // http request
                                                String name = req.getParameter("name");

                                                //
                                                //              .
                                                // servlet container hello controller     controller
                                                HelloController helloController = genericApplicationContext.
getBean(HelloController.class);
                                                //  (  )
                                                String ret = helloController.hello(name);

                                                //resp.setStatus(HttpStatus.OK.value()); // 200
                                                resp.setContentType(MediaType.TEXT_PLAIN_VALUE);
                                                resp.getWriter().println(ret); //getWriter.print: object to
string .
                                        }
                                        else{
                                                resp.setStatus(HttpStatus.NOT_FOUND.value());
                                        }
                                }
                        }).addMapping("/*"); // /        -> front controller
                });
```

```
                // Tomcat servlet container
                webServer.start();
        }

}
```

HelloController

```
package tobyspring.helloboot;

import java.util.Objects;

//    spring boot annotation

/**
 * controller   :
 * ex) name parameter  null  check
 */
public class HelloController {
    // controller     ->  (Spring container) hello controller
    //
    private final HelloService helloService;


    // spring container Hello controller
    //     helloService
    // -> container     Hello service
    // ->
    public HelloController(HelloService helloService) {
        this.helloService = helloService;
    }

    public String hello(String name){
        //if(name == null) throw NullPointerException;

        // Objects.requireNonNull(obj) : obj null
        // null
        return helloService.sayHello(Objects.requireNonNull(name));
    }
}
```

HelloService interface

```
package tobyspring.helloboot;

/**
 * intellij refactor -> extract interface  class interface
 */
public interface HelloService {
    String sayHello(String name);
}
```

SimpleHelloService

```
package tobyspring.helloboot;


//
public class SimpleHelloService implements HelloService {
    @Override
    public String sayHello(String name){
        return "Hello " + name;
    }
}
```

DispatcherServlet으로 전환

spring container 적용하고 bean을 spring container에 등록한 뒤 DI가 적용되도록 만들어서 Hello service가 동작하는 것 확인함

이제:

servlet containerless로 가고싶은디?

- 매핑이 하드코딩 되어있음
- 요청의 파라미터(쿼리 스트링 등) 추출 후 넘기기도 하드코딩 되어있음(바인딩)

요런걸 언제까지나 servlet 관련 코드에 집어 넣을 순 없잖아?

spring을 이용해서 다른 전략으로 바꿔보자

dispatcher servlet 사용

dispatcher servlet: front controller의 많은 기능을 수행해주는 서블릿 클래스

HelloBootApplication

```
package tobyspring.helloboot;

import org.apache.catalina.startup.Tomcat;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.WebServer;
import org.springframework.boot.web.servlet.ServletContextInitializer;
import org.springframework.boot.web.servlet.server.ServletWebServerFactory;
import org.springframework.context.support.GenericApplicationContext;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.web.context.support.GenericWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;



public class HellobootApplication {

        public static void main(String[] args) {
                //    (   )     == application context == spring container
                // container
                GenericWebApplicationContext applicationContext = new GenericWebApplicationContext();

                //
                applicationContext.registerBean(HelloController.class);
                applicationContext.registerBean(SimpleHelloService.class); // controller       ?

                //       -> application context bean object
                applicationContext.refresh();

                //Tomcat servlet webserver
                //ServletWebServerFactory serverFactory = new TomcatServletWebServerFactory(); ->
                TomcatServletWebServerFactory serverFactory = new TomcatServletWebServerFactory();

                //servlet container
                //servlet container      object
                WebServer webServer = serverFactory.getWebServer(servletContext -> {

                        //servlet container
                        servletContext.addServlet("dispatcherServlet",
                                        // front controller
                                        //  servlet spring container   ? -> genericApplicationContext
                                        //        (GenericWebApplicationContext)
                                        //   ( dispatch)
                                        // TODO:  URL    bean(controller)        ->
                                                // mapping  servlet        controller class  mapping    ( )
                                        new DispatcherServlet(applicationContext)
                        ).addMapping("/*");
                });

                // Tomcat servlet container
                webServer.start();
        }

}
```

애노테이션 매핑 정보 사용

servlet container code 대신에 controller 클래스 안에다가 매핑 정보를 집어 넣는 방법을 써보자

HellobootApplication

```
package tobyspring.helloboot;

import org.apache.catalina.startup.Tomcat;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.WebServer;
import org.springframework.boot.web.servlet.ServletContextInitializer;
import org.springframework.boot.web.servlet.server.ServletWebServerFactory;
import org.springframework.context.support.GenericApplicationContext;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.web.context.support.GenericWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;



public class HellobootApplication {

        public static void main(String[] args) {
                GenericWebApplicationContext applicationContext = new GenericWebApplicationContext();

                applicationContext.registerBean(HelloController.class);
                applicationContext.registerBean(SimpleHelloService.class);
                applicationContext.refresh();

                TomcatServletWebServerFactory serverFactory = new TomcatServletWebServerFactory();
                WebServer webServer = serverFactory.getWebServer(servletContext -> {
                        servletContext.addServlet("dispatcherServlet",
                                        new DispatcherServlet(applicationContext)
                        ).addMapping("/*");
                });
                webServer.start();
        }
}
```

HelloController

```
package tobyspring.helloboot;

import org.springframework.web.bind.annotation.*;

import java.util.Objects;

@RequestMapping
public class HelloController {
    private final HelloService helloService;
    public HelloController(HelloService helloService) {
        this.helloService = helloService;
    }

    @GetMapping("/hello")
    @ResponseBody
    public String hello(String name){
        return helloService.sayHello(Objects.requireNonNull(name));
    }

}
```

스프링 컨테이너로 통합

servlet container를 만들고 초기화하는 작업을 스프링 컨테이너가 초기화 되는 과정 중에 일어나도로록 해보자

왜?

spring boot가 그렇게 함, 멋있으니까!!

HellobootApplication

```
package tobyspring.helloboot;

import org.apache.catalina.startup.Tomcat;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.WebServer;
import org.springframework.boot.web.servlet.ServletContextInitializer;
import org.springframework.boot.web.servlet.server.ServletWebServerFactory;
import org.springframework.context.support.GenericApplicationContext;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.web.context.support.GenericWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;



public class HellobootApplication {

        public static void main(String[] args) {
                GenericWebApplicationContext applicationContext = new GenericWebApplicationContext(){ //   ()
                        @Override
                        protected void onRefresh() { //spring container   servlet container
                                super.onRefresh();

                                TomcatServletWebServerFactory serverFactory = new
TomcatServletWebServerFactory();
                                WebServer webServer = serverFactory.getWebServer(servletContext -> {
                                        servletContext.addServlet("dispatcherServlet",
                                                        new DispatcherServlet(this)
                                        ).addMapping("/*");
                                });
                                webServer.start();
                        }
                };

                applicationContext.registerBean(HelloController.class);
                applicationContext.registerBean(SimpleHelloService.class);
                // spring container  (template method?, hook method?)
                applicationContext.refresh();


        }
}
```

자바코드 구성 정보 사용

우리가 만든 코드를 어떻게 오브젝트로 만들어서 컨테이너 내에 컴퍼넌트로 등록해두고 빈 간의 의존관계를 어떻게 어느 시점에 맺어줄것인가 에 대한 구성 정보를 spring container에 제공해야함

구성정보 제공 방법

factory method(어떤 오브젝트를 생성하는 로직을 담고 있음) 사용

factory method에서 bean 오브젝트를 다 생성, 의존 관계 주입도 하고 리턴하는 오브젝트를 스프링 컨테이너에게 빈으로 등록해서 사용하도록 한다.

왜 빈 오브젝트를 직접 만들어서 넣어주려고 하는가? (그럴 필요 없다며):

일반적으로 이 방법을 쓰지 않아도 되지만 어떤 경우에는 빈 오브젝트를 만들고 초기화하는 작업이 상당히 복잡한 경우가 있음. 이걸 복잡한 설정정보로 나열하는 대신에 자바코드로 작성하면 훨씬 간결해지고 이해하기 쉬움

HellobootApplication

```
package tobyspring.helloboot;

import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.WebServer;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.context.support.GenericWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;


@Configuration // spring container  class    factory method  class
public class HellobootApplication {
        @Bean  //  bean       spring container
        public HelloController helloController(HelloService helloService) { // helloController     factory
method -> spring container
                return new HelloController(helloService);
        }

        @Bean
        //return
        //
        public HelloService helloService(){ // helloService     factory method
                return new SimpleHelloService();
        }


        public static void main(String[] args) {
                //GenericWebApplicationContext    configuration
                // AnnotationConfigWebApplicationContext : annotation  java     application context
                AnnotationConfigWebApplicationContext applicationContext = new
AnnotationConfigWebApplicationContext(){
                        @Override
                        protected void onRefresh() {
                                super.onRefresh();

                                TomcatServletWebServerFactory serverFactory = new
TomcatServletWebServerFactory();
                                WebServer webServer = serverFactory.getWebServer(servletContext -> {
                                        servletContext.addServlet("dispatcherServlet",
                                                        new DispatcherServlet(this)
                                        ).addMapping("/*");
                                });
                                webServer.start();
                        }
                };

                //applicationContext.registerBean(HelloController.class);
                //applicationContext.registerBean(SimpleHelloService.class); ->   java code

                //
                applicationContext.register(HellobootApplication.class); // java code     bean object
                applicationContext.refresh();


        }
}
```

중요한것은?

@Configuration이 붙은 이 클래스가 annotationconfig를 이용하는 application context에 처음 등록된다는 사실임

왜냐하면 @Configuration이 붙은 클래스는 빈 factory method를 가지는 것 이상으로 전체 애플리케이션을 구성하는데에 필요한 중요한 정보들을 많이 넣을 수 있기 때문

@Component 스캔

이전까지 스프링 컨테이너에 빈을 등록하는 방식

1. class 정보를 registerBean() method에 넘겨줌
2. 팩토리 메소드를 만들어서 직접 인스턴스를 생성함

더 간결하게 빈을 등록할 수 있는 방법이 없을까?

@Component, @ComponentScan

HellobootApplication

```
package tobyspring.helloboot;

import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.WebServer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;


@Configuration

// @Component      ! ->
//
@ComponentScan
public class HellobootApplication {
        public static void main(String[] args) {
                AnnotationConfigWebApplicationContext applicationContext = new
AnnotationConfigWebApplicationContext(){
                        @Override
                        protected void onRefresh() {
                                super.onRefresh();

                                TomcatServletWebServerFactory serverFactory = new
TomcatServletWebServerFactory();
                                WebServer webServer = serverFactory.getWebServer(servletContext -> {
                                        servletContext.addServlet("dispatcherServlet",
                                                        new DispatcherServlet(this)
                                        ).addMapping("/*");
                                });
                                webServer.start();
                        }
                };

                applicationContext.register(HellobootApplication.class);
                applicationContext.refresh();


        }
}
```

HelloController

```
package tobyspring.helloboot;

import org.springframework.stereotype.Component;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.Objects;

//@RequestMapping

//   !     !
// -> spring container
//@Component
//@MyComponent
//@Controller
@RestController // -> @Controller + @ResponseBody
                //  @RequestMapping   dispatcherServlet @RestController         ->
public class HelloController {
    private final HelloService helloService;
    public HelloController(HelloService helloService) {
        this.helloService = helloService;
    }

    @GetMapping("/hello")
    @ResponseBody
    public String hello(String name){
        return helloService.sayHello(Objects.requireNonNull(name));
    }

}
```

- 장점
  - 매번 구성정보를 추가할 필요없이 내가 작성하는 클래스가 빈으로 등록됨
- 단점
  - 빈으로 등록되는 클래스 많아지게 되면 내가 이 애플리케이션을 실행했을때 정확하게 어떤 것들이 등록되는지 찾아보기 번거롭다

그럼에도 우리가 작성하는 애플리케이션 로직을 담은 코드를 빈으로 등록하는 방식에서는 거의 표준으로 쓰임 (편리해서)

번거롭다는 단점은 패키지 구성 관리와 모듈 잘 나눠서 관리를 잘하면 커버칠 수 있음

@Component의 재밌는 특징

@Component를 메타 애너테이션으로 가지고 있는 애너테이션을 붙여도 @Component이 붙은 것과 동일한 효과를 낸다

메타 애너테이션이란?

보통 애너테이션은 클래스나 메소드 앞에 붙음. 하지만 애너테이션도 자바 코드로 만들어진 것이니까 애너테이션 코드 위에 애너테이션을 붙힐 수 있다.

새로운 애너테이션을 만들어보자!

MyComponent

```
package tobyspring.helloboot;

import org.springframework.stereotype.Component;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME) //     ?
@Target(ElementType.TYPE) //     // TYPE => class interface  TYPE
@Component //   @Component  -> @MyComponent  @Component
public @interface MyComponent {
}
```

spring에서 제공하는 메타 애너테이션 예시

- @Controller
- @Service
- 등등

Bean의 생명주기 메소드

`TomcatServletWebServerFactory serverFactory = new TomcatServletWebServerFactory();` 와 `new DispatcherServlet(this)` 는 애플리케이션 기능을 위한 애는 아니지

하지만 이게 없으면 앱이 시작도 안하지

위 두개도 스프링 빈으로 등록해보자

스프링 컨테이너가 이걸 관리를 하면 나중에 굉장히 유연한 구성이 가능해짐

실습

등록하는 방법 중에서 Factory method를 사용하는 방법, bean annotation이 붙은 메소드를 정의하는 방법 사용할 거임

HellobootApplication

```
package tobyspring.helloboot;

import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.WebServer;
import org.springframework.boot.web.servlet.server.ServletWebServerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;


@Configuration

// @Component      ! ->
//
@ComponentScan
public class HellobootApplication {
        @Bean
        public ServletWebServerFactory servletWebServerFactory(){
                return new TomcatServletWebServerFactory();
        }
        @Bean
        public DispatcherServlet dispatcherServlet(){
                /*
                * application context   ?
                *  main method onRefresh application context  ?
                * 40 ~ 41
                * */
                return new DispatcherServlet();
        }

        public static void main(String[] args) {
                AnnotationConfigWebApplicationContext applicationContext = new
AnnotationConfigWebApplicationContext(){
                        @Override
                        protected void onRefresh() {
                                super.onRefresh();

                                ServletWebServerFactory serverFactory = this.getBean(ServletWebServerFactory.
class);
                                DispatcherServlet dispatcherServlet = this.getBean(DispatcherServlet.class);
                                //dispatcherServlet.setApplicationContext(this); ->  ?,    ?
                                //   , ???
                                        // spring container  ->   dispatcher servlet application context
                                        // bean
                                        // dispatcher servlet hierarchy  ApplicationContextAware
                                                //
                                        //   setApplicationContext
                                        //  spring container  dispatcher servlet   application context set

                                WebServer webServer = serverFactory.getWebServer(servletContext -> {
                                        servletContext.addServlet("dispatcherServlet", dispatcherServlet).
addMapping("/*");
                                });
                                webServer.start();
                        }
                };

                applicationContext.register(HellobootApplication.class);
                applicationContext.refresh();


        }
}
```

HelloController

```
package tobyspring.helloboot;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.Objects;

//@RequestMapping

//   !     !
// -> spring container
//@Component
//@MyComponent
//@Controller
@RestController // -> @Controller + @ResponseBody
                //  @RequestMapping   dispatcherServlet @RestController         ->

public class HelloController { //implements ApplicationContextAware { -> // ApplicationContextAware
application context
    private final HelloService helloService;
    private final ApplicationContext applicationContext; // ApplicationContextAware  application context   final

                                                         // -> final
                                                         //
                                                         //  container   ApplicationContext        ( )
    public HelloController(HelloService helloService, ApplicationContext applicationContext) { // 2.   (   )
        this.helloService = helloService;
        this.applicationContext = applicationContext;
    }

    @GetMapping("/hello")
    @ResponseBody
    public String hello(String name){
        return helloService.sayHello(Objects.requireNonNull(name));
    }

    //1. setApplicationContext application context
//    @Override
//    public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {
//        this.applicationContext = applicationContext;
//    }
}
```

SpringBootApplication

HellobootApplication

```
package tobyspring.helloboot;

import org.springframework.boot.web.embedded.tomcat.TomcatServletWebServerFactory;
import org.springframework.boot.web.server.WebServer;
import org.springframework.boot.web.servlet.server.ServletWebServerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;


@Configuration
@ComponentScan
public class HellobootApplication {
        @Bean
        public ServletWebServerFactory servletWebServerFactory(){ //factory method
                return new TomcatServletWebServerFactory();
        }
        @Bean
        public DispatcherServlet dispatcherServlet(){ // factory method
                return new DispatcherServlet();
        }

        public static void main(String[] args) {
                // 1.   -> * main  *
                        // +                    ->  (MySpringApplication)
                // 2.          (main   )
                MySpringApplication.run(HellobootApplication.class, args); // command line argument   args
                /*
                * run()
                * @Configuration
                * @ComponentScan factory method  spring container application
                * */
        }


}
```

MySpringApplication

```
package tobyspring.helloboot;

import org.springframework.boot.web.server.WebServer;
import org.springframework.boot.web.servlet.server.ServletWebServerFactory;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class MySpringApplication {
    public static void run(Class<?> applicationClass, String... args) {
        AnnotationConfigWebApplicationContext applicationContext = new AnnotationConfigWebApplicationContext(){
            @Override
            protected void onRefresh() {
                super.onRefresh();

                ServletWebServerFactory serverFactory = this.getBean(ServletWebServerFactory.class);
                DispatcherServlet dispatcherServlet = this.getBean(DispatcherServlet.class);
                WebServer webServer = serverFactory.getWebServer(servletContext -> {
                    servletContext.addServlet("dispatcherServlet", dispatcherServlet).addMapping("/*");
                });
                webServer.start();
            }
        };

        applicationContext.register(applicationClass);
        applicationContext.refresh();
    }
}
```

지금까지 해온 작업들이,,,

spring boot가 standalone으로 서블릿 컨테이너까지 포함하는 spring application을 동작시키는 그 원리가 담긴 코드이다.