**Spring**

**Springプロジェクト**

・springスタータープロジェクト spring boot を使ったプロジェクト

・springレガシープロジェクト spring boot を使わず個々のフレームワークを個別に組み込むプロジェクト

**Templstes**

・Simple Java

・Simple Spring Maven

・Simple Spring Web Maven

**Mavenプロジェクト**

基本的にアーキタイプを選択し、maven-archetype-quickstartを選択する

pomにjunitやエンコーディングなどの設定が最初から記述されている

**MavenプロジェクトのJREシステムライブラリを1.8にする**

・propertiesタグに追加

**<properties>**

**<maven.compiler.source>1.8</maven.compiler.source>**

**<maven.compiler.target>1.8</maven.compiler.target>**

**</properties>**

・maven-compiler-pluginを追加

**<build>**

**<plugins>**

**<plugin>**

**<groupId>org.apache.maven.plugins</groupId>**

**<artifactId>maven-compiler-plugin</artifactId>**

**<version>3.8.1</version>**

**<configuration>**

**<source>1.8</source>**

**<target>1.8</target>**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

**DI**

**DIコンテナへのBeanの登録**

|  |  |
| --- | --- |
| **XMLベース** | **<bean>タグ** |
| **Javaベース** | **@Configurationと@Bean** |
| **アノテーションベース** | **@Componentとcomponent-scan** |

**XMLベース**

**<?xml version="1.0" encoding="UTF-8"?>**

**<beans xmlns="http://www.springframework.org/schema/beans"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">**

**<bean id="userRepository" class="com.example.demo.UserRepositoryImpl" />**

**<bean id="passwordEncoder " class="com.example.demo.BCriptPasswordEncoder " />**

**<bean id="userService" class="com.example.demo.UserServiceImpl">**

**<constructor-arg ref=”userRepository”>**

**<constructor-arg ref=”passwordEncoder”>**

**</bean>**

**</beans>**

・idがBean名

・classがインスタンス(完全修飾クラス名)

**Javaベース**

**@Configuration**

**public class AppConfig {**

**@Bean**

**UserRepository userRepository () {**

**return new UserRepository ();**

**}**

**@Bean**

**PasswordEncoder passwordEncoder () {**

**return new PasswordEncoder ();**

**}**

**@Bean**

**UserService userService () {**

**return new UserService ();**

**}**

**}**

・メソッド名がBean名となる

**アノテーションベース**

**@Component**

**public class UserRepositoryImpl implements UserRepository {**

**// ・・・**

**}**

JavaConfig

**@Configuration**

**@ComponentScan(“com.example.demo”)**

**public class AppConfig {**

**}**

XML

**<?xml version="1.0" encoding="UTF-8"?>**

**<beans xmlns="http://www.springframework.org/schema/beans"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xmlns:context="http://www.springframework.org/schema/context"**

**xsi:schemaLocation="**

**http://www.springframework.org/schema/beans**

**http://www.springframework.org/schema/beans/spring-beans.xsd**

**http://www.springframework.org/schema/context**

**http://www.springframework.org/schema/context/spring-context.xsd">**

**<context:component-scan base-package="com.example.demo" />**

**</beans>**

**ApplicationContext**

|  |  |
| --- | --- |
| ClassPathXmlApplicationContext | XMLベース |
| AnnotationConfigApplicationContext | Javaベース |
| アノテーションベース |

**ApplicationContext context = new ClassPathXmlApplicationContext(“applicationContext.xml”);**

**ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.java);**

・Bean DIコンテナに登録するコンポーネント

・Bean定義 Configuration

・ルックアップ DIコンテナからBeanを取得すること

getBean(class)

getBean("id")

getBean("id", class)

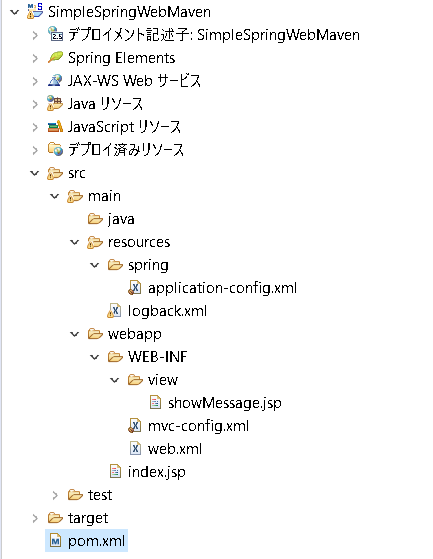
**インジェクション**

・セッターインジェクション

・コンストラクタインジェクション

・フィールドインジェクション

**Spring Web プロジェクト**



**デフォルト依存関係**

<artifactId>

spring-context

spring-tx

slf4j-api

logback-classic

hibernate-entitymanager

spring-test

junit

**pom修正点**

・Javaのバージョン

・Springのバージョン

・servletのバージョン

・servlet-apiのアーティファクトID

**web.xml修正点**

・バージョン変更

・application-configの場所

・ContextLoaderListener

・エンコーディングフィルター

**リソース**

**StaticApplicationContext**

とりあえずApplicationContextを用意したい場合

**Resource**

Resource res = app.getResource(“classpath:sample.txt”);

**リソース文字列**

|  |  |
| --- | --- |
| classpath: | src/main/resources フォルダ（相対パス） |
| file: | ファイルシステム（絶対パス） |
| http: | ネットワーク上のリソース |

・src/main/javaとsrc/main/resourcesはビルドすると**target/classes**にまとめられる

**ClassPathXmlApplicationContext**

src/main/resourcesから読み込む場合

**FileSystemXmlApplicationContext**

ファイルシステムから読み込む場合

**プロパティ**

**プロパティファイルの登録**

|  |  |
| --- | --- |
| JavaConfig | @PropertySourceを使う |
| XML | < context :property-placeholder>を使う |

**Injection**

@Valueを使う

**Java Config**

**@Configuration**

**@PropertySource(“classpath:application.properties”)**

**public class AppConfig {**

**}**

・@PropertySourceは複数記述できる

**XML**

**<?xml version="1.0" encoding="UTF-8"?>**

**<beans xmlns="http://www.springframework.org/schema/beans"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xmlns:context="http://www.springframework.org/schema/context"**

**xsi:schemaLocation="**

**http://www.springframework.org/schema/beans**

**http://www.springframework.org/schema/beans/spring-beans.xsd**

**http://www.springframework.org/schema/context**

**http://www.springframework.org/schema/context/spring-context.xsd">**

**<context:property-placeholder location=”classpath:application.properties”/>**

**</beans>**

・location属性でプロパティファイルをカンマ区切りで複数指定できる。

**@Value**

**@Value(“${jdbc.url}”)**

**private String jdbcUrl;**

@Value("${key\_name:default\_value}")

**AOP**

ロギングとかに使える

|  |  |
| --- | --- |
| アドバイス | 処理 |
| ジョインポイント | タイミング |
| ポイントカット | 条件 |

**dependency**

・spring-aop

・aspectjrt

・aspectjweaber

**enable**

|  |  |
| --- | --- |
| JavaConfig | @EnableAspectJAutoProxyを使う |
| XML | <aop:aspectj-autoproxy>を使う |

**Aspectクラス**

・@Aspect

・@Component

・@Before, @After, @Around

**@Aspect**

**@Component**

**public class LoggingAspect {**

**@Before(“execution(\* \*..\*ServiceImpl.\*(..))”)**

**public void startLog(JoinPoint jp) {**

**System.out.println(“メソッド開始:” + jp.getSignature());**

**}**

**}**

・アスペクトメソッドの引数はJoinPoint

・@Aroundの場合はProceedingJoinPoint

・ポイントカット式は色々な表現がある

**SpringMVC**

**コントローラー**

**@Controller**

・Webページ用

・JSPやテンプレートエンジンのViewに遷移

・戻り値はView名を指定

String

@RequestMapping("/index")

public String index() {

return "index";

}

ModelAndView

@RequestMapping("/index")

public ModelAndView index() {

ModelAndView modelAndView = new ModelAndView();

modelAndView.addObject(new User("紀伊", "太郎"));

modelAndView.setViewName("index");

return modelAndView;

}

フォワード

@RequestMapping("/forward")

public String forward1() {

return "forward:/index";

}

リダイレクト

@RequestMapping("/redirect")

public String redirect() {

return "redirect:/index";

}

@ResponseBody

@RequestMapping(value="/orders", method=RequestMethod.GET)

@ResponseBody

public List<Account> accountSummary() {

return accountManager.getAllAccounts();

}

・戻り値がレスポンスのボディを構成する

・Content-Typeは以下次第

①produces属性

　②Acceptヘッダー

・Javaオブジェクトの場合にはJSONかXMLに勝手にシリアライズされる

<https://stackoverflow.com/questions/28646332/how-does-the-spring-responsebody-annotation-work-in-this-restful-application-ex>

ResponseEntity

@RequestMapping("/responseEntity")

public ResponseEntity<String> responseEntity() {

HttpHeaders headers = new HttpHeaders();

headers.add("header1", "heaer1-value");

HttpStatus status = HttpStatus.NOT\_FOUND;

return new ResponseEntity<>("text content", headers, status);

}

HTTPステータスやコンテンツタイプ以外のレスポンスヘッダも指定したい時は、戻り値をResponseEntityにする。

**@RestController**

・WebAPI用

・@Contorollerと@ResponseBodyの合成

・戻り値はレスポンスのコンテンツになる

@RestController

@RequestMapping("/book")

public class BookRestController {

@GetMapping("/{id}", produces = "application/json")

public Book getBook(@PathVariable int id) {

return findBookById(id);

}

参考リンク

<https://qiita.com/tag1216/items/3680b92cf96eb5a170f0>

**@RequestMapping**

・クラス、メソッドに着ける

・ショートカット

@GetMapping

@PostMapping

@PutMapping

@DeleteMapping

・属性

|  |  |
| --- | --- |
| value | URIを指定 |
| method | HTTPメソッドを指定 |
| consumes | リクエストのContent-Typeを指定 |
| produces | レスポンスのContent-Typeを指定 |

※consumesとproducesにはMediaType定数を使うのが良い

**メソッド引数につけるアノテーション**

@RequestParam

@PathVariable

@CookieValue

@RequestHeader

@RequestBody

@SessionAttribute

**@RequestParam**

@GetMapping("/hello")

public String hello(@RequestParam("name") String name) {

URLに含まれるクエリパラメータや、メッセージボディーに含まれるポストパラメータを受け取ることができる。

例）/hello?name=world

**@PathVariable**

@GetMapping("/hello/{name}")

public String hello(@PathVariable("name") String name) {

URLに含まれる動的なパラメータを受け取ることができる。

**@CookieValue**

@GetMapping("/hello")

public String hello(@CookieValue("name") String name) {

**@RequestHeader**

@GetMapping("/hello")

public String hello(@RequestHeader("User-Agent") String userAgent) {

**@RequestBody**

@PostMapping("/hello")

public String hello(@RequestBody String body) {

**@SessionAttribute**

@GetMapping("/hello")

public String hello(@SessionAttribute("age") Integer age) {

**参考リンク**

<https://qiita.com/MizoguchiKenji/items/2a041f3a3eb13274e55c>