**Spring**

**STSのプロジェクト**

|  |  |
| --- | --- |
| **Spring starter project** | Spring boot を使ったプロジェクト |
| **Spring legacy project** | Spring boot を使わないプロジェクト |

**Mavenプロジェクト**

**アーキタイプ**は**maven-archetype-quickstart**を選択する

pomに**junit**や**エンコーディング**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>**に**<maven.compiler.source>**と**<maven.compiler.target>**を追加する |
| **<maven-compiler-plugin>**を追加し**<configuration>**に**<source>**と**<target>**を追加する。 |

**DIコンテナ**

|  |
| --- |
| SpringのDIコンテナは**ApplicationContext** |

**ApplicationContext**の実装クラス

|  |  |
| --- | --- |
| **AnnotationConfigApplicationContext** | Java Configでインスタンス化 |
| **ClassPathXmlApplicationContext** | クラスパスにあるXml Configでインスタンス化 |
| **FileSystemXmlApplicationContext** | ファイルシステムにあるXml Configでインスタンス化 |

**Bean**

|  |
| --- |
| DIコンテナに登録するコンポーネント |

**ルックアップ**

|  |
| --- |
| DIコンテナからBeanを取得すること |
| **ApplicationContext**#**getBean()**メソッド |

**DIコンテナへのBeanの登録方法**

|  |
| --- |
| **XML Config** |
| **Java Config** |
| **コンポーネントスキャン** |

**Xml Config**

|  |
| --- |
| **<beans>**がルートタグ |
| **<bean>**でBeanを定義する |
| **id**でBean名を指定する |
| **class**で型を指定する。値は**完全修飾クラス名** |

**Java Config**

|  |
| --- |
| **@Configuration**がついたクラス |
| **@Bean**がついたメソッドでBeanを定義。**メソッド名**がBean名となる。 |
| **@Configuration**は**@Component**を継承しているため**Component-Scan**の対象 |

**コンポーネントスキャン**

|  |  |  |
| --- | --- | --- |
| Xml Config | **<context:component-scan>** | **base-package**でスキャン対象のパッケージを指定 |
| Java Config | **@ComponentScan** | **basePackages**でスキャン対象のパッケージを指定 |

スキャン対象の@

@Component

@Controller

@Service

@Repository

@Configuration

@RestController

**オートワイヤリング**

|  |  |
| --- | --- |
| **@Autowired** | DIコンテナからBeanをバインドする@ |
|  | **フィールド**に付ける |
|  | **required**属性に**false**を指定すると必須でなくなる |

|  |  |
| --- | --- |
| **NoSuchBeanDefinitionException** | オートワイヤリングしようとしたがDIコンテナに合致する型のBeanがなかった場合に発生するException |

|  |  |
| --- | --- |
| **NoUniqueBeanDefinitionException** | オートワイヤリングしようとしたがDIコンテナに合致する型のBeanが複数あった場合に発生するException |

|  |  |
| --- | --- |
| **@Qualifier** | **@Autwired**のついた**フィールド**につける |
|  | Bean名を指定できる |

**Configurationのインポート**

|  |  |
| --- | --- |
| Xml Config | <import> |
| Java Config | @Import |

|  |  |
| --- | --- |
| **<import>** | Xml Configをインポートする |
|  | **resource**属性でインポートするXmlを指定する |

|  |  |
| --- | --- |
| **@Import** | Java Configをインポートする |
|  | **@Configuration**の付いたクラスに付ける |
|  | 複数指定する場合は配列で指定 |

**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 |
| file: | ファイルシステム |
| 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

**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 forward() {

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>

|  |  |
| --- | --- |
| **@PropertySource** | プロパティファイルを登録する@ |
| @Configurationのついたクラスに付与 |
| 複数回使用可能 |

|  |  |
| --- | --- |
| **<context:property-placeholder>** | プロパティファイルを登録するxmlタグ |
| location属性でファイルを指定 |
| カンマ区切りで複数指定可能 |

|  |  |
| --- | --- |
| **@Value** | フィールドにプロパティを注入する@ |

|  |  |
| --- | --- |
| **Environment** | プロパティにアクセスできるクラス |
| getPropertyメソッド |

|  |  |
| --- | --- |
| **application.properties** | Spring Bootにおいてsrc/main/resource配下に置くと自動で登録されるプロパティファイル |
| テストのときはsrc/test/resource配下に置く |

|  |  |
| --- | --- |
| **--spring.config.location** | Spring Bootにおいてコマンドラインでプロパティファイルを指定 |

|  |  |
| --- | --- |
| **@TestPropertySource** | テストクラスでプロパティファイルを登録する場合に使うアノテーション |
| src/test/resource配下のapplication.propertiesに代わる |
| "Key"="Value"の形式でプロパティを直接指定することも可能 |

|  |  |
| --- | --- |
| **@ContextConfiguration** | テストクラスでコンテキストを生成する場合に使うアノテーション |
| classes属性でjava configを指定 |
| locations属性でxml を指定 |

|  |  |
| --- | --- |
| **@RunWith(SpringRunner.classs)** | junitで@Autwiredなどspringの@が使用可能になる@ |

|  |  |
| --- | --- |
| **@SpringBootTest** | SpringBootのテストで使うアノテーション |
| コンテキストを生成する |

|  |  |
| --- | --- |
| **@SpringBootApplication** | @Configuration |
| @EnableAutoConfiguration |
| @ComponentScan |
| を合わせたアノテーション |