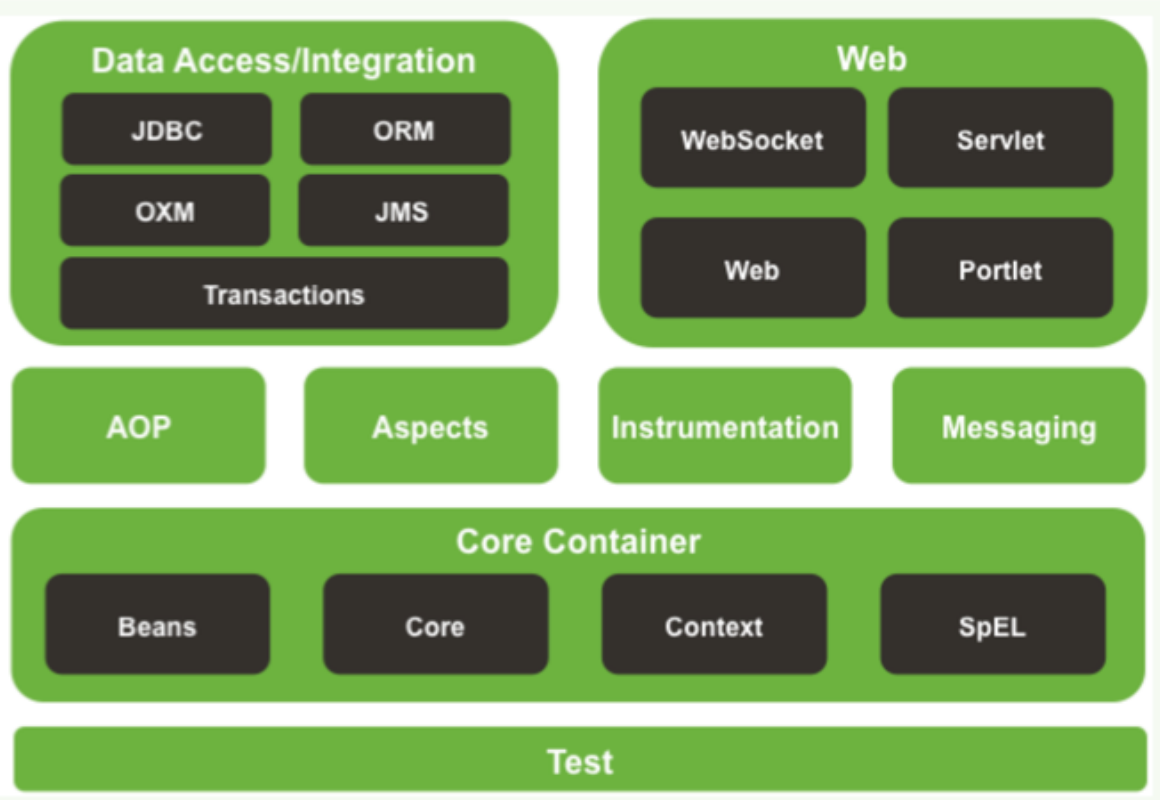
03 Spring

Spring kromě DI poskytuje i anotace pro typické funkcionality, které potřebujeme u typických aplikací



Stejně jako každý dům potřebuje vyřešit topení, elektřinu, vodu…

03 Konfigurace

Typy konfigurace ve Springu:

- Konfigurace založená na XML: Definuje beany v externích XML souborech (starší přístup).
- Konfigurace založená na Javě (@Configuration): Definuje beany a závislosti přímo v Java třídách (moderní přístup).
- Konfigurace založená na anotacích: Používá anotace jako @Component, @Service, @Repository a @Autowired pro automatickou registraci a propojení beanů.
- Spring Boot Auto-Configuration: Ve Spring Boot automatická konfigurace poskytuje přednastavené konfigurace komponent pro skeleton klasické aplikace, která má např. perzistenci, bezpečnost, rest API

03 Konfigurace – XML based

```xml
<!-- applicationContext.xml -->
<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 definition for a service class -->
    <bean id="myService" class="com.example.MyService">
        <!-- Injecting dependencies -->
        <property name="repository" ref="myRepository" />
    </bean>

    <!-- Bean definition for a repository class -->
    <bean id="myRepository" class="com.example.MyRepository" />
</beans>
```

03 Konfigurace – Java based

```java
@Configuration
public class AppConfig {

    @Bean
    public MyService myService() {
        return new MyService(myRepository());
    }

    @Bean
    public MyRepository myRepository() {
        return new MyRepository();
    }
}
```

Explicitně vytváříme beany a ty propojujeme

03 Konfigurace – annotation based

```java
@Component
public class MyService {
    // Business logic
}

@Repository
public class MyRepository {
    // Data access logic
}

@Service
public class MyOtherService {
    private final MyRepository myRepository;

    @Autowired
    public MyOtherService(MyRepository myRepository) {
        this.myRepository = myRepository;
    }
}
```

IoC container si scanuje třídy a propojení provádí sám

03 Konfigurace

Konfigurace založená na anotacích: Nejvhodnější pro jednoduché až středně velké aplikace, kde se preferuje snadnost použití a minimální konfigurace. Spoléhá na skenování komponent a anotace pro automatické propojení komponent, což dělá konfiguraci méně explicitní, ale snadnější na správu v menších projektech.

Konfigurace založená na Javě: Poskytuje plnou kontrolu a flexibilitu, což ji činí ideální pro větší nebo složitější aplikace. Každý bean je explicitně definován typově bezpečným způsobem, což dělá konfiguraci přehlednější a snáze refaktorovatelnou.

03 DI Annotations

Spring uses dependancy injection to configure and bind your appliation together.

T **@Configuration** - used to mark a class as a source of the bean definitions.

T **@ComponentScan** - makes Spring scan the packages configured with it for the @Configuration classes.

T **@Import** - loads additional configuration. This one works even when you specify the beans in an XML file.

T **@Component** - turns the class into a Spring bean at the auto-scan time.

T **@Service** - tells Spring that it's safe to manage @Components with more freedom than regular components.

C F M **@Autowired** - wires the application parts together, on the fields, constructors, or methods in a component.

M **@Bean** - specifies a returned bean to be managed by Spring context. The returned bean has the same name as the factory method.

M **@Lookup** - tells Spring to return an instance of the method's return type when we invoke it.

T M **@Primary** - gives higher preference to a bean when there are multiple beans of the same type.

C F M **@Required** - shows that the setter method must be configured to be dependency-injected with a value at configuration time.

C F M **@Value** - used to assign values into fields in Spring-managed beans. It's compatible with the constructor, setter, and field injection.

T M **@DependsOn** - makes Spring initialize other beans before the annotated one.

T M **@Lazy** - makes beans to initialize lazily. @Lazy annotation may be used on any class directly or indirectly annotated with @Component or on methods annotated with @Bean.

T M **@Scope** - used to define the scope of a @Component class or a @Bean definition and can be either singleton, prototype, request, session, globalSession, or custom scope.

T **@Profile** - adds beans to the application only when that profile is active.

Legend: T - Class F - Field Annotation C - Constructor Annotation M - Method P - Parameter

03 JPA Annotations

Annotation	Purpose	Parameters
@Entity	Marks a class as a JPA entity	`name`, `catalog`, `schema`, `readOnly`, `dynamicInsert`, `dynamicUpdate`
@Id	Marks the primary key field of an entity	`name`, `columnDefinition`, `allocationSize`
@GeneratedValue	Specifies how the primary key value should be generated	`strategy`, `generator`
@Table	Specifies the name of the database table that the entity is mapped to	`name`, `catalog`, `schema`, `uniqueConstraints`
@Column	Specifies the mapping of a field to a database column	`name`, `nullable`, `unique`, `length`, `precision`, `scale`, `columnDefinition`
@OneToMany	Defines a one-to-many relationship between two entities	`targetEntity`, `cascade`, `fetch` ,`mappedBy`
@ManyToOne	Defines a many-to-one relationship between two entities	`targetEntity`, `cascade`, `fetch`, `optional`, `mappedBy` , `joinColumns`
@ManyToMany	Defines a many-to-many relationship between two entities	`targetEntity`, `cascade`, `fetch`, `mappedBy`, `joinTable`, `joinColumns`, `inverseJoinColumns`
@JoinTable	Specifies the join table for a many-to-many relationship	`name`, `catalog`, `schema`, `joinColumns`, `inverseJoinColumns` , `uniqueConstraints`
@JoinColumn	Specifies the join column for a many-to-one or one-to-one relationship	`name`, `referencedColumnName`, `nullable`, `unique`, `insertable` , `updatable`, `columnDefinition`, `table`

03 Rest annotations

Annotation	*Description*
`@RestController`	Marks a class as a RESTful web service controller.
`@GetMapping`	Maps HTTP GET requests to a specific method in the controller.
`@PostMapping`	Maps HTTP POST requests to a specific method in the controller.
`@PutMapping`	Maps HTTP PUT requests to a specific method in the controller.
`@DeleteMapping`	Maps HTTP DELETE requests to a specific method in the controller.
`@PathVariable`	Binds a method parameter to a path variable in the request URL.
`@RequestParam`	Binds a method parameter to a query parameter in the request URL.
`@RequestBody`	Binds the request body to a method parameter in the controller.
`@ResponseStatus`	Specifies the HTTP response status code for a method in the controller.
`@ExceptionHandler`	Defines a method to handle a specific type of exception thrown by the controller.

03 Security annotations

Annotation	Description
`@EnableWebSecurity`	Enables Spring Security for a web application.
`@Configuration`	Indicates that a class is a configuration class that provides Spring bean definitions.
`@Secured`	Restricts access to a method to users who have been granted a specific set of authorities.
`@PreAuthorize`	Restricts access to a method based on a SpEL expression that evaluates to `true`.
`@PostAuthorize`	Restricts access to a method based on a SpEL expression that evaluates to `true` after the method has been called.
`@RolesAllowed`	Restricts access to a method to users who have been granted a specific set of roles.
`@EnableGlobalMethodSecurity`	Enables global method security for a Spring Security-enabled application.
`@AuthenticationPrincipal`	Provides access to the currently authenticated principal (user) in a controller method.
`@Order`	Specifies the order in which security filters should be applied to HTTP requests.