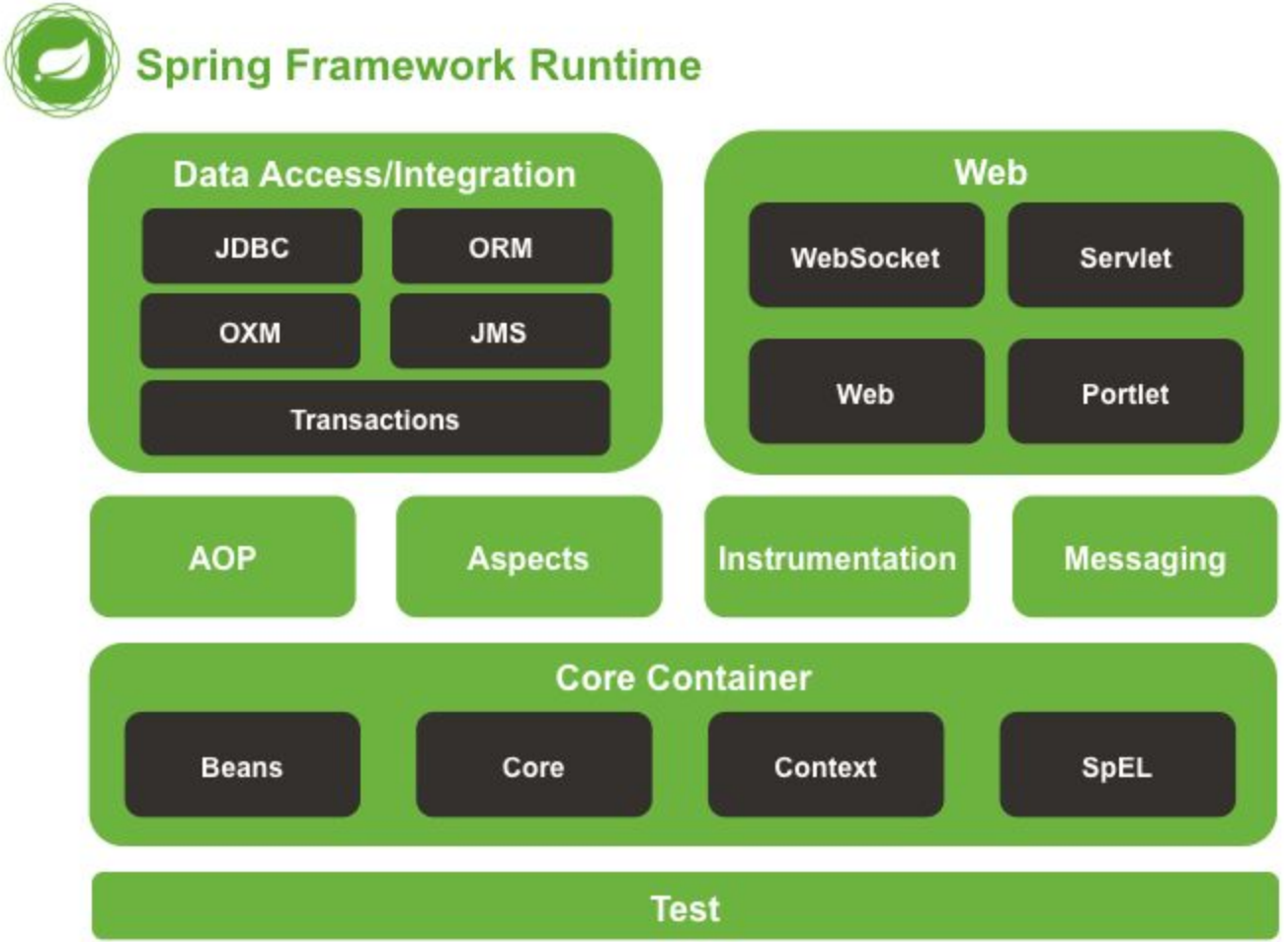
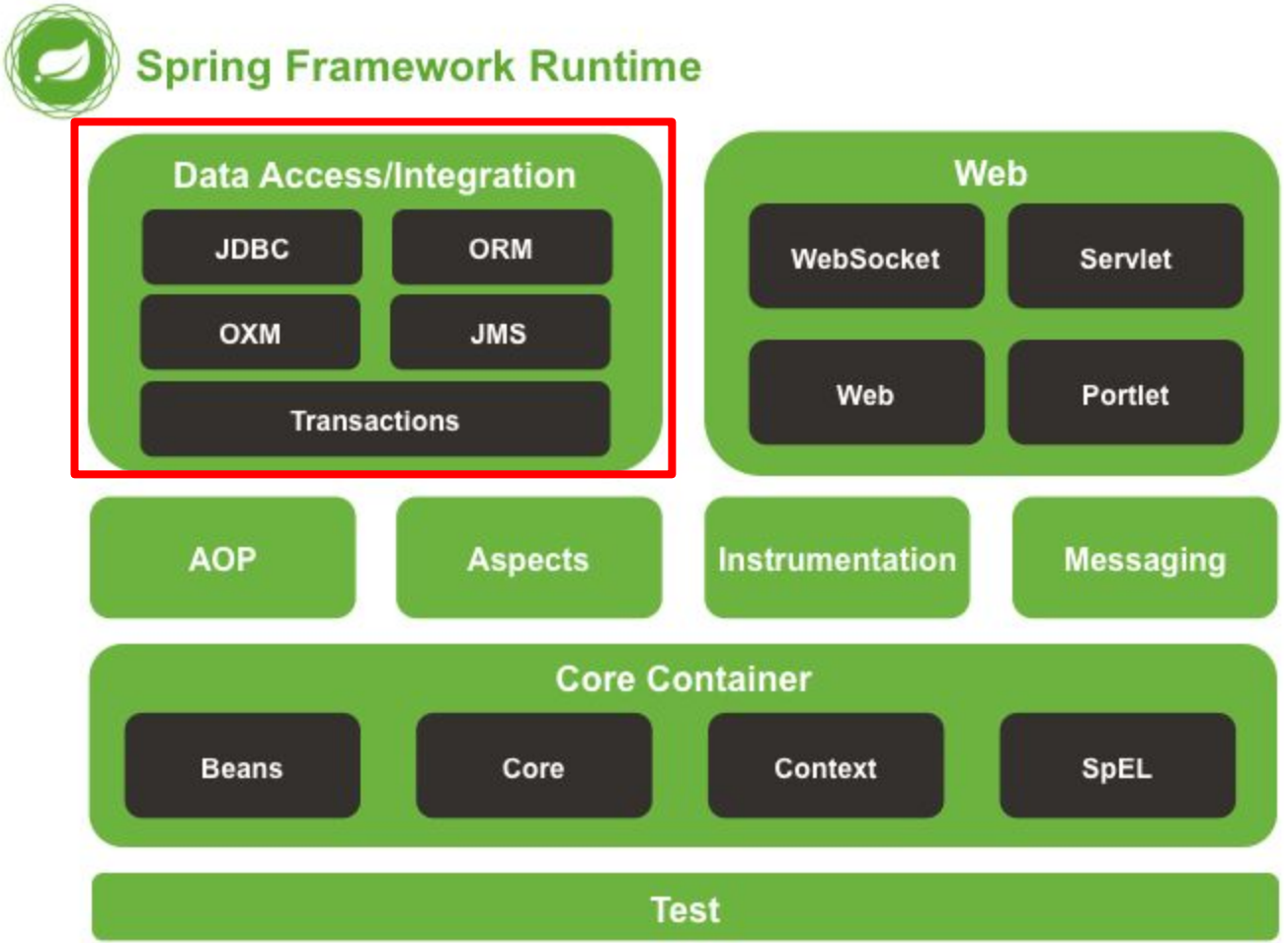
Intro Spring Data Access

For Spring 4.3.x


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

Spring ORM

- Module of Spring which allows you to integrate data access code with Spring applications
- Allows us to integrate Hibernate with Spring features such as:
  - Dependency injection
  - AOP
  - Testing
- Allows for Spring-managed transactions
- Data access configuration (DB credentials, Hibernate configuration) migrated to beans.xml file

ORM Beans

- DataSource
  - Contains database-specific configuration
    - Driver
    - Username/Passord
    - URL
  - Is injected into a SessionFactory bean
- SessionFactory
  - Contains Hibernate-specific configuration
    - Hbm2ddl.auto
    - Show.sql
    - Etc…
  - Is autowired into a TransacationManager bean
- Transaction Manager
  - Manages transactions for classes and methods marked @Transactional

@Transactional

- Indicates that a method will be handled in the context of a Spring transaction; it essentially tells Spring that a class (every method in it) or a specific method is going to be handled as a Transaction
- Sessions, however, are still handled by Hibernate; there is simply a contextual session which exists in the context of a Spring-managed transaction
- We don’t have to manually commit or rollback transaction anymore; Spring handles this for us with AOP

Contextual Sessions

- Hibernate has a feature called contextual sessions, wherein Hibernate itself manages one current Session per transaction.

- This is roughly equivalent to Spring’s synchronization of one Hibernate Session per transaction.

Exception Translation

- Spring translates specific database exceptions to its own generic exceptions, with DataAccessException as the root.

- This saves us trouble and allows us to handle diverse database access exceptions in the same way.

Validation with JSR 303

- JSR 303 is a java spec for validation that integrates with Spring
- Validation makes sure our beans are valid -- that their properties are set in accordance with our wishes.
- Under the hood, Spring uses Hibernate’s validator
- We can do most validation using annotations on model classes
- We can also validate by implementing the Validator interface

Validation Annotations

- @NotNull
- @NotEmpty
- @Min()
- @Max()
- @Size()
- @DateTimeFormat()
  - Specify a certain format as an argument, i.e. MM/dd/yyyy
- @Pattern()
  - Use a regex pattern as an argument

```java
public class PersonValidator implements Validator {

    /**
     * This Validator validates *just* Person instances
     */
    public boolean supports(Class clazz) {
        return Person.class.equals(clazz);
    }

    public void validate(Object obj, Errors e) {
        ValidationUtils.rejectIfEmpty(e, "name", "name.empty");
        Person p = (Person) obj;
        if (p.getAge() < 0) {
            e.rejectValue("age", "negativevalue");
        } else if (p.getAge() > 110) {
            e.rejectValue("age", "too.darn.old");
        }
    }
}
```