# معرفی

## چرا Framework؟

توسعه نرم افزار یک فرایند پیچیده و حساس است که در صورتی که به درستی کنترل و مدیریت نشود زمان بر و طاقت فرسا خواهد شد. باگ های نرم افزار، تغییر کامپوننت های نرم افزار، اضافه کردن امکانات جدید و ... در صورتی که به درستی مدیریت نشوند می توانند تبدیل به یک کابوس بزرگ برای تیم تولید شوند. Framework نرم افزاری با تعیین یک چارچوب یکپارچه برای نرم افزار و پیاده سازی بخش های کلی محصول سعی در آسان کردن پیاده سازی و اعمال مدیریت رو کد های تولید شده دارد.

## معماری مورد استفاده

معماری استفاده شده در Framework، معماری مبتنی بر سرویس(SOA) است. این معماری که در سال های اخیر با استقبال گسترده ای مواجه شده است، نرم افزار را به دو بخش سرویس دهنده و سرویس گیرنده تقسیم می کند. برنامه های وب ای که مطابق با این معماری پیاده سازی شده اند عمدتا از وب سرویس ها به عنوان بستر ارائه سرویس استفاده می کنند. به عنوان مثال یک برنامه وب که از طریق موبایل هم در دسترس است، دو نوع سرویس گیرنده دارد که یکی از آنها یک برنامه جاوااسکریپتی است و دیگری یک برنامه اندروید یا آی او اس. هر دوی این برنامه ها از طریق سرویس Rest با سرویس دهنده ارتباط برقرار می کنند.

لایه بندی سرور برای پیاده سازی این سرویس ها متشکل از 4 لایه است. این لایه ها (به ترتیب از منبع داده تا سرویس گیرنده) عبارتند از:

### لایه منبع داده

وظیفه بازخوانی داده ها از منبع داده را دارد. منبع داده می تواند یک پایگاه داده، فایل متنی، فایل XML و یا هر چیز دیگری باشد. لایه داده دسترسی به این منبع داده را فراهم می کند. در یک پیاده سازی موفق، چنانچه منبع داده مربوط به یک موجودیت تغییر کند (مثلا از پایگاه داده به فایل) تنها این کلاس دستخوش تغییر خواهد شد.

### لایه منطق برنامه

منطق برنامه در کلاس های این لایه اتفاق می افتد. در واقع عمل اصلی مربوط به Use caseهای برنامه، در کلاس های این لایه پیاده سازی می شوند. همچنین مدیریت تراکنش کاربر، کنترل دسترسی و... از جمله منطق هایی هستند که در این لایه پیاده سازی می شوند.

### لایه سرویس

لایه منطق قابل استفاده توسط سرویس گیرنده ها نیست چرا که از کلاس موجودیت استفاده می کند و به همین دلیل و یک سری دلایل دیگر اصطلاحا به سیستم گره خورده است. کلاس موجودیت به دلیل امکاناتی که ORMها در اختیار قرار می دهند، یک کلاس دستکاری شده (Proxy) است (فلسفه وجود این لایه مشابه فلسفه استفاده از DTO است). با توجه به آنچه گفته شد(!) وظیفه لایه سرویس، ارائه سرویس قابل استفاده توسط سیستم های خارجی است.

### لایه وب سرویس

لایه وب سرویس آخرین لایه است که مبتنی بر تکنولوژی مورد استفاده سرویس ارائه می دهد.

## معرفی ماژول های Framework

Framework متشکل از یک سری ماژول است که در ادامه بررسی خواهیم کرد. این ماژول ها به دسته های زیر تقسیم می شوند:

### ماژول های پایه

این ماژول ها امکانات پایه ای سیستم از قبیل I18N، JSON، JDBC، Dozer Mapping، Caching و ... را فراهم می کنند.

### ماژول های وب سرویس

این ماژول ها برای ارائه وب سرویس های Rest و JAX-WS استفاده می شوند.

### ماژول های فایل

این ماژول ها برای مدیریت فایل های برنامه استفاده می شوند.

### ماژول های اجرای موازی

این ماژول ها برای مدیریت عملیات سمت سرور و امکان موازی سازی این عملیات استفاده می شوند.

### ماژول های گزارش گیری

این ماژول ها برای گزارش گیری استفاده می شوند.

### ماژول های امنیت

این ماژول ها برای کنترل امنیت پروژه استفاده می شوند.

### ماژول های گردش کاری

این ماژول ها برای مدیریت گردش کاری در سیستم استفاده می شوند.

# امکانات پایه

## ماژول هسته

ماژول هسته عمدتا یک سری کلاس پایه پروژه است که در سایر ماژول ها و پروژه ها مورد استفاده قرار می گیرند.

### کلاس پیکره بندی

این کلاس یک فایل properties به نام **vaseline.properties** تعریف می کند که دربرگیرنده مشخصات پیکره بندی مورد استفاده در پروژه خواهد بود.

همچنین این کلاس امکان گرفتن یک Bean را خارج از محدوده Spring فراهم می کند:

**public static** <T> T getBean(Class<T> beanClass) {  
 **return** *applicationContext*.getBean(beanClass);  
}

این متد که static است، می تواند بدون Injection مورد استفاده قرار گیرد. به عنوان مثال ماژول گزارش گیری یک مبدل برای تبدیل FileReportSourceImplClient به FileReportSourceImplServer دارد که برای ایجاد فایل سمت سرور نیاز به IFileApi دارد. از آنجا که مبدل های Dozer نمی توانند Spring Bean باشند در کلاس مبدل مجبور شده ایم که با استفاده از متد static این bean را بازیابی کنیم:

destination.setFileApi(VaselineCoreConfig.*getBean*(IFileApi.**class**));

دقت داشته باشید که استفاده از این متد توصیه نمی شود و بایستی حتی الامکان از استفاده از آن پرهیز کرد، چرا که چارچوب انتخاب شده برای Framework، Spring می باشد و باید تمامی کلاس ها به صورت Spring Bean مورد استفاده قرار گیرند.

### پکیج ent

این پکیج شامل interfaceهای پایه برای موجودیت های سیستم می باشد. این کلاس ها عبارتند از:

#### کلاس IBaseEntity

این کلاس دو متد را برای موجودیت ها در نظر گرفته است که getter و setter آی دی موجودیت هستند. نوع آی دی به عنوان generic به کلاس پاس داده می شوند.

#### کلاس IBaseHasModifyDateEntity

این کلاس که کلاس IBaseEntity را extend می کند دو متد دیگر اضافه می کند که getter و setter تاریخ آخرین تغییر موجودیت هستند.

#### کلاس IBaseHasCreateDateEntity

این کلاس که کلاس IBaseEntity را extend می کند دو متد دیگر اضافه می کند که getter و setter تاریخ ایجاد موجودیت هستند.

### پکیج exc

این پکیج کلاس های مربوط به Exception Handling را در خود جای داده است. کلاس های موجود در این پکیج عبارتند از:

#### کلاس BaseVaselineServerException

برای خطاهای سمت سرور یک کلاس پدر در نظر گرفته شده است. این خطا که Checked Exception می باشد، در لایه منطق می تواند به وجود بیاید. بنابراین تمام متدهای لایه منطق بایستی این خطا را در تعریف خود ذکر کنند.

#### کلاس ICoreExceptionHandler

همانطور که گفته شد لایه منطق می تواند خطای BaseVaselineServerException را به وجود بیاورند، علاوه بر این خطاها، خطاهای Unchecked هم ممکن است در این لایه اتفاق بیفتد. این خطاها در لایه سرویس catch می شوند و به یک bean از نوع ICoreExceptionHandler پاس داده می شوند، این bean، خطای مربوطه را به یک BaseVaselineClientException تبدیل می کند که خطای قابل فهم توسط استفاده کننده های خارجی است (به عنوان مثال کد خطا و پیغام مناسب را در خود دارد). بنا بر آنچه گفته شد این کلاس تنها یک متد دارد:

BaseVaselineClientException convertException(Exception exception);

#### کلاس CoreExceptionHandlerImpl

این کلاس، پیاده سازی ICoreExceptionHandler است. این کلاس متد convertException را پیاده سازی می کند. برای پیاده سازی تبدیل خطا به خطای سمت کلاینت، این از دو Map استفاده می کند. یک Map برای نگهداری مبدل خطاها و یک Map برای نگهداری خطاهایی که در سمت سرور اتفاق افتاده است که در صورت تمایل، کاربر بتواند خطا را گزارش کند. مبدل خطا یک interface است که تبدیل یک خطای خاص را به عهده دارد. کلاس CoreExceptionHandlerImpl یک Map برای مبدل خطاها دارد که کلید این Map کلاس خطا و مقدار آن یک نمونه از IBaseExceptionConverter است. هنگامی که یک خطا به متد convertException می رسد، ابتدا کلاس خطا گرفته می شود. چنانچه IBaseExceptionConverter ای در Map به ازای کلاس خطا وجود داشته باشد خطا توسط آن تبدیل می شود، اگر نباشد کلاس پدر خطا گرفته می شود و دوباره Map چک می شود. این فرایند آنقدر ادامه پیدا می کند تا به کلاس Exception برسیم. به ازای کلاس Exception، یک مبدل وجود دارد که خطای "خطای ناشناخته در سمت سرور" را باز می گرداند. پس از تبدیل خطا یک UUID تولید می شود و خطای سرور با این UUID در Map مربوط به گزارش خطا ذخیره می شود.

متد getException: این متد شناسه خطای مربوط به خطای سمت سرور را می گیرد و خطای مربوطه را باز می گرداند. این شناسه همان UUID است که هنگام تبدیل خطا تولید شده و خطای سمت سرور با این شناسه در Map مربوطه ذخیره شده است.

متد registerHandler: این متد کلاس خطا و نمونه مبدل خطای مربوط به یک نوع خطا را می گیرد و مبدل خطا را برای کلاس خطای داده شده نگه می دارد.

#### کلاس IBaseExceptionConverter

این کلاس یک interface است که وظیفه تبدیل یک خطای خاص را به عهده دارد. این interface دو generic ورودی دارد، اولی نوع خطای سمت سرور و دومی نوع خطای سمت کلاینت است. کلاس پیاده سازی این interface بایستی خطای سرور را به خطای کلاینت تبدیل کند. این کلاس یک متد دارد که خطای سمت سرور را به عنوان ورودی می گیرد و خطای سمت کلاینت را باز می گرداند.

#### کلاس BaseExceptionConverterImpl

کلاس پیاده سازی پایه برای IBaseExceptionConverter. این کلاس

## ماژول I18N

## ماژول JSON

## ماژول Mapper

## ماژول Caching

## ماژول JDBC

## ماژول Hibernate

## ماژول Validation

## ماژول معماری

## ماژول پیاده سازی معماری

# امکانات وب سرویس

# امکانات مربوط به فایل

# امکانات اجرای موازی

# امکانات گزارش گیری

# امکانات امنیتی

# امکانات گردش کاری