عنوان مقاله: تفاوت DDD، میکروسرویس (Microservice)، الگوهای طراحی (Design pattern) و معماری تمیز (Clean Architecture)

نویسنده مقاله: تیم فنی نیک آموز

تاریخ انتشار: ۷ فروردین ۱۴۰۳

منبع: https://nikamooz.com/the-difference-between-ddd-microservice-design-pattern-clean-architecture/

دنیای توسعه نرم افزار پر از مفاهیم و رویکردهای مرتبط و پیچیده است. هنگام ایجاد اپلیکیشن‌های پیچیده، دسترسی به ابزار و تکنیک‌های مناسب، اهمیت فراوانی دارا است. در این مطلب، تفاوت DDD، میکروسرویس (Microservice)، الگوهای طراحی (Design pattern) و معماری تمیز (Clean Architecture) را به‌همراه کاربردها و مزایای هر یک شرح می‌دهیم. اگر به حوزه توسعه نرم‌افزار علاقه‌مند هستید و می‌خواهید شناخت عمیقی‌تری از آن به‌دست آورید، این مقاله مناسب شما است.

Clean Architecture چیست؟

معماری تمیز (Clean Architecture) مجموعه‌ای از اصول و قوانینی است که برای طراحی نرم‌افزار استفاده می‌شود و در آن، تأکید روی استقلال، قابل تست‌بودن و نگهداری است. نمی‌توان معماری تمیز را یک فریمورک یا تکنولوژی خاص تلقی کرد؛ بلکه این معماری، شیوه تفکر درمورد نحوه ساختاردهی کدها به‌حساب می‌آید.

در معماری تمیز، مفاهیمی همچون وجود لایه‌های مستقل، قانون استقلال، جداسازی منطق کسب و کار از وابستگی‌های خارجی مطرح می‌شود. از بارزترین نقاط مثبت و مزایای استفاده از Clean Architecture ، می‌توان به افزایش امکان نگهداری و درک بهتر کدها، بهبود امکان تست و آزمایش منطق هسته کسب‌وکار، سازگاری و انعطاف‌پذیری اپلیکیشن به تکنولوژی‌های جدید اشاره کرد. در **مقاله معماری تمیز (Clean Architecture) چیست ؟ ۵ مرحله راه اندازی آن** به بررسی بیشتر این معماری پرداخته‌ایم، درصورت علاقه‌مندی می‌توانید به آن رجوع کرده تا با اهمیت و نحوه پیکربندی آن به‌صورت گام‌به‌گام آشنایی پیدا کنید.

کاربردهای Clean Architecture

شاخص‌ترین کاربردهای Clean Architecture عبارتند از:

- ایزوله‌سازی منطق هسته کسب‌وکار از سایر بخش‌ها و تسهیل روند ویرایش و درک کدها
- ساده‌سازی فرآیند تست، به‌ویژه Unit Testing
- ترویج اتصالات سست (Loose Coupling) میان اجزا و افزایش هرچه بیشتر انعطاف‌پذیری
- بهبود تعاملات و مشارکت اعضای تیم و شفاف‌سازی وظایف شغلی آن‌ها

مزایای استفاده از Clean Architecture

مزیت‌های معماری تمیز عبارتند از:

- اولویت‌دهی و تمرکز روی منطق هسته کسب‌وکار
- بهبود تعاملات تیمی و افزایش کارایی تیم
- استقلال منطق هسته کسب از سایر کامپوننت‌های سیستم
- افزایش قابل اکتفا بودن و قدرت کدها با آزمودن آن‌ها
- بهبود ماژولاریتی کد و درک بهتر سیستم

Design Pattern چیست؟

الگوهای طراحی (Design Patterns) راه حل‌هایی با قابلیت استفاده مجدد هستند که شما می‌توانید آن‌ها را به‌منظور رفع مشکلات رایج طراحی نرم‌افزار به کار ببرید. دیزاین پترن راه‌حلی همه‌منظوره برای تمامی شرایط نیست؛ بلکه به‌نوعی Template ها یا Blueprint هایی محسوب می‌شوند که با ارائه یک رویکرد ساختاریافته، به چالش‌های خاص توسعه رسیدگی خواهند کرد. برای آشنایی با انواع دیزاین پترن و درک بهتر مفهوم آن، مطالعه **مقاله دیزاین پترن چیست؟** می‌تواند برای شما مفید باشد.

کاربردهای Design Pattern

موارد استفاده الگوهای طراحی در ادامه لیست شده‌اند:

- امکان استفاده از آن به‌عنوان نقطه شروع در مواجه با مشکلات رایج طراحی
- ترویج قابلیت استفاده مجدد از کد با کمک راه‌حل‌های از قبل‌تعریف‌شده
- بهبود قابلیت کیفیت و نگهداری کد
- تسهیل مقیاس‌پذیری و همچنین، افزایش انعطاف‌پذیری کد

مزایای استفاده از Design Pattern

مزیت‌های استفاده از دیزاین پترن عبارتند از:

- تسریع فرآیند توسعه و تمرکز روی پیاده‌سازی توابع و فیچرهای اصلی
- تسهیل درک و نگهداری Codebase و افزایش ماژولاریتی کد
- امکان Scale کردن اپلیکیشن در طول زمان و تطبیق‌پذیری آن به تغییرات آتی
- بهبود نحوه تعامل تیم‌های توسعه و گفتگوی مؤثر درخصوص تصمیمات طراحی

Microservice چیست؟

میکروسرویس (Microservice) یک رویکرد توسعه نرم افزار است که یک برنامه را به‌عنوان مجموعه‌ای از سرویس‌های کوچک و مستقل ساختاردهی می‌کند. هر سرویس روی یک قابلیت تجاری واحد متمرکز است و ازطریق API های ازپیش‌تعریف‌شده، با سایر سرویس‌ها ارتباط برقرار می‌کند. مطلب جامع میکروسرویس می‌تواند به‌عنوان یک راهنمای کامل و قابل درک در شناخت میکروسرویس‌ها به شما کمک کند.

کاربردهای Microservice

موارد استفاده از میکروسرویس‌ها به شرح زیر است:

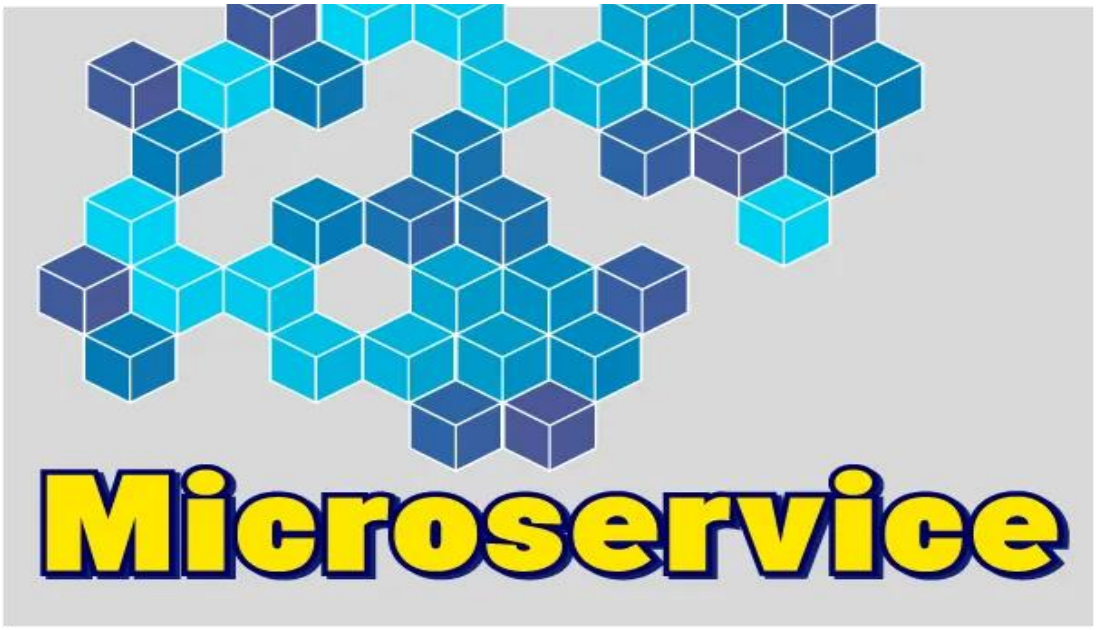
- **اپلیکیشن‌های بزرگ و پیچیده:** میکروسرویس‌ها برای ساخت و نگهداری اپلیکیشن‌های بزرگ و پیچیده‌ای که به‌سرعت تغییر می‌کنند، مناسب است. طراحی ماژولار میکروسرویس‌ها، باعث ساده‌سازی فرآیند توسعه، تست و استقرار می‌شود.
- **سیستم‌های با قابلیت مقیاس‌پذیری بالا:** هر میکروسرویس می‌تواند براساس نیازهای خاص خود، به‌صورت مستقل Scale شود. این موضوع، تخصیص کارآمد منابع را امکان‌پذیر کرده و اطمینان می‌دهد که برنامه می‌تواند لودهای متغیر را مدیریت کند.
- **تسریع چرخه توسعه:** توسعه و استقرار مستقل میکروسرویس‌ها منجر می‌شود تا چرخه توسعه سریع‌تر انجام شود. درحقیقت، تیم‌ها می‌توانند هم‌زمان روی سرویس‌های مختلف کار کنند که این موضوع، درنهایت به زمان کوتاه‌تری برای ورود فیچرهای جدید منجر خواهد شد.

مزایای استفاده از Microservice

مزایای میکروسرویس عبارتند از:

- تسریع چرخه توسعه و تطبیق‌پذیری آسان‌تر با نیازمندی‌های متغیر
- ایزوله‌سازی بهتر خطا و افزایش تاب‌آوری سیستم در برابر خطاهای احتمالی
- افزایش مقیاس‌پذیری و امکان بهبود اپلیکیشن تحت لودهای کاری مختلف
- انعطاف‌پذیری در انتخاب تکنولوژی

برای بررسی عمیق‌تر جزئیات مربوط به مزیت‌های معماری میکروسرویس، می‌توانید به مقاله مزایای میکروسرویس رجوع کنید.



DDD چیست؟

طراحی دامنه محور (Domain Driven Design | DDD) یک رویکرد توسعه نرم‌افزار است که بر ایجاد همکاری نزدیک بین توسعه‌دهندگان نرم‌افزار و متخصصان حوزه کسب‌وکار (دامنه) تمرکز دارد. هدف اصلی در DDD، پرکردن حد فاصل میان دنیای فنی و حوزه کسب‌وکار خاصِ اپلیکیشن مربوطه است. شایان ذکر است که پیش‌تر یک مطلب تحت عنوان پیاده سازی معماری Domain Driven Design و تست نویسی + راهنمای گام به گام ، به‌منظور آشنایی با مراحل Implementation طراحی دامنه‌محور منتشر کرده‌ایم.

کاربردهای DDD

به‌صورت کلی، موارد زیر مهم‌ترین کاربردهای Domain Driven Design محسوب می‌شوند:

- مدل‌سازی دقیق قوانین مالی در سیستم‌های مالی و ایجاد نرم‌افزاری باثبات و قابل اعتماد
- امکان استفاده از آن در اپلیکیشن‌های مدیریت پروژه
- مدل‌سازی جریان کالاها، سفارشات و موجودی انبار در مدیریت زنجیره تأمین
- مدل‌سازی داده‌های بیمار، داروها و سایر مفاهیم کلیدی در حوزه مراقبت‌های بهداشتی
- قابل استفاده در سیستم‌های رزرو

به‌طور کلی، DDD برای هر اپلیکیشنی مفید است که نیاز به درک عمیق دامنه و همکاری نزدیک بین بخش فنی و کسب‌وکار دارد.

مزایای استفاده از DDD

مزیت‌های طراحی دامنه‌محور عبارتند از:

- کاهش پیچیدگی و تجزیه دامنه به Context های کوچک‌تر
- تمرکز بر ایجاد مرزهای شفاف و بهبود درک و نگهداری
- بهبود کیفیت کد و امکان تست‌نویسی بهتر
- تمرکز روی ارزش‌های کسب‌وکار و انعکاس دادن دقیق نیازمندی‌ها در نرم‌افزار

تفاوت DDD – Microservice – Design Pattern – Clean Architecture

در ادامه، به تفصیل به تفاوت DDD، میکروسرویس، الگوهای طراحی و معماری تمیز از نقطه نظرهای مختلف می‌پردازیم.

تمرکز

- **DDD:** طراحی دامنه‌محور روی مدل‌سازی هسته اصلی حوزه کسب‌وکار و ایجاد یک زبان مشترک بین توسعه‌دهندگان و متخصصان دامنه تمرکز دارد.
- **میکروسرویس:** معماری میکروسرویس بر معماری برنامه به‌عنوان مجموعه‌ای از خدمات کوچک و مستقل که با یکدیگر ارتباط برقرار می‌کنند، متمرکز است.
- **الگوهای طراحی:** دیزاین پترن بر ارائه راه حل‌هایی با قابلیت استفاده مجدد برای مشکلات رایج طراحی نرم‌افزار تمرکز دارد.
- **معماری تمیز:** Clean Architecture بر ساختاردهی برنامه به‌گونه‌ای که دغدغه‌ها ازهم جدا شده باشند و وابستگی سست ترویج شود، متمرکز است.

سطح انتزاع

- **DDD:** رویکردی سطح بالا برای طراحی کلی نرم‌افزار است.
- **میکروسرویس:** یک سبک معماری برای ساخت اپلیکیشن‌ها تلقی می‌شود.
- **الگوهای طراحی:** راه حل‌های سطح پایین و خاصی برای مشکلات رایج طراحی نرم‌افزار هستند.
- **معماری تمیز:** رویکردی سطح متوسط است که اصول معماری سطح بالا را تعریف می‌کند.

ارتباط

- **DDD:** می‌توان طراحی دامنه‌محور را در داخل میکروسرویس و به‌منظور مدل‌سازی دامنه هر سرویس استفاده کرد.
- **دیزاین پترن:** می‌توان از الگوهای طراحی در DDD و میکروسرویس و برای حل مشکلات خاص در معماری انتخاب‌شده، بهره‌مند شد.
- **معماری تمیز:** این معماری، چارچوبی برای پیاده‌سازی DDD و میکروسرویس ارائه می‌دهد و به‌واسطه آن، جداسازی دغدغه‌ها و قابلیت تست‌پذیری ترویج خواهد شد.

مثالی از تفاوت معماری تمیز – الگوهای طراحی – DDD و میکروسرویس ها

برای درک بهتر تفاوت معماری تمیز، الگوهای طراحی، DDD و میکروسرویس ها، به مثال‌های زیر توجه کنید:

- DDD همچون نقشه ساختمانی است که اتاق‌ها، کارکردها و چیدمان کلی را تعریف می‌کند.
- میکروسرویس مانند ساختن خانه با ماژول‌های جداگانه برای آشپزخانه، حمام و سایر بخش‌ها است.
- الگوهای طراحی مانند راه حل‌های ازپیش‌ساخته برای اجزای خاص مانند پنجره‌ها یا دره‌های منزل هستند.
- معماری تمیز، اصل معماری زیربنایی است که تضمین می‌کند خانه ساختاریافته، قابل نگهداری و سازگار با محیط باشد.

بدیهی است که هر یک از آن‌ها به‌طور جداگانه حائز اهمیت باشند و در روند توسعه نرم‌افزار کاربردهای چشم‌گیری از خود به جا بگذارند.

جمع بندی : تفاوت DDD، میکروسرویس (Microservice)، الگوهای طراحی (Design pattern) و معماری تمیز (Clean Architecture)

در این مقاله، تفاوت DDD، میکروسرویس (Microservice)، الگوهای طراحی (Design pattern) و معماری تمیز (Clean Architecture) بررسی شدند و به مزیت‌ها و کاربردهای هر یک اشاره شد. با توجه به اهمیت این مفاهیم در دنیای توسعه نرم‌افزار، بهتر است شما مواردی همچون نقاط تمایز آن‌ها را به‌خوبی درک کنید.