# مقدمه

امروزه در معماري نرم‌افزار الگوهاي معماري متنوعي پديد آمده‌اند كه بهبود فرآيند معماري نرم‌افزار كمك مينمايند. يكي از الگوهاي معماري رايج در معماري نرم‌افزار الگوي معماري رويداد محور [[1]](#footnote-1) که به طور خلاصه معماری رویداد محور نامیده می‌شود . در ادامه اين تحقيق ابتدا به تعريف اين معماري پرداخته مي‌شود . سپس اجزاء مختلف معماري رويداد محور مشاهده مي‌گردد . در بخش بعدي به كاربرد اين الگوي معماري پرداخته مي‌شود . سپس مزايا و معايب معماري رويداد محور توضيح داده شده است . و در بخش آخر نكاتي كه بايد درحين استفاده از اين الگوي معماري بايد مورد توجه قرار گيرد پرداخته مي‌شود .

1. فزار ‌ا

# الگوی معماری رویداد محور

## تعریف معماری رویداد محور

معماری رویداد محور یک رویکرد طراحی مدرن است که حول داده‌هایی متمرکز است که رویدادها را توصیف می‌کنند[8] . بسیاری از طراحی‌های اپلیکیشن‌های مدرن رویداد محور هستند ، مانند چارچوب‌های تعامل با مشتری که باید از داده‌های مشتری به صورت بلادرنگ استفاده کنند . اپلیکیشن های رویداد محور می توانند با تمام زبان‌های برنامه‌نویسی به وجود بیایند زیرا معماری رویداد محور یک رویکرد است نه یک زبان . معماری رویداد محور حداقل میزان دوگانگی را ممکن می‌سازد،که آن را به گزینه خوبی برای معماری‌های کاربردی مدرن و توزیع شده تبدیل می کند[4].

## تعریف رویداد

یک رویداد را می توان به عنوان "چیزی که اتفاق افتاده است یا تغییر قابل توجهی در وضعیت" توصیف کرد. رویدادها تغییرناپذیرند، یعنی نمی‌توان آن‌ها را تغییر داد یا حذف کرد و به ترتیب ترتیب وقوعشان مرتب می‌شوند[5].

## اجزای اصلی تشکیل دهنده معماری رویداد

معماری رویداد محور دارای سه جزء اصلی است که عبارتند از :

1-تولییدکننده رویداد

تولییدکننده‌های رویدادها درالگوی معماری رویداد محورصرفاً خالق رویدادها هستند . هر چیزی که می‌تواند یک رویداد تولید کند، تولیدکننده در نظر گرفته می‌شود، خواه یک حسگر مبتنی بر سخت‌افزارباشد ، یا یک درخواست HTTP، یا فرآیندهای تجاری و یا هر شکل دیگری از برنامه کاربردی که می‌تواند رویداد جدیدی را در معماری ما معرفی کند. [7].

2-‌‌ مسیریاب رویداد

مسیریاب رویداد به عنوان صف رویداد یا گذرکاه پیام عمل می‌کند که رویدادها را قبل از ارسال به مصرف‌کنندگان رویداد ، آن‌ها را به طور موقت ذخیره ، فیلتر، پردازش و مسیریابی می كند[59]

3- مصرف‌کنندگان رویداد

نقش مصرف‌کنندگان رویداد پردازش و اجرای رویدادهایی است که توسط تولییدکنندگان رویداد به وجود آمده‌اند.

نمونه‌های متداول مصرف‌کنندگان رویداد می‌توانند يك اعلان خودكار اسلك ،يك تابع براي تجزيه و تحليل و يا دستكاري داده‌ها ، برنامه‌اي براي توليد و ارسال ايميل‌ها براي خبرنامه‌ها و يا ورود به يك سيستم باشد [9]

## معماری رویداد محور چگونه کار می‌کند ؟

همانطور که توضیح داده شد معماری رویداد محور از سه جزء تولیدکننده رویداد ، مصرف‌کننده رویداد و مسیریاب رویداد تشکیل شده است . یک تولییدکننده رویداد یک رویداد را تشخیص یا حس می‌‌كند و رخداد را به عنوان يك پيام نشان مي‌دهد . تولييد‌كننده رويداد اطلاعي از مصرف‌كننده رويداد يا نيتجه يك رويداد را تشخیص نمی‌دهد .

پس از مرحله شناسایی ، یک رویداد از تولیدکننده رویداد از طریق مسیریاب رویداد به مصرف کنندگان رویداد منتقل می شود، جایی که یک پلت فرم پردازش رویداد ، رویداد را به صورت ناهمزمان پردازش می کند. مصرف کنندگان رویداد باید در صورت وقوع یک رویداد مطلع شوند. آنها ممکن است رویداد را پردازش کنند یا فقط تحت تأثیر آن قرار گیرند.

پلتفرم پردازش رویداد پاسخ صحیح به یک رویداد را اجرا می کند و فعالیت را در یک جریان سرازیری به مصرف کنندگان مناسب ارسال می کند. این فعالیت جریان سرازیری جایی است که نتیجه یک رویداد دیده می شود.

یک از مشهورترین پلتفرم‌های جریان‌داده توزیع شده آپاچی کافکا است که یک انتخاب محبوب برای پردازش رویداد است. این پلتفرم می تواند انتشار، اشتراک، ذخیره و پردازش جریان های رویداد را به صورت بلادرنگ انجام دهد. آپاچی کافکا طیف وسیعی از موارد کاربرد را پشتیبانی می کند که در آن توان عملیاتی و مقیاس پذیری بالا حیاتی است و با به حداقل رساندن نیاز به ادغام نقطه به نقطه برای به اشتراک گذاری داده در برنامه های خاص، می تواند تأخیر را به میلی ثانیه کاهش دهد.

میان افزارهای مدیریت رویداد دیگری نیز در دسترس هستند که می‌توانند به عنوان یک پلتفرم جریان رویداد مورد استفاده قرار گیرند[3] .

## چه زمان‌هايي از معماري رويداد محور استفاده مي‌شود؟

در موارد زیر از معماری رویداد محور استفاده میشود که در بخش زیر قابل مشاهده است .

1. تجمیع سیستم‌های ناهمگن

اگر سیستم هایی موجود باشند که روی پشته‌های مختلف اجرا می‌شوند ،آن هم بدون اینکه از هم اطلاع داشته باشند . این قابلیت موجود است که از معماری رویداد محور برای به اشتراک گذاشتن اطلاعات استفاده کرد بدون اینکه نیازی به استفاده از جفت‌شدگی[[2]](#footnote-2) باشد .مسیریاب رویداد به طور غیر مستقیم قابلیت همکاری را در بین سیستم ها ایجاد می‌کند ، بنابراین آنها می‌توانند پیام‌ها و داده‌ها را در حالي كه ناشناس باقي مي‌مانند ، مبادله كنند [2].

2-‌‌‌‌‌‌‌ ‌ گنجایش خروجی بالا

اگر سیستم‌های زیادی وجود دارند که به عنوان مصرف‌کننده برای پاسخ‌دهی به یک رویداد کار می‌کنند ، می‌شود که از معماری رویداد محور استفاده کرد تا دیگر نیاز نباشد تا برای ارسال رویداد به هر مصرف کننده کد سفارشی و جداگانه‌ای نوشت . آن هم بدین‌صورت که مسیریاب رویداد ،رویداد را به هر یک از مصرف‌كنندگان ارسال مي‌كند و آنها مي‌توانند رویداد را به طور موازی و با اهداف خاص که برای آنها تعریف شده است ،پردازش کنند [2] .

3- ‌نظارت بر منابع

برای برسی منابع ، می‌شود از معماری رویداد محور برای نظارت و دریافت هشدار در مورد هرگونه ناهنجاری ، تغییر و به‌روزرسانی استفاده کرد بدون اینکه نیازی باشد از بررسی مداوم استفاده کرد . این منابع می‌تواند شامل سطل‌های ذخیره‌سازی ، جداول پایگاه داده، توابع ‌بدون سرور،گره‌های محاسباتی و غیره باشد [2].

4- تکثیر داده‌ها بین سیستم‌هایی با موقعیت جغرافیایی متفاوت

اين قابليت وجود دارد كه برای هماهنگ‌کردن سیستم‌ها بین تیم‌هایی که در مناطق و حساب‌های مختلف فعال هستند و مستقر شده‌اند از معماري رويداد محور استفاده كرد . با استفاده از مسیبریاب رویداد برای انتقال داده‌ها بین سیستم‌ها ، می‌شود سرویس ها را مستقل از سایر تیم‌های توسعه ، مقیاس و استقرار داد[2] .

## سبك‌هاي مختلف الگوي معماري رويداد محور

سبك هاي مختلف الگوي معماري رويداد محور براساس نحوه عملكرد خود بر دو مدل مختلف تقسيم مي‌شوند كه در ادامه به آنها پرداخته خواهد شد[1] :

1. مدل انتشار اشتراك

اين يك زير ساخت پيام‌رساني مبتني بر اشتراك در جريان ‌رويداد است [3]. در این مدل هنگامی که یک رویداد منتشر می شود، رویداد را برای هر مشترک ارسال می کند. پس از دریافت یک رویداد، نمی توان آن را دوباره پخش کرد و مشترکین جدید رویداد را نمی بینند[1].

1. مدل جريان رويداد

رویدادها در یک گزارش نوشته می شوند. رویدادها کاملاً سفارشی (در یک پارتیشن) و بادوام هستند. مشتریان در جریان مشترک نمی شوند، در عوض مشتری می تواند از هر بخشی از جریان بخواند. مشتری مسئول پیشبرد موقعیت خود در جریان است. این بدان معناست که یک مشتری می تواند در هر زمان بپیوندد و می تواند رویدادها را دوباره پخش کند[1] .

مدل جريان رويداد نيز خود بر حسب نوع جريان رويداد به سه گونه ديگر تقسيم مي‌گردد كه عبارتند از:

1. پردازش رويداد گسترده

از یک پلتفرم جریان داده، مانند Azure IOT Hub یا Apache Kafka، به عنوان خط لوله برای جذب رویدادها و تغذیه آنها به پردازشگرها استفاده کنید. پردازشگرهای جریان برای پردازش یا تبدیل جریان عمل می کنند. ممکن است چندین پردازنده جریانی برای زیرسیستم های مختلف برنامه وجود داشته باشد. این رویکرد برای حجم کاری اینترنت اشیا مناسب است [1] .

1. پردازش رويداد ساده (وب سايت رد هت )

پردازش رویداد ساده زمانی است که یک رویداد بلافاصله اقدامی را در مصرف کننده رویداد ایجاد می کند [3]. برای مثال، می‌توانید از توابع Azure با یک تریگر Service Bus استفاده کنید، به طوری که هر زمان که پیامی در یک موضوع Service Bus منتشر می‌شود، یک تابع اجرا می‌شود [1] .

1. پردازش رويداد پيچيده

در پردازش رویداد پیچیده یک مصرف کننده با استفاده از فناوری هایی مانند Azure Stream Analytics یا Apache Storm، مجموعه ای از رویدادها را پردازش می کند و به دنبال الگوهایی در داده های رویداد می گردد[1].

## مزایای معماری رویداد محور

1. قابلیت تحمل خطا و بافرکردن

رویدادها ناهمزمان است، که باعث می‌شود رویدادها به محض اینکه اتفاق می‌افتند توسط تولییدکننده رویداد ارسال می‌گردند. مؤلفه مصرف کننده رویداد در این رویدادها مشترک می‌شوند و یا از جریان رویداد می‌خوانند . بنابراین اگر مصرف کننده ای از کار بیفتد، برنامه در غیاب آن به کار خود ادامه می دهد. رویدادها در مسیریاب رویداد ذخیره می‌شوند تا زمانی که مصرف کننده از موقعیت شکست بیرون بیاید و بتواند رویدادهای معلق شده را دریافت کند [5].

1. اتصال سست [[3]](#footnote-3)یا جداسازی تولید‌کنندگان و مصرف کنندگان

سیستم‌هایی که از معماری رویداد محوراستفاده می‌کنند دارای اجزایی هستند که به‌طور سست‌ متصل شده‌اند. این جداسازی به تفکیک منطقی تولید و مصرف رویدادها کمک می کند،بطوریکه نه تولیدکنندگان نگران نحوه مصرف رویدادها هستند، نه مصرف‌کنندگان نگران تولییدکنندگان رویدادها . به دلیل این اتصال سست، اجزا یا میکروسرویس های مختلف را می توان به زبان های دیگر توسعه داد یا از فناوری های مختلف استفاده کرد. همچنین به مقیاس بندی برنامه نیز کمک می شود. تولیدکنندگان یا مصرف‌کنندگان را می توان بر اساس الزامات بدون تغییر منطقی در سایر اجزاء اضافه یا حذف کرد [5] .

1. مقرون به صرفه

معماری‌های رویداد محور مبتنی بر عرضه هستند، بنابراین همه چیز بر حسب تقاضا اتفاق می‌افتد، زیرا رویداد خود را در روتر نشان می‌دهد. به این ترتیب، برای نظرسنجی مداوم برای بررسی یک رویداد هزینه ای پرداخت نمی کنید. این به معنای مصرف کمتر پهنای باند شبکه، استفاده کمتر از پردازنده است [2].

1. قابلیت مقیاس‌پذیری زیاد

اغلب می توان جریان های رویداد را به زیرجریان های نامرتبط تقسیم کرد و آنها را به صورت موازی پردازش کرد. همچنین می‌توانیم تعداد مصرف‌کنندگان را برای پاسخگویی به تقاضای بار ، در صورت افزایش حجم رویدادها، مقیاس کنیم. پلتفرم‌هایی مانند کافکا پردازش رویدادها را به ترتیب دقیق امکان‌پذیر می‌سازند و در عین حال امکان موازی‌سازی عظیم در سراسر جریان را فراهم می‌کنند. [4].

5- تجربیات کاربری بهتر

رابطهای برنامه نویسی کاربردی رویداد محور با حذف بسیاری از مسئولیت‌هایی که قبلاً به کاربران محول شده بود، منجر به تجربه تعاملی بهتری برای کاربر نهایی با نیازهای تعاملی مدرن می‌شوند. البته، این ممکن است پیچیدگی را در تولید کننده افزایش دهد، اما خروجی کار ارزش آن را دارد [5].

## معایب معماری رویداد محور

1-محدود به پردازش ناهمزمان

اگرچه معماری رویداد محور یک الگوی قوی برای جداسازی سیستم‌هاست ، اما کاربرد آن به پردازش‌ رویداد‌های ناهمزمان محدود می‌شود. معماری رویداد محور در جایی که آغاز کننده باید قبل از ادامه به کار باید منتظر پاسخ باشد، نمی‌تواند به خوبی به عنوان جایگزینی برای تعاملات درخواست-پاسخ کار کند [4] .

2-ایجاد پیچیدگی اضافی

درجایی که روش های قدیمی مشتری-سرور و درخواست-پاسخ فقط جزء اصلی را شامل می‌شود ، پذیرش معماری رویداد محور به یک کارگزار برای میانجیگری تعاملات بین تولییدکنندگان و مصرف‌کنندگان رویداد نیاز دارد [4].

3-‌ توسعه و خطایابی پیچیده

توسعه به دلیل ماهیت ناهمزمان الگو و همچنین ایجاد قرارداد ونیاز به شرایط مدیریت خطای پیشرفته‌‌‌‌‌‌‌‌‌‌‌‌‌تر در کد برای پردازشگرهای رویدادی که پاسخگو نیستند تا حدودی پیچیده می باشد[6] .

4-‌ آزمون پذیری پایین

گرچه تست واحد به طور کلی چندان سخت نیست . اما به یک سری ابزارهای تست خاص برای ایجاد رویدادها نیاز است . به علاوه بابت ماهیت آسنکرون این الگو ، آزمون‌پذیری پیچیده‌تر می‌شود [6] .

## نكاتي كه بايد هنگام استفاده از معماري رويداد محور بايد مورد توجه قرار داد؟

معماري رويداد محور يك نوشدارو نيست و مانند هر ابزار قدرتمند ديگري مستعد استفاده نادرست است . فهرست زير نبايد به عنوان معايب آشكار معماري رويداد محور خوانده شود ، بلكه بيشتر به عنوان مجموعه اي از مشكلات كه توسعه‌دهندگان و معماران بايد هنگام طراحي و اجراي سيستم‌هاي رويداد محور ازآن آگاه باشند [4].

1-رقص پيچيده

با مولفه‌هایی که به‌طور ضعیف جفت شده‌اند، می‌توان وارد موقعیتی شد که معماری ممکن است شبیه ماشین روب گلدبورگ باشد، که در آن کل منطق کسب‌وکار به‌عنوان مجموعه‌ای از اثرات جانبی که به‌عنوان رویدادها پنهان می‌شوند پیاده‌سازی می‌شوند: یک جزء ممکن است رویدادی را ایجاد کند که باعث ایجاد یک رویداد شود. پاسخ در مؤلفه دیگری که رویداد دیگری را ایجاد می کند، مؤلفه دیگری را تحریک می کند و غیره. درک و استدلال این سبک از تعامل بین اجزا به سرعت دشوار می شود[4].

2- پنهان كردن دستورات به عنوان رويداد

یک رویداد تصویری خام از چیزی است که اتفاق افتاده است. نحوه رسیدگی به رویداد را تعیین نمی کند. از سوی دیگر، یک فرمان یک دستور مستقیم است که به یک جزء خاص خطاب می شود. از آنجایی که دستورات و رویدادها هر دو پیام هایی هستند، به راحتی می توان یک فرمان را به عنوان یک رویداد نادرست معرفی کرد[4].

3-ناشناس ماندن مصرف كنندگان

رویدادها باید ویژگی‌های مرتبط را به گونه‌ای دربر گیرند که نحوه پردازش آن رویدادها را محدود نکنند. گفتن این کار آسان تر از انجام آن است. گاهی اوقات ممکن است اطلاعات بیشتری را در اختیار داشته باشیم که در تئوری می‌تواند به یک رکورد رویداد اضافه شود، اما مشخص نیست که افزودن آن اطلاعات به رکورد مفید است یا اینکه منجر به نفخ بی‌فایده می‌شود[4].

# مراجع:

ماکروسافت 1

آمازون 2

رد هت 3

مدیوم 4

نیلل 5

https://www.neebal.com/blog/advantages-and-disadvantages-of-event-driven-architecture

مرجع 6 اوریلی

مرجع 7 کتاب **Mastering JBoss Drools 6**

**https://hazelcast.com/glossary/event-driven-architecture/ مرجع 8**

مرجع 9

https://www.koyeb.com/blog/understanding-event-driven-architecture-and-serverless-opportunities

1. Event Driven Architecture Pattern(EDAP) [↑](#footnote-ref-1)
2. coupling [↑](#footnote-ref-2)
3. Loose coupling [↑](#footnote-ref-3)