سخن سردبیر

سخن سردبیر معمولا از آن بخش‌هایی است که هیچوقت خوانده نمی‌شود و خودم به شخصه همیشه سرمقالات و پیشوندها و بخش‌های این‌چنینی کتاب‌ها و مجلات را نخوانده رها می‌کنم. یکی از بزرگترین مشکلات دانشکده ما، عدم وجود فضاهایی است که در آن بتوان به صورت مفید و کارآمد، دانشجویان و اساتید دانش خود را به اشتراک بگذارند. البته نه آن دانشی که فقط تئوری باشد، بلکه دانشی کارآمد و صنعتی که متاسفانه دیگر در کلاس‌های درس پیدا نمی‌شوند. اولین جایی که می‌توان چنین فضایی ایجاد کرد، همین نشریه علمی دانشجویی است. هرچند که در این نشریه می‌توان به مشکلات فرهنگی دانشجویان هم پرداخت اما از نظر من، که سردبیر فعلی این نشریه هستم، پرداختن به جنبه علمی آن سودمندتر از پرداختن به مسائل فرهنگی است، که بسیاری از آنها از کنترل ما خارج‌اند.

در این شماره سعی کرده‌ایم که مطالبی با دامنه گسترده و با سطوح مختلفی آماده کنیم تا هرکس به نوبه خود بتواند از مطالب نوشته شده استفاده علمی ببرد. مطالب تخصصی‌تر نیز سعی شده با دقت بیشتری آماده شوند و توسط اساتید تأیید و بازبینی علمی شوند تا کمتر دارای اشکال باشند.

این یک نشریه علمی دانشجویی است و طبیعتاً وجود آن نیازمند دانشجویانی است که سواد خود را به اشتراک می‌گذارند. پس از دانشجویان و اساتید تقاضامندیم که به غنی‌تر شدن این نشریه کمک کنند تا بتوانیم سطح علمی دانشگاه را بالاتر برده و روحیه و نشاط علمی را تقویت کنیم.

تهیه شدن این نشریه زمان زیادی از نویسندگان، ویراستاران، طراحان و اساتیدی که مطالب را بازبینی کرده‌اند برده و بدون کمک آنها تهیه این نشریه ممکن نبوده، اما بدون کمک و حمایت‌های شرکت دانش بنیان اعوان نیز تهیه این نشریه با این کیفیت و حجم مطالب عملاً غیرممکن بود و همینجا از حمایتشان تشکر می‌کنم.

اما در مورد مطالب یک نکته‌ای حائز اهمیت است. اگر به مقاله جادی و همینطور به مصاحبه‌ای که با سرپرست تیم پردازش تصویر آزمایشگاه رباتیک دانشگاه انجام شده نگاهی بیندازیم به اهمیت علوم پایه و درس‌هایی مثل آمار و ریاضیات مهندسی پی می‌بریم. متاسفانه یکی از مشکلات تدریس اساتید، اشاره نکردن و به کار نبردن مفاهیم در حوزه‌های مرتبط با رشته دانشجویان است. صرف گفتن چند مثال فیزیکی ساده و گفتن این جمله که "در آینده از این مفاهیم استفاده خواهید کرد" کافی نیست.

در صورتی که مایل به همکاری با ما هستید و می‌خواهید مطلبی را با دیگران اشتراک بگذارید با ایمیل mhrimaz@email.kntu.ac.ir در تماس باشید.

دانشگاه صنعتی خواجه نصیرالدین طوسی
انجمن علمی کامپیوتر
نشریه عصر رایانه
شماره ۱۹
بهار ۹۶

حامی مالی
اعوان
مشاوران صنعت نرم افزاری

فهرست

www.cekntu.ir
telegram: @ceassociation
instagram: @ceassociation

اگر من به دانشگاه برمیگشتم

جادی

من الان نزدیک چهل سالمه. شاید به نظر شما پیر باشم ولی از نظر خودم فرق خاصی بین الان و سال های دانشگاه حس نمی‌کنم. شاید دلیلش کامپیوترها باشن که باعث می شن آدم همیشه با دنیای جدید در ارتباط باشه. شایدم یک جور نگرش به دنیا که سن و سال رو خیلی به رسمیت نمی‌شناسه. سال‌های دانشگاه برای من خیلی نزدیک به نظر می‌رسه و کاملا حس می کنیم همین پنج شش سال قبل بود که توی راهروها شیطنت می‌کردیم، بعد از حضور غیاب از پنجره فرار می‌کردیم یا دلمون خوش بود که قبل از حضور و غیاب خودمون رو به کلاس رسونده‌ایم یا موفق شده‌ایم سوالات امتحانی رو قبل از شروع امتحان پیدا کنیم. من تقریبا تمام مدت دانشگاه رو یا دودر می‌کردم یا به انواع تکنیک‌ها سر کلاس نمی‌اومدم و زندگی‌ام رو یا تو حیاط دور میزها می‌گذروندم یا تو سایت کامپیوتر در حال کشف چیزهایی که دوست داشتم؛ البته به جز قسمتی که به عشق و عاشقی می‌گذشت.

اون سال ها برای من بسیار جذاب و پر هیجان بود و اگر بهشون برگردم چیزهای بسیار کمی رو توش تغییر میدم. گفتم چیزهای خیلی کمی رو تغییر می‌دم؟ دقیقا. چیزهایی هست که تغییرشون می‌دم. من اگر به سال‌های لیسانسم که دانشجوی مخابرات بودم برگردم، مطمئناً بهتر درس می‌خونم. مطمئن هستم اگر قصد شما اپلای کردن و رفتن باشه باید خیلی بهتر از من درس بخونین ولی من حتی در زندگی فعلی‌ام هم فکر می‌کنم کاش بهتر درس خونده بودم. حداقلش این بود که استرس شب‌های امتحان و فصل‌های امتحان رو نمی‌داشتم. یادمه روزهایی به انصراف فکر می‌کردم؛ فقط چون درست درس نمی‌خوندم. فکر کنم با هفته‌ای دو ساعت درس خوندن به راحتی می‌تونستم درس‌هام رو پاس کنم. شاید باور نکنین ولی من هنوزم گاهی خواب می‌بینم سر امتحان مغناطیس هستم بدون اینکه بلد باشم انتگرال بگیرم. خوب درس خوندن حداقلش این بود که این استرس رو از من دور می‌کرد.

همچنین اگر این سال‌ها در دانشگاه می‌بودم حتما مشارکت خیلی زیادی توی پروژه‌های اوپن سورس می‌کردم. من اون سال‌ها به عنوان فری‌لنسر رایگان برای شرکت‌ها و پروژه‌های مختلف کار کردم و رزومه‌ام خوب بود ولی الان به نظرم حل باگ‌های پروژه‌های بزرگ یا حتی مشارکت بزرگ‌تر در اونها می‌تونه فوق العاده باشه. یک روش برای یاد گرفتن، کسب اعتبار، لینک عالی درست کردن و همچنین ساختن یک رزومه غیرقابل مقاومت برای هر شرکتی که دنبال نیرو باشه.

یک چیز دیگه که حتما می‌خوام در موردش حرف بزنم هم پایان نامه است. من یک پایان نامه راحت برداشتم؛ نتیجه هیجان انگیز بود البته ولی خودِ کار برای من ساده حساب می‌شد و توی چند شب تمومش کردم چون دقیقا تخصصم بود. من هنوزم طرفدار یک پایان نامه معقول و ساده هستم. شکستن شاخ غول جاش توی پایان نامه نیست - مگر اینکه بخواین باهاش مهاجرت کنین. قرار نیست پایان نامه تبدیل به یک منبع استرس بشه. اومدین دانشگاه که بهترین سال های عمر رو زندگی کنیم. دورانی که از دست خانواده راحت‌تر هستیم، دورانی که جامعه هنوز با ما کاری نداره، دورانی که اولین بخش زندگی مختلط ما است و دورانی که کسی نمی‌دونه چه ساعتی کجا هستیم و کنترل همه چیز تقریبا توی دستای خودمونه. من اگر به دانشگاه برگردم بازم مثل قبل تا جایی که بشه کشش می دم و ۵ سال و نیم دانشگاه می‌مونم. چی بهتر از دانشگاه؟ تازه اونم با حداقل استرس.

در ضمن این رو هم بگم که من با وجود معدل خیلی ضایع حدود ۱۲ در دوره لیسانس، چندین بیست مهم هم توی کارنامه داشتم که البته فکر کنم هر دو به یک اندازه خجالت آورن. بیست های من از درس‌هایی بود که کتابشون رو به انگلیسی می‌خوندم چون دوستشون داشتم. مدار منطقی، معماری کامپیوتر، میکروپروسسور و چیزهای مشابه. مطمئن هستم که اگر الان دانشگاه بودم کتاب‌های بیشتری رو از منابع اصلی می‌خوندم. اینجوری هم لذت بیشتری می‌بردم و هم استرس بلد نبودن پایه ها برام کمتر می‌شد. این بلد نبودن پایه‌ها بدترین بخش دانشگاه من بود. متاسفانه در سیستم آموزش داغون ما، پذیرفته است که شما با نمره‌ای مثل ۱۰ قبول بشین. مثلا توی دبستان شما اجازه دارین با حفظ کردن فقط نصف جدول ضرب (بخصوص مشهورهایی مثل پلنگ و شیش تا) نمره ۱۰ بگیرین و برین کلاس بعدی در حالی که در کلاس بعدی «انتظار» نظام آموزشی اینه که شما کل جدول ضرب رو بلد باشین. به نظرم این اصلی ترین استرس ما در تحصیل است. من می‌تونم بدون یاد گرفتن انتگرال دوگانه ریاضی رو با مثلا ۱۲ پاس کنم ولی بعد سر تمام

واحدهایی که تصورشون اینه که چون من ریاضی رو پاس کردم پس حتما خدای انتگرال دوگانه هستم، عصبی باشم. همین اتفاق در رشته من یعنی مخابرات با درس معادلات دیفرانسیل افتاد. من ۳ بار دیفرانسیل رو افتادم و بالاخره با ترفندهای خاص بدون دونستن سری فوریه و دوستانش نمره‌ام رو گرفتم و چندین ترم سر سیگنال سیستم و بقیه درس‌ها عذاب کشیدم. من اگر به دانشگاه برمی‌گشتم غیر ممکن بود بذارم این تیکه تکرار بشه.

اوه... و با اینکه من در کلاس‌های فوق برنامه خیلی فعال بودم؛ اگر به دانشگاه برگردم از اونم فعال‌تر می‌شم. بهونه‌هایی مثل «درس زیاده» و اینها واقعیت ندارن. من تا جایی که می تونستم به کلاس های مختلف می‌رفتم. از موسیقی تا کوه و لینوکس و انواع مجلاتی که منتشر می‌کردیم و از همه باحال‌تر گروه‌های مطالعاتی. هدف تا حدی وسعت بخشیدن به دیدگاه آدم است ولی در واقع اون پشت مشوق اصلی دیدن آدم‌های جدید و پیدا کردن دوست های متنوع‌تر است. می دونین که چی می گم؟ (: من مطمئن هستم که کتاب‌های زیادی می‌خوندم تا بتونم بهتر حرف بزنم و جهان رو بفهمم و مطمئن هستم که اگر فقط دو سه روز پشت سر هم در دانشگاه بهم خوش نمی‌گذشت، سعی می کردم کشف کنم چرا بهم خوش نگذشته و چی رو باید تغییر بدم که بهم خوش بگذره. دانشگاه تقریبا بهترین شرایط ممکن رو داره. گفتم که؛ کسی با شما کاری نداره ولی شما می تونین با همه کار داشته باشین. من اگر دانشگاه بودم طولش می‌دادم و هر چقدر می‌تونستم متنوعش می‌کردم. ببینم شما چیکار می کنین.

خاطره یک سال لذت‌بخش در دانشگاه صنعتی خواجه نصیرالدین طوسی

دکتر صادق علی‌اکبری
دبیر علمی انجمن جاواکاپ

دو سال پیش بود. با این که مشغول جمع‌بندی رساله دکترا در دانشگاه صنعتی شریف بودم و زمان کمی برای کارهای خارج از دانشگاه داشتم، و با این که چند نیمسال بود که تدریس در دانشگاه شریف را نمی‌پذیرفتم، این تجربه را از دست ندادم و تدریس در دانشکده مهندسی کامپیوتر دانشگاه خواجه نصیر را با جان و دل پذیرفتم. تجربه‌ی تدریس دو نیمسال در این دانشکده، حقیقتاً متفاوت و جالب بود. اساتید این دانشکده‌ی جوان، بسیار حمایت‌گر و دلسوز بودند و در بهبود اوضاع دانشکده می‌کوشیدند. اکثر دانشجویان هم فوق‌العاده با استعداد و بی‌ادعا بودند. چهره‌های ستاره‌های علمی آن کلاس‌ها که مشتاقانه در کلاس و خارج از کلاس، درس را پیگیری و مطالعه می‌کردند در ذهن دارم.

خاطره‌ی خوب این دانشگاه با من بود، تا این که امسال مسابقه علمی جاواکاپ را بین برنامه‌نویسان جاوای کشور در دانشگاه شهید بهشتی برگزار کردیم. در حالی که برنامه‌نویسان شرکت‌های بزرگ و دانشجویان دانشگاههای مختلف در این مسابقه شرکت کرده بودند، برخی از رتبه‌های برتر این مسابقه از دانشجویان دانشگاه خواجه نصیر بودند. باز هم این عبارت‌ها در توصیف دانشجویان خواجه نصیر در ذهن متبلور می‌شد: بااستعداد، پرتلاش و کم‌ادعا.

اگر از من بپرسید، میان دانشجویان دانشگاههای خوب ایران، از جمله دانشگاه خواجه‌نصیر، تفاوت قابل‌توجهی از نظر استعداد و توانمندی نیست. من در شریف و خواجه‌نصیر و شهیدبهشتی تدریس کرده‌ام. دانشجویان این دانشگاهها (و امیرکبیر و تهران و ...) می‌توانند بزرگان عرصه صنعت فناوری اطلاعات کشور ما باشند، اگر خودشان بخواهند. درست است که دانشگاه می‌تواند روند پیشرفت دانشجویان را تسریع و تسهیل کند، ولی واقعیت این است که انتخاب راه پیشرفت، که توأم با تلاش و مجاهدت است، در دست و ذهن خود دانشجویان است.

به امید انجام کارهای بزرگ توسط دانشجویان خوب دانشگاه خواجه نصیر

مصاحبه با سرپرست پردازش تصویر تیم رباتیک

حسین ریماز

اول از همه یکمی خودتون رو معرفی کنید تا بچه‌ها بیشتر باهاتون آشنا بشن و از فعالیت هاتون بگید.

سلام، پرنیا شکری هستم، دانشجوی ورودی ۹۲ برق. از ترم اول از طریق کلاس‌هایی که آزمایشگاه برگزار می‌کرد با آزمایشگاه رباتیک آشنا شدم و واردش شدم و از قبل هم هیچ آشنایی با رباتیک نداشتم. الان سرپرست پردازش تصویر تیم ربات های پرنده هستم. کار هایی که من الان دارم انجام میدم و کلاً تیم ما روش کار می کنه در راستای اینه که ربات رو هرچه بیشتر Autonomous کنیم، یعنی وابستگی اش به یک انسان برای کنترل کمتر بشه و خودش بتونه خودش رو هدایت کنه. اینکارو با استفاده از پردازش تصویر انجام میدیم و اخیراً داریم به سمت استفاده از شبکه‌های عصبی و یادگیری ماشینی میریم.

در مورد تیم روباتیک ارس ، نیاز مندی ها و پوزیشن های خالیش و توانایی‌های مورد نیاز و همچنین موفقیت‌های این تیم رو کمی توضیح بدید.

تیم روباتیک ارس دوتا آزمایشگاه مجزا است، یکی برای بچه های کارشناسی ارشد و یک هم برای بچه های کارشناسی که من توی آزمایشگاه بچه‌های کارشناسی هستم. این آزمایشگاه هدفش این بوده که بچه‌های کارشناسی که وارد دانشگاه میشن، فقط با تئوری کار نکنن و از درسهایی که میخونن به صورت عملی استفاده کنن و مطالب پیاده‌سازی بشن. مثلاً تمام پروژه های درسی ای که بچه‌ها دارن از جمله خودم توی پروژه‌های آزمایشگاه می گنجه و میشه پیاده سازیشون کرد. اینجا فقط چیز هایی که درس میدن رو پیاده نمی کنن و بچه ها خیلی چیزای جدید رو که خودشون علاقه دارن رو میتونن یاد بگیرن و پیاده کنن.

به نظر من مهم‌ترین چیزهایی که میشه اینجا در کنار مطالب علمی یاد گرفت شاید بشه گفت دوتا چیزه. اول کارگروهیه. تو دانشگاه بستر خیلی کمی وجود داره که چنین چیزی رو به ما یاد بدن و اکثر پروژه‌هایی که برمی‌داریم تک نفره هستن و چیزایی هست که کمتر کارگروهی توشون دیده میشه اما توی تیم کارگروهی خیلی نمود پیدا می کنه. دوم اینکه ما یاد میگیریم که از کاری که هیچی ازش بلد نیستیم نترسیم و بریم سراغش. مثلاً من خودم رشتم برق بود و هیچی از پردازش تصویر نمی دونستم، ولی وارد تیم شدم و وقتی دیدم که کارایی که انجام میدن برام جذابه رفتم سمت پردازش تصویر، با اینکه اصن توی درسای من نبود و من هیچی ازش نمی دونستم. روند ایجا اینطوریه که بچه‌های قدیم مطالبی که میدونن رو به ما انتقال میدن و نسل جدید راهشونو با کارای جدید تر ادامه میده. تو این راه سرچ کردن حرف اولو میزنه. تقریباً الان دیگه تو کار ما تقریباً هر مطلبی رو میشه از اینترنت یاد گرفت.

اگر بخوام یکم از تیم و موفقیت‌هاش بگم، ما در حال حاضر سه تیم، ربات های فوتبالیست، ربات های امداد گر و ربات های پرنده رو داریم. آخرین دستاوردی که آزمایشگاه داشت توی مسابقات جهانی IMAV ۲۰۱۶ بود و در کشور چین برگزار شد که برای تیم روبات های پرنده بود که اونجا مقام اول ربات های پرنده داخل سالن رو کسب کردیم. بعد اون همایش ICRoM بود که بچه‌های تیم امداد گر تونستن مقام دوم توی قسمت خلاقیت با رباتی که ساخته بودن رو کسب کنن. البته این تنها دست آورد های اخیر آزمایشگاه بود.از سال ۸۲ تیم های مختلفی توی دانشگاه حضور داشتن که از سال ۹۱ این تیم ها تجمیع پیدا کردند و این آزمایشگاه تشکیل شده. مقام های مختلف زیادی هم در سال‌های گذشته کسب کردیم. مثلاً قهرمانی IRANOPEN ۲۰۱۶ , ۲۰۱۵ ربات های پرنده داخل سالن ، نایب قهرمانی و عنوان بهترین ربات خودکار ربات های امدادگر تو مسابقه ی AUTCup ۲۰۱۵ ، مقام اول ربات های کاوشگر SharifCup ۲۰۱۵ که تیم اسمال سایز شرکت کرده بود و مقام سوم SharifCup ۲۰۱۵ برای تیم ربات های پرنده و افتخارات دیگه ای که همشون رو یادم نیست.

در حال حاضر علاقه‌مند هستیم که وارد کارهای صنعتی هم بشیم. از اونجایی که بیشتر استفاده از ربات های پرنده در خارج از سالن (Out Door) هست، تیم داره به اون حوزه هم

ورود پیدا میکنه.
هرکسی که علاقه‌مند باشه میتونه وارد تیم بشه. وارد شدن به تیم آسونه ولی موندن توش سخته. اینجا کسی که میاد باید پشتکار داشته باشه. کار اینجا خصوصا نزدیکای مسابقه خیلی سنگین میشه. به نظرم کسی که میخواد وارد اینجا بشه اول از همه باید پشتکار داشته باشه و دوم حس مسئولیت پذیری داشته باشه. چون اگر مسئولیتی که بهش داده میشه رو نتونه درست انجام بده کل تیم ضرر می کنه. ولی خوب من واقعاً با تمام این سختی‌ها راضی هستم و اصلاً پشیمون نیستم که وقتم رو گذاشتم، اگه به عقب برمیگشتم باز هم همین راهو انتخاب میکردم. کارگروهی که اینجا یاد گرفتم واقعاً ارزشش رو داشت و با آدم‌های خیلی خوبی آشنا شدم و ایده‌های جدیدی رو مطرح کردیم و با کمک بچه‌هایی که از رشته‌های مختلف کنار هم

جمع شدن تونستیم اون ایده‌ها رو پیاده کنیم. کارایی که یک دانشجوی برق به تنهایی نمی تونست انجام بده.

مهندسی برق رو دوست داشتید که انتخاب کردید یا توی جو کنکور و انتخاب رشته این انتخاب رو کردید؟

یکمی پیچیده است! من دبیرستان که بودم خیلی نجوم دوست داشتم، خیلی! ولی خوب خانواده‌ام نمیذاشتن که برم سمت رشته ی نجوم. در کنار اون هم فیزیک ذرات بنیادی رو هم دوست داشتم که خوب خانوادم نه میذاشتن برم نجوم نه فیزیک! بعد من دیدم توی برق-الکترونیک، گرایش های نیمه هادی ها خیلی شبیه فیزیک ذرات بنیادیه و خوب من دیدم برام جالبه و خوب گفتم برق بخونم و برم الکترونیک و تو این زمینه کار کنم.
وقتی اومدم دانشگاه ترم اول دیدم که یه گروه رباتیک هست که یه کلاسی گذاشته و خوب معمولاً ترم اول دانشجو ها ذوق دارن و اکثر کلاس‌ها رو شرکت می کنن. تقریباً یه ترم دوره مقدماتی رو گذروندم و بیشتر توی اون دوره کارای برقی می‌کردیم تا نرم افزاری و با بورد ها کار می‌کردیم و برنامه نویسی میکرو کنترلر و ... و برام جالب بود که آدم میتونه یه رباتی بسازه که کاریی که دوست داره رو انجام بده. چون یه پیش زمینه ای از نجوم داشتم، و این چیزا رو هم دیدم تو ذهنم هدف جدیدی برام ایجاد شد. سازمان های فضایی که شاید معروف ترینش برای ما ناسا باشه برای اکتشافاتشون به فضاپیما ها و روبات هایی احتیاج دارن که درواقع که بیشش همین کاریه که اینجا تو آزمایشگاه ما داره انجام میشه اما خیلی پیشرفته تر. از اونجا رباتیک شد هدف من. چون علاقه به نجوم داشتم و این میتونست اون کنجاوی هام رو برطرف کنه و خیلی هم دوست داشتم و دارم عضوی از پروژه‌های فضایی باشم. هرچی بیشتر جلو رفتم اون فکرای قبلیم که میخواستم وارد گرایش الکترونیک بشم از بین رفت و الان هم الکترونیکی نیستم و رفتم کنترل چون بیشتر به مباحثی که من بهشون علاقه داشتم از رباتیک ربط داشت. الان هم برای ارشد دیگه کلاً میرم کامپیوتر نه برق! شاید اگه از اول رشتم کامپیوتر بود بهتر بود اما خب یه جورایی خوبه که حداقل الان دارم چیزی که دوست دارم رو پیدا می‌کنم نه دیرتر، اگه برم تا دکترا و بعد بفهمم چی دوست داشتم خب خوب نیست. اینکه الان رشتم برقه یه خوبی ای که داره اینه که یه دید برقی نسبت به موضوعات دارم منتها بدیش اینه که این درس‌های کامپیوتری خصوصا نرم افزاری ها میگذرونن رو من نگذروندم تو برنامه نویسی و خودم باید اونها رو بخونم و خوب یکمی سخت تر هست.

دوست نداشتید که کامپیوتر میخوندید؟ درس‌های کامپیوتری براتون جذابیت بیشتری دارند یا برقی؟

چرا چرا! خیلی دوست داشتم!!! البته یه مشکلی که به نظرم هست مخصوصاً توی بچه‌های نرم افزاری اینه که خیلی دیدی نسبت به سخت‌افزار کار ندارن بعد شاید توقع های عجیب غریب داشته باشن. مثلاً معمولاً ما مشکلمون تو آزمایشگاه این بود که نرم افزاری هایی که میومدن توی تیم زود از تیم میرفتن و خیلی این صبر رو نداشتن که خوب این یه رباته و خوب کد زدن روش مشکلات خودش رو داره. مثلاً بهینه بودن کد ها و سریع بودنشون خیلی مهمه و خوب ما که برقی بودیم بیشتر این مشکلات رو متوجه میشدیم و تحمل می کردیم. اما خوب فکر کنم بهتر بود که کامپیوتر میخوندم! تو طول ترم ها مثلاً جاوا رو هم برداشتم به عنوان درس اختیاری اما طبیعیه که نمیتونم همه ی درسای اصلی کامپیوترو بگذورنم. برام تو رشته کامپیوتر یه سری از درسا جذابن و تو برق هم یه سری از درسای برقی. من خودم اگه برگردم عقب و اگه میتونستم یه رشته جدیدی اختراع می‌کردم که مخلوطی از درسای برقی و کامپیوتری بود!

بهترین استادتون کی بوده؟ چرا؟ چه تأثیر مثبتی داشته روتون؟

سخته! نمیشه چندتا باشه؟ اولین نفری که واقعاً تأثیر زیادی داشته روم دکتر تقی راد ، سرپرست آزمایشگاه KN2C هستن. و خوب ایشون خیلی تأثیر زیادی روی زندگی من داشته توی این چند سال لیسانس. به نظر من خوبی این آزمایشگاه اینه که مثل آزمایشگاه‌های دیگه که بیشتر برای بچه‌های ارشده و استاد میگه که بچه‌ها باید چیکار کنن نیست. اینجا دانشجو ها خودشون تصمیم می گیرن که چیکار کنن و استاد بیشتر در نقش یک راهنما است و این اجازه رو بهت میدن که اشتباه کنی. بعضی جاها دکتر هم میدونن که شاید ما داریم اشتباه می کنیم، حتی بهمون هم میگن اما این اجازه رو به ما میدن که خودمون انتخاب کنیم و تجربه کنیم و خیلی هم حمایتمون کردن.

درباره استاد های دیگه اگه بخوام بگم، دکتر درمانی من باهاشون اصول میکرو داشتم و اصول میکرو هم یه جورایی به کارای الکترونیکی روباتیک ربط داره. ایشون خیلی قشنگ و مفهومی درس میدادن و آدم جذب میشد. مفهموم خیلی از کارایی رو که قبلا کدشون رو میزدم ، با درس دادن ایشون یاد گرفتم. کلاسشون تجربه لذت بخشی بود برام.

این ترمم درس بینایی ماشین داشتم با دکتر ابریشمی مقدم، ازون جایی که دارم پردازش تصویر کار می‌کنم درس خیلی خوبی بود برام چون باز تو این درس ریز اون الگوریتم هایی که داشتم ازش استفاده می‌کردم رو مفهموشو بیشتر یاد گرفتم و خیلی برام جالب بود ازونجایی که ایشون خیلی هم خوب توضیح میدادن.

چه زمانی اصن علاقه‌مند به رباتیک شدید و چطوری وارد تیم رباتیک شدید؟

همون ترم اول که اومدم فکر کنم یک ماه از ترم گذشته بود شایدم کمتر و دیدم که کلاس گذاشتن و دیدی هم نداشتم و همینجوری رفتم که ببینم چطوریه. یه کلاس مقدماتی ای بود که برام جالب بود، بعد اون کلاس یه امتحانی بود که خوب چون من خیلی علاقه داشتم اون امتحان رو هم خیلی خوب دادم. بعد دوره های پیشرفته هم شرکت کردم بعد وارد یه تیمی شدم به اسم CanSat که الان نیست البته و قرار بود به صورت موقت توی اون تیم برم و یه مسابقه‌ای شرکت کنیم. چون اون موقع روال آزمایشگاه اینجوری بود که بچه‌های جدید اول وارد تیم های کوچیک می شدن و بعد از یه سال کار کردن و یه مسابقه رفتن وارد تیم اصلی میشدن اما الان اینجوری نیست و به بچه‌ها اول یه پروژه‌هایی میدیم و بچه‌ها وارد تیم اصلی میشن.

چطوری به پردازش تصویر علاقه‌مند شدید و چطوری یاد گرفتید؟

بعد اون تیم های کوچیک باید انتخاب می‌کردیم که وارد چه تیمی میخوایم بشیم. روند اینطوری بود که بچه‌های قدیمی تر هر تیم تو یه جلسه برامون درباره کارهای تیمشون ارائه هایی انجام میدادن و از فرصت های موجود توی تیمشون برامون توضیح میدادن. خودم بین اسمال سایز و ربات های پرنده مونده بودم. توضیحات پردازش تصویری که ارائه شد برام خیلی جذاب بود و دورنمایی هم که از اون ربات پرنده میدادن این بود که هوش باهاش میتونه مخلوط شه. من دیدم که خیلی جالبه برام که آدم بتونه یه موجودی خلق کنه که بتونه یه کم شبیه انسانا رفتار کنه و بشه بهش هوش اضافه کرد. بعد اومدم وارد تیم ربات های پرنده شدم، مقدمات کار پردازش تصویر رو هم از بچه‌های قدیمی تو چند جلسه کلاس یادگرفتیم. بعدش دیگه ما رو یکم آزاد گذاشتن و هدف‌ها و تسک ها رو گفتن و دیگه خودمون باید میرفتیم انجام بدیم. خوب اولش یکمی گیج بودیم اما خیلی خوب بود که اینکارو کردن چون یاد گرفتیم که خودمون رو پای خودمون وایسیم و به کسی متکی نباشیم. و خودمون سرچ می‌کردیم و کم کم دستمون اومد که باید چیکار کنیم. کم کم با رفتن بچه‌های قدیمی دیگه سرپرستی تیم به عهده من قرار گرفت.

یکمی راجب پردازش تصویر و تکنیک ها و الگوریتم ها و تکنولوژی‌هایی که استفاده کردید توضیح بدید.

ما بیشتر برای پردازش تصویر از کتابخونه OpenCV و زبان C++ استفاده می کنیم. خوب اولین مزیتش سرعتشه و هم اینکه داکیومنت های زیادی وجود داره و تیم های مطرح دیگه دنیا هم از همین زبان استفاده می کنن. الگوریتم هایی که تا الان استفاده کردیم یه بخشیش الگوریتم های تشخیص (Detection) هست.حالا میتونه این تشخیص یه شکل ساده باشه که خوب توابع ساده‌ای هم برای اینجور کارا هست. حالا یکمی سخترش تشخیص انسان ، یا دستش یا صورتش که خوب بازم توابعی هست که OpenCV از قبل به صورت Train شده داره و میشه از اون ها هم استفاده کرد. برای الگوریتم هایی که به حرکت ربات کمک می کنن ما خیلی از الگوریتم OpticalFlow استفاده می کنیم. تقریباً برنامه‌ای

که توی همه شرایط داره اجرا میشه این الگوریتم هست. با استفاده از دوربینی که زیر ربات هست و زمین رو میبینه، و این الگوریتم میتونیم بردار های حرکتی (مثل بردار سرعت و شتاب) رو استخراج کنه. این کار میتونه به سیستم IMU و سیستم کنترلی کمک کنه که بتونه بهتر ربات رو کنترل کنه. مثلاً ربات سرجاش وایسته یا مثلاً با یه سرعت ثابت حرکت کنه. در اون سیستم‌های کنترلی، سنسور ها معمولاً خطا هایی دارن که با کمک پردازش تصویر و تلفیقشون با فیلتر کالمن نتیجه بهتری میشه گرفت.

غیر از اون بحث های Exploration, Navigation هست. که استارت اینکار اول توی تیم امداد گر خورده بود چون اون ربات ها باید بتونن فرضا یک محیط زلزله زده رو بررسی کنن و نقاط مختلف و افراد آسیب دیده رو شناسایی کنن. ما توی ربات های پرنده نیاز داریم که بتونیم یه محیط سه بعدی رو نقشه اش رو در بیاریم.اکثر اینکار های سنگین رو بیشتر تابستون شروع می کنیم. مثلاً همین تابستون روی این بحث‌ها خیلی کار کردیم و از SLAM های سه بعدی استفاده کردیم. الان هم از دوربین های real sense و xtion برای اینکار استفاده می کنیم. در حال حاضر تیم در حال حرکت به محیط های خارج سالن (outdoor) هست. که خوب چالش هایی که باید حلشون کنیم خیلی زیاد و جالبه. یکی از خوبیای outdoor اینه که میتونیم از اطلاعات GPS استفاده کنیم اما خوب با این حال باید بتونیم موانع رو تشخیص بدیم و نحوه عبور ازشون رو مشخص کنیم.

چالش هایی که تیم روباتیک الان داره حلشون می کنه چیه؟ میشه در موردشون توضیح بدید؟

کلاً کار ما چالشه!!!

توی تیم کاری که تا الان کرده بودیم خیلی هوشمند نبود و بیشتر الگوریتم بود. اما الان داریم سمت یادگیری ماشین میریم و نیاز داریم که پیاده‌سازی های بهینه ای برای اون الگوریتم ها داشته باشیم.

یکی دیگه از چالش هامون حفظ جایگاه تیم هست! و این بهترین بودن و موندن خیلی سخته و نیاز به تلاش خیلی زیادی داره. برای بقیه تیم ها مثل تیم امداد گر و اسمال سایز هم چالش های مشابه وجود داره.

فکر می‌کنید چی باعث شده که تیم روباتیک موفق باشه و بتونه رتبه های جهانی بیاره؟

اولین موضوع کار گروهیه قطعا و کاری که ما کردیم چیزی نبود که بشه با یکی دو نفر انجامش داد. هماهنگ شدن افراد و کارهاشون و هماهنگی اعضای تیم نقطه قوت کار ماست. دومین موضوع اینه که از چیز های جدید نترسیم. مثلاً قبل تابستون پلتفرم ربات ها یه چیز دیگه بود و برای تابستون و این مسابقه رفتیم سراغ چیز های جدید. مثلاً مینی کامپیوتر هایی که استفاده می‌کردیم مبتی بر ARM بودن اما ما قدرت پردازشی بیشتری میخواستیم که بتونیم بیشتر پردازش ها رو روی خود ربات انجام بدیم به جای اینکه تصاویر روی پایگاه زمینی پردازش بشن. نمی‌خواستیم دیگه مشکل قطع و وصل شدن وای فای و delay برنامه‌ها رو داشته باشیم. برای همین که مجبور شدیم بریم سراغ مینی کامپیوتر هایی که اصن تو ایران نبودن و از خارج سفارش دادیم و برای خودمون خیلی جدید بود! باید سیستم کد ها و ارتباط برنامه هامون رو عوض می‌کردیم که خوب چالش برانگیز بود. نکته بعدی پشتکار اعضای تیم هست. فشار خیلی زیادی روی بچه‌های تیم بود. روز هایی بوده که تا ۱۰ شب دانشگاه بودیم و دیگه خود نگهبانا مارو میفرستادن خونه! نکته بعدی

برنامه‌ریزی تیم هست و ما همیشه به پلن های جایگزین داشتیم که بتونیم توی شرایط سخت راه حل‌های دیگه ای داشته باشیم. و همین پلن ها بود که مارو نجات داد. مثلاً برای مسابقات چیزی که تو ذهنمون بود بعضی جاها خیلی متفاوت با اون چیزی شد که اجرا شد.

سخت ترین شرایطی که توی تیم رباتیک تجربه کردید چی بود و چطوری تونستید از پسش بر بیاید؟

سخت ترین شرایط برای من مسابقه جهانی بود چون ما اولین تیمی از KN2C بودیم که تو چنین سطحی داشتیم مسابقه میدادیم. و دانشگاه حمایت مالی زیادی کرده بود و از ما انتظار زیادی میرفت که بتونیم موفق بشیم. در همون زمان هم باید اعضای جدید رو مدیریت می کردیم. و تحمل این فشار و استرس برای من مشکل بود. هنوزم وقتی فیلم‌های رو می‌بینم اون حس استرس رو احساس می کنم. به نظرم این تجربیات خیلی خوبی برای من بود و خیلی یاد گرفتم ازش که اینحور شرایطو چطوری میشه مدیریت کرد. این تجربه تو آینده فکر می‌کنم خیلی

میتونه مفید باشه.

به کسانی که تازه میخوان پردازش تصویر یاد بگیرن چه مسیر یادگیری ای رو پیشنهاد می کنید؟

به نظرم با کتاب‌های OPENCV شروع کنید و با یکی از زبان‌های Python , C++ میتونن شروع کنن. خود سایت OpenCV هم داکیومنت ها و آموزش‌های خوبی داره که می تونن از اون استفاده کنن.کتاب OpenCV by Example هم کتاب خوبیه که اون رو هم میتونن بخونن.

مؤثر ترین عامل پیشرفتتون چی بوده؟ دانشگاه یا بچه ها؟ هرکدوم چه تأثیری روی شما داشتن؟

مؤثر ترین که خود بچه‌ها و آزمایشگاه. شاید اگر برمی گشتم عقب و این آزمایشگاه نبود واقعاً ترجیح میدادم که خواجه نصیر نیام اگه میشد. واقعاً خوبی این دانشگاه برای من این آزمایشگاهشه. آدمایی که توش هستن آدم‌های خوش فکری هستن و کار هایی که انجام میدن خیلی ارزشمنده و همراهی باهاشون خیلی تجربه لذت بخشیه. به نظرم در نهایت اون حس پشتکار و مسئولیت باعث میشه که آدم بتونه تو این تیم پیشرفت کنه. چقدر دختر بودن روی فعالیتتون تأثیر داشته؟ مثلاً نگاه دیگران؟

نظرتون راجب پیش‌فرض هایی مثل اینکه پسر های توی برنامه نویسی بهترن؟ توصیه‌ای برای دخترانی مثل خودتون دارید؟

اولش که وارد آزمایشگاه شدم هم پسر ها بودن هم دختر ها اما خوب پسر ها بیشتر بودن. خودم دید خاصی نداشتم که پسر ها بهترن یا دختر ها. اما هرچی جلو رفتم دختر ها زودتر کنار می کشیدن اما من کنار نمی کشیدم! چون پسر ها بیشتر بود

تعدادشون دختر ها شاید احساس خجالت و تنهایی می کردن و حس خوبی نداشتن و کم کم از آزمایشگاه فاصله می گرفتن. اما خودم همچین احساسی نداشتم. متاسفانه خیلی از بچه ها همچین حسی داشتن که به نظر من غیر منطقی هست و در آینده هم همچین موقعیت هایی برامون توی بازار کار هم پیش میاد.

به نظرم اینطوری نیست که پسر ها کد نویس های بهتری هستن. شاید دخترا بیشتر دوست دارن که بهشون بگن که چیکار کنن و اونا برن اونکار رو به بهترین شکل انجام بدن اما پسر ها کمی خلاقیت بیشتری دارن و خودشون میرن یه کاری برای خودشون می کنن و ریسک می کنن و شاید دخترا ترس بیشتری از شروع کار دارن. البته اینو خیلی بسط نمیدم صرفاً نظر شخصی خودمه. و البته هستن دختر هایی که بهتر از پسر ها ایده میدن و کار میکنن.

بزرگ‌ترین حسرتتون چی بوده؟ مثلاً اگر برمیگشتید به ترم ۱ چه کارایی بوده که حسرت انجام دادن یا ندادنشون رو داشتید؟

ای کاش ترم های اول بیشتر درس میخوندم و درس‌های پایه‌ای رو بیشتر میخوندم مثل ریاضی ۱ و ۲ و سیگنال و ... اون موقع نمی‌دونستم اون درس‌ها به چه دردی میخوره مثلاً فکر نمی‌کردم درس سیگنال به چه دردی میخوره اما بعدش که بیشتر کار کردم اون مفاهیم رو دیدم. کاش اون موقع یکی بود که بهم می‌گفت اینا به چه دردی میخوره. خوب الان مجبورم برم سراغشون که طبیعتاً بیشتر ازم وقت میگیره که خوب اون موقع که فرصتش رو داشتم خوب نخوندم.

حستون نسبت به دانشگاه خواجه نصیر؟

معمولاً خواجه نصیریا بچه‌هایی هستن که انتظار بیشتری از خودشون داشتن اما کنکورشون رو خراب کردن و اومدن اینجا و معمولاً یکمی افسرده شدن و راضی نیستن از جایی که هستن. جو کلی هم خیلی سرزنده نیست و پراکندگی دانشکده ها خیلی این موضوع رو تشدید کرده. اما آزمایشگاه ما از نظر من خیلی فضای خوبی داره به خاطر اون کار تیمی که توش انجام میشه. خواجه نصیر رو بخاطر آزمایشگاهش دوست دارم.

بزرگترین حسن و بزرگترین بدی دانشگاه خواجه نصیر چیه؟

همونجور که گفتم بزرگترین حسنش برای من آزمایشگاهه، بخاطر دادن فرصت اشتباه و بحث‌های علمی و کارگاهی گروهیش. بزرگترین بدی دانشگاه از نظر من اینه که به دانشجو ها یاد نمیده چطوری زندگی خوبی داشته باشن. من نمی‌فهمم هدفشون از تربیت دانشجو چیه! یعنی حس می‌کنم که یه دوره! یعنی دانشجو تربیت می کنن که استاد بشه که اون استاده یه سری دانشجو تربیت کنه که باز اونا استاد شن و یه سری دانشجو تربیت کنن و ... . من دور و برم که می‌بینم حداقل تو رشته خودم اکثر بچه‌ها افسرده هستن و نمی دونن درس‌هایی که میخونن برای چی دارن میخونن حتی اونایی که رنک هستن نمی‌دونن چیزایی که خوندن استفاده اش برای چیه. خیلی جاها درسا فقط تئوری هستن.

دوست داشتم دانشگاه این جور آزمایشگاه‌ها رو گسترش بده و بچه‌های بتونن با کار درگیر بشن و بچه‌ها با هم کار های گروهی علمی انجام بدن. به نظرم این چیزیه که خیلی دانشگاه بهش نمی‌پردازه. بعضی وقتا بعضی از استادا از اینکه ما روباتیکیم ناراحتن و اصن باهمون لج هم می کنن!

ادامه تحصیل در خارج یا داخل؟ چرا؟

من خودم در خارج ادامه میدم انشاللّه. دلیلش برام دو بخش داره. بخش علمیش که هم از لحاظ امکاناتی و اون چیز هایی که من بهش علاقه دارم شاید توی خارج کشور بیشتر روش کار بکنن و خوب اونجا پیشرفت بیشتری بتونم داشته باشم. قسمت دومش که شاید مهم‌ترم باشه آینه که از لحاظ زندگی من یک سری فاکتور ها توی ذهنم هست که به نظرم میاد شاید بتونم خارج از ایران راحت‌تر زندگی بکنم. البته خوب شایدم اینطوری نباشه اما چیزی که مسلمه اینه که می ارزه که امتحانش کنم، اگه اینجوری بود که خوب همونجا میمونم و اگرم نبود که بر میگردم.

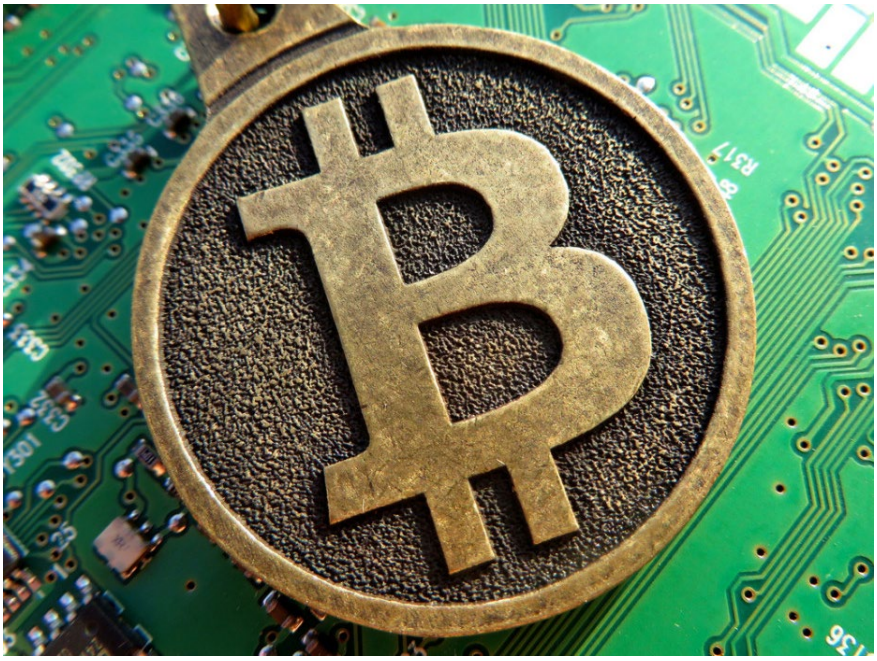
به کسانی که اول راه هستن چه نصیحتی داری تا موفق تر بشن؟

درس‌های ترم های اول به نظر خیلی تئوری میاد و غیر کاربردی. اما بعداً متوجه میشین واقعاً اون قدرا هم غیر کاربردی نبودن. به نظر من سعی کنین خیلی خوب اون مفاهیم رو یاد بگیرید و اگه معدل براتون مهمه تو اون ترمای اول راحت‌تر میتونین معدل بالا داشته باشید چون ترم های بالا تر درس‌ها خیلی سخت میشن.

پیشنهاد میکنم بچه‌های ترم های اول وارد همچین محیط هایی مثل آزمایشگاه ما بشن. براشون تجربه خوبی خواهد بود.

بیت کوین معدن طلای دیجیتال

کیمیا سعادت



بیت کوین نوع جدیدی پول است که در سال ۲۰۰۹ توسط برنامه نویسی ناشناس با نام مستعار «ساتوشی ناکوموتو» منتشر شد. با خواندن واژه‌ی پول احتمالا ذهن شما به سمت پول کاغذی و واحدهای پول مرسوم (مانند دلار) می‌رود که عموما مردم برای ذخیره و نگهداری این پول،ها ، آن‌ها را به بانک می‌سپارند.

باید گفت تنها وجه اشتراک بیت کوین و پول های معمول ، کاربرد آن‌هاست. بیت کوین مانند بقیه پول‌ها ارزش مالی دارد و برای خرید کالاها و خدمات واقعی استفاده می‌شود.

با این‌حال بیت کوین تفاوت‌های زیادی با پول معمولی دارد که برخی از این تفاوت‌ها باعث محبوبیت آن شده است.

۱.بیت کوین پولی دیجیتال و مجازی است.

بیت کوین به صورت فیزیکی تولید نمی‌شود و الکترونیکی ساخته و نگهداری میشود. (به زبان ساده‌تر: بیت کوین مجموعه‌ای از رشته‌های طولانی کد است که ارزش مالی دارند!)

قبل از بیت کوین اغلب پول‌های مجازی برای خرید کالاهای خاص مجازی به کار می‌رفته‌اند (مثلا در اکثر بازی‌ها نوعی پول جدید برای خرید آیتم‌های بازی ایجاد شده است) اما بیت کوین پولی مجازی است که با آن می‌توان کالاهایی واقعی خرید.

۲.بیت کوین غیرمتمرکز است.

این نوع پول به شرکت یا دولت خاصی وابسته نیست و توسط آن‌ها کنترل نمی‌شود.

۳.انتقال بیت کوین شخص به شخص است.

برای انتقال پول معمولا واسطه‌ای مثل بانک یا سیستم کنترلی وجود دارد که بر پرداخت‌ها نظارت می‌کند. اما در بیت کوین انتقال پول از شخصی به شخص دیگر بدون واسطه صورت می‌گیرد.

٤.ناشناس ماندن

در شبکه‌ی بیت کوین نیازی به استفاده از مشخصات خود ندارید و در نتیجه نام شما فاش نمی‌شود.

برای شروع استفاده از بیت کوین لازم است یک کیف پول مجازی داشته باشید.

کیف پول مجازی یک پایگاه داده‌ی کوچک است که با نصب برنامه‌ی بیت کوین بر روی کامپیوتر یا موبایل شما (local) قرار می‌گیرد.

در این حالت کیف پول مجازی شما آسیب پذیراست زیرا در صورت ویروسی شدن یا خرابی فیزیکی کامپیوتر یا موبایل تمام بیت کوین‌های شما از دست می‌روند.

همچنین می توان از طریق سایت‌های موجود، کیف پول مجازی را به صورت آنلاین ساخت.

در این شیوه هم باید به سازنده‌ی سایت و امنیت آن اعتماد داشته باشیم.

انتقال پول یا تراکنش

همانطور که گفتیم بیت کوین برای اداره‌ی خود به سیستمی مرکزی وابسته نیست و عملیات بیت کوین توسط همه ی کامپیوترهای عضو شده در شبکه ی بیت کوین (گره‌ها) انجام می‌شود.

یک لاگ فایل را در نظر بگیرید که شامل همه‌ی حساب‌های ساخته شده در بیت کوین و میزان موجودی آن هاست. یک کپی از این فایل بر هر کامپیوتر (یا گره) در شبکه‌ی بیت کوین نگهداری می‌شود.

برای فرستادن پول شما به شبکه اطلاع می‌دهید که میزان پول موجود در حساب شما باید پایین و پول موجود در حساب گیرنده باید بالا برود.

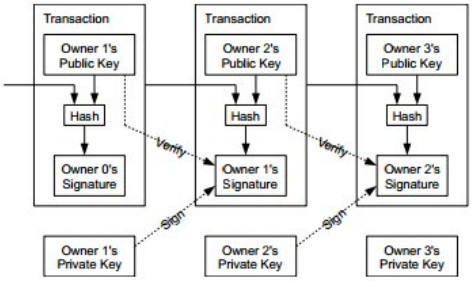
گره ها (یا کامپیوترها) این تراکنش را در کپی موجود از لاگ فایل خود اضافه می‌کنند و سپس تراکنش را به گره بعدی انتقال می‌دهند.

در این سیستم همه‌ی اعضای شبکه‌ی بیت کوین (هر کامپیوتر) اطلاعات حساب‌ها را نگه داری می‌کنند درحالی که در سیستم معمول تنها بانک حساب ها را نگه داری می‌کند. این موضوع باعث پیدایش تفاوت‌هایی میان دو سیستم شده است. مثلا برخلاف بانک که فقط درباره ی تراکنش‌های خود اطلاع دارید در بیت کوین همه درباره تراکنش‌های هر فرد مطلع‌اند.

در حالت اول شما به بانک خود اعتماد دارید و اگر اتفاقی بیفتد می‌توانید از آن شکایت کنید. اما در بیت کوین با گروهی غریبه روبرو هستید پس نمی‌توانید به کسی اعتماد کنید. البته سیستم بیت کوین توابع ریاضی خاصی دارد که سیستم را از هر جهت محافظت و شما را از اعتماد بی نیاز می‌کند.

در هر تراکنش دقیقا چه اتفاقی می افتد؟

فرض کنیم فردی به نام x می‌خواهد کالایی را



از فرد y بخرد و پول آن را با بیت کوین پرداخت کند.

وقتی x می‌خواهد به y پولی پرداخت کند پیغامی، میزان پول و نام حساب ها را به سیستم اطلاع می‌دهد.

پیام تراکنش: ۵ بیت کوین از x به y بفرست.

هر گره که این پیام را دریافت می‌کند کپی خود از لاگ فایل را به روز کرده و پیام تراکنش را انتقال می‌دهد.

سوالی که ممکن است ذهن شما را درگیر کند این است که هر گره یا کامپیوتر که پیام را دریافت می‌کند چگونه می‌تواند بفهمد پیام معتبر است و صاحب واقعی بیت کوین پیام را فرستاده است؟

بیت کوین نیازمند نوعی پسورد برای باز کردن حساب و خرج کردن پول است. به این پسورد امضای دیجیتال می‌گویند.

امضای دیجیتال همانند امضای معمولی معتبر بودن یک پیام را ثابت می کند ولی شیوه‌ی اثبات آن به وسیله ی یک الگوریتم ریاضی است که از تقلب در حوزه ی دیجیتال جلوگیری می‌کند.

یک امضای دیجیتال منحصربه فرد برای هر انتقال پول یا تراکنش لازم است.

این امضا از دو کلید متفاوت (ولی مرتبط) استفاده می‌کند. یکی از این کلیدها کلید خصوصی است که برای تولید امضا به کار می‌رود و بعدی کلید عمومی است که دیگران می‌توانند از آن استفاده کنند.

می‌توانید این گونه در نظر بگیرید که کلید خصوصی همان پسورد ماست و امضای دیجیتال که از آن ساخته می شود ثابت می‌کند شما پسورد را دارید بدون این که مجبور باشید آن را فاش کنید.

کلید های عمومی در حقیقت آدرس‌های ارسال هستند یعنی وقتی به کسی پول ارسال می‌کنید آن را به کلید عمومی او می‌فرستید.

وقتی می‌خواهید پولی را به کسی بفرستید ابتدا باید پول خود را خرج کنید. پول موجود در حساب شما هم قبلا توسط یک تراکنش به کلید عمومی آن تراکنش یعنی حساب شما فرستاده شده است.

برای خرج پول شما باید ثابت کنید که صاحب واقعی کلید عمومی‌ای هستید که به آن پول فرستاده شده و این کار را با ساخت یک امضای دیجیتال از پیام تراکنش و کلید خصوصی خود انجام می‌دهید. (به زبان ریاضی تابعی با دو ورودی پیام تراکنش و کلید خصوصی را داریم که به ما به عنوان خروجی، امضای دیجیتال می‌دهد).

بقیه ی گره‌ها در شبکه می‌توانند از آن امضا استفاده کنند تا بفهمند که با کلید عمومی شما متناظر است.

به وسیله‌ی ریاضیات موجود در امضای دیجیتال آن ها قادرند تایید کنند که فرستنده صاحب یک کلید خصوصی است (بدون دیدن آن کلید!).

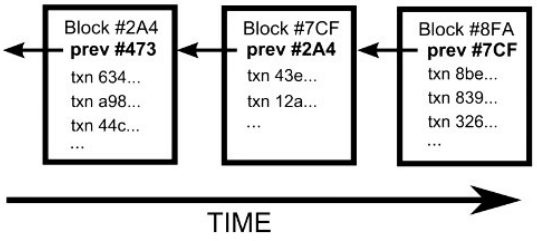
چون امضای دیجیتال به پیام تراکنش هم بستگی دارد برای هر تراکنش منحصر به فرد می شود. و به همین دلیل کسی نمی‌تواند دوباره برای تراکنش دیگری آن را استفاده کند. همین وابستگی به پیام تراکنش باعث می‌شود که اگر کسی بخواهد تقلب کند و پیام را عوض کند، منجر به نامعتبر شدن امضای دیجیتال شود.

تا اینجا می‌دانیم که امضای دیجیتال برای اینکه از معتبر بودن یک انتقال مطمئن شویم، استفاده می‌شود. اما تمام این توضیحات در واقع به شدت ساده شده بود. در واقعیت ماجرا هیچ لاگ فایلی از میزان پول هر حساب در سیستم نگه داری نمی‌شود!

اگر این که هر کس چقدر پول دارد را نگه نداریم چگونه بدانیم هر فرد پول کافی برای فرستادن به شخص دیگری را دارد؟

به جای نگه داشتن میزان پول هر شخص، مالکیت حساب ها به وسیله ی لینکی به تراکنش های قبلی تایید میشود.

برای اینکه x پنج بیت کوین به y ارسال کند فرد x باید به انتقال‌های قبلی که پنج بیت کوین را از طریق آن ها به دست آورده است مراجعه کند.



به این تراکنش‌های قبلی، ورودی گویند. بقیه ی گره‌ها برای تایید کردن یک تراکنش ورودی را بررسی می‌کنند که مطمئن شوند x گیرنده بوده و ۵ بیت کوین را دریافت کرده است.

با این حساب معتبر بودن هر تراکنش به تراکنش‌های قبلی بستگی دارد. ولی چطور می‌توان به تراکنش‌های قبلی اعتماد داشت؟ شما باید ورودی‌های تراکنش‌های قبلی را هم بررسی کنید. وقتی که برای اولین بار برنامه‌ی کیف پول مجازی را نصب می‌کنید هر تراکنش تا به حال انجام شده در سیستم را دانلود می‌کند و معتبر بودن هر کدام را تا تراکنش اول بررسی می‌کند. از آنجایی که با غریبه‌ها سر و کار دارید لازم است تا هر تراکنش را خودتان بررسی کنید. این عملیات می‌تواند یک شبانه روز طول بکشد اما فقط یکبار انجام می‌شود.

پس در واقع به جای لاگ فایل‌هایی در سراسر سیستم و بر روی هر گره (کامپیوتر) که میزان حساب ها را نگهداری کنند، گره های بیت کوین یک لیست بزرگ از تراکنش ها را نگهداری می کنند .

داشتن بیت کوین به این معنی است که تراکنش هایی در این لیست به اسم شما وجود دارد که خرج نشده است یا به عبارت دیگر به عنوان ورودی در انتقال‌های دیگر از آن استفاده نشده است.

نکته ی جالب این ساختار این است که برای دانستن این که در حساب خود چقدر بیت کوین دارید باید از میان هر تراکنشی که تاکنون انجام شده است حرکت کنید و همه‌ی ورودی‌ها به حساب شما که خرج نشده‌اند را جمع کنید.

مشکل دو بارخرج کردن

تاکنون دو مشکل ایجاد شده در سیستم غیرمتمرکز بیت کوین را شرح داده و راه حل آن را گفته‌ایم.

با استفاده از امضای دیجیتال می‌دانیم که تنها صاحب واقعی یک حساب می‌تواند پیام تراکنش را ایجاد کرده باشد.

برای این که مطمئن شویم فرستنده پولی برای فرستادن دارد، هر ورودی (تراکنش های قبلی) را بررسی می‌کنیم و مطمئن می شویم خرج نشده باشد.

اما همچنان یک مسأله در سیستم وجود دارد و آن ترتیب تراکنش‌ها است.

با توجه به اینکه تراکنش‌ها گره به گره در شبکه‌ی بیت کوین انتقال داده می شوند پس ضمانتی برای اینکه باهمان ترتیبی که تولید شده‌اند به شما برسند وجود ندارد. شیوه ای نیست که بتوان گفت یک تراکنش قبل از تراکنش دیگری انجام شده است. این موضوع راه را برای سوءاستفاده‌ی سودجویان باز می‌گذارد.

به عنوان مثال کاربر X می‌خواهد از y کالایی را بخرد. کاربر x یک تراکنش انجام می‌دهد و پول کالا را برای y میفرستد. سپس منتظر می‌ماند تا y کالای سفارش داده شده را برای او بفرستد. بعد از آن یک تراکنش دیگر را با استفاده از همان ورودی که برای y فرستاد، برای خودش می‌فرستد.

بعضی از گره‌ها در شبکه به دلیل اختلافات زمانی، دومین تراکنش (ارسال پول x به خودش) را قبل از تراکنش اول دریافت کرده و وقتی تراکنش واریز پول به y می‌رسد آن را نادرست می‌شمارند، زیرا این تراکنش تلاش می‌کند از همان ورودی دوباره استفاده کند.

در این مثال y هم پول و هم کالای خود را از دست داده است.

در شبکه بین گره‌ها درباره ی اینکه کدامشان پول را دارند اختلاف وجود دارد زیرا راهی نیست که ثابت کنند کدام تراکنش اول آمده است.

به دلیل مشکل مطرح شده باید راهی باشد که همه ی گره‌ها ترتیب تراکنش‌ها را بدانند و با یک ترتیب موافقت کنند!

راه حل سیستم بیت کوین راهی هوشمندانه بود که در ادامه می‌بینیم.

زنجیره ی بلاک‌ها : ترتیب تراکنش‌ها

بیت کوین ترتیب تراکنش‌ها را با قراردادنشان در گروه هایی به نام بلاک تنظیم می‌کند و این بلاک ها را به هم لینک می‌کند که به آن زنجیره‌ی بلاک‌ها می‌گویند.

هر بلاک به بلاک قبلی خود اشاره می‌کند. اگر بلاک ها را به عقب برگردیم، همه ی تراکنش ها را تا اولین تراکنش های سیستم می‌بینیم.

هر بلاک می تواند چندین تراکنش را در خود داشته باشد.
تراکنش‌های موجود در یک بلاک در یک زمان اتفاق افتاده‌اند. و تراکنش هایی که در بلاکی قرار ندارند هنوز تایید نشده‌اند.
هر گره می‌تواند تعدادی از این تراکنش‌های تایید نشده را در یک بلاک جمع کند و آن را به عنوان بلاک بعدی در زنجیره‌ی بلاک‌ها، به شبکه پیشنهاد دهد.
از آن جایی که گره‌های مختلف هر کدام می‌توانند پیشنهادهای متفاوتی ایجاد کنند انتخاب‌های متفاوتی برای بلاک بعدی وجود دارد.
پس شبکه چطور تصمیم می‌گیرد؟
برای فهمیدن راه حل بیت کوین باید ابتدا با مفهوم هش به زبان ساده آشنا شویم.
یک تابع هش در بیت کوین تابعی برگشت ناپذیر است که یک رشته را می‌گیرد و آن را به یک عدد یا رشته منحصر به فرد تبدیل می‌کند.
مثلا تابع هشی که بیت کوین استفاده میکند (SHA256) به صورت زیر کار میکند:

SHA256("bitcoin")= 6b88c087247aa2f07ee1c5956b8e1a9f4c7f892a70e324f1bb3d161e05ca107b

با اندکی تغییر دادن رشته ورودی، خروجی تماماً تغییر می‌کند و قابل پیش بینی نیست. فرض کنید می‌خواهیم یک خروجی خاص را ایجاد کنیم. تنها راه پیدا کردن یک مقدار خاص به عنوان خروجی این است که برای رشته‌ی ورودی حدس‌های تصادفی بزنیم.

بیت کوین چگونه از این تابع هش استفاده کرد؟

برای اینکه بلاک جدیدی توسط تمام کامپیوترهای شبکه تایید شود نیازمند مشارکت همه‌ی آن ها هستیم.
در زنجیره ی بلاک‌ها آخرین بلاک همواره بر سر زنجیره در شبکه قرار دارد.
اگر می‌خواهیم بلاک جدیدی معتبر شناخته شود کامپیوتری بر روی شبکه باید پیام تراکنشی برای آن ایجاد کند که به بلاک قبلی تایید شده متصل شود.
برای جلوگیری از پذیرفته شدن بلاک‌های ساختگی باید مهر تایید هر بلاک برای هر کاربر مستقل بر روی شبکه مشکل باشد ولی برای کل شبکه راحت انجام شود. هر کامپیوتر به تنهایی برای حل کردن یک بلاک باید چندین سال حدس بزند. به همین دلیل برای دور زدن این سیستم به توان کامپیوتری حداقل نصف شبکه نیاز داریم که عملا غیر ممکن است.
برای حل بلاک بعدی، کل شبکه سعی می‌کند به وسیله‌ی اضافه کردن تراکنش های جدید به آخرین بلاک (که در واقع سر تمام بلاک های پیشین است) یک خروجی از تابع هش در محدوده‌ای خاص و از پیش تعیین شده تولید کند.
با وجود شبکه‌ی بیت کوین و همه ی کامپیوترها در آن که در حال حدس زدن هستند به طور میانگین ده دقیقه طول میکشد که کسی یک جواب پیدا کند.
اولین کسی که بلاک را حل کند، بلاک پیشنهادی‌اش را اعلام میکند و می‌تواند آن را به عنوان بلاک بعدی در زنجیره ی بلاک‌ها قرار دهد.

استخراج پول

بیت کوین ها از کجا می‌آیند؟ برای فرستادن پول شما به یک تراکنش قبلی که در آن گیرنده بودید مراجعه می‌کنید ولی اولین بیت کوین‌ها چگونه وارد سیستم شده اند؟
تولید بیت کوین ها شیوه ای آرام و تصادفی دارد. برای ایجاد بیت کوین به هر کس که با استفاده از توان کامپیوتری خود یک بلاک را حل می‌کند، جایزه داده می‌شود. برای همین به حل کردن بلاک‌ها استخراج می‌گوییم.
سیستم بیت کوین را مانند یک معدن طلا در نظر بگیرید. در ابتدا این جایزه ۵۰ سکه بیت کوین بود.
یعنی به کسانی که با گذاشتن توان کامپیوتری یک بلاک را حل می کردند ۵۰ سکه بیت کوین تعلق می‌گرفت و بدین صورت پول به سیستم وارد می‌شد.
در سال ۲۰۱۳ این مقدار نصف شده و ۲۵ سکه بود. (همانند یک معدن طلا که طلا های اصلی آن کشف شده و به دست آوردن طلا های جدید سخت تر شده است) در سال ۲۰۱۷ ، ۱۲.۵ سکه و به تدریج این مقدار انقدر نصف می‌شود که میزان ورود سکه به سیستم تقریبا صفر شود.
در نهایت در تمام شبکه ۲۱ میلیون بیت کوین وجود خواهد داشت که بین مردم منتقل می‌شوند و سکه ی جدیدی تولید نخواهد شد.
ناکوموتو انگیزه ی خود را از ساخت بیت کوین خشم از وضعیت اقتصادی و دخالت دولت‌ها در آن اعلام کرده بود. اما بیت کوین به عنوان پولی مستقل و غیرقابل ردیابی، همانطور که دست دولت ها را از کنترل پرداخت ها کوتاه کرده است، راه را برای انجام معاملات غیرقانونی هم باز گذاشته است.
بیت کوین هسته ی اصلی سایت Silk Road بود که یک بازار سیاه برای خرید اقلام غیرقانونی به شمار می‌رفت.
بسیاری از مردم نمی‌توانند به بیت کوین اعتماد کنند چون توسط یک سیستم مرکزی کنترل نمی‌شود و تماما وابسته به کاربرانش است.
اما فراموش نکنیم با توجه به مزایای گفته شده، بیت کوین انقلابی در پول های دیجیتال است.
پولی که مستقل از حکومت و دولت توسط مردم دنیا اداره می شود.

مقدمه ای بر MongoDB

حسین ریماز | تایید علمی: دکتر حسین خواسته و دکتر سعید فرضی

در این مقاله مقدمه‌ای از پایگاه داده MongoDB و نحوه نصب و کار با آن خواهیم گفت. اما قبل از آنکه بتوانیم از MongoDB حرف بزنیم نیاز داریم که بعضی از مفاهیم را مرور کنیم تا به درک بهتری از کارکرد این پایگاه داده برسیم.

مقدمه

تاریخچه ظهور پایگاه‌های داده، روند پیشرفت آن و ظهور و افول تکنولوژی ها و استانداردهای مربوط به آن مفصل‌تر از آن است که بتوان در یک مقاله همه آن‌ها را تعریف کرد. اما شاید بتوان تاریخ تکامل پایگاه‌های داده را به سه نسل زیر تقسیم کرد:

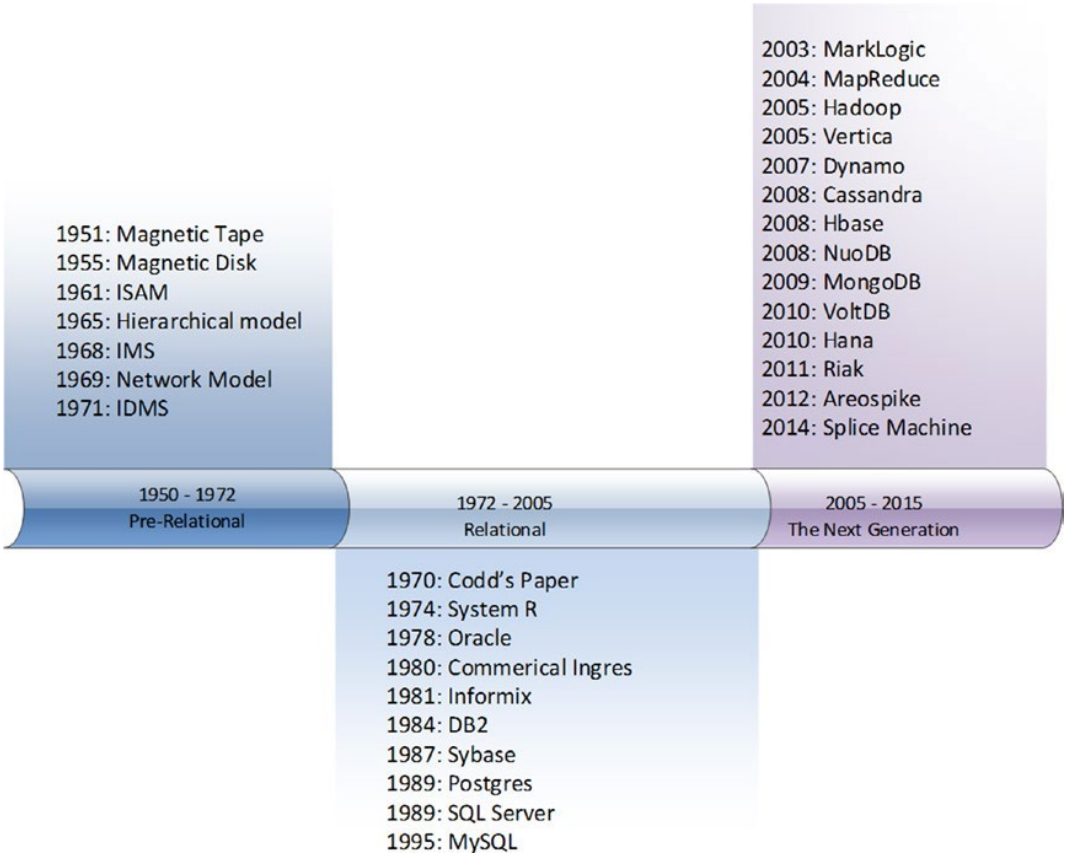


Illustration 1: Generations of DBMSs [1]

نسل های ابتدایی دیتابیس‌ها صرفاً سیستم‌های ساده‌ای برای ذخیره و بازیابی داده‌ها بودند اما با معرفی سیستم پایگاه داده رابطه‌ای (RDBMS) و مدل رابطه ای در سال ۱۹۷۰ توسط ادگار کاد (Edgor Codd) وارد نسل دوم پایگاه‌های داده شدیم. مدل رابطه‌ای در این دوره به خوبی می‌توانست داده‌ها را به شکل سطرها و ستون‌ها و در غالب جداولی مدل کند. اعمال مدیریتی و پرس و جو ها نیز توسط زبان SQL (Structured Query Language) انجام می‌شود.

اما با ظهور اینترنت، چالش های زیادی هم از لحاظ مقیاس پذیری برنامه‌هایی که خصوصا تحت وب اجرا می‌شد و هم از لحاظ مدل سازی داده‌های پیچیده‌ای که با ظهور اینترنت به وجود آمده بودند پیش روی برنامه نویسان و محققان قرار گرفت.

نتیجه این چالش‌ها نسل سومی از پایگاه‌های داده، موسوم به پایگاه‌های داده غیر رابطه‌ای ظهور کردند.

NoSQL (Not only SQL) تعریف دقیقی ندارد اما اشاره‌اش به پایگاه های داده غیر رابطه ایست و بیان کننده این است که تنها پایگاه‌های رابطه‌ای نیستند که می‌توان از آن برای ذخیره سازی و مدیریت داده‌ها استفاده کرد. درواقع هر کدام از این دیتابیس‌ها به گونه خاص خود داده‌ها را مدل سازی و مدیریت می‌کنند که این امکان به برنامه نویسان داده می‌شود که مناسب‌ترین پایگاه داده را متناسب با نوع کاربرد خود انتخاب کنند.

در پایگاه های داده رابطه‌ای تراکنش ها باید ۴ خصوصیت زیر را داشته باشند:

- **Atomic:** به معنی اینکه هر تراکنش یا باید کامل اجرا شود یا اصلاً اجرا نشود.
- **Consistent:** هر تراکنش پایگاه داده را از یک وضعیت سالم و معتبر به یک وضعیت سالم و معتبر دیگر می برد.
- **Isolated:** تراکنش هایی که بر روی یک دیتاست اجرا می‌شوند از یک دیگر مستقل هستند. بنابراین کار یک تراکنش بر روی دیگر تراکنش تداخلی ایجاد نمی‌کند و اجرای همزمان تراکنش‌ها مشکلی برای پایگاه داده ایجاد نمی‌کند.
- **Durable:** یعنی تغییرات اعمال شده در سیستم ماندگار خواهد بود و در صورت بروز هرگونه خطا و اشکال از بین نخواهد رفت.

این ۴ خصوصیت به اختصار ACID نامیده می‌شود که تمام پایگاه های داده رابطه‌ای بر مبنای آن، تراکنش ها را انجام می‌دهند.

اما همین خصوصیات بود که اشکالات متعددی برای مقیاس پذیری و کارایی دیتابیس‌های رابطه‌ای در اپلیکیشن‌های عظیم تحت وب ایجاد می‌کرد.

اما چرا؟

CAP Theorem

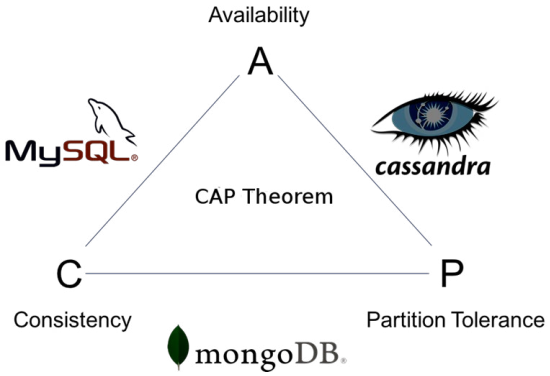
این نظریه در سال ۲۰۰۰ توسط Eric Brewer مطرح شد. درک این نظریه اساسی‌ترین موضوع برای

توسعه دهندگان و معماران برای مواجهه با سیستم‌های توزیع شده است. این نظریه می‌گوید در طراحی یک برنامه‌ای که در محیطی توزیع شده اجرا می‌شود ۳ شرط مورد نیاز است:

• **Consistency:** یعنی اطلاعات بعد از اعمال عملیات ها و تغییرات وضعیت صحیحی داشته باشند و تمام کاربران تغییر اعمال شده را مشاهده کنند.

• **Availability:** یعنی سیستم همواره در دسترس باشد.

• **Partition Tolerance:** یعنی سیستم همواره بتواند به کار خود ادامه دهد، حتی اگر به بخش‌های مجزایی تقسیم شده باشد که آن بخش‌ها نتوانند باهم ارتباط برقرار کنند. به عبارتی دیگر سیستم بتواند به صورت توزیع شده در شبکه ای از کامپیوترها کار کند و با ازدسترس خارج شدن بعضی از این کامپیوترها، سیستم هنوز بتواند به درستی کار کند. بر اساس این نظریه، یک سیستم توزیع شده تنها می‌تواند دو شرط از سه شرط فوق را برآورده کند.



BASE

پایگاه های داده رابطه‌ای بر مبنای ACID بودند، اما در مقابل NoSQL دیتابیس ها بجای ACID بودن خصوصیات دیگری موسوم به BASE را انتخاب کردند. این خصوصیات عبارت اند از:

• **Basically Available:** به این معنا که سیستم با همان تعریفی که در CAP گفته شد دردسترس است.

• **Soft State:** بیانگر این است که اگر حتی هیچ تغییری و ورودی‌ای به سیستم اعمال نشود، حالت سیستم در طول زمان تغییر می‌کند. این ویژگی متناسب با قابلیت Eventual Consistency است.

• **Eventual Consistency:** یعنی سیستم در نهایت به وضعیت صحیح خود خواهد رفت، و حتی هنگامی که هیچ ورودی ای به سیستم اعمال نمی‌شود، سیستم می‌تواند در حال گذار به وضعیت صحیح باشد.

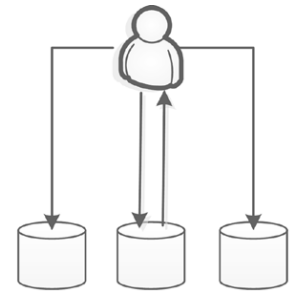
اما دقیقاً مشکل کجا بود؟

دنیای سخت گیرانه ACID با دنیای خطا پذیر BASE باهم کاملا تفاوت دارند. اکثر سیستم های کنونی توزیع شدگی به واسطه اینترنت دارند (مانند بسیاری از سایت های بزرگ دنیا مثل توییتر و آمازون). در بسیاری از این سیستم ها سازگاری لحظه‌ای چندان مهم نیست و چیزی که مهم است، دردسترس بودن و سرعت پاسخگویی سایت است. سایتی مثل توییتر را تصور کنید، این سایت به صورت توزیع شده اطلاعات خود را در سرورهای مختلف در نقاط مختلف جغرافیایی ذخیره می‌کند. کاربری از آمریکا توییتی ارسال می کند. اگر بخواهیم با همان مدل‌های قدیمی و رابطه‌ای کار کنیم باید منتظر بمانیم تا تغییرات در تمام دیتابیس‌ها اعمال شود! این می‌تواند منجر به کاهش سرعت و مقیاس پذیری سیستم ما شود. اما در حقیقت اصلاً نیازی نیست که کاربری که در ایران است همان چیزی را ببیند که کاربری در اروپا مشاهده می‌کند. اما در نهایت جفت آن‌ها دیر یا زود توییت ارسال شده کاربر آمریکایی را خواهند دید. این مفهوم Eventual Consistency است.

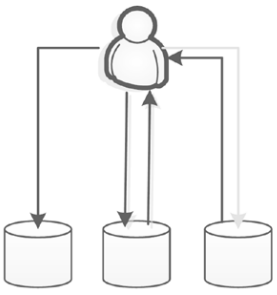
اکثر NoSQL دیتابیس ها از Eventual Consistency پیروی می‌کنند یا این امکان را به طراح می‌دهند که آن را انتخاب کند، اما نحوه پیاده‌سازی آن در هر NoSQL به صورت خاصی است.

از سه حرف NRW برای تشریح مدل Eventual Consistency ای که NoSQL دیتابیس پیاده‌سازی کرده استفاده می کنند.

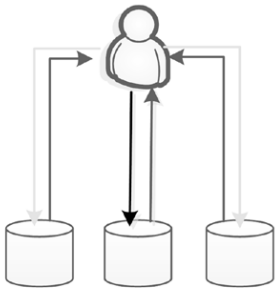
N عددی است که تعداد کپی‌هایی که دیتابیس از داده اصلی نگه می‌دارد را نشان می‌دهد.



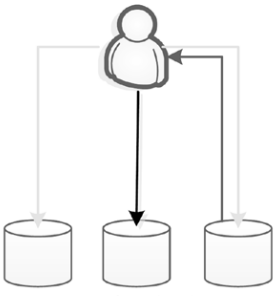
N=3 W=3 R=1
Slow writes, fast reads, consistent
There will be 3 copies of the data.
A write request only returns when all 3 have written to disk.
A read request only needs to read one version.



N=3 W=2 R=2
Faster writes, still consistent (quorum assembly)
There will be 3 copies of the data.
A write request returns when 2 copies are written – the other can happen later.
A read request reads 2 copies make sure it has the latest version.



N=3 W=1 R=N
Fastest write, slow but consistent reads
There will be 3 copies of the data.
A write request returns once the first copy is written – the other 2 can happen later.
A read request reads all copies to make sure it gets the latest version.
Data might be lost if a node fails before the second write.



N=3 W=1 R=1
Fast, but not consistent
There will be 3 copies of the data.
A write request returns once the first copy is written – the other 2 can happen later.
A read request reads a single version only: it might not get the latest copy.
Data might be lost if a node fails before the second write.

Illustration 2: NRW Configuration

R تعداد کپی هایی از داده است که اپلیکیشن نیاز دارد به آن‌ها استناد کند تا پاسخ یک عملیات خواندن را بدهد.

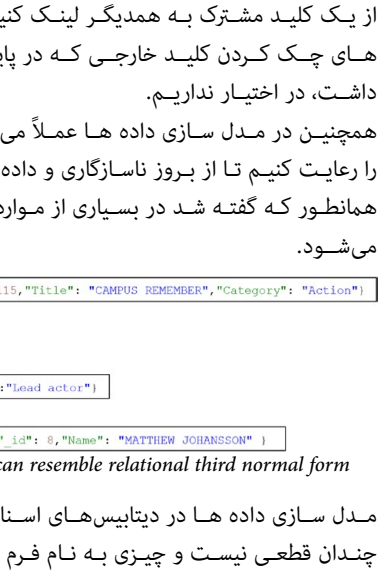
W تعداد کپی هایی از داده است که نیاز است به طور موفقیت آمیز نوشته شود تا آن عملیات موفق آمیز تلقی شود.

درک این سه پارامتر و تأثیری که بر کارایی دیتابیس می‌گذارند مساله‌ای بسیارمهم است.

یکپارچگی

(Consistency) می‌تواند در دو سطح خواندن و نوشتن پیاده‌سازی شود.

NoSQL دیتابیس های انواع مختلفی دارند که هر کدام متناسب با کاربرد خاصی استفاده می‌شوند و عبارتند است: Graph Store , Key-Value Store , Document Store, Column Store.



پایگاه های داده اسنادی

معمولاً پایگاه های داده اسنادی در فرمت XML یا JSON فایل‌ها را ذخیره می‌کنند که استفاده از JSON محبوب‌تر است. MongoDB و CouchDB معروف‌ترین پایگاه‌های داده اسنادی هستند. در پایگاه‌های داده اسنادی که از فرمت JSON استفاده می‌کنند، سلسه مراتب مدل ذخیره سازی معمولاً به شکل زیر است:

Document: یک داکیومنت معادل یک سطر در پایگاه داده رابطه‌ای است و واحد ذخیره سازی اطلاعات است. یک داکیومنت شامل چندین key-value هایی است که می‌تواند شامل داکیومنت‌ها و آرایه های تو در تو از داکیومنت ها باشد. که چنین ساختاری اجازه مدل سازی روابط پیچیده را به ما خواهد داد.

Collection: یک کالکشن مجموعه‌ای از داکیومنت هاست که معمولاً وجه اشتراکی بین این داکیومنت‌ها وجود دارد که تقریباً می‌توان گفت مشابه جداول در پایگاه های داده رابطه‌ای هستند. اما هیچ نیازی نیست که تمام داکیومنت‌های موجود در یک کالکشن از یک نوع و به یک فرمت باشند. اما معمولاً داکیومنت‌هایی که بیانگر اطلاعات یکسانی است را در یک کالکشن نگه‌داری می کنند.

با وجود اینکه می‌توان سطوح نرمال سازی را در مدل سازی داده‌ها در این روش پیاده‌سازی کرد اما معمولاً نیازی به نرمال سازی نیست و حتی در مواردی داده‌ها را نرمال نمی‌کنیم و آن‌ها را به صورت داکیومنت های تو در تو نگه داری می کنیم (Denormalization).

به عنوان مثال پایگاه داده فیلم‌ها و بازیگران را در نظر بگیرید. در مدل رابطه‌ای اطلاعات را در جداول جدا از هم نگه داری می‌کنیم و با استفاده از یک join table اطلاعات را به هم متصل می‌کنیم.

اما در مدل سازی JSON Document می‌توانیم با key-value ها این رابطه را

Illustration 4: JSON Movie Collection [۳]

در مدل سازی فوق از الگویی به نام Document Embedding استفاده کردیم و با ترکیب داکیومنت ها، آرایه از بازیگران را درون یک فیلم جا دادیم. اما در بعضی موارد انجام این عمل ممکن نیست. در اینصورت میتوانیم در کالکشنی جدا اطلاعات بازیگران را لینک کنیم. به عنوان مثال در MongoDB به صورت پیش‌فرض حجم یک داکیومنت نمی‌تواند بیشتر از ٦٤ مگابایت باشد.

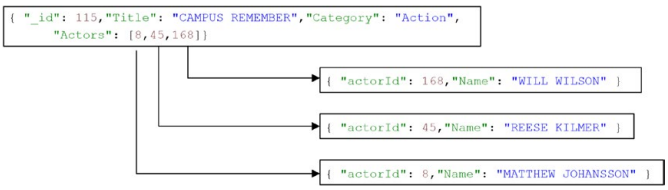


Illustration 5: Linking document in a document database

به همین دلیل ممکن است تصمیم بگیریم که داکیومنت‌ها را با استفاده از یک کلید مشترک به همدیگر لینک کنیم اما باید توجه کنیم مکانیزم های چک کردن کلید خارجی که در پایگاه های داده رابطه‌ای وجود داشت، در اختیار نداریم.

همچنین در مدل سازی داده ها عملاً می‌توانیم سطح سوم نرمال سازی را رعایت کنیم تا از بروز ناسازگاری و داده‌های تکراری جلوگیری کنیم. اما همانطور که گفته شد در بسیاری از موارد از نرمال سازی داده‌ها اجتناب می‌شود.

Illustration 6: Document linking can resemble relational third normal form

مدل سازی داده ها در دیتابیس‌های اسنادی مانند دیتابیس‌های رابطه‌ای چندان قطعی نیست و چیزی به نام فرم نرمال و استاندارد وجود ندارد و نمی‌توان مدل سازی‌ای را درست یا غلط خواند. همچنین در دیتابیس‌ها رابطه‌ای، مدل سازی داده‌ها با توجه به ماهیت داده‌ها انجام می‌شود اما در دیتابیس های اسنادی مهم‌ترین نکته، در نظر گرفتن نحوه دسترسی به داده‌ها (Data Access Pattern) است.

به عنوان مثال فرض کنید که همیشه با نشان دادن یک فیلم، لیست بازیگرها هم به همراه آن نمایش می‌دهیم. اگر اینچنین باشد، شاید بهتر باشد که با قرار دادن لیست بازیگران درون داکیومنت اطلاعات فیلم، باعث تسریع روند بازیابی داده مربوطه و در نتیجه باعث افزایش سرعت پاسخگویی سیستم شویم. در غیر اینصورت می‌بایستی برای پیدا کردن اطلاعات بازیگر به صورت جدا، زمان بیشتری صرف کنیم.

تـا اینجـا سـعی کردیـم کـه مفاهیـم مـورد نیـاز بـه صـورت سـطحی و گـذرا پوشـش داده شـوند تـا درک صحیحـی از MongoDB داشـته باشـیم. حـالا وقـت ایـن رسـیده کـه کمـی بـا MongoDB کار کنیـم.

SQL and MongoDB Terminology

SQL	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field
Index	Index
Joins wihin Table	Embedding and referencing
Primary Key: A column or set of columns can be specified	Primary Key: Automatically set to _id field

نصب و راه اندازی

MongoDB را می‌توانیـد در اکـثر پلتفـرم هـا و سیسـتم عامـل هـا راه انـدازی کنیـد. مراحـل نصـب بـرای هـر سیسـتم عامـل بـه صـورت کامـل در سـایت MongoDB توضیـح داده شـده‌اند کـه بـا دنبـال کـردن مراحـل بـه سـادگی خواهیـد توانسـت MongoDB را نصـب کنیـد.[4]

بـرای راه انـدازی MongoDB کافـی اسـت کـه بـا وارد کـردن کامنـد mongod (در ویندوز اجـرای فایـل mongod.exe در پوشـه bin مسـیر نصـب شـده MongoDB) را اجـرا کنیـد. بـا انجـام اینـکار، دیتابیـس بـه صـورت پیش‌فـرض بـر روی پـورت ۲۷۰۱۷ راه انـدازی می‌شـود.

بعـد از نصـب و راه انـدازی شـما نیـاز داریـد کـه از طریـق کنسـول، یـا یـک برنامـه نوشـته شـده بـه دیتابیـس متصـل شـوید و اعـمال خـود را انجـام دهیـد.

CRUD Operations

بـرای متصـل شـدن بـه MongoDB از طریـق کنسـول، دسـتور mongo (در ویندوز فایـل mongo.exe در پوشـه bin محـل نصـب شـده MongoDB) را اجـرا کنیـد.

```
$ mongo
MongoDB shell version: 3.2.11
```

حـال شـما وارد MongoDB Shell شـده‌اید و می‌توانیـد بـا دیتابیـس ارتبـاط برقـرار کنید.بـرای مشـاهده دیتابیـس هـای موجـود بـا وارد کـردن دسـتور show dbs می‌توانیـد لیسـت تمـام دیتابیـس هـای موجـود را مشـاهده کنیـد.

```
> show dbs
db      0.000GB
test    0.000GB
```

بـرای ادامـه کار بایـد مشـخص کنیـم کـه بـا کـدام دیتابیـس کار خواهیـم کـرد و از دسـتور use dbname بـرای کار بـا دیتابیـس مـورد نظـر خـود اسـتفاده مـی کنیـم.

```
> use test
switched to db test
```

حـال می‌توانیـم ببینیـم کـه چـه کالکشـن هایـی بـرروی دیتابیـس مـورد نظـر مـا ذخیـره شـده. بـرای اینـکار از دسـتور show collections اسـتفاده مـی کنیـم.

```
> show collections
sample
```

در دیتابیـس مـا کـه نـام آن test اسـت، یـک کالکشـن بـه نـام sample ذخیـره مـی شـود.

بـرای وارد کـردن یـک داده در درون ایـن کالکشـن کافـی اسـت کـه دسـتور زیـر را وارد کنیـم.

```
> db.sample.insert({name:"Hossein", family:"Rimaz"})
WriteResult({ "nInserted" : 1 })
```

درواقـع فرمـت ایـن کوئـری بـه شـکل db.collection.insert() اسـت. در صورتـی کـه کالکشـن معیـن شـده وجـود نداشـته باشـد، دیتابیـس بـه صـورت خـودکار آنـرا ایجـاد خواهـد کـرد.

```
> db.users.insert({name: "Hossein",age:21,status: "Student"})
WriteResult({ "nInserted" : 1 })
> db.users.find().pretty()
{
        "_id" : ObjectId("588cbebb486af00acfa9d1cb"),
        "name" : "Hossein",
        "age" : 21,
        "status" : "Student"
}
```

همانطـور کـه دیدیـم داده جدیـد بـه صـورت موفـق در کالکشـن users وارد شـد. بـرای بازیابـی داده‌هـای وارد شـده از دسـتور db.collection.find می‌توانیـم اسـتفاده کنیـم. همچنیـن می‌توانیـم شروط خاصـی را تعریـف کنیـم و فیلـد هـای خاصـی را از آن خـارج کنیـم (Projection).

```
> db.users.find({age : {$lt:24}},{name:1}).limit(10)
{ "_id" : ObjectId("588cbebb486af00acfa9d1cb"), "name" : "Hossein" }
```

کوئری فوق فقط نام ۱۰ نفر از افرادی که زیر ۲۴ سال دارند را نمایش می دهد.

نکتـه‌ای کـه هنـوز بـه آن اشـاره نکرده‌ایـم فیلـد id_ اسـت. ایـن فیلـد در صورتـی کـه توسـط کاربـر مشـخص نشـود بـه صـورت خـودکار تولیـد می‌شـود و منحـصر بـه فـرد بـودن داده وارد شـده را نشـان می‌دهـد و مثـل شـماره شناسـنامه‌ای بـرای آن داکیومنـت اسـت.

بـرای آپدیـت کـردن داکیومنت‌هـا می‌توانیـد از دسـتور db.collection.update اسـتفاده کنیـم.

```
> db.users.update({age: {$lt:24}},{$set:{ University: "KNTU"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.users.find().pretty()
{
        "_id" : ObjectId("588cbebb486af00acfa9d1cb"),
        "name" : "Hossein",
        "age" : 21,
        "status" : "Student",
        "University" : "KNTU"
}
```

همانطـوری کـه می‌بینیـم کوئـری فـوق افـرادی کـه سـن آن‌هـا زیـر ۲۴ سـال اسـت بـه عنـوان دانشـجوی خواجـه نصیـر در نظـر مـی گیـرد. خالـی از لطـف نیسـت کـه بگوییـم در یـک کالکشـن لزومـی نـدارد تمـام داکیومنـت هـا فیلـد هایـی یکسـان و از یـک جنـس داشـته باشـند. مثلاً پـس از اجـرا کـردن ایـن کوئـری بسـیاری از افـراد فیلـد University را نخواهنـد داشـت.

برای حذف داکیومنت ها نیز از دستور db.collection.remove استفاده می کنیم.

```
> db.users.remove({status: "Student"})
WriteResult({ "nRemoved" : 1 })
> db.users.count()
0
```

با اجرا کردن این دستور تنها داکیومنتی که در کالکشن بود پاک می‌شود.

تـا اینجـای کار بـا دسـتورات پایـه ای CRUD آشـنا شـدیم. البتـه هـر یـک از ایـن دسـتورات پارامترهـای دیگـری نیـز ممکـن اسـت دریافـت کنـد کـه بـرای مطالعـه بیشـتر آن‌هـا می‌توانیـد بـه داکیومنت‌هـای MongoDB مراجعـه کنیـد و خودتـان ایـن دسـتورات را امتحـان کنیـد[5].

مسیر پیش رو

تـا اینجـا بـا مفاهیـم پایـه‌ای دیتابیـس هـای غیـر رابطـه‌ای آشـنا شـدیم و MongoDB کـه جـزو مهم‌تریـن دیتابیس‌هـای اسـنادی اسـت را بررسـی کردیـم و بـا نحـوه راه

انـدازی آن آشـنا شـدیم و در چنـد مثـال اعـمال مدیریتـی سـاده را انجـام دادیـم.
اما داستان به اینجا ختم نمی‌شود ومفاهیم پیشرفته بسیاری وجود دارد که بررسی آن‌ها در یک مقاله ممکن نیست.
بـرای مطالعـه بیشـتر می‌توانیـد بـه داکیومنت‌هـای خـود سـایت MongoDB مراجعـه کنیـد. همچنیـن می‌توانیـد دوره هـای ویدئویـی کـه در سـایت[۷] ایـن دیتابیـس وجـود دارد بگذرانیـد. همچنیـن می‌توانیـد بـه کتاب‌هـای زیـر بـرای ادامـه مطالعـات خـود مراجعـه کنیـد:

1- Plugge, E., Hows, D., Membrey, P., Hawkins, T. The definitive guide to MongoDB (2015). CA: Apress.

2- Banker, K., Bakkum, P., Verch, S., Garrett, D., and Hawkins T. MongoDB in Action, Second Edition (2016). NY: Manning.

مراجع:

1- Harrison, Guy (2015). Next Generation Databases. CA: Apress. 4.

2- Harrison, Guy (2015). Next Generation Databases. CA: Apress. 50.

3- Harrison, Guy (2015). Next Generation Databases. CA: Apress. 59.

4- https://docs.mongodb.com/manual/installation/

5- https://docs.mongodb.com/manual/crud/

6- Edward, S.G., Sabharwal, N. (2015). Practical MongoDB. CA: Apress 13-18.

7- https://university.mongodb.com/

جاوا ۹ و JShell

حسین ریماز

ورژن جدید جاوا یعنی جاوا ۹ قرار بود که امسال در کنفرانس JavaOne امسال معرفی شود و به صورت رسمی منتشر شود اما بخاطر بررسی بیشتر پروژه Jigsaw جاوا ۹ باز هم قرار است با تأخیر منتشر شود. اما تقریباً بقیه قابلیت‌های جاوا ۹ نهایی شده‌اند و امکانات متعدد و API های جدیدی به همراه بهینه سازی‌های فراوانی که در جاوا ۹ اضافه شده‌اند هم‌اکنون به صورت آزمایشی در اختیار علاقه‌مندان قرار گرفته است.

Jshell یکی از این امکانات جدید است که در جاوا ۹ معرفی شده و قرار است در این مقاله این ابزار کاربردی جدید را بررسی کنیم و سپس نشان دهیم چگونه می‌توانیم با استفاده از کتابخانه JSoup یک صفحه وب را تحلیل یا اصطلاحاً Scrape کنیم. هرچند که قرار نیست به اصلاً کار با کتابخانه JSoup را به شما نشان دهیم اما برای جذاب‌تر کردن کار و نشان دادن قدرت جدیدی که در اختیار ما قرار گرفته لیست ۲۵۰ فیلم برتر IMDb را استخراج خواهیم کرد.

Jshell اصطلاحاً یک REPL است. درواقع یک ابزار تحت کنسول است که به طور مداوم و در حلقه‌ای بی پایان دستورات وارد شده را می‌خواند و آن‌ها را ارزیابی می‌کند و نتایج این ارزیابی ها را به ما نشان می‌دهد و در نهایت دوباره اینکار را تکرار می کند. چنین مفهومی پیش از این در بسیاری از زبان‌ها مانند Python , Groovy و Scala و دیگر زبان‌ها وجود داشته و این امکان را به توسعه‌دهنده می‌دهد تا بدون باز کردن یک IDE و ساختن پروژه و سختی‌های این چنینی مستقیماً بتواند کد های خود را در یک محیط تعاملی در درون کنسول بنویسد و با سرعت بیشتری بتواند این کدها را اجرا و تست کند.

برای بهره مندی از این قابلیت جدید شما نیاز به این دارید که JDK ۹ را دانلود کنید برای اینکار می‌توانید به سایت[1] OPEN JDK مراجعه کنید و همانطوری که جاوا را قبلاً نصب نموده‌اید، جاوا ۹ را هم به همان شکل نصب کنید.

Hello World Jshell

برای اینکه بتوانیم وارد jShell بشویم باید کامند jshell را در کنسول اجرا کنیم. حال می‌توانیم بدون هیچ دردسری به دنیا سلام کنیم!

```
jshell> System.out.println(“Hello JShell World”)
Hello JShell World
```

به همین سادگی به دنیا سلام کردیم!

عبارات کنترلی

برای شروع احتمالاً فیبوناچی نوشتن شروع کلیشه ای خوبی باشد! برای اینکار باید یک آرایه تعریف کنیم:

```
jshell> int[] a = new int[10]
a ==> [I@7d9d1a19
```

حال که آرایه را ساخته‌ایم کافیست دو مقدار اولیه آن را مقدار دهی کنیم و به همان روش Dynamic Programming شروع به پر کردن خانه‌های بعدی این آرایه کنیم تا سری فیبوناچی را تشکیل دهیم.

```
jshell> a[1] = 1
$5 ==> 1
jshell> for(int i=2;i<a.length;i++){
   ...> a[i] = a[i-2] + a[i-1];
   ...> }
jshell> System.out.println(Arrays.toString(a))
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

در همینجا دو نکته وجود دارد که دانستنش خالی از لطف نیست:

• می‌توانید با فشردن کلید TAB دستوری که در حال وارد کردن آن هستید را به صورت خودکار کامل کنید یا لیستی از دستورات ممکن را دریافت کنید.

• در دستورات یک خطی نیازی به وارد کردن سمیکالن (;) وجود ندارد اما اگر دستورات چند خطی باشد لازم است که سمیکالن گذاشته شود.

متدها

ساختن متد های در Jshell کار چندان سختی نیست و همانگونه که قبلاً متدها را تعریف می‌کنیم در اینجا نیز به همان شکل اینکار را انجام می دهیم:

```
jshell> int[] produceFibonacci(int size){
   ...> int[] fibonacci = new int[size];
   ...> fibonacci[0] = 0;
   ...> fibonacci[1] = 1;
   ...> for(int i=2;i<fibonacci.length;i++){
   ...> fibonacci[i] = fibonacci[i-1] + fibonacci[i-2];
   ...> }
   ...> return fibonacci;
   ...> }
|  created method produceFibonacci(int)
```

حال متد ما به درستی تعریف شده و می‌توانیم از آن استفاده کنیم. با استفاده از دستور /methods میتوانیم لیستی از متد های تعریف شده را مشاهده کنیم:

```
jshell> /methods
|    printf (String,Object...)void
|    produceFibonacci (int)int[]
```

پس متد ما به درستی تعریف شده و می‌توانیم آن را به سادگی فراخوانی کنیم:

```
jshell> produceFibonacci(10)
$2 ==> [I@4883b407
jshell> System.out.println(Arrays.toString($2))
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

متد ما به درستی اجرا شد. همانطوری که می‌بینید، در دستورات فوق آن را در ۲$ ذخیره Jshell خروجی متد در جایی ریخته نشده در اینصورت خود کرده و ما هم می‌توانیم به آن دسترسی داشته باشیم. همینطور با استفاده از دستور /vars می‌توانیم لیستی از متغیر های تعریف شده را ببینیم:

```
jshell> /vars
|    int[] $2 = [I@4883b407
```

همینطور برای پاک کردن تمام متد ها و متغیر هایی که تعریف شده می‌توانیم از دستور /reset استفاده کنیم:

```
jshell> /reset
|  Resetting state.
jshell> /vars
```

کلاس‌ها

تعریف کلاس‌ها هم کار آسانی است:

```
jshell> class Person{
  ...> private String name;
  ...> private int age;
  ...> public Person(String name, int age){
  ...> this.name = name;
  ...> this.age = age;
  ...> }
  ...> public String toString(){
  ...> return name + " has " + age + " years old ";
  ...> }
  ...> }
| created class Person
```

حال می‌توانیم از کلاس تعریف شده شی بسازیم:

```
jshell> new Person("Hossein" , 20)
$3 ==> Hossein has 20 years old
```

امکان استفاده از Abstract Class و اینترفیس و دیگر قابلیت‌های شی گرایی در Jshell هرچند به صورت محدود تر وجود دارد. اما باید توجه کرد که اصولاً در چنین محیط‌هایی امکان نوشتن برنامه‌های خیلی بزرگ وجود ندارد. کلاس Student را از کلاس Person ارث بری می کنیم:

```
jshell> class Student extends Person{
  ...> private String fieldOfStudy;
  ...> public Student(String name, int age, String fieldOfStudy){
  ...> super(name,age);
  ...> this.fieldOfStudy = fieldOfStudy;
  ...> }
  ...> public String toString(){
  ...> return super.toString() + "Studying" + fieldOfStudy ;
  ...> }
  ...> }
| created class Student
```

کلاس Student نیز به درستی تعریف شده و می‌توانیم از آن استفاده کنیم. برای دیدن لیستی از کلاس‌های تعریف شده می‌توانیم از دستور /types استفاده کنیم.

```
jshell> /types
|  class Test
|  class Person
|  class Student
```

ساختن یک شی از Student

```
jshell> new Student("Hossein" , 20 , "Computer Engineering")
$5 ==> Hossein has 20 years old Studying Computer Engineering
```

Imports و Classpath

برای استفاده از متد ها و کلاس‌های تعریف شده در پکیج های دیگر نیاز دارید که آن‌ها را import کنید.به صورت پیشفرض تعدادی از این پکیج‌های پراستفاده تعریف شده‌اند. می‌توانید لیست پیکج‌های import شده را با استفاده از دستور /imports ببینید:

```
jshell> /imports
|  import java.util.*
|  import java.io.*
|  import java.math.*
|  import java.net.*
|  import java.util.concurrent.*
|  import java.util.prefs.*
|  import java.util.regex.*
```

برای اینکه بتوانید از کتابخانه‌های دیگر استفاده کنید باید آدرس فایل jar آن‌ها را به classpath اضافه کنید و پکیج های مربوطه آن‌ها را import کنید. ما میخواهیم از کتابخانه Jsoup برای خزش بر صفحات وب و استخراج محتویات آن‌ها استفاده کنیم. می‌توانید این کتابخانه را از سایت آن دانلود کنید.

برای اضافه کردن به classpath باید از دستور /classpath استفاده کنید و در ادامه آن آدرس فایل jar را مشخص کنید.

```
jshell> /classpath Desktop/jshell/jsoup-1.9.2.jar
| Path ‹Desktop/jshell/jsoup-1.9.2.jar› added to classpath
jshell> import org.jsoup.nodes.
Attribute Attributes BooleanAttribute Comment DataNode Document
DocumentType Element Entities FormElement Node TextNode XmlDeclaration
jshell> import org.jsoup.nodes.
```

همانطور که می‌بینید ما فایل jsoup-1.9.2.jar را به classpath اضافه کردیم و سعی کردیم که یک کلاس از پکیج org.jsoup.nodes را import کنیم تا از آن استفاده کنیم.

۲۵۰ فیلم برتر IMDb

بدون شک یکی از مهم‌ترین تفریحات دانشجویان دیدن فیلم و سریال است! تا اینجا تمام چیزی که لازم بود از Jshell بدانیم را یاد گرفتیم. الان می‌خواهیم کمی تفریح کنیم و لیست ۲۵۰ فیلم برتر سایت IMDb را استخراج کنیم و در فایلی ذخیره کنیم تا در ایام فراغتمان این فیلم‌ها را ببینیم.

Jsoup یکی از معروف ترین و قوی‌ترین کتابخانه‌ها برای انجام اینکار است. در این مقاله قرار نیست که این کتابخانه را کامل بررسی کنیم و حتی اجزا آن را معرفی کنیم و کار با آن را یاد بدهیم. برای یادگرفتن آن می‌توانید به مثال‌ها و داکیومنت های موجود در این سایت استفاده کنید.[۲]

کار چندان سختی در پیش نداریم، اول از همه نیاز به ساختن یک Document داریم اما قبل از آن باید آن را import کنیم:

```
jshell> import org.jsoup.*
jshell> import org.jsoup.nodes.Document
jshell> Document imdbTop = Jsoup.connect("http://www.imdb.com/chart/top").get();
imdbTop ==> <!doctype html>
<html xmlns:og="http://ogp.me/ns#" xmlns:fb="http://www.face …
```

در اینجا ما از لینک مربوط به ۲۵۰ فیلم برتر یک Document ساختیم. همانطور که مشاهده می‌کنید اطلاعات موجود در آن صفحه به صورت خودکار توسط کتابخانه دانلود می شود.

الان همه چیز برای پارس کردن محتویات این صفحه و استخراج اطلاعات دلخواه ما آماده است. کافی است که ما بر روی المان (Element) های این داکیومنت گذر کنیم و از Selector ها و قابلیت‌های کتابخانه برای استخراج اطلاعاتمان استفاده کنیم:

```
jshell> for(Element row : imdbTop.select("table.chart.full-width tr")){
  ...> String title = row.select(".titleColumn a").text();
  ...> String rating = row.select(".imdbRating").text();
  ...> System.out.println(title + "\t" + rating);
  ...> }
The Shawshank Redemption 9.2
The Godfather 9.2
The Godfather: Part II 9.0
```

The Dark Knight 8.9
12 Angry Men 8.9
Schindler›s List 8.9
Pulp Fiction 8.9
The Lord of the Rings: The Return of the King 8.9
The Good, the Bad and the Ugly 8.9
Fight Club 8.8
…

همانطور که دیدید اطلاعات را به سادگی و فقط با یک حلقه ساده استخراج کردیم و کار ما تقریباً تمام شده و تنها کافیست این اطلاعات استخراج شده را در فایلی ذخیره کنیم.
می‌توانیم این اطلاعات را از کنسول کپی کنیم و در یک فایل ذخیره کنیم اما طبیعتاً خیلی بهتر است که مستقیماً این کار را انجام دهیم.
برای اینکار نیاز داریم که این رشته‌ها را در جایی ذخیره کنیم:

```
jshell> List<String> topMovies = new ArrayList<>(250);
topMovieList ==> []
```

و کمی حلقه کد را تغییر دهیم تا نتایج را در این لیست ذخیره کند.

```
jshell> for(Element row : imdbTop.select("table.chart.full-width tr")){
   ...> String title = row.select(".titleColumn a").text();
   ...> String rating = row.select(".imdbRating").text();
   ...> topMovies.add(title + "\t" + rating);
   ...> }
```

حال با استفاده از امکانات موجود در Java NIO می‌توانیم با یک خط به سادگی محتویات این لیست را ذخیره کنیم.

```
jshell> import java.nio.charset.StandardCharsets
jshell> import java.nio.file.Files;
jshell> import java.nio.file.Paths;
Files.write(Paths.get("Desktop/Top250.txt"),topMovies,StandardCharsets.UTF_8)
```

و تمام! از تماشای این فیلم‌ها لذت ببرید!

1- https://openjdk.java.net/projects/jdk9/
2- https://jsoup.org/cookbook/

سیستم های پیشنهاد دهنده

مصطفی خلجی | تایید علمی: دکتر چیترا دادخواه

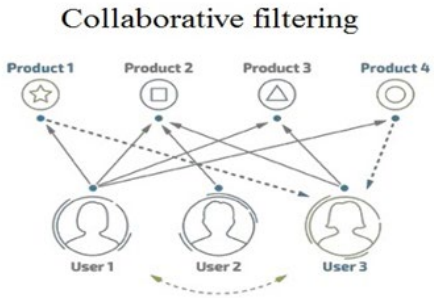
درحال حاضر، اینترنت حجم فراوانی از داده ها به عنوان فرصتی مناسب پیش روی کاربران قرار داده است، درصورت نبود مدیریتی کارآمد برروی انبوه داده‌های در دسترس، این امتیاز خود مانعی برای پیشرفت خواهد بود. به طوری که امروزه با توجه به حجم روز افزون داده‌ها و اطلاعات، نیاز به سیستم‌هایی که توانایی هدایت کاربران به سمت کالا و سرویس مورد نظر را داشته باشند، بیش از پیش احساس می‌شود. سیستم‌های پیشنهاد دهنده[1]، سیستمهای هوشمندی هستند که در فضای اینترنت با شناسایی علایق و اولویت‌های کاربر، اطلاعات موجود را پالایش کرده و پیشنهادات مناسب را به تک تک یا گروهی از کاربران ارائه می‌کنند. سیستم‌های پیشنهاد دهنده ابزاری برای توانمند کردن کاربران در بهره برداری از فضای وب محسوب می‌شوند و با استفاده از سیستم‌های پیشنهاد دهنده، امکان جستجو به دنبال مفاهیمی وجود دارد که در جستجوی عادی، دسترسی به آنها میسر نیست. با رشد روز افزون تجارت دردنیای وب، آموزش الکترونیکی، افزایش ارتباط و اشتراک کاربران با یکدیگر و پیدایش شبکه‌های اجتماعی، لزوم طراحی و پیاده سازی چنین سیستم‌هایی غیرقابل انکار است، به این منظور تکنیک‌های متعددی مورد استفاده قرار گرفته‌اند که اکثریت آنها برپایه دو رویکرد مبتنی بر پالایش همکارانه[2] و پالایش محتوا محور[3] هستند. هدف اصلی تمامی این تکنیک‌ها افزایش دقت[4] در پیشنهاداتی است که به کاربران خود می‌دهند. معیار های دیگری از جمله: جدید بودن[5]، غیر قابل انتظار[6] نیز در این حوزه قابل بررسی می‌باشد.

پالایش همکارانه

یکی از مهمترین و پراستفاده‌ترین روش های فیلترینگ در سیستم‌های پیشنهاد دهنده روش همکارانه می‌باشد که پایه و مبنای کار در بسیاری از راهکارهای دیگر نیز بشمار می‌رود. روش کار این الگوریتم در واقع به همان صورتی است که ما در تصمیم گیری‌های روزمره خودمان عمل می‌نماییم. بعنوان مثال کالایی را می خریم که بیشتر مورد پسند دیگران واقع شده باشد (مثلا در دنیای تجارت الکترونیک امتیاز[7]های بیشتری به آن داده شده باشد). بنابراین می‌توان گفت در روش همکارانه آنچه مد نظر قرار داده می شود، بیشتر تجربه دیگران است تا خود فرد.

در این روش ابتدا باید اجازه داد تا کاربران در سیستم مشارکت نمایند و به آیتم های مختلف موجود در سیستم امتیاز دهند. البته این امتیاز دادن‌ها می‌تواند به صورت ضمنی نیز اتفاق افتد و توسط سیستم تشخیص داده شود. بعنوان مثال یک نوع امتیاز دادن ضمنی می‌توان بدین شکل باشد که آیتم‌هایی که بیشتر بارگذاری شده‌اند احتمالا از محبوبیت بیشتری برخوردار بوده‌اند و در نتیجه امتیاز بیشتری نسبت به بقیه به آنها داده می‌شود.

توجه داشته باشید که در این راهکار سیستم بر مبنای امتیازات، آیتم ها را رتبه بندی می کند و آیتم‌هایی با بیشترین امتیاز را به کاربر پیشنهاد می‌دهد. به همین دلیل در صورتی که سیستم تازه شروع به کار کرده باشد و یا آیتم جدیدی به سیستم اضافه شود، اطلاعات کافی از آیتم‌ها در دسترس نخواهد بود و در نتیجه نمی‌توان به درستی امتیاز دهی و رتبه بندی را انجام داد. این یکی از مشکلات اساسی و مهم در اینگونه سیستم‌ها می‌باشد که با عنوان شروع سرد[8] شناخته می‌شود. البته این سیستم‌ها از مشکل دیگری نیز رنج می‌برند که پراکندگی داده‌ها[9] می‌باشد. بدین معنی که اطلاعات در سیستم توسط همه کاربران بدست نیامده است و نمی‌توان بدرستی و با قطعیت گفت که چه آیتمی مقبولیت بیشتری دارد. در شکل زیر، نحوه پیشنهاد بر اساس سلایق دیگر کاربران، به کاربر جاری پیشنهاداتی صورت می‌گیرد.



شبکه های اجتماعی[10] ساختاری اجتماعی است که از گره‌هایی (که عموما فردی یا سازمانی هستند) که توسط یک یا چند نوع خاص از وابستگی، مانند ایده‌ها و تبادلات مالی، دوست‌ها، خویشاوندی، لینک‌های وب، سرایت بیماری‌ها (اپیدمولوژی)، به هم متصل هستند، تشکیل شده است. استفاده از خدمات شبکه‌های اجتماعی روز به روز محبوبیت بیشتری پیدا میکند. هم اکنون سایت های شبکه های اجتماعی، بعد از پرتال‌های بزرگی مثل Yahoo یا MSN و موتورهای جستجو مثل گوگل، تبدیل به پر استفاده‌ترین خدمات اینترنتی شده‌اند. سیستم پیشنهاد دهنده در شبکه های اجتماعی کاربرد فراوانی دارد به طور نمونه در شبکه اجتماعی فیس بوک پیشنهادات دوستی بر مبنای سیستم پیشنهاد دهنده موجود در فیس بوک می‌باشد.

1- Recommender Systems
2- Collaborative Filtering
3- Content-based Filtering
4- Accuracy
5- Novelty
6- Serendipity
7- Rate
8- Cold Start
9- Data Sparsity
10- Social Networks

پایتون چیست؟!

علیرضا عظیم زاده میلانی

پایتون یک زبان برنامه‌نویسی سطح بالا(High-Level)، مفسری(Interpreted)، محاوره‌ای (Interactive) و شی‌گرا اسکریپتی (Object-Oriented scripting) می‌باشد. زبان پایتون بدین خاطر طراحی شده تا قابلیت خوانایی بالاتری نسبت به سایر زبان‌های برنامه‌نویسی داشته باشد (استفاده از کلید واژه‌های انگلیسی کاملا معنادار، و در تعدادی کمتر نسبت به دیگر زبان‌ها(کمتر از ۴۰ کلید واژه).

• **مفسر:** مفسری بودن پایتون بدین معناست که برنامه‌های شما در زمان اجرا از نظر قواعد دستوری و ساختاری مورد بررسی قرار می‌گیرند و دیگر نیازی به کامپایل شدن برنامه پیش از هر اجرا نیست (دو زبان برنامه نویسی مفسری: PHP و Perl).

• **محاوره‌ای:** پایتون برخلاف سایر زبان‌های برنامه‌نویسی به شما اجازه می‌دهد بصورت مستقیم با محیط prompt پایتون ارتباط برقرار کنید و دستورهای خود را وارد نمایید و در همان لحظه خروجی دستورات را مشاهده نمایید:

```
>>> 2 + 6
8
>>> print “Hi  KNTU”
Hi  KNTU
```

• **شی‌گرا:** در برنامه‌نویسی شی‌گرا امکاناتی مانند کپسوله‌سازی، قابلیت استفاده مجدد، حل مسائل پیچیده و... این امکان را به ما می‌دهد تا بتوانیم داده‌های خود را پنهان نماییم، وابستگی به توابع را کمتر کنیم، انعطاف برنامه را افزایش دهیم و... . تمام اینها مبنای پایه زبان پایتون است.

ویژگی‌های پایتون:

• **ساده در یادگیری:** پایتون دارای کلید واژه‌های اندک، ساختاری ساده و نحوهای (Syntax) از پیش تعریف شده‌ی واضحی است. این ویژگی به دانشجوها، برنامه‌نویسان و... اجازه می‌دهد تا در کمترین زمان ممکن، کارهای خود را پیاده‌سازی نمایند.

• **ساده در خواندن:** کدهای پایتون وضوح بیشتری برای‌چشم دارد و آسان‌تر قابل مشاهده است.

• **ساده در نگهداری:** موفقیت پایتون بخاطر سادگی در نگهداری و پشتیبانی از سورس کدش است.

• **قابل انتقال:** برنامه‌هایی که به زبان پایتون نوشته می‌شوند، می‌توانند در سایر پلتفرم‌ها نیز اجرا شوند (Windows، Linux/Unix، Macintosh، An-droid و...).

• **قابل توسعه:** شما می‌توانید ماژول‌های سطح پایین نیز به مفسر پایتون خود اضافه نمایید. این ماژول‌ها، برنامه‌نویسان پایتون را توانمند می‌سازد که امکانات بیشتری به ابزارها و ماژول‌های کنونی‌شان اضافه نمایند تا برنامه نهایی از کارایی بیشتری برخوردار باشد.

• **پایگاه‌داده:** این زبان واسط های بسیار متعددی جهت برقراری ارتباط با دیتابیس‌های رایگان و تجاری در اختیار برنامه‌نویسان می‌گذارد (Oracle، Microsoft SQL Server 2003 and Later، IBM-DB2، SAP-DB و...).

• **برنامه نویسی GUI:** زبان پایتون به شما این امکان را می‌دهد تا بتوانید برنامه‌های کاربردی تحت وب یا دسکتاپ (حتی با واسط گرافیکی- Graphical User Interface) نیز طراحی نمایید.

• **متن باز.**

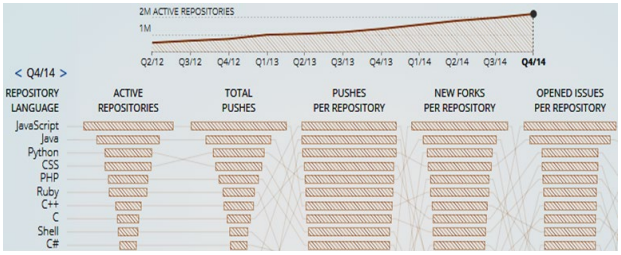
• **مُد محاوره‌ای.**

علاوه بر ویژگی‌های بالا، پایتون ویژگی‌های خوب دیگری مانند: پشتیبانی از متدهای برنامه‌نویسی ساختاری و تابعی، استفاده کردن از برنامه نوشته شده به‌صورت اسکریپت یا کامپایل کردن آن به ByteCode جهت ساخت برنامه‌های کاربردی در مقایس بزرگ، پشتیبانی از تکنیک «Automatic Garbage Collection»، و یکپارچه شدن با دیگر زبان‌های برنامه‌نویسی(C/++، CORBA، ActiveX و Java)، کدنویسی در قطعات الکترونیکی (Embeded Raspberry، Arduino) و... در خودش دارد.

با توجه به توضیحات بالا، اگر چه، ممکن است که بسیاری از برنامه‌نویسان قدیمی که با زبان‌هایی نظیر Java و C سروکار داشته‌اند، برنامه‌نویسی با پایتون را چندان جدی نگیرند و حتی آن را کودکانه تصور کنند؛ اما این تفکر از قابلیت‌های این زبان نخواهد کاست. در عوض، تعداد بسیاری از برنامه‌نویسان(همچون برنامه‌نویسان شرکت Google) پایتون را به‌دلیل سادگی، خوانایی و امکانات فراوانش به هر زبان دیگری ترجیح می‌دهند. دوستداران پایتون که غالباً Pythonistas نامیده می‌شوند، معتقدند: این زبان به‌قدری کامل و کار با آن لذت‌بخش است که برای تمام نیازهای برنامه‌نویسی می‌توان به آن مراجعه کرد. برخی از مهمترین دلایل فنی این دوستداران را می توانید در فصل نخست کتاب مطالعه نمایید.

در جدول‌های زیر می‌توانید گزارشی از رتبه‌بندی زبان‌های برنامه‌نویسی که توسط سه سایت معتبر در حوزه IT ارائه شده است، مشاهده نمایید.

۱) منبع سرویس GitHut بیش از دو میلیون مخزن کد فعال در سایت GitHub را آنالیز، و گزارش زیر را ارائه کرده است:



۲) سایت codeeval هر سال آماری مبنی بر محبوب‌ترین و رایج‌ترین زبان‌های برنامه‌نویسی را منتشر می‌کند، که این آمار از میان هزاران برنامه‌نویس فعال در صنعت IT جمع‌آوری شده است:

2015 Rank		Change	2014 Rank	Change	2013 Rank	Change	2012 Rank
1	Python	0	1	0	1	0	1
2	Java	0	2	0	2	0	2
3	C++	0	3	0	3	0	3
5	Ruby	-1	4	0	4	0	4
4	C#	2	6	2	8	1	9
7	C	0	7	-1	6	4	10
6	JavaScript	-1	5	2	7	-1	6
8	PHP	0	8	-3	5	0	5

برای مطالعه بیشتر به کتاب آموزش کاربردی برنامه نویسی به زبان پایتون که مولف آن مهندس عظیم زاده میلانی از دانشجویان ارشد شبکه های کامپیوتری دانشگاه خودمان هستند مراجعه کنید.

برای سفارش کتاب به این لینک مراجعه کنید:

http://bit.ly/py_azimzadeh

جاوا JNI

سهیل چنگیزی

JNI چیست؟

Java Native Interface یک رابط دوطرفه می باشد که اجازه فراخوانی متد هایی که در زبان محلی (C یا C++) موجود می باشند و تعامل آن متد ها با کد جاوا به کاربر میدهد.

چرا JNI ؟

در اینکه زبان java زبان قدرتمندی است شکی نیست ولی در یک سری موقعیت ها نمی تواند به تنهایی نیاز های ما را برطرف کند. به عنوان مثال:

- بعضی مواقع به سرعت بیشتری نیاز داریم پس به جای java می بایست از زبان هایی که به صورت مستقیم با ماشین ارتباط برقرار می کنند استفاده کنیم.
- ممکن است بعضی کتابخانه های موردنیاز ما در زبان های دیگری نوشته شده باشند.

ممکن است زبان java یک سری platform ها پشتیبانی نکند.

میخواهیم یک کد HelloWorld ساده را که به زبان C++ نوشته شده را در جاوا کامپایل و اجرا کنیم.

چگونه؟؟!!

ایجاد سورس کد جاوا: ابتدا یک فایل java. با نام HelloWorld ایجاد میکنیم تابع مورد نیاز ما قرار نیست ورودی یا خروجی داشته باشد همچنین باید از واژه native برای تعریف متدی که قرار است بدنه آن در زبان محلی باشد استفاده کنیم. متدهای native شبیه به متدهای abstract عمل می کنند یعنی نیاز به بدنه ندارند. حال در تابع main باید یک شی از کلاس HelloWorld بسازیم تا متد محلی را صدا بزنیم.

بخش استاتیک زودتر از بقیه کد اجرا می شود و قرار است کتابخانه داینامیک را بارگذاری کند.

```
class HelloWorld {
    private native void print();
    public static void main(String[] args) {
        new HelloWorld().print();
    }
    static {
        System.loadLibrary("HelloWorld");
    }
}
```

کامپایل کد جاوا و ایجاد header: با دستور javac کد بالا را کامپایل می کنیم تا فایل class. ایجاد شود سپس با اجرای دستور javah -jni بر روی فایل کلاس فایل header را ایجاد می‌کنیم. محتویات فایل header به صورت زیر می‌باشد.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class HelloWorld */

#ifndef _Included_HelloWorld
#define _Included_HelloWorld
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     HelloWorld
 * Method:    print
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_HelloWorld_print
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

همانطور که مشاهده می‌کنید در خط دوم این header فایل کتابخانه jni.h اضافه شده. این کتابخانه تمام اطلاعات موردنیاز تابع‌های محلی برای همکاری با کد جاوا را فراهم می‌سازد. همچنین شامل متغیرهای ضروری و ماکرو و typedef های لازم می‌باشد تا و پیچیدگی mapping متغیرهای جاوا به زبان محلی را پنهان کند.

همچنین در این header فایل prototype یک تابع مشاهده می‌شود که ورودی‌های آن یک اشاره‌گر از جنس استراکت JNIEnv (اسم این استراکت در زبان JNINativeInterface C می‌باشد)، که این استراکت شامل تمام تابع‌های لازم برای ایجاد اشیا، پرتاب اکسپشن‌ها، تبدیل آرایه‌های محلی به/از آرایه های جاوا و یا تبدیل رشته های محلی به/از رشته های جاوا است. با استفاده از این اشاره‌گر می‌توان هر کاری را که زبان جاوا می‌تواند انجام دهد را انجام

دهیم. ورودی دوم این تابع بستگی به استاتیک یا غیراستاتیک بودن متد محلی دارد. اگر متد استاتیک باشد یک رفرنس به کلاس می‌باشد (jclass) و اگر غیراستاتیک باشد یک رفرنس به شی می باشد(jobject).

JNIEXPORT و JNICALL دو ماکرو هستند برای مشخص کردن قرارداد فراخوانی و لینک شدن متد محلی در جاوا و پیاده سازی آن در زبان محلی استفاده می‌شوند. در سیستم عامل های linux و mac این ماکروها خالی می‌باشند. اما به دلیل اینکه ویندوز از قرارداد فراخوانی متفاوتی استفاده می‌کند (کتابخانه های داینامیک با پسوند dll. ایجاد می‌شوند) این ماکرو ها به ترتیب(dllexport)declspec_ و stdcall_ میباشند. نام این تابع نیز طبق یک قرارداد ایجاد می شود ابتدا باید از پیشوند Java_ سپس نام کلاس و در آخر نام متد محلی باید استفاده شود.

پیاده سازی کد محلی: ابتدا یک فایل cpp. با نام HelloWorld ایجاد می‌کنیم تابع‌ای که قرار است پیاده سازی کنیم باید کاملا شبیه prototype آن در فایل HelloWorld.h باشد و بدنه آن پیام ما را چاپ کند.

ایجاد کتابخانه مشترک/داینامیک: برای اینکه کد پیاده سازی شده در زبان محلی قابل تلفیق با جاوا شود باید کد محلی را به یک کتابخانه مشترک/ داینامیک کامپایل کنیم. از آنجایی که پسوند این کتابخانه در سیستم عامل‌ها

```
#include <jni.h>
#include <iostream>
#include "HelloWorld.h"
using namespace std;

JNIEXPORT void JNICALL
Java_HelloWorld_print(JNIEnv *, jobject){
    cout << "Hello World from C++ :| \n";
    return;
}
```

متفاوت می باشد این فرآیند وابسته به سیستم عامل خواهد بود و کتابخانه ایجاد شده در linux غیرقابل استفاده بر روی windows می باشد.

پسوند کتابخانه های مشترک/داینامیک در لینوکس so. و در مک jnilib. و در ویندوز dll. می‌باشد.

نکته قابل توجه دیگر کتابخانه jni.h است که در کد محلی استفاده شده است و باید آدرس آن را به کامپایلر g++ بدهیم تا کد را کامپایل کند. آدرس این کتابخانه در فولدر include در مکانی که جاوا نصب شده قرار دارد. همچنین باید از دستور c- برای ایجاد شی باینری استفاده کنیم.

g++ “PATH” -c HelloWorld.cpp

حال باید دستور زیر را در ترمینال وارد کنیم, از دستور dynamiclib- برای ایجاد کتابخانه از شی باینری استفاده می کنیم)کد زیر برای ایجاد کتابخانه در مک می‌باشد):

g++ -dynamiclib -o libhelloworld.jnilib HelloWorld.o

اجرای برنامه: حال با دستور java فایل کلاس HelloWorld را اجرا کنید.

ارسال آرگومان ها

چگونه متغیر هایی که در زبان محلی تعریف نشده‌اند (مانند boolean و ...) از جاوا به زبان محلی ارسال می‌شوند؟

برای حل این مشکل جاوا این متغیرها را به متغیرهای تعریف شده در زبان محلی map کرده. به عنوان مثال به جای unsigned char از boolean استفاده شده:

typedef unsigned char jboolean

همچنین اشیایی چون آرایه و رشته نیز map شده اند. که سلسله مراتب آن ها در شکل زیر نشان داده شده.

Java Type	C/C++ type	Native name	Description
boolean	unsigned char	jboolean	unsigned 8 bits
byte	signed char	jbyte	signed 8 bits
char	unsigned short	jchar	unsigned 16 bits
short	short	jshort	signed 16 bits
int	int	jint	signed 32 bits
long	long (for 64-bit) long long (else)	jlong	signed 64 bits
float	float	jfloat	32 bits
double	double	jdouble	64 bits
void	void	void	

معایب JNI

```
jobject                    (all Java object)
  ├──jclass                (java.lang.Class objects)
  ├──jstring               (java.lang.String objects)
  ├──jarray                (arrays)
  │     ├──jobjectarray    (array of object)
  │     ├──jbooleanarray   (array of boolean)
  │     ├──jbytearray      (array of byte)
  │     ├──jchararray      (array of char)
  │     ├──jshortarray     (array of short)
  │     ├──jintarray       (array of int)
  │     ├──jlongarray      (array of long)
  │     ├──jfloatarray     (array of float)
  │     ├──jdoublearray    (array of double)
  └──throwable             (java.lang.Throwable object)
```

- رفع مشکل و دیباگ کد بسیار سخت تر خواهد شد.
- برنامه‌ها وابسته به یک پلتفرم خاص خواهند شد (به دلیل کتابخانه داینامیک متفاوت) که برای حل این مشکل باید تمام پیاده سازی‌های مربوط به هر پلتفرم را جداگانه انجام دهیم. تا جاوا تشخیص دهد از کدام استفاده کند.
- garbage collector برای بخش کد محلی کار نمی‌کند پس باید به صورت دستی اشیا را کنترل کنیم.

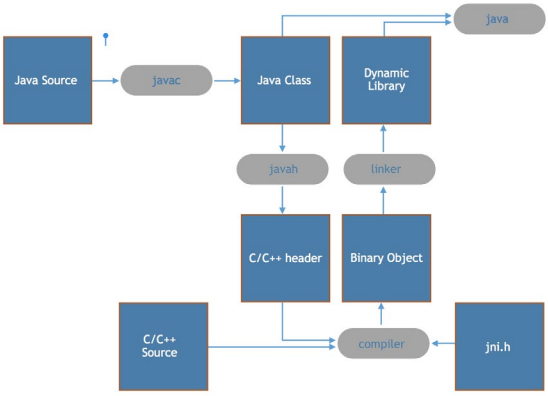
متد های JNI هزینه بالایی دارند و در صورت چند بار صدا زدن‌ممکن است سرعت برنامه کاهش پیدا کند.

منابع:

http://docs.oracle.com/javase/1.5.0/docs

http://www.pacifer.com/mmead/cs510jip/jni

http://mrjoelkemp.com/2012/01/getting-started-with-jni-and-con-osx-lion



docker به زبان ساده

حسین رحمتی

تایید علمی: دکتر سعید صدیقیان

همه‌چیز از آن جایی شروع شد که وب خیلی سریع گسترش پیدا کرد و این فضای پویا باعث بوجود آمدن اپلیکیشن‌های تحت وب در مقیاس خیلی بزرگ شد. این اپلیکیشن‌ها در درجه اول نیاز داشتند که برروی یک سرور اجرا (Run) بشوند و افراد بتوانند از آنها استفاده کنند. به این فرآیند که یک اپلیکیشن در اختیار همه قرار بگیرد Software Deployment می‌گویند.

در فرآیند Deployment یک سری سلسله مراتب وجود دارد:

- Release: آماده شدن خروجی برای انتقال به سرور و در اختیار دیگران قرار گرفتن.
- Install and Activate: این بخش فرآیند اجرا (execute) کردن اپلیکیشن است.
- Deactivate: این بر خلاف مرحله قبلی است و به معنی قطع‌کردن روند اجرای اپلیکیشن‌مان است.
- Adapt: فرآیند بهبود دادن نرم‌افزاری است که قبلاً نصب شده است.
- Update: جایگزینی نسخه قدیمی‌تر با نسخه جدیدتر.
- Uninstall: برخلاف Install و به معنی پاک کردن نرم‌افزار از سیستم.
- Retire: پایان یافتن دوره پشتیبانی و حذف کلی نرم‌افزار.

در گذشته هر اپلیکیشن بر روی یک سرور مجزا اجرا می‌شد ولی با گذشت زمان و به لطف پیشرفت تکنولوژی ما می‌توانیم هر اپلیکیشن را به صورت کاملا مستقل و ایمن بر روی یک سرور اجرا نماییم. با این حال هر بار از ما درخواست برای راه اندازی یک اپلیکشن تجاری و یا سرویس می‌شود ما نیاز به خرید یک سرور پیدا می‌کنیم و می‌بایست مدل و مشخصات مورد نیاز را تنها حدس زده و سپس اقدام به تهیه سرور کنیم که ممکن است مشخصات سرور بعد از راه اندازی سرویس بیشتر یا حتی کمتر از میزان مورد نیاز ما باشد. سالها بعد این مشکل از طریق سرویس‌های مجازی سازی همچون Vmware حل گردید. با آنکه مجازی سازی از طریق این مجازی سازها یکی از حرفه‌ای ترین روش ها برای راه اندازی سرویس ها، مدیریت منابع و... می‌باشد ولی قطعا مشکلاتی نیز دارد!

حال این سؤال پیش می‌آید که مشکل این سیستم Deployment چیست؟

عواملی از جمله هزینه زیاد لایسنس جهت راه اندازی سرویس مجازی سازی، هزینه لایسنس جهت نصب هر Patch ، OS کردن و مانیتور کردن هر OS و حتی نیاز به نصب Os جداگانه برای هر سرویس که منجر به از بین رفتن منابع خواهد شد (منابعی که صرف Up کردن Os خواهد شد) و از طرفی زمان Boot طولانی از جمله مواردی هستند که مجازی سازی هایی همچون Vmware را دچار چالش می کنند.

از مشکلات موجود دیگر اینکه ممکن است ما از یک فریم‌ورک یا یک ورژن خاص از یک زبان برنامه نویسی استفاده کرده باشیم ولی چیزی که روی سرور هست با چیزی که ما از آن استفاده کرده‌ایم متفاوت باشد و ما نیز دسترسی برای تغییر آن را نداشته باشیم.

مشکل احتمالی دیگر اینکه ممکن است Development یا Test ها بر روی سیستم خاص یا سیستم‌عامل خاص انجام بدهیم و اما چیزی که قرار است روی سرور Production باشد متفاوت باشد.

چاره پیشنهادی: استفاده از کانتینرها

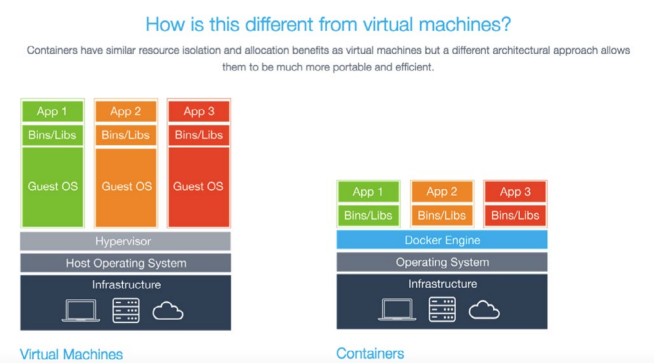
کانتینر چیست؟

کانتینرها چاره این مشکل هستند؛ که چگونه بدون اینکه نرم‌افزار ما به مشکل بربخورد، به صورت کاملاً منظم از یک محیط به یک محیط دیگر منتقل بشود. این محیط می‌تواند در هر کدام از فاز های Development, Test یا Production باشد.به لطف وجود Container ها مجازی سازی در سطح سیستم عامل انجام می‌شود. در واقع شما هیچ سیستم عامل جدیدی ایجاد نمی‌کنید بلکه به بسته نرم‌افزاری این امکان را می دهید که از کرنل و یا هسته اصلی سیستم عامل بهره برد و که این عامل باعث افزایش کارایی سیستم خواهد شد . بنابراین Containerها بدون نیاز به OS بوده و در حقیقت از هسته اولیه سیستم عامل بهره می‌برند.

البته مفاهیم کانتینر چیز جدیدی نیست. از ده سال پیش در LXC مطرح شد و داشت استفاده می‌شد. بعدها تکنولوژی‌های دیگری مثل Free-BSD و Solaris هم اضافه شدند و کانتینرهای مخصوص خودشان را راهی بازار کردند. این روزها مفاهیم جدیدی تحت عنوان اپلیکیشن کانتینرها مطرح شده‌اند مثل Kubernetes و Docker.

Docker چیست؟

داکر یک اپلیکیشن کانتینر است. یعنی اینکه می‌توانیم اپلیکیشن‌مان را در قالب یک پکیج مستقل از سایر پکیج‌ها اجرا کنیم. در این پکیج همه



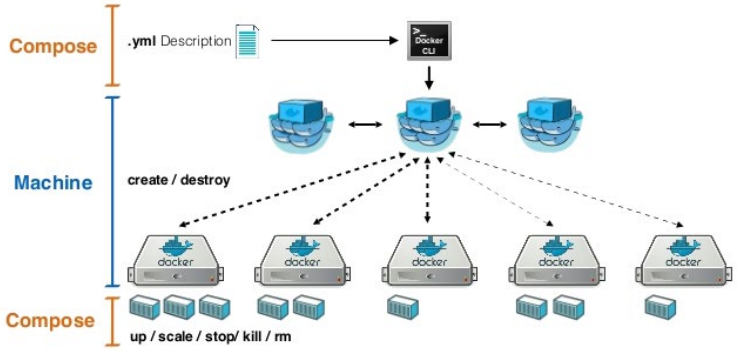
چیزهایی که برای اجرای آن اپلیکیشن لازم است (کد، کتابخانه، ابزار سیستمی، dependencyها و ...) با پیکربندی‌های انجام شده به صورت پیش‌فرض، همگی قرار می‌گیرند و این باعث می‌شود پکیج ما مستقل از سیستمی که دارد بر روی آن اجرا می‌شود عمل کند.

داکر با Golang[1] نوشته شده، برای استفاده از شبکه از سیستم Bridge[2] استفاده میکند.

یک فرق اساسی و مهم این سیستم نسبت به قبل این شده است که سیستم می‌تواند در Load یکسان، کارایی بیشتری داشته باشد. کانتینرها بر خلاف سیستم قبلی(مجازی سازی) از یک سیستم‌عامل مشترک استفاده می‌کنند. قبلاً در سیستم مجازی‌سازی لازم بود برای هر اپلیکیشن ما یک سیستم‌عامل مجازی داشته باشیم و این باعث می‌شد کارایی (Performance) سیستم کم بشود و تعداد محدودتری اپلیکیشن روی سرور اجرا بشوند.

به فایل‌هایی که شامل تمامی نیازمندی‌های اپلیکیشن، کد، کتابخانه و ... می‌شوند Docker Image می گوییم. (نکته: Docker Image حالت استاتیک است و وقتی از این Image ها Run بگیریم آن را Docker Container می‌نامیم). استفاده از داکر یک فرایند سه مرحله‌ای Build-Ship-Run است. (تولید-انتقال به سرور-اجرا کردن).

داکر متن‌باز است و این باعث شده کامیونیتی خیلی قوی ای داشته باشد. خودِ توسعه‌دهنده‌های داکر یک مخزن (Repository) به اسم Docker Hub[3] ایجاد کرده‌اند که افراد و شرکت‌ها می‌توانند Imageهای خود را در آن قرار بدهند و بقیه از آنها استفاده کنند.
البته این Repository برای ایرانیان تحریم است! و برای استفاده ازآن باید از VPN استفاده کرد یا از ریپوزیتوری های شخصی Mirror استفاده کرد.
نکته مهم در مورد کانتینرها Stateless بودن‌شان است. یعنی حالتشان را حفظ نمی‌کنند. این باعث می‌شود که هر بار که ما آن‌ها را اجرا می کنیم، به تغییراتی که دفعه قبل ایجاد کرده‌ایم و فایل‌هایی که ساختیم و ... دسترسی نداشته باشیم. برای چاره‌ی این موضوع در داکر باید یک دایرکتوری برای قرارگیری این فایل‌ها مشخص کنیم و از طریق Docker Volume این فضا ها را به Docker Engine معرفی کنیم.
یک مساله مهم در مورد داکر این است که می‌توانیم داکر ایمیج های دلخواه خودمان را داشته باشیم. این ایمیج‌ها با استفاده از Dockerfile ساخته می‌شوند. داکرفایل‌ها، فایل‌های متنی‌ای هستند که Docker Engine آن‌ها را تفسیر می‌کند و Docker Image را برایمان می‌سازد.
وقتی داکر را نصب می‌کنیم یک پکیج از چند نرم‌افزار داریم:
docker engine – docker swarm – docker compose – docker machine – docker registry و ...
در این بین Docker Compose یک ابزاری است که بجای اینکه تعداد زیادی داکر فایل داشته باشیم و بخواهیم جداگانه آن‌ها را run کنیم یکی از این فایل‌ها با پسوند yml. می‌سازیم و آن فایل را با ابزار مخصوص Run می‌کنیم.



Docker Machine برای زمانی است که یک واسط بین Docker Engine و Docker Container ها داریم. مثلاً زمانی که قصد داریم از داکر در ویندوز یا مک OS استفاده کنیم به طور خودکار Docker Machine ایجاد می‌شود.
Docker Swarm یک سیستم کلاسترینگ داکر است. این سیستم یک سری Node ایجاد می‌کند و بر مبنای معماری Master-Slave ، بعضی از این Node ها نقش Manager دارند و وضعیت اجرا و پایداری کانتینرها در سیستم‌های Slave را مدیریت می‌کنند، اگر این کانتینرها غیرفعال شده باشند مجدداً آن‌ها را راه‌اندازی می‌کنند.
داکر برای همه سیستم‌عامل ها موجود می‌باشد. برای نصب به سایت رسمی‌اش مراجعه کنید[4] و قبل از نصب حتماً حتماً نحوه نصب را بخوانید (مخصوصا در لینوکس).
بی‌شک بهترین و کامل‌ترین مرجع برای استفاده از داکر بخش مستندات رسمی[5] خود داکر است که می‌توانید به آن دسترسی داشته باشید. البته چند سایت فارسی[6] هم برای این منظور بوجود آمده اند که برخی مطالب ابتدایی را به طور کامل توضیح داده اند.
چند کتاب خوب هم در این زمینه چاپ شده است که ما به چندتا از آنها اشاره می کنیم:

1- Docker: Up & Running: Shipping Reliable Containers in Production By Karl Matthias, Sean P. Kane. O›Reilly Media; 1 edition (2015).
2- Docker in Action By Jeff Nickoloff. Manning Publications; 1 edition (2016).
3- Using Docker: Developing and Deploying Software with Containers By Adrian Mouat. O›Reilly Media; 1 edition (2016).
4- Docker for Java Developers Package, Deploy, and Scale with Ease By Arun Gupta. O›Reilly Media; 1 edition (2016).

منابع:

1- https://golang.org/
2- https://en.wikipedia.org/wiki/Bridging_(networking)
3- https://hub.docker.com/
4- http://docker.com/
5- http://docs.docker.com/
6- http://elastico.io/

معرفی موتور بازی سازی UNITY

محمد سینا کیارستمی | تایید علمی: دکتر علی احمدی

هیچ‌گاه فراموش نمی‌کنم، اولین بازی کامپیوتری که بازی کردم «فیفا ۲۰۰۰» با گزارش عادل فردوسی‌پور بود. هنوز خوب یادم است که گاهی اوقات از روی عصبانیت و عمداً، آن‌قدر روی دروازه‌بان تیم حریف خطا می‌کردم تا بازی نیمه‌کاره تمام می‌شد. از همان موقع این سوال در ذهنم نقش بست که «چگونه این بازی‌ها ساخته می‌شوند؟». در دنیای کودکی خودم پاسخ‌هایی ساده، گاه با چاشنی تخیل، صرفا برای آرام کردن چند ماهه‌ی ذهنم ساخته و پرداخته می‌کردم. زمان گذشت و کم‌کم فهمیدم که از چه راهی به پاسخ این سوال برسم. راهی که مرا با صنعت بزرگ بازی‌سازی و سپس با پدیده‌ای به نام موتور بازی آشنا کرد.

شاید کم و بیش درباره‌ی موتورهای بازی شنیده یا خوانده باشید. شاید در حد چند کلمه آن هم موقع خواندن نقد و بررسی بازی مورد علاقه‌تان؛ اما به شما توصیه می‌کنم با آن‌ها بیشتر آشنا شوید حتی اگر نمی‌خواهید یک بازی‌ساز شوید. شاید با خواندن این مطلب دِین خود را تا حدی به موتورهای بازی‌سازی ادا کرده باشید؛ چراکه آن‌ها بودند که واسطه شدند و خلاقیت ذهن بازی‌سازها را به‌حقیقت بدل کردند و بسیاری از خاطرات زیبای کودکی ما را رقم زدند. بسیاری از خنده‌ها، گریه‌ها، شب‌بیداری‌ها و سوزش چشم‌های همه‌ی ما متعلق به آن‌هاست. با معرفی موتور قدرتمند Unity همراه ما باشید.

تاریخچه

تا قبل از دهه‌ی ۸۰ میلادی، ساخت بازی‌های ویدیویی به چنین و چنانی امروزه نبود. در آن زمان تنها چند شرکت خاص آن هم با هزینه‌هایی گزاف این دست بازی‌ها را تولید می‌کردند؛ بازی‌هایی که بعضا با سخت افزار مخصوص خود عرضه می‌شدند. از اوایل دهه ۸۰ میلادی بود که با سرعت گرفتن روند رو به رشد تکنولوژی کامپیوترها، سرعت رشد و توسعه ساخت بازی‌های کامپیوتری نیز افزایش پیدا کرد تا اینکه ابزارهای مستقل و مفیدی جهت ساخت بازی‌های ویدیویی دو بعدی معرفی شدند که معروف‌ترین آن‌ها عبارت اند از:

- Pinball Construction set (1983)
- ASCII's War Game Construction set (1983)
- Thunder Force Construction set (1984)
- Adventure Construction set (1984)
- Garry Kitchen's Game Maker (1985)
- War Game Construction set (1986)
- Shoot'Em_up Construction set (1987)
- Arcade Game Construction kit (1988)
- ASCII's RPG Maker Engines (1988)

از اواسط دهه‌ی ۹۰ میلادی موتورهای بازی که امروزه نسل جدیدترشان را می‌بینیم، به منظور ساخت بازی‌های ویدیویی سه‌بعدی به‌وجود آمدند که می‌توان به «موتورهای اکشن اول شخص» (First Person Shooter engine) به عنوان اولین‌های این عرصه اشاره کرد. از شناخته شده‌ترین بازی‌های آن دوران نیز می‌توان به «دووم» (Doom) و «دووم ۲» (Doom II) اشاره کرد که هر دو توسط موتور «آی‌دی تک ۱» (idTech 1) ساخته شدند. موتورهای «آنریل ۱»، «دارک استار»، «آی‌دی تک ۲»، «ماراتن ۲»، «کوآک»، «بیلد» و در نهایت «گولد سورس» (Gold Source)، خالق بخش عظیمی از خاطرات بازی کردن ما یعنی «ضد حمله» (Counter Strike)، همگی از بهترین‌های حوزه خود در بین سال‌های ۱۹۹۰ تا ۲۰۰۰ میلادی بودند.

آی‌دی تک، لیت تک و آنریل نسخه‌های جدیدتر خود را در سال‌های بعد عرضه کردند. در کنار این غول‌های پردازش گرافیکی، موتورهای جدیدی نیز مانند IW، که هسته مرکزی «ندای وظیفه (Call of Duty)»بود، به دنیای بازی‌سازی معرفی شدند. امروزه نیز دائما خبرهای مختلفی از شاهکارهای جدید «یونیتی» (Unity) و «فراست بایت» (Frostbite) مانند «میدان جنگ ۱» (Battlefield1) را می‌شنویم.

تا اینجا با هم تا حدودی با سرگذشت موتورهای بازی‌سازی آشنا شدیم. حال این سوال پیش می‌آید که اصلا چرا به موتورهای بازی قلب و هسته مرکزی بازی می‌گویند؟ پاسخ این سوال واضح است! در واقع سازندگان بازی‌ها، با استفاده از یکی از همین موتورها، بازی خود را خلق می‌کنند. اما چگونه؟ برای پاسخ به این سوال از یک مثال خیلی ساده و قابل فهم استفاده می‌کنیم: قطعا همه‌ی شما حداقل یک بار بازی‌های شهرسازی یا از این قبیل بازی‌ها را امتحان کرده‌اید، حال فرض کنید که آن بازی موتور بازی‌سازی شماست و شهری که درحال ساخت و توسعه‌ی آن هستید بازی‌تان است؛ با این تفاوت که ساختن بازی آن‌قدرها هم ساده نیست. برای بررسی بیشتر چیستی و نحوه عملکرد موتورهای بازی، به بررسی یکی از بهترین‌های آن‌ها می‌پردازیم.

معرفی و بررسی موتور بازی Unity | آقای خاص!

موتور یونیتی امروزه یکی از معروف‌ترین و رایج‌ترین موتورهای بازی‌سازی در این صنعت است که اولین نسخه آن یعنی Unity ۱,۰,۰ در سال ۲۰۰۵ میلادی توسط «یوآخین آنته»، «دیوید هلگاسون» و «نیکولاس فرانسیس» (David Helgason ،Joachin Ante و Nicholas Francis) به همگان معرفی شد.

برخی ویژگی‌های این موتور قدرتمند عبارتند از: قابلیت ساخت بازی‌های دو بعدی و سه بعدی، قابلیت ساخت بازی‌ها در ژانر و مکانیزم‌های متفاوت، قابلیت نورپردازی و رندرینگ فوق‌العاده، دارای موتور پردازش و ویرایش صدا، دارای تعداد زیادی از ابزارهای مفید (asset) رایگان و پیش ساخته، مستندسازی کامل و حرفه‌ای، پشتیبانی از ساخت بازی تحت پلتفرم‌های ویندوز، مک، اندروید، لینوکس و وب و از همه مهم‌تر دارای رابط کاربری ساده و روان.

یونیتی برای پیاده‌سازی گرافیکی دو بعدی از دو مدل «بیت مپ» (Bitmap) و «وکتور» (Vector) استفاده می‌کند. مدل پیاده‌سازی گرافیکی بیت مپ به این صورت است که رنگ هر پیکسل در یک نقشه ذخیره می‌شود و در نهایت از شکل مورد نظر یک نقشه‌ی پیکسلی رنگی ایجاد می‌کند. مدل پیاده‌سازی گرافیکی وکتور نیز به این صورت است که یک شکل گرافیکی را به‌صورت مجموعه‌ای از اطلاعات هندسی ذخیره می‌کند. فرض کنید شما دایره‌ای با شعاع ۵ و با ضخامت حاشیه‌ی ۱ رسم کرده‌اید؛ در مدل پیاده‌سازی وکتور اطلاعات به‌صورت عبارت «یک دایره با شعاع ۵ و ضخامت ۱» (a circle with radi-us 5 and border 1) ذخیره می‌شود.

برای کار با فایل‌های صوتی شما می‌توانید از دو روش «صدای دیجیتال» (Digitized Audio) و «هم‌گذاری» (Synthesized) استفاده کنید که در مقاله‌ی بعدی مفصلاً درباره آن‌ها حرف خواهیم زد. برای فرمت فایل‌های قابل استفاده در موتور، مختصر می‌توان به .mp3 ، .aif ، .wav ، .aup ، sesx و agg. اشاره کرد.

تمام مواردی که تا اینجا مطرح شد به شما این امکان را می‌دهند که عکس یا مدل‌های شبیه‌سازی سه بعدی یا دو بعدی خود را به عنوان محیط یا اجسام موجود در بازی به موتور اضافه کنید و با توجه به نیاز خود از آن‌ها استفاده نمایید. همچنین فایل‌های صوتی مورد نیاز خود را نیز می‌توانید همانند عکس‌ها و مدل‌های گرافیکی به راحتی از طریق قسمت ابزارهای مفید به موتور اضافه کنید. مثلا فایل صدای ترمز کردن ماشین را به موتور اضافه کرده و سپس به آن می‌گویید زمانی که کاربر در بازی

عمل ترمز کردن را انجام داد (فرض کنید با فشاردن کلید s روی کیبورد) این فایل صدا را پخش کند؛ البته لازمه‌ی چنین تعاملی با موتور، دانش برنامه نویسی است. بخش کار با صدا و طراحی گرافیکی بازی یک طرف، بخش برنامه‌نویسی بازی یک طرف. تمامی این روابط در یک بازی در واقع همان اجرای توابع و قطعه کدهای از پیش‌نوشته شده توسط برنامه نویس است. به طور مثال هروقت کاربر با موس چپ کلیک کرد، اسکریپت شلیک کردن فراخوانی شود. برنامه‌نویسی در یونیتی به دو زبان «سی شارپ» (C#) و «جاوا اسکریپت» (JavaScript) انجام می‌پذیرد که سی شارپ در مقابل جاوا اسکریپت به دلیل قدرت و توانایی بالاتر و امکانات بیشتر، از اقبال بیشتری برخوردار است.

یونیتی با ابزارهای قدرتمند دیگری مانند «ویژوال استودیو» (Visual Studio) نیز هماهنگ است که امکان برنامه‌نویسی بازی برای شما را بسیار راحت‌تر می‌کند. کافی است از طریق منوی ابزارهای مفید در نوار بالای موتور یک فایل جدید C# Script باز کنید. سپس می‌توانید هر آنچه که بازی شما برای تبدیل شدن به یک بازی کامل نیاز دارد را کد نویسی کنید. البته لازم به تذکر است که بسیاری از ابزارها و امکاناتی که شما برای ساختن یک بازی نیاز دارید از قبل مهیا شده است. در سایت یونیتی نیز یک فروشگاه کامل از لوازم و طراحی‌ها تعبیه شده است. به‌طور مثال چند مدل درخت را می‌توانید با پرداخت هزینه‌ای جزیی تهیه کنید و از آن در طراحی محیط بازی خود استفاده نمایید. رنگ‌آمیزی محیط و اشیاء درون بازی نیز کار بسیار ساده‌ای است. تنها با ایجاد ماده (Material) از طریق ابزار مفید، بدون حتی یک خط کد زدن تمام محیط خود را آن‌گونه که می‌خواهید رنگ‌آمیزی کنید.

روشن است که بازی‌ها هرچقدر حرفه‌ای‌تر و پیچیده‌تر می‌شوند؛ به طراحی گرافیکی و برنامه‌نویسی پیچیده‌تری نیازمند هستند. در این مقاله تلاش کردیم شما را کمی بیشتر با این ناشناخته‌های همه‌کاره آشنا کنیم. نحوه عملکرد آن‌ها را از طریق بررسی یکی از بهترین‌های این عرصه یعنی موتور یونیتی تا حدودی به شما شناساندیم. در هرحال هر مخاطبی باید تا حدودی در مورد چگونگی ساخت و توسعه بازی‌ها اطلاعات داشته باشد تا بازی‌هایی را که بازی می‌کند درک کند و فقط یک بازیکن نباشد؛ چه بسا یک منتقد خوب و یا حتی یک بازی‌ساز خوب شود.

از آنجایی که به‌خوبی می‌دانیم ذهن شما هم‌اکنون سرشار از سوالات بی‌جواب است نوید این را می‌دهیم که در قسمت بعدی این مقاله به اکثر آن‌ها پاسخ دهیم. در قسمت بعدی مفصل در مورد بازی‌سازی با موتور یونیتی بحث خواهیم کرد و ساختن یک بازی نسبتا جالب و ساده را از صفر تا صد برای شما توضیح خواهیم داد.

کامپایل کردن همراه با بهینه سازی در GCC

امیرراد کیمیایی

gcc یک کامپایلر بهینه ساز است.این کامپایلر حاوی مجموعه ای از گزینه ها است که یا سرعت فایلهای قابل اجرای را افزایش می دهند،یا سایز آنها را کاهش می دهد.

برای بهینه سازی کدهایی که قرار است کامپایل شوند باید نکات زیادی در نظر گرفته شود: توجه به پردازنده های مختلف و assembly های مختلف آنها که می توانند با هم ناسازگار باشند، تعداد registerها که مشخصا در نحوه ی ذخیره‌ی نتایج میانی تاثیر دارند و همچنین ترتیب‌های مختلف برای دستورات assembly مختلف که توسط کامپایلر تولید می شوند، همه نکاتی هستند که کامپایلر باید در هنگام تولید فایل خروجی بهینه در نظر بگیرد که توسط gcc نیز لحاظ می‌شود.

اولین نوع بهینه سازی که gcc از آن بهره می برد در سطح سورس کد استفاده می‌شود و نیازی به اطلاع از دستورات سطح ماشین (مرتبط با معماری پردازنده) ندارد. در راستای این بهینه سازی تکنیکهای زیادی وجود دارند که به دو عدد از آنها یعنی حذف زیرعبارت عمومی(common subexpression elimination) و function inlining خواهیم پرداخت.

Common subexpression elimination

یکی از راه های بهینه سازی در سطح کد این است که محاسبه‌ی یک عبارت را با دستورات کمتر، با استفاده دوباره از نتایج محاسبه شده، انجام داد. برای مثال در نمونه‌ی پایین:

```
x = cos(v) * (1 + sin(u / 2)) + sin(w) * (1 - sin(u / 2))
```

می تواند به کمک یک متغیر موقت به صورتی نوشته شود که از ارزیابی غیر ضروری و اضافه ی لفظ sin(u/2) جلوگیری کند:

```
t = sin(u / 2)
x = cos(v) * (1 + t) + sin(w) * (1 - t)
```

به این دوباره نویسی حذف زیرعبارت عمومی می گویند(CSE) و هنگامی که بهینه سازی فعال شود به صورت خودکار انجام می شود. این ابزاری قوی محسوب می‌شود زیرا هم زمان سرعت را زیاد و اندازه کد را کمتر می‌کند.

Function inlining

نوعی دیگر از بهینه سازی سطح کد، که به آن function inlining گفته می‌شود در عملکرد توابعی که زیاد فراخوانی می‌شوند تاثیر زیادی دارد. هنگامی که یک تابع استفاده می شود، زمانی اضافه نیاز است تا CPU آن فراخوانی را انجام دهد: باید آرگمان‌های تابع را در رجیسترها و مکان‌های مناسب حافظه ی ذخیره کرده، به ابتدای تابع پرش کرده (یا حتی صفحات حافظه مجازی مورد نیاز را به حافظه فیزیکی یا گش CPU بیاورد اگر نیاز باشد)، شروع به اجرای کد کرده و سپس هنگامی که فراخوانی تابع به انتها می رسد، به نقطه ی قبلی اجرا بر می گردد. به این فرآیند برای اجرای تابع سربار فراخوان تابع می گویند. با استفاده از function inlining این سربار با جایگزینی هر فراخوانی با کد خود تابع حذف می‌شود.

در بیشتر مواقع، از سربار فراخوانی یک تابع می توان چشم‌پوشی کرد. اما این مسئله وقتی بزرگ می شود که توابعی با دستورات نسبتا کم وجود دارند، و آن توابع بخش زیادی از زمان اجرا را شامل می‌شوند.

Inlining در همه ی مواقعی که برای یک تابع تنها یک نقطه‌ی فراخوانی در برنامه وجود دارد مطلوب است. همچنین اگر فراخوانی یک تابع بیشتر از جابه‌جا کردن بدنه ی آن به صورت in-line دستورات بیشتری بخواهد (حافظه) نیز همواره مفید است. این یک وضعیت معمول در بسیاری از توابع دسترسی در C++ است،که می توانند به اندازه زیادی از Inlining بهره ببرند.

تابع پایین یک مثال معمول از تابعی است که از inlining بهره مناسبی می برد.این تابع تنها توان دوی ورودی را محاسبه می کند:

```
double sq(double x) {
   return x * x;
}
```

این تابع کوچک است، و سربار فراخوانی آن می تواند با زمان اجرای آن مقایسه شود. به عنوان مثال اگر این تابع در حلقه‌ای مانند قطعه کد زیر استفاده شود سربار بسیار زیاد می شود:

```
for (i = 0; i < 1000000; i++) {
   sum += sq(i + 0.5);
}
```

با اجرای بهینه سازی خواهیم داشت:

```
for (i = 0; i < 1000000; i++) {
   double t = (i + 0.5);
   sum += t * t;
}
```

GCC توابعی که مناسب inlining هستند را با استفاده از مکاشفاتی انتخاب می کند، مانند اندازه ی تابع که به اندازه ی مناسبی کوچک باشد. به عنوان یک بهینه سازی، inlining فقط در هر فایل (object.o) انجام می شود.کلمه کلیدی inline می تواند توسط خود برنامه نویس نیز در کد استفاده شود که به طور مستقیم بیان شود تا یک تابع در هر جا که می شود، inline شود. (حتی به هنگام استفاده در فایل های دیگر).

Speed-space tradeoffs

با اینکه بعضی از انواع بهینه سازی مانند حذف زیرعبارت عمومی می‌توانند هم حجم برنامه را کاهش داده و هم سرعت را افزایش دهند، انواع دیگر بهینه سازی کدهای سریعتر را در ازای افزایش حجم فایل اجرایی ارائه می‌دهند.بهینه سازی‌هایی با مصالحه فضایی-زمانی (speed-space tradeoff) همچنین می‌توانند بالعکس حجم کد را کاهش و سرعت را کندتر کنند.

Loop unrolling

یک مثال از بهینه سازی با مصالحه فضایی-زمانی بازکردن حلقه است. برای مثال به کد زیر توجه کنید:

```
for (i = 0; i < 8; i++) {
   y[i] = i;
}
```

این حلقه به طور کلی ۹ بار اجرا شده است و بخش زیادی از زمان اجرا به چک کردن شرط حلقه تعلق دارد. راه حلی بهتر برای نوشتن همین کد به صورتی دیگر این است که حلقه را باز کنیم:

```
y[0] = 0
y[1] = 1;
y[2] = 2;
y[3] = 3;
y[4] = 4;
y[5] = 5;
y[6] = 6;
y[7] = 7;
```

این نوع کد نیاز به هیچ چک کردنی ندارد و با ماکسیمم سرعت اجرا

می‌شود. از آنجایی که هر انتسابی مستقل از دیگری می‌باشد، همچنین به کامپایلر اجازه می دهد که از موازی سازی نیز بر روی پردازنده‌هایی که این قابلیت را دارند، استفاده کند. باز کردن حلقه نوعی از بهینه سازی است که سرعت فایل اجرایی را بالا برده و معمولا اندازه را افزایش می دهد (مگر اینکه حلقه بسیار کوتاه باشد مانند یک یا دو بار تکرار).
باز کردن حلقه می‌تواند حتی در زمانی که حد بالایی حلقه مشخص نیست نیز اجرا شود. برای مثال قطعه کد زیر همانند بالا را با حد نامشخصی در نظر بگیرید:

```
for (i = 0; i < n; i++) {
   y[i] = i;
}
```

می تواند توسط کامپایلر به صورت زیر نوشته شود:

```
for (i = 0; i < (n % 2); i++) {
   y[i] = i;
}
for (; i + 1 < n; i += 2)  {
   y[i] = i;
   y[i + 1] = i + 1;
}
```

حلقه ی اول برای i=0 هنگامی که n فرد است مقدار انتساب را انجام می‌دهد و حلقه ی دوم ادامه‌ی حلقه‌ی اصلی را انجام دهد. به این توجه کنید که حلقه‌ی دوم نیاز به مقدار اولیه ندارد زیرا ادامه ی کار حلقه‌ی اول را انجام می‌دهد.انتساب‌های حلقه‌ی دوم می‌توانند به صورت موازی انجام شوند، و تعداد تست‌هایی که به عنوان شرط در حلقه انجام می شود، تقریبا با ضریب ۲ کمتر می شود. ضرایب بیشتر نیز می‌توانند مورد استفاده قرار بگیرند که در این صورت باید انتساب‌های بیشتری در داخل حلقه را باز کرد که سایز کد را بیشتر می‌کند.

Scheduling

پایین ترین سطح scheduling است، که در آن کامپایلر بهترین ترتیب دستورات مستقل را مشخص می‌کند. بیشتر CPU ها این امکان را دارند که یک یا بیشتر از یک دستور را قبل از تمام شدن دستورهای دیگر اجرا کنند. بسیاری از CPU ها نیز از pipelining پشتیبانی می کنند، که چندین دستور به صورت همزمان بر روی CPU اجرا می‌شوند.
هنگامی که scheduling فعال می شود، دستورات باید به گونه ای قرار داده شوند که نتایج آنها برای دستورات بعدی در زمان مناسب در دسترس باشد، و همچنین امکان اجرای موازی به صورت ماکسیمم را فراهم کنند. Scheduling سرعت یک فایل اجرایی را بدون افزایش سایز آن فراهم می کند، اما نیاز به حافظه و زمان اضافه در فرآیند کامپایل دارد (به دلیل پیچیدگی آن).

سطوح بهینه سازی

برای اینکه زمان و استفاده حافظه در هنگام کامپایل کنترل شود، GCC چندین سطوح بهینه سازی را فراهم می‌کند که از ۰ تا ۳ عددگذاری شده اند و همچنین گزینه‌های مستقل بسیاری نیز برای گونه های مشخصی از بهینه سازی وجود دارند.
یک سطح بهینه سازی با گزینه ی OLEVEL- در کامند لاین انتخاب می شود،که LEVEL عددی بین ۰ تا ۳ می باشد.اما هر کدام از این سطوح چه تاثیراتی بر خروجی کد و کامپایل آن دارند؟
O0 - یا نبود گزینه O- (حالت پیش فرض)
در این سطح GCC هیچ بهینه سازی را اجرا نمی‌کند و کد را به مستقیم ترین شکل ممکن کامپایل می کند.هر دستور در کد منبع مستقیم بدون هیچ گونه جابه‌جایی در نحوه ی قرار گیری، به دستورات معادل (IL) تبدیل می شود. این بهترین گزینه است هنگامی که می خواهیم برنامه را دیباگ کنیم.
O1- یا O-
این سطح عمومی ترین اشکال بهینه سازی را که نیازی به هیچ مصالحه زمانی فضایی ۰ باشد. O- ندارند، فعال می کند. با این گزینه فایل اجرایی باید کوچک‌تر و تندتر از در این مرحله استفاده نمی scheduling هایی بالاتر مانندهایی با هزینه‌بهینه سازی زمان کمتری ببرد، از O0- شوند. کامپایل کردن با این سطح نیز ممکن است نسبت به .های ساده باید پردازش شوندآنجایی که داده‌ی کمتری بعد از انجام بهینه سازی
O2-
این سطح بهینه سازی های بیشتری را علاوه بر سطح قبل فعال می کند. این بهینه سازی های اضافه شامل scheduling می شود. اما در این مرحله نیز از بهینه سازی‌هایی که مصالحه فضایی-زمانی دارند استفاده نمی‌شود و در نتیجه حجم فایل اجرایی بیشتر نخواهد شد. کامپایلر زمان بیشتری برای کامپایل و همچنین حافظه ی بیشتری نسبت به سطح O1- می‌طلبد.این گزینه معمولا بهترین گزینه برای توزیع و عرضه ی یک برنامه است زیرا بیشترین سطح بهینه سازی را بدون افزایش حجم فایل اجرایی فراهم می‌کند. این سطح، سطح پیش فرض برای انتشار در بسته های GNU می باشد.
O3-
این گزینه بهینه سازی‌های هزینه برتری مانند inline کردن توابع به کار می‌برد (علاوه بر بهینه سازی‌ها در سطوح قبل). این سطح علاوه بر افزایش سرعت می‌تواند سایز فایل اجرایی را نیز افزایش دهد. در بعضی از شرایط غیرمطلوب این گزینه می‌تواند باعث کندشدن برنامه نیز بشود.

-funroll-loops

این گزینه باز کردن حلقه ها را فعال می کند،و مستقل از دیگر گزینه های بهینه سازی است.با توجه به اینکه این گزینه حجم فایل را افزایش می دهد،مفید بودن آن باید در موارد مختلف بررسی شود تا کارایی آن مشخص شود.
Os-
این گزینه بهینه سازی هایی را فعال می کند که حجم را کاهش می دهند و هدف تولید کوچک ترین فایل های اجرایی ممکن است. در بعضی موارد، به علت استفاده ی بهتر از cache ممکن است این گزینه باعث بهبود سرعت نیز بشود.
با در نظر گرفتن تاثیرات هر گزینه می توان آنها را به درستی انتخاب کرد اما به طور کلی بهتر است از O0- برای دیباگ کردن و از O2- برای توسعه و عرضه استفاده شود.
برای نشان دادن تاثیر هر کدام از سطوح مختلف بهینه سازی و گزینه‌ها آنها را برروی برنامه ی زیر امتحان می کنیم:

```
# include < stdio.h > double powern(double d, unsigned n) {
   double x = 1.0;
   unsigned j;
   for (j = 1; j <= n; j++) x *= d;
      return x;
   }
   int main(void) {
      double sum = 0.0;
      unsigned i;
      for (i = 1; i <= 100000000; i++) {
         sum += powern(i, i % 5);
      }
      printf("sum = %g\n", sum);
      return 0;
   }
```

حال با استفاده از کامند لاین کد را با گزینه های مختلف کامپایل و با دستور time زمان اجرای فایل اجرایی را به دست می آوریم:

```
gcc test.c -O0
time ./a.out
real  0m1.023s
user  0m1.016s
sys  0m0.004s
gcc test.c -O1
time ./a.out
real  0m0.455s
```

```
user  0m0.448s
sys  0m0.004s
gcc test.c -O2
time ./a.out
real  0m0.192s
user  0m0.188s
sys  0m0.002s
gcc test.c -O3
time ./a.out
real  0m0.188s
user  0m0.184s
sys  0m0.002s
gcc test.c -O3 -funroll-loops
time ./a.out
real  0m0.188s
user  0m0.183s
sys  0m0.002s
```

توجه کنید که در خروجی پارامتر اصلی سنجش زمان user می باشد. real و sys به ترتیب کل زمان سپری شده از زمان اجرای برنامه تا اتمام آن (شامل زمان‌هایی می شود که CPU برای فرآیندهای دیگر نیز صرف کرده) و زمان سپری شده برای فراخوانی‌های مرتبط با سیستم عامل را نشان می‌دهند. تست‌های فوق بر روی پردازنده Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz انجام شدند.

منابع:

http://www.network-theory.co.uk/docs/gccintro/gccintro_45.html

An Introduction to GCC: For the GNU Compilers GCC and G++, Brian Gough, 2004

معرفی MonoGame

حسن صادقین

معرفی :

XNA یک Framework براساس DirectX است که برای طراحی بازی های دو بعدی و سه بعدی است که برای این منظور از زبان C# بهره می‌برد. محدودیت های XNA به این صورت بود که بازی ها طراحی شده تنها در Windows و Xbox360 قابلیت اجرایی داشتند که این محدودیت‌ها مشکلات بسیاری را برای تولید کننده ها فراهم آورد تا اینکه دیگر تقریبا استفاده‌ای از آن نمی‌شود.

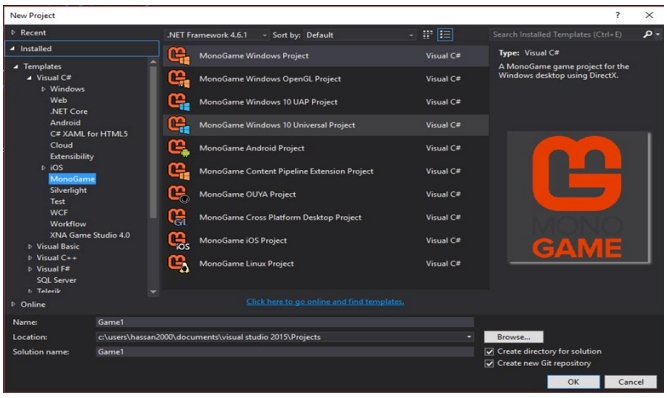
در سال ۲۰۰۹ با توسعه XNA به منظور اجرا در تمامی سیستم عامل ها از جمله Windows,Linux,Mac,IOS,PS4,Xbox One,... یک Framework دیگر به وجود آمد که تمامی ویژگی های XNA را داشت با این تفاوت که در این پیاده سازی علاوه بر استفاده از DirectX از OpenGL نیز استفاده شده بود که این باعث قابلیت اجرا شدن خروجی ها در سیستم عامل های گوناگون بود.

آخرین ورژن این موتور ۳,۵ است که می‌توان در سایت رسمی آن monogame.net دریافت و استفاده کرد.

چرا Monogame :

- از بهترین ویژگی های آن استفاده از زبان C# و تمامی قابلیت ها .NET برای آن است.
- متن باز بودن آن
- قابلیت خروجی در بسیاری از کنسول ها و سیستم عامل ها
- کارایی بسیار خوب آن در منابع

برای استفاده از Monogame نیاز به داشتن Visual Studio ویا MonoDevelop است که بعد از دانلود این IDE ها و نصب خود Monogame شما انتخاب های گوناگونی دارید برای ایجاد پروژه که در عکس پایین آمده است دارید :



بعد از ساختن پروژه در قسمت solution کلاس Game1 را خواهید دید که مهم ترین کلاس بازی است و تمامی کار ها در آن انجام می‌شود. این کلاس از کلاس game ارث بری می‌کند که تمامی توابع لازم برای پیاده سازی منطق بازی و گرافیک بازی را در خود دارد که به این توابع اشاره میکنیم:

Initialize : همین طور که از اسمش پیداست در این توابع مقدار دهی‌های اولیه را انجام می‌دهیم.

LoadContent : از این تابع نیز برای بارگذاری تمامی شئ‌هایی که میخواهیم در بازی نشان داده شوند از جمله عکس ها و فونت ها و... استفاده می‌کنیم.

UnloadContent : در این تابع هنگامی که بازی بسته شد باید تمامی محتوا هایی را که در بازی بارگذاری کردیم را آزاد کنیم.

Update : این تابع که مهم ترین تابع در این کلاس است برای تمامی منطق بازی از جمله ورودی‌ها و روابط ورودی‌ها با اشیا در بازی استفاده می‌شود.

در این تابع ما تمامی اطلاعاتی که داریم برای مثال موقعیت بازیکن را به روز می‌کنیم. از دیگر کار هایی که در این تابع باید انجام دهیم صداگذاری است.

این تابع به ازای هر Frame یک‌بار انجام می‌شود که باعث می‌شود در هر فریم ما اطلاعت را به روز کنیم و سپس آنها را نمایش دهیم.

این تابع یک ورودی از نوع GameTime دارد که بعداً به آن می‌پردازیم.

Draw : در این تابع تمامی تمامی محتوایی که داریم رو به تصویر می‌کشیم و در صفحه نمایش نشان می‌دهیم. با استفاده از تابع update و draw ما میتوانیم در هر فریم اطلاعات را به روز کنیم و نمایش بدهیم.

2D Graphics : همانند بسیاری از Framework ها و موتورهای بازی سازی دیگر XNA نیز مبدا صفحه نمایش را بالا سمت چپ در نظر می‌گیرد که با پایین آمدن ارتفاع بیشتر و با به طرف راست رفتن عرض بیشتر می‌شود.

اگر دقت کرده باشید یک فولدری به نام Content وجود دارد که درون آن یک فایلی به نام Content.mgcb است که ما با استفاده از این می‌توانیم تمامی محتوای خود را از جمله عکس و فونت و حتی Shader ها را نیز در بازی بارگذاری کنیم. این فایل خود تمامی محتوایی که درون آن بگذاریم را تبدیل به فرمت xnb میکند و اگر به پوشه bin بازی برویم تمامی این محتوا را خواهیم دید.

بعد از بارگذاری یک عکس برای مثال hero.png در این فایل ما می‌توانیم این عکس را در بازی بکشیم برا این کار می‌بایست از کلاس Texture2D استفاده کنیم.

```
private Texture2D Hero;
protected override void LoadContent() {
   spriteBatch = new SpriteBatch(GraphicsDevice);
   Hero = Content.Load<Texture2D>("hero");
}
```

همان طور که دیدید ما در تابع loadContent یک شئ از کلاس SpriteBatch ساختیم که این کلاس از مهم ترین کلاس ها برای کشیدن محتوا در حالت دو بعدی است.

برای کشیدن تصویر کافیست در تابع draw داشته باشیم

```
protected override void Draw(GameTime gameTime){
   GraphicsDevice.Clear(Color.CornflowerBlue);
   spriteBatch.Begin();
   spriteBatch.Draw(Hero, new Rectangle(0, 0, 800, 480), Color.White);
   spriteBatch.End();
   base.Draw(gameTime);
}
```

تابع SpriteBatch.Draw() انواع مختلف با ورودی‌های مختلفی دارد که به توضیح مهم ترین آنها می‌پردازیم :

یکی از ورودی‌های اصلی آن یک شئ از Texture2D است که می‌خواهیم آن را بکشیم.

دیگر ورودی یک شئ از Rectangle است به نام destnitionrectangle که همانطور که از اسمش پیداست شکل مورد نظر را در مستطیلی که ما می‌دهیم می‌کشد.

دیگر ورودی مهم آن یکی شئ دیگر از Rectangle است به نام sourcerectangle که ما در این مستطیل مشخص می‌کنیم کدام بخش از عکس خود را بکشیم.

یکی دیگر از ورودی های آن layerdepth است که ما با مشخص کردن آن با عددی بین ۰ و ۱ عمق آن را تعیین میکنیم که نوع کشیدن برای مثال از عمق بیشتر به کمتر وبرعکس در تابع spritebatch.begin() می‌توانیم تعیین کنیم.

از دیگر ورودی های آن مانند تعیین کردن مرکز جابجایی و چرخاندن آن نسبت به آن مرکز و... است.

XNA خود به صورت پیش فرض سعی بر نگه داشتن Framerate بازی در۶۰ فریم دارد

که هر فریم ۰,۰۱۶۶۶۶۷ ثانیه میشود. البته مدت زمان هر فریم تا فریم بعدی ممکن است متفاوت شود پس میتوان فهمید که در هر ثانیه ۶۰ بار تابع update اجرا میشود. می توان این Framerate را که به صورت پیش فرض در ۶۰ قفل است را عوض کرد و یا Framerate ثابتی نگذاشت. اما تابع draw قوانین متفاوتی دارد که هر وقت خود سیستم خواست یعنی در واقع نیاز دانست آن را اجرا میکند. با این تفاسیر سناریو های متفاوتی وجود دارد که میتوان به آنها اشاره کرد:

۱. فرض کنید که تابع update و draw در هر فریم یک بار اجرا شوند و بعد از ۱ ثانیه پس باید ۶۰ بار اجرا شده باشند که این ساده ترین سناریو موجود است.

۲. فرض کنید که تابع update و draw کمتر از ۶۰/۱ ثانیه نیاز داشته باشند برای اجرا در این صورت نیز جواب ساده است باید صبر کنیم تا این زمان زیاد آورده شده اتمام شود.

۳. فرض کنید که مجموع زمان اجرای این دوتابع از ۶۰/۱ ثانیه بیشتر شود در این صورت اگر کامپیوتر از لحاظ سخت افزاری ضعیف باشد ما می‌توانیم از متغیر IsRunningSlowly استفاده کنیم که بعضی از محتوا را بارگذاری نمی‌کند و یا جزئیات را کم می‌کند، به صورتی که بازی روان اجرا شود.

اما اگر یک تک فریم باشد که هزینه بالایی دارد؛ برای مثال محتوای جدید زیادی بارگذاری می‌شود مانند انفجارهای متنوع در بازی که باعث می‌شود آن فریم مقداری بیشتر طول بکشد که در اینصورت XNA خود نیز تابع update را به صورت اتوماتیک چند بار دیگه فرا می‌خواند که باعث می شود یک پرش در صحنه دیده شود.

اگر هم بعد از همه این کار ها باز هم دچار افت فریم شدیدی باشیم باید سخت افزار را ارتقاء ببخشیم.

کلاس GameTime:

اگر به تابع های draw و update نگاهی بکنید خواهید فهمید که این توابع ورودی از نوع کلاس GameTime دارند. این کلاس به ما دو ویژگی مهم خواهد داد از جمله : TotalGameTime,ElapsedGameTime

ما از این دو ویژگی در بسیاری از موارد از جمله در انیمیشن ها استفاده خواهیم کرد.

TotalGameTime : این ویژگی مقدار کل زمانی که بازی آغاز شده است را دربر دارد که می‌توانیم این زمان را به روز, ساعت و... بدست آوریم.

ElapsedGameTime : این ویژگی مقدار کل زمانی را که از اخرین فراخوانی تابع update میگذرد را به ما می‌دهد که با استفاده از این ویژگی میتوانیم انیمیشن های روانتری بسازیم.

FrameRate : دو نوع فریم ریت در ساخت بازی میتوانیم استفاده کنیم.

Fixed Time Step : که در این روش ما فریم ریت ثابتی داریم که به صورت پیش فرض در XNA به این صورت است و فریم ریت در ۶۰ قفل است پس برای مثال فرض کنید که عکسی داریم که میخواهیم آن را با گذشت زمان جابجا کنیم. برای مثال فرض کنید می‌خواهیم ۱۴ واحد یا پیکسل در ثانیه آن عکس را جابجا کنیم پس چون بازی در ثانیه ۶۰ بار تابع update را اجرا میکند پس ما باید ۰,۲۳۳۳=۶۰/۱۴ در هر فریم جابجا شویم یعنی در واقع داریم :

```
public Game1() {
   graphics = new GraphicsDeviceManager(this);
   Content.RootDirectory = “Content”;
   IsFixedTimeStep = true;
}
private Vector2 Position =new Vector2(0,0);
private Vector2 velocity = new Vector2(0.2333f, 0);
protected override void Update(GameTime gameTime){
   Position += velocity;
   base.Update(gameTime);
}
```

پس با این کار داریم در هر فریم عکس را ۰,۲۳۳۳ واحد جابجا میکنیم که در ثانیه ۱۴ واحد میشود. توجه کنید که اگر تابع update , draw کارش زود تر تمام شود XNA خود ایستاده تا به حد مورد نظر برسد یعنی همیشه بازی در ۶۰ فریم اجرا می شود اگر توانست که فریم ریت بهتری دهد منتظر می‌ماند.

Variable Time Step : در این روش ما فریم ریت ثابتی نخواهیم داشت و توابع update , draw هر چه سریع تر انجام می‌شوند و ما قفلی بر روی آنها نگذاشته‌ایم. اما در این روش اگر بخواهیم که عکس را جابجا کنیم باید مفهموم واحد بر فریم را کنار بگذاریم چون دیگر نمی‌دانیم که بازی در چه فریم ریتی اجرا می‌شود و ثابت نیست برای این منظور باید از مفهموم واحد بر ثانیه استفاده کنیم.

با استفاده از ویژگی ElapsedGameTime ما میتوانیم بفهمیم که هر فریم چند ثانیه طول می‌کشد پس با استفاده از کد زیر می‌توانیم عکس را جابجا کنیم.

```
public Game1(){
   graphics = new GraphicsDeviceManager(this);
   Content.RootDirectory = “Content”;
   IsFixedTimeStep = false;
}
private Vector2 Position =new Vector2(0,0);
private Vector2 velocity = new Vector2(14, 0);
protected override void Update(GameTime gameTime){
   Position += velocity * gameTime.ElapsedGameTime.TotalSecond;
   base.Update(gameTime);
}
```

با این کار ما در واقع داریم می‌بینیم که هر فریم چه قدر طول می‌کشد برای مثال ۲ میلی ثانیه یکی از فریم ها طول می‌کشد پس چون سرعت جابجایی عکس ما ۱۴ واحد بر ثانیه بود و ۲ میلی ثانیه فریم ما طول کشید پس با یک حساب ساده می‌توان فهمید که ۱۴*۲/۱۰۰۰=۰,۰۲۸ واحد باید در این فریم جابجا شود.

Inputs :

در XNA برای دسترسی داشتن به ورودی‌های کاربر کلاس‌هایی در نظر گرفته شده است که با استفاده از آن ها میتوانیم به تمامی ورودی های کیبورد و موس و دسته را پاسخ دهیم.

برای دست یابی به کلید هایی که کاربر فشرده است میتوانیم با استفاده از کلاس Keyboard و همین طور برای دست یابی برای ورودی های دسته میتوانیم از کلاس GamePad استفاده کنیم.

یکی از روش‌های خوب برای ذخیره کردن ورودی‌ها map کردن (در C# Dictionary,Enum) ورودی ها به اعداد باینری ۱,۲,۴,۸,... است که با زدن چند دکمه همزمان می‌توان به ورودی های کاربر پاسخ داد.

Shader : Shader ها برنامه هایی هستند که مستقیما به GPU فرستاده می‌شوند و نحوه نمایش دادن پیکسل ها را مشخص می‌کنند . در XNA برای نوشتن این Shader ها نیاز است که در Content یک فایل از نوع Effect درست کنیم و در آن برنامه خود را بزنیم.

در Monogame باید Shaderها را به زبان HLSL که برای DirectX است بزنیم خود Monogame مبدلی از HLSL به GLSL که برای OpenGL است دارد. در کتاب خانه های XNA چند نمونه از Shader ها ازجمله بعضی از نور ها برای استفاده وجود دارد که در کلاس BasicEffect است.

3D World : برای اینکه جهانی سه بعدی داشته باشیم نیازی به اطلاعاتی در زمینه ماتریس ها داریم . می‌دانیم که ماتریس ها در صورتی در هم ضرب می شوند که اگر ماتریس اولی m*n باشد ماتریس دوم حتما نیز باید به صورت n*k باشد تا بتوانند در هم ضرب بشوند.

یک نقطه در فضای ۳ بعدی داری سه مشخصه است x,y,z که با آن Point می‌گوییم.

یک بردار نیز داری سه مولفه x,y,z است که به آن Vector می‌گوییم.

برای اینکه بتوانیم بر روی این نقطه ها و بردار ها عملیات انجام دهیم نیاز به تعریف Transformation Matrix داریم. برای اینکه بتوانیم این ماتریس ها را در نقاط و بردار های مان ضرب کنیم باید این ماتریس ها ۳*۳ باشند اما برای اینکه بتوانیم تمامی کار های جابجایی و چرخاندن و بزرگ نمایی را انجام دهیم به یک ماتریس ۴*۴ نیازمندیم (با کمی جستجو میتوانید بفهمید چرا) پس برای اینکه بتوانیم نقاط مان را در این ماتریس ها ضرب کنیم باید نقاط ما ۴ مولفه داشته باشند که ما یک مولفه w را خود اضافه میکنیم که این مولفه همیشه باید ۱ باشد. به بعضی از این ماتریس ها نگاهی می‌اندازیم:

Translation matrices : این ماتریس برای جابجایی نقاط است است.

نقطه یک مکان از جهان ما را مشخص می‌کند اما یک بردار جهت را مشخص می‌کند پس جابجایی بردار برای ما تاثیری ندارد پس ما یک بردار را هرگز جابجا نمی‌کنیم.

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

همان طور که در شکل میبینید اگر ما این ماتریس را در نقطه ای ضرب کنیم این نقطه به اندازه X,Y,Z در جهات x,y,z به ترتیب جابجا می‌شود اما اگر دقت کنید با ضرب این ماتریس در یک نقطه مقدار w نیز ۱ باقی خواهد ماند برای مثال

$$\begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 1*10+0*10+0*10+10*1 \\ 0*10+1*10+0*10+0*1 \\ 0*10+0*10+1*10+0*1 \\ 0*10+0*10+0*10+1*1 \end{bmatrix} = \begin{bmatrix} 10+0+0+10 \\ 0+10+0+0 \\ 0+0+10+0 \\ 0+0+0+1 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

Scaling matrices : این ماتریس ها مختصات نقاط و یا جهات بردار ها را بزرگ یا کوچک میکنند

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

همان طور که در شکل میبینید مختصات در این مولفه قطر اصلی ضرب می‌شوند و باز مولفه w ۱باقی می‌ماند.

Rotaion matrices : یکی از مهم ترین ماتریس هاست که می‌تواند نقطه و یا بردار را در جهات مختف بچرخاند.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

همان طور که در شکل دیدید سه ماتریس مختلف برای سه جهت مختلف داریم که به ترتیب برای x,y,z هستند.

ترتیب ضرب کردن این ماتریس‌ها در بردار ها و نقاط مهم است که به این صورت ابتدا نقطه یا بردار خود را بزرگ می‌کنیم سپس می‌چرخانیم و بعد جابجا می‌کنیم.

Transformedvector=Translation * Rotation * Scale * originalvector

برای درک بهتر فرض کنید که ابتدا خود ۹۰ درجه به راست چرخیده سپس یک قدم به عقب بروید و یا یک قدم به عقب بروید و ۹۰ درجه خود را به راست بچرخانید.

حال با دانستن این ماتریس ها نیاز است که فضاهای مختلف را بدانیم:

هر شکلی که ما بخواهیم وارد بازی خود کنیم خود آن شکل دارای مختصاتی است که به آن Model space می‌گویند. پس برای اینکه جهانی داشته باشیم پر از اشکال متفاوت و مختلف باید یک مختصات یکسان و واحدی داشته باشیم که به این فضا World space می‌گویند و در آخر یک فضای دیگر داریم که چشم ما آنرا می‌بینند یا دوربین دارد آن را می‌بینند که به آن View space ویا Camera space می‌گویند.

حال برای آنکه تمامی چیز هایی که در مقابل دوربین هستند را در صفحه نمایش ببنیم نیاز به یک ماتریس دیگر به نام Projection matrix داریم این ماتریس تمامی محتوایی که در فضای View ویا همانی که دوربین در حال دیدن آن است را به صفحه نمایش منتقل می‌کند تا بتوانیم در صفحه نمایش آنها را نمایش دهیم.

برای ساختن یک دوربین باید ابتدا ماتریس View را بسازیم که یکی از راحت ترین روش ها استفاده از مفهموم Lookat است که در واقع جهتی است که دوربین به آن نگاه می‌کند. در XNA میتوانیم با استفاده از کد پایین این View matrix را بسازیم.

ViewMatrix = Matrix.CreateLookAt(CameraPosition, CameraLookAt, Vector3.Up)

Lookat ما یک بردار است که میتواند رو به یک شئ باشد که با جایجایی آن شئ مثلا شخصیت بازی Lookat ما نیز تغییر کند وmatrix View نیز تغییر کند.

برای ماتریس World نیز میتوانیم ماتریس همانی را انتخاب کنیم که مبدا مختصات را در ۰،۰،۰ می‌گذاریم.

برای ساختن ماتریس Projection گزینه هایی داریم از جمله Prespective و Orthographic که در Orthographic میتوان گفت ما فاصله‌ایی از دوربین نداریم. مانند بازی های دو بعدی اما Prespective را می‌توان گفت داری عمق است و دور یا نزدیک بودن معنی پیدا می‌کند مانند اکثر بازی های ۳ بعدی روز. برای ساخت این ماتریس می‌توان در XNA از کد پایین استفاده کنیم.

ProjectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4, Game.GraphicsDevice.Viewport.AspectRatio, 0.5f, 100.0f) ;

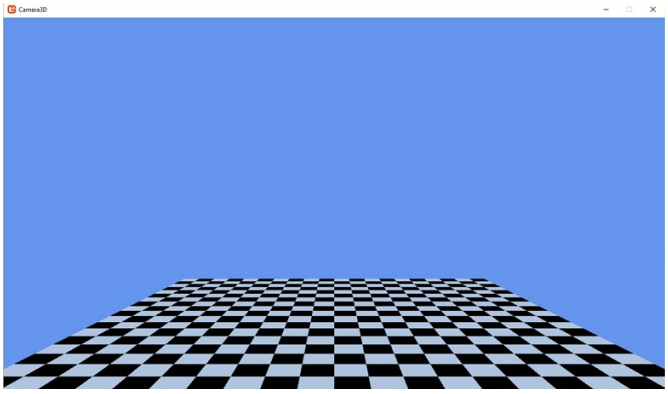
که برای اطلاعات بیشتر خود جستجو کنید.

حال با داشتن تمامی این ماتریس ها می‌توان با دادن آنها به گرافیک با استفاده از Shader ها یک دوربین داشت .

منابع:

https://www.scratchapixel.com/index.php?redirect

http://www.monogame.net/documentation/?page=Tutorials



الگوهای طراحی چرا و چگونه؟!

حامد خشه چی و حسین ریماز | تایید علمی: دکتر مهدی اثنی عشری

الگوهای طراحی یا Design Pattern ها یکی از مباحثی است که هر توسعه دهنده و برنامه نویسی نیاز دارد تا با آنها آشنا باشد. آشنایی و تسلط بر این الگوهای طراحی تقریبا یکی از نیازمندی هایی است که در هر آگهی استخدامی با آن مواجه خواهید شد. در این مقاله و در مقالات بعدی سعی خواهیم کرد تا در هر شماره تعدادی از این الگوهای طراحی و نحوه پیاده سازی آنها به زبان جاوا را بیان کنیم.

تاریخچه

مدت زمانی طولانی مهندسان نرم افزار همه به مشکلاتی یکسان در هنگام توسعه و تولید نرم افزارها برمی‌خوردند. هر توسعه دهنده و معماری بنا به تجربه خود به نحوی این مشکلات را حل می‌کرد و هیچ گونه استانداردی برای حل این مشکلات وجود نداشت. از مشکلاتی که از عدم وجود استاندارد ایجاد می شد، می توان به زمان‌هایی اشاره کرد که عضوی جدید (چه فردی با تجربه ، چه فردی تازه کار) به پروژه اضافه می شد و به علت عدم آشنایی فرد جدید با ساختار پروژه مدت زمان مدیدی را صرف آشنایی با آن می کرد، که به هیچ عنوان دلخواه کسی نبود. همچنین داشتن یک فرهنگ واژگانی مشترک باعث می شد که به سادگی توسعه دهندگان، صرفاً با بیان اسم الگوی طراحی مد نظرشان، با یکدیگر ارتباط برقرار کنند. وجود یک استاندارد باعث به وجود آمدن روش های کارآمد، بهینه و تست شده می‌شود. همچنین با وجود چنین روش های استانداردی فرایند توسعه و نگهداری نرم افزار نیز تسریع می‌شود.

همانطور که گفتیم، الگو های طراحی راه حل‌هایی قابل تکرار، برای مشکلاتی رایج در فرایند توسعه و طراحی نرم افزار هستند. اما از طرفی الگوهای طراحی مثل کتابخانه یا کد های آماده نیستند که به سادگی مشکل را حل کنند، بلکه الگو های طراحی مثل غالب ها یا توصیفاتی هستند که به ما می گویند که چگونه مشکل را حل کنیم.

در سال ۱۹۹٤ گروهی چهار نفره که بعدها به GOF (Gang of Four) معروف شدند و متشکل از : Erich Gamma, Richard Helm, Ralph Johnson John Vlissides بودند کتابی در مورد الگوهای طراحی نوشتند که پایه گذار آنچه که ما امروزه از الگوهای طراحی می‌دانیم شد. این چهار نفر ۲۳ الگوی طراحی را در سه دسته Creational, Structural و Behavioral معرفی کرده‌اند. هرکدام از این دسته‌ها در پی حل مشکلاتی متفاوت اما از یک جنس هستند. الگوهای Creational در پی حل مشکلاتی هستند که مربوط به ساختن نمونه از اشیاء و کلاس‌ها هستند و این فرایند را کنترل می‌کنند. از طرفی Structural پترن ها در پی معرفی روش هایی هستند که چگونه اشیا و کلاس‌ها در کنار هم قرار بگیرند تا اجزاء بزرگ‌تر و پیچیده‌تری ساخته شوند. الگو های طراحی Behavioral نیز در پی تعریف چگونگی رفتار اشیاء و کلاس‌ها و همچنین چگونگی شیوه ارتباط آنها با یکدیگرند.

Observer

اولین الگوی طراحی که در مورد آن صحبت می‌کنیم این الگوی طراحی است که در دسته بندی الگو های طراحی Behavioral قرار می گیرد. هدف این الگو تعریف کردن یک وابستگی یک به چند بین اشیا است. به گونه ای که اگر در حالت یک شی تغییری رخ داد، تمامی اشیا وابسته به آن، به صورت خودکار از آن تغییر آگاه شوند.

مفهوم

در این الگوی طراحی چندین شی مشاهده‌گر (Observer) وجود دارند که در حال مشاهده تعدادی موضوع (Object) یا شی قابل مشاهده (Observable) هستند. در واقع با ایجاد تغییر در حالت یک شی Observable تمام Observer ها از آن آگاه می شوند و متناسب با آن تغییر واکنشی نشان خواهند داد.

اشیا Observer می توانند به سادگی یک شی را مشاهده کنند و تغییرات آن را رصد کنند یا در صورت نیاز دیگر از این تغییرات آگاه نشوند و دیگر آن شی را رصد نکنند. در برنامه نویسی گرافیکی خصوصا در JavaFX این الگوی طراحی نقش بسیار زیادی ایفا می کند. تصور کنید که برنامه شما لیستی از اسامی را از دیتابیس می‌خواند و آنها را در برنامه نشان می‌دهد. کاربر می‌تواند هر یک از این این اطلاعات نمایش داده شده را که در یک فرضا در یک جدول نشان داده می‌شود ویرایش کند. با ویرایش هر یک از این فیلدهای رابط گرافیکی، دیتابیس برنامه نیز باید ویرایش شود. چنین اعمالی از طریق Propertyها و Observable Collectionهای JavaFX به سادگی قابل انجام است.[٦]

روش پیاده سازی

یکی از روش های پیاده سازی این الگوی طراحی استفاده از یک لیست از observerها است که در صورت تغییر در شی مورد نظر ، متد update این اشیا را صدا کنیم.

پیاده سازی

```
import java.util.ArrayList;
import java.util.List;
interface IObserver {
   void update(int i);
}
class Observer1 implements IObserver {
   @Override
   public void update(int i) {
      System.out.println("Observer1: myValue in Subject is now: " + i);
   }
}
class Observer2 implements IObserver {
   @Override
   public void update(int i) {
      System.out.println(
          "Observer2: observes ->myValue is changed inSubject to:" + i);
   }
}
interface ISubject {
   void register(IObserver o);
   void unregister(IObserver o);
   void notifyObservers(int i);
}
class Subject implements ISubject {
   private int myValue;
   public int getMyValue() {
      return myValue;
   }
```

```
    public void setMyValue(int myValue) {
        this.myValue = myValue;
        //Notify observers
        notifyObservers(myValue);
    }
    List<IObserver> observersList = new ArrayList<>();

    @Override
    public void register(IObserver o) {
        observersList.add(o);
    }
    @Override
    public void unregister(IObserver o) {
        observersList.remove(o);
    }
    @Override
    public void notifyObservers(int updatedValue) {
        for (int i = 0; i < observersList.size(); i++) {
            observersList.get(i).update(updatedValue);
        }
    }
}
class ObserverPattern {
    public static void main(String[] args) {
        System.out.println("*** Modified Observer Pattern Demo***\n");
        Subject sub = new Subject();
        Observer1 ob1 = new Observer1();
        Observer2 ob2 = new Observer2();
        sub.register(ob1);
        sub.register(ob2);
        sub.setMyValue(5);
        System.out.println();
        sub.setMyValue(25);
        System.out.println();
        //unregister ob1 only
        sub.unregister(ob1);
        //Now only ob2 will observe the change
        sub.setMyValue(100);
    }
}
```

نتیجه پیاده سازی

```
*** Observer Pattern Demo3 ***

Observer1: myValue in Subject1 is now: 50
Observer2: observes ->myValue is changed in Subject1 to :50

Observer2: observes ->myValue is changed in Subject2 to :250
Observer3 is observing:myValue is changed in Subject2 to :250

Observer1: myValue in Subject1 is now: 550

Observer2: observes ->myValue is changed in Subject2 to :750
Observer3 is observing:myValue is changes in Subject2 to :750
```

Singleton

دومیـن الگـوی طراحـی کـه در مـورد آن صحبـت می‌کنیـم ایـن الگـوی طراحـی Singleton نـام دارد کـه در دسـته الگـو هـای Creational قـرار می‌گیـرد. گاهـی اوقـات نیـاز داریـم تـا مطمـئن شـویم کـه از یـک کلاس فقـط و فقـط یـک نمونـه داشـته باشـیم و دسترسـی بـه آن نمونـه از هـر جایـی امـکان داشـته باشـد.

مفهوم

هـمان طـور کـه در تعریـف مشـخص اسـت بایـد از کلاس تنهـا یـک شـی داشـته باشـیم و هـر موقـع بـه آن کلاس نیـاز داشـتیم آن شـی را صـدا کنیـم. صرف دسترسـی دادن بـه صـورت اسـتاتیک شـاید مشـکل را تـا حـدی حـل کنـد امـا نیـاز بـه داشـتن فقـط و فقـط یـک نمونـه را بـر طـرف نمی‌کنـد. اسـتفاده صحیـح و درسـت از ایـن الگـوی طراحـی خـود جـای بحـث جداگانـه ای دارد و حتـی عـده ای Singleton را یـک پادالگـو (Anti-Pattern) مـی داننـد[۱و۲و۳و۴]. بـه عنـوان مثـال یکـی از مشـکلاتی کـه در صـورت اسـتفاده بیـش از حـد از ایـن الگـوی طراحـی ایجـاد مـی کنـد سـخت تـر کـردن فراینـد تسـت نـرم افـزار اسـت. یکـی از مـوارد اشـتباه اسـتفاده از ایـن الگـو برطـرف کـردن نیـاز بـه دسترسـی global بـه یـک شـی اسـت. فـارغ از اینکـه آن شـی بایـد ماهیتـاً فقـط و فقـط یـک نمونـه داشـته باشـد یـا نـه. در واقـع تحـت هـر شرایطـی نبایـد از ایـن الگـو اسـتفاده کـرد و دامنـه کاربـرد ایـن الگـو نیـز آنقـدر زیـاد نیسـت کـه در یـک برنامـه چندیـن کلاس نیـاز بـه تعریـف بـه صـورت Singleton داشـته باشـند.

ایـن روش معمـولا بـرای مدیریـت منابـع و متمرکـز کـردن آنهـا اسـتفاده می‌شـود. مثـلا ایجـاد یـک Log File در کل برنامـه و اسـتفاده از آن می‌توانـد یکـی از کاربردهـای ایـن الگـوی طراحـی باشـد. امـا بازهـم تکـرار مـی کنیـم کـه یکـی از مـوارد اشـتباه اسـتفاده از آن دادن دسترسـی global بـه آن منبـع اسـت، کـه راه حـل بهـتر آن پـاس دادن آن منبـع یـا Dependency Injection اسـت.

روش پیاده سازی

دو روش رایج برای پیاده سازی این الگو وجود دارد :

۱) در ابتدا شی را بسازیم (Eager) .

۲) تا زمان اولین فراخوانی ساختن شی را به تعویق بیندازیم (Lazy) .

پیاده سازی Eager

```
public class EagerSingleton {
    private static final EagerSingleton instance = new EagerSingleton();
    //private constructor to avoid client applications to use constructor
    private EagerSingleton() {
    }
    public static EagerSingleton getInstance() {
        return instance;
    }
}
```

پیاده سازی Lazy

```
public class LazySingleton {
    private static LazySingleton instance;
    private LazySingleton() {}
    public static LazySingleton getInstance() {
        if (instance == null) {
            instance = new LazySingleton();
        }
        return instance;
    }
}
```

پیـاده سـازی فـوق Thread-Safe نیسـت. مـی تـوان بـا synchronized کـردن متـد getInstance() آن را Thread-Safe کـرد. امـا پیـاده سـازی رایـج دیگـری نیـز موجـود اسـت کـه Bill Pugh بـه آن دسـت یافـت و Thread-Safe نیـز هسـت. ایـن روش اصطلاحـا [5] Double-checked locking نـام دارد. در نـگاه اول شـاید تفـاوت چندانـی بـا پیـاده سـازی Eager وجـود نداشـته باشـد امـا بـه علـت سـاختار مدیریـت حافظـه JVM و نحـوه بارگـذاری کلاس هـا در واقـع ایـن پیـاده سـازی هـم Lazy بـوده و هـم Thread-Safe.

پیاده سازی Bill Pugh

```
public class BillPughSingleton {
    private BillPughSingleton(){}
    private static class SingletonHelper{
        private static final BillPughSingleton INSTANCE = new BillPughSingleton();
    }
    public static BillPughSingleton getInstance(){
        return SingletonHelper.INSTANCE;
    }
}
```


1- http://misko.hevery.com/2008/08/17/singletons-are-pathological-liars/

2- http://wiki.c2.com/?SingletonsAreEvil

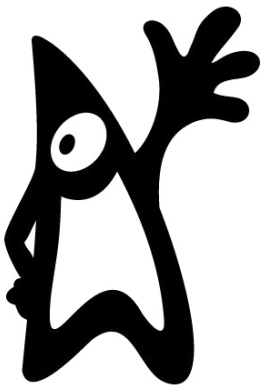
3- https://blogs.msdn.microsoft.com/scottdensmore/2004/05/25/why-singletons-are-evil/

4- https://testing.googleblog.com/2008/11/clean-code-talks-global-state-and.html

5- https://en.wikipedia.org/wiki/Double-checked_locking

6- http://www.developer.com/java/data/understanding-java-observable-and-javafx-observable.html

خودتان را محک بزنید



حسین ریماز

سعی کنید بدون کمک IDE و اجرا کردن کدها و کمک اینترنت به سوالات زیر پاسخ دهید و جواب های خود را به همراه دلیل آن برای ما ارسال کنید.

سوال ۱: کدام یک از گزینه‌های زیر میتوانند به صورت موفقیت آمیز به جای line n1 قرار بگیرند؟ (هر خط جداگانه وارد میشود)

```
// line n1
switch (x) {}
```

a) boolean x = false;

b) short x = 99;

c) int x = 0;

d) long x = 0;

e) StringBuilder x = new StringBuilder(“x”);

سوال ۲: قطعه کد زیر داده شده است:

```
List<String> ls = Arrays.asList(“KNTU”, “Java”, “User”, “Group”);
Set<String> s1 = new HashSet<>(ls); // line n1
Set<String> s2 = new TreeSet<>(ls); // line n2
System.out.println(s1.equals(s2));
```

کدام یک از گزینه‌های زیر صحیح است:

الف) خط n1 باعث به وجود آمدن خطای کامپایل است.

ب) هر دو خط n1 , n2 باعث به وجود آمدن خطای کامپایل می شوند.

پ) در خط n۱ اکسپشن پرتاب خواهد شد.

ت) خروجی true خواهد بود.

ث) خروجی false خواهد بود.

سوال ۳: کدام یک از گزینه های زیر در مورد الگوهای طراحی صادق است (تمام گزینه‌های درست را انتخاب کنید)

الف) الگوهای طراحی، قطعه کدهایی هستند که می توان آنها را در برنامه بدون تغییر کپی کرد.

ب) الگوهای طراحی، راه حل‌های مفهومی قابل تکرار هستند.

پ) الگوهای طراحی، میانبرهایی (Shortcut) برای صحبت در مورد کدها هستند.

ت) از هر الگوی طراحی فقط یکبار می‌توان در برنامه ها استفاده کرد.

ث) الگوهای طراحی، کتابخانه هایی هستند که می توان از آنها در برنامه‌هایمان استفاده کنیم.

سوال ۴: کد زیر داده شده است:

```
3. import java.util.*;
4. class Dog { int size; Dog(int s) { size = s; } }
5. public class FirstGrade {
6.     public static void main(String[] args) {
7.         TreeSet<Integer> i = new TreeSet<Integer>();
8.         TreeSet<Dog> d = new TreeSet<Dog>();
9.
10.        d.add(new Dog(1)); d.add(new Dog(2)); d.add(new Dog(1));
11.        i.add(1); i.add(2); i.add(1);
```

```
12.     System.out.println(d.size() + " - " + i.size());
13.  }
14. }
```

خروجی برنامه کدام یک از گزینه های زیر است؟

a) 1 – 2

b) 2 – 2

c) 2 – 3

d) 3 – 2

e) 3 – 3

f) Compilation fails

g) An exception is thrown at runtime

سوال ۵: چه تغییر در کد زیر نیاز است اعمال شود تا خروجی برنامه 12345 شود؟ (تمامی گزینه‌های ممکن را انتخاب کنید)

```
Stream.iterate(1, x -> x++).limit(5).map(x -> x).collect(Collectors.joining());
```

1) تغییر Collectors.joining() به Collectors.joining (“”)

2) تغییر map(x -> x) به map(x -> “” + x)

3) تغییر x -> x++ به x -> ++x

4) اضافه نمودن forEach(System.out::print) بعد از صدا زدن collect()

5) گذاشتن تمام کد در درون System.out.print()

6) هیچکدام، کد فوق به درستی خروجی 12345 را نشان می‌دهد.

سوال ۶: برای کامل کردن کد زیر به کدام یک از فانکشنال اینترفیس‌های زیر نیاز داریم؟ (تمام گزینه های ممکن را انتخاب کنید)

```
6: _____ x = String::new;
7: _____ y = (a, b) -> System.out.println();
8: _____ z = a -> a + a;
```

a) BiConsumer<String, String>

b) BiFunction<String, String>

c) BinaryConsumer<String, String>

d) BinaryFunction<String, String>

e) Consumer<String>

f) Supplier<String>

g) UnaryOperator<String>

h) UnaryOperator<String, String>