**BU NE LOG?**

Üniversiteden mezun olduktan sonra irili ufaklı pek çok projede çalıştım.Temel mühendislik yaklaşımı; bir problem var yani gereksinimler listesi, bunu bir tasarım yaklaşımı ile çözüyorsunuz.İşin doğası gereği geliştirdiğiniz üründe genellikle hatalar bunlunabiliyor.Bu hataları tasarım ya da kodlama aşamasında yakalamaya çalışyorsunuz. “debug” ederek kodlama aşamasının ilk başlarında pek çok hatayı yakalabiliryorsunuz ama kod büyüdükçe işler biraz daha karmaşık hale gelebiliyor.Hele büyük bir ekip içierisinde çalışıyorsanız diğer kişilerin geliştirdiği koddaki hataları yakalamak iyice zorlaşıyor. Bu durum karşında sistematik şekilde ortak kullanılan bir loglama yaklaşımı ile uygulamanın yaşantısını izlemek gerekiyor.

Projelerde bu konu genellik çok fazla önemsenmiyor.Çünkü müşteri gereksinimlerinde doğrudan log alınmasını istiyorum diye bir gereksinimle karşılaşmıyorsunuz.Bakım sürecinin temel taşlarından biri olan log verilerinin daha iyi toplanması konusunu daha fazla ciddiye almak gerektiğini düşünenlerdenim diyebiliriz.

Ne demek istiyorum? Yazılım dünyasında kullanılan teknolojiye bağlı olarak çok sayıda loglama kütüphanesi var ve çoğu geliştirici her gün bir veya daha fazlasını kullanabiliyor. Mesela Java geliştiricileri için en yaygın kullandığı kütüphanelerden ikisi log4j ve logback'dir. Basit ve kullanımları kolay ve harika çalışan bu yazılım kütüphaneleriyle log alma işlemi oldukça kolay bir şekilde yapılabiliyor. Ancak sadece log almak, yazılımın sağlık durumunu takip etmek için belli bir boyuttan sonra yeterli gelmiyor. Benzer durum diğer programlama dilleri içinde geçerlidir.Uygulamaların geliştirme aşaması tamamlandıktan sonra log dosyalarıyla çalışmak zorunda kaldığımızda bir kaç tane acı gerçek karşımıza çıkıyor.Bunlar;

* Verinin çok fazla olması
* Bu çok fazla veriye erişmemiz gerekmesi
* Bu çok fazla verinin birden çok sunucuya veya servise/uygulamaya(gömülü uygumalar için) yayılmış olması
* Belirli/Özel bir işlemin uygulamalara dağıtılmış olması- bu yüzden daha fazla log içinde arama yapmanız gerekmesi.
* Loglar içerisinde sorgula yapmanızın oldukça zor olması ; SQL ile sorgulama yapmayı isteseniz bile, kullanılabilir hale getirmek için metinlerin hepsini indekslemeniz gerekecektir.(Püff..)
* Düzenli olmamasından dolayı okunmasının oldukça zor olması.(Karma karışık bir log dosyası bir işe yaramaz genellikle..)
* Yardımcı olabilecek ayrıntıları içermemesi. (Log.Info (" Fonksiyona girdi ") Çok yardımcı olacak bir log değil mi☺ Genelde geliştiriciler böyle logları pek severler.1 hafta sonra neden yazıldığı anlaşılmayan loglar.)
* Log dosyalarının saklanması gibi işleri yönetmenin maliyetli olması.

şeklinde özetlenebilir. Bu noktada log alma işlemini ciddiye almak gerektiğini siz de farkına varmışınızdır artık. ☺

Çalışmayan veya hata veren bir uygulamanın, log mesajları (istisnalar dahil), çoğunlukla uygulamanın neden düzgün çalışmadığını hızlı bir şekilde keşfetmek için çok önemli bir vazife görmektedir.Elbetde,performans izleme araçları ile hafıza kaçaklarını ve performans dar boğazlarını bulabiliriz.Fakat bu araçlar sizin o anda karşılaştınız bir problemi çözmek için yeterli olamayabilirler.Örneğin neden telemetri verisi işlenmiyor ya da MIL-STD-1553 sürücüsü neden zamanında veriyi göndermiyor gibi..Bilerek gömülü sistemlerden örnek veriyorum.Kaynakların kısıtlı kullanılması gereken sistemlerde log alma işlemine kaynak ayırmak epey tartışmalı bir konu.Buna daha sonra ilerleyen kısımlarda zaman zaman deyinmeye çalışacağım.

Loglama kültürü ve yaklaşımı nasıl olmalı peki;

Mümkün oldukça herşeyin log’unu almalıyız.Bu noktada ikiye bölünebiliriz.Fazladan yük getirmez mi? Kapsama bağlı olarak loglama yapılsa daha iyi olmaz mı?Ben şimdilik herşey diyorum.!

Bütün logların, tüm geliştiriciler tarafından kullanılabilecek ve kolayca ayrıştırılmasını mümkün kılacak merkezi bir konumda birleştirmeliyiz. Ayrıca, yazılımın geliştirmemize yardımcı olacak günlük ve istisna durumu loglarını daha iyi ele almak için yeni yollar bulmaya çalışmalıyız.İçinden bu nasıl olacak?.. zaten bilsem/bulsam hemen uygularım.. diyenleri duyar gibiyim.Bu konuda bende önerilerinize açığım.

Herşeyi loglamak; Aşağıda genellik geliştiricilerin loglama mesajını görebilirsiniz.

}catch (exception e)

{

Logger.error(e.getMessage(),e)

}

En azında adam exception handling yapıyor diyebiliriz.Bir diğer sık karşılaştığımız log örneğide;

Logger.debug(“İşlem tamamlandı”) ..Gerçekten çok açıklayıcı bunu anlamıyorsanız.Sorun kesinlikle sizde.:)

Bu kötü örneklerden sonra;şöyle demeliyiz ,amacımız: debug ortamına bağlanamadığımız fiziksel ortamlarda bağlama uygun,ilişkili verileri kayıt altına almak ve böylelik o samanlıkta iğne arama olayını bir nebzede olsa küçük bir mıknatıs ile yapmaya çalışmak olarak tanımlayabiliriz.Bu bağlamda tavisyelerim aşağıda yer alıyor.

**Tavsiye 1:** Kaynak kod içerisinden her seviyede (debug ve trace seviyesinde de) bilgi toplamalısınız.

Bir fonksiyona geçen parametrelerin (hata durumunda) kayıt altına alınması gerekir.(Kod örneklerim daha çok pseudo kod gibi değerlendirmenizi tavsiye ediyorum.Derlediğinizde doğrudan çalışmayacaktır.)

Apple makeApple(String type, Double volume) {

Logger.debug("Elma yaratma işlemine giriş.");

try {

Apple anApple= new Apple(type, volume.doubleValue());

Logger.debug("{}", anApple->toString());

return anApple;

}

catch (Exception e)

{

Logger.error(e.getMessage(), e);

//Sadece hata ile ilgili değil.Hata durumuna neden olan parametre bilgilerininde //loglanması gerekiyor.Fonksiyona ne girdi ne çıkıyor bilinmesi faydalı olacaktır.

}

return null;

}

Bu fonksiyonu şu şekilde çağrıldığını varsayılım;

Object.makeApple(“Amasya”, Double.valueOf(3.14));

Object.makeApple(“Golden”, null);

DEBUG ..: Elma yaratma işlemine giriş.

DEBUG ..: Elma [type=Amasya, volume=3.14]

DEBUG ..: Creating a Apple

ERROR ..: java.lang.NullPointerException

2.satır hangi parametreler ile ne iş yapıldığını anlamak için işe yarar bir veri oluşturuyor.Bunun için log aldığımız yerde toString fonksiyonunun çağırılmasını sağlıyoruz.Bu işlem farklı dillerde farklı özellikler ile sağlanabilir.(Java’da toString fonksiyonu sınıfa eklenince büyük ihtimal ile çağırmaya gerek kalmayacaktır.)

**Tavsiye 2:** “Diagnostic Contexts” kullanarak (çok istemcili/parçacıklı yapılarda) bağlam hakkında daha fazla bilgi loglanmasını sağlamalısınız.

Bir önceki tavsiyemiz içerisinde yer alan örnekteki loglama işleminin birden fazla thread veya istemci ile yapıldığını varsayalım.Üsteki şekilde loglama işlemi yaptığımızda hangi istemcinin bu işlemi gerçekleştirdiği hakkında bir fikrimiz olmayacaktır.İstemci olarak sadece istemci-sunucu mimarisindeki istemci gelmesin, bir modül tarafından sunulan herhangi bir hizmet olarak düşünebilirsiniz.Bu şekilde aslında geniş kapsamlı değerlendirebilirsiniz. Gömülü sistemlerden,dağıtık uygulamalara kadar geniş bir yelpazeyi kapsayabilir.Ama gömülü sistemler için biraz daha farklı ele almak gerekecektir.

Gömülü sistemlerde Macro tarzında Log alma fonksiyonlarının eklenmesini öneriyorum.Klasik bir log alma kütüphanesi geliştirilebilir ama bu kütüphane predefine macroların kullanıldığı bir arayüz sunarsa loglama işlemi daha kolay yönetilebilir.Böylece log alma işleminin memory foot print’i derleme zamanında ayarlanabilir.Bu durum domainden domain’e farklılık gösterebilir. IoT alanında cihaz içerisinde SD kartlar konulabilirken,uzay alanında emniyet –kritiklik unsurunun çok yüksek olmasından ve uzay koşullarından dolayı depolama alanı olarak MRAM benzeri yapılar kulllanılmaktadır.Bu pahalı bileşenlerde log bilgilerini depolamak çok iyi bir tasarım olmayacaktır.Lakin yazılım doğrulama aşamasının devam ettiği koşullarda loglama işleminin ,özellikle yazılım entegrasyon testlerinde, aktif olmasını tavsiye ediyorum.Bu durum “Verification&Validation”,”test as you fly” kavramları gibi pek çok konunun “trade off”’unu gerektiriyor.En basitinden log alma işlemini macrolar ile yazılımdan derleme zamanında devre dışı bıraktığınızda kod kapsama analizi ve object kodunuz değişecektir. Bu durum kalifikasyon ihtiyacının yüksek olduğu projeler için çok tercih edilmeyebilir.Bu konuyu şimdilik bu kadarla bırakıyorum.!

Java ve C++ ‘da bağlam kullanımı konusunda aşağıdaki linklerden faydalanabilirsiniz.

<https://www.boost.org/doc/libs/1_67_0/libs/log/doc/html/log/design.html>

https://logback.qos.ch/manual/mdc.html.

Buraya kadarki kısımda aklımıza gelen ilk soru, herşeyin logunu alırsak sisteme ek yük gelmeyecek mi? Bu noktada loglama seviyelerini doğru kullanmak gerekiyor.Böylelikle geliştirme ve yayın aşamalarında farklı loglama seviyelerini sadece konfigürasyon değişikliği ile kullanıp,ek yük oluşmasını azaltabiliriz.Ama unutmamalıyız ki loglama işlemi, hiç bir hata durumu kadar yük getirmeyecektir.Diğer bir hususta asekron olarak loglama işlemini gerçekleştiriyorsanız loglama maliyeti yine düşecektir.Son olarak, ilk başlarda söylediğim gibi “Bütün logları, tüm geliştiriciler tarafından kullanılabilecek ve kolayca ayrıştırılmasını mümkün kılacak merkezi bir konumda birleştirmeliyiz”.

**Tavsiye 3:**Loglarınızı yapısal bir formatta saklamalısınız.

Loglarınız JSON formatında saklayabilirsiniz. Aşağıdaki link içinde kullanmayı tercih edebileceğiz formatların karşılaştırmaları yer alıyor.Yapısal formatda saklanan loglarınız JIRA ve ElasticSearch gibi uygulamalara entegre ederek “arama yapma”, “detayları keşif etme”, “takip etme”(hata oranları, çözümlenmiş hatalar & yeni hatalar gibi) işlevlerini gerçekleştirerek loglardan daha fazla faydalanabilirsiniz.

**ÖZET:**

* Herşeyi loglayın; Kaynak kod içerisinden her seviyede (debug ve trace seviyesinde de) bilgi toplamalısınız.
* “Diagnostic Contexts” kullanarak (çok istemcili/parçacıklı yapılarda) bağlam hakkında daha fazla bilgi loglanmasını sağlamalısınız.
* Loglarınızı yapısal bir formatta saklamalısınız. Örnek:JSON

Bundan sonra bu ne log demezsiniz.Sağlıkla kodlamaya devam..