Debugowanie

➢ ➢Poznawanie głównych procesów uruchamiania i debugowania

➢ ➢Obsługa narzędzia Android Device Monitor Tool

➢ ➢Przechodzenie przez ważne skróty i techniki Android Studio

W poprzedniej Części dowiedziałeś się, jak używać i refaktoryzować najpopularniejsze narzędzia dla programistów Androida - SDK Manager, Android Virtual Device (AVD) Manager i Navigation Editor. W tej części omówimy podstawy procedury debugowania Android Studio. Ponadto pokażemy, jak nawigować za pomocą Android Device Monitor i przedstawimy listę przydatnych skrótów i technik przycisków, które można zastosować podczas tworzenia aplikacji w Android Studio. Zwykle im bardziej złożona jest Twoja aplikacja, tym większe prawdopodobieństwo, że będzie zawierała błędy. Nic nie denerwuje użytkownika bardziej niż aplikacja, która ulega awarii, nie działa w określonych warunkach lub przeszkadza w realizacji zadania, które miała wykonać. Naiwnym podejściem do programowania jest myślenie, że Twój kod zawsze będzie ewoluował zgodnie z wyznaczonymi dla niego ścieżkami. Ważne jest, aby zrozumieć, od czego odbiega Twój kod, ale nawet świadomość tego nie może zagwarantować, że pójdzie dokładnie bezbłędnie. A ponieważ nie można przewidzieć wszystkich potencjalnych wadliwych ścieżek podczas programowania, pomaga to w zrozumieniu różnych narzędzi diagnostycznych i technik związanych z tworzeniem systemu Android. W tym rozdziale szczegółowo omówimy debugger i przyjrzymy się innym narzędziom analitycznym, które można wykorzystać do naprawienia błędów i uzyskania wglądu w potencjalne słabości, które mogą Cię powstrzymywać podczas pracy.

URUCHAMIANIE I DEBUGOWANIE

Android Studio Debugger to świetne narzędzie, które umożliwia wykonywanie następujących czynności i nie tylko:

• Wybierz urządzenie, na którym chcesz debugować aplikację

• Ustal punkty przerwania w kodzie Java, Kotlin i C/C++

• Przeglądanie zmiennych i szacowanie wyrażeń w czasie wykonywania

Jednak zanim zaczniesz debugować, musisz wykonać następujące proste przygotowania. Po pierwsze, zaczynasz od włączenia debugowania na swoim urządzeniu. Jeśli używasz emulatora, musi to być domyślnie włączone. Ale w przypadku podłączonego urządzenia musisz włączyć debugowanie w opcjach programisty urządzenia. Gdy to zrobisz, upewnij się, że uruchomiłeś wariant kompilacji, który można debugować. Należy użyć wariantu kompilacji, który zawiera debugowalną wartość true w konfiguracji kompilacji. Zwykle wystarczy wybrać domyślny wariant „debugowania”, który jest zawarty w każdym projekcie Android Studio. Jeśli jednak zamierzasz używać nowych typów kompilacji, które powinny być debugowalne, musisz dodać „debuggable true” do typu kompilacji, jak pokazano tutaj:

```
android {

buildTypes {

customDebugType {

debuggable true

...
```

}

Zauważ, że ta sama właściwość dotyczy również modułów z kodem C/C++. Jeśli Twoja aplikacja opiera się na module biblioteki, który chcesz również debugować, ta biblioteka powinna być spakowana z możliwością debugowania true, aby przechowywać jej symbole debugowania. Aby upewnić się, że możliwe do debugowania warianty projektu aplikacji otrzymują możliwy do debugowania wariant modułu biblioteki, zaleca się opublikowanie niedomyślnych wersji biblioteki. Kolejną rzeczą, którą musisz wykonać, jest ustawienie punktów przerwania w kodzie aplikacji. Na pasku narzędzi wybierz urządzenie do debugowania aplikacji z menu rozwijanego urządzenia docelowego. Jeśli nie masz skonfigurowanych urządzeń, musisz albo podłączyć urządzenie przez USB, albo utworzyć AVD, aby korzystać z emulatora Androida. Na pasku narzędzi wybierz opcję Debuguj. Jeśli zobaczysz okno dialogowe z pytaniem, czy chcesz „przełączyć się z uruchamiania na debugowanie", oznacza to, że Twoja aplikacja jest już uruchomiona na urządzeniu i zostanie uruchomiona ponownie, aby rozpocząć debugowanie. Jeśli wolisz pozostawić uruchomione to samo wystąpienie aplikacji, kliknij Anuluj debugowanie i zamiast tego dołącz debuger do uruchomionej aplikacji.

W przeciwnym razie Android Studio zbuduje pakiet aplikacji Android (APK), podpisze go kluczem debugowania, zainstaluje na wybranym urządzeniu i aktywuje. W przypadku dodania kodu C i C++ do projektu, Android Studio uruchamia również debuger LLDB w oknie debugowania, które jest używane do debugowania kodu natywnego. A ponieważ do debugowania kodu Java/Kotlin i kodu C/C++ wymagane są różne narzędzia debugera, debuger Android Studio pozwala wybrać typ debugera, którego chcesz użyć. Domyślnie Android Studio decyduje, który debuger zastosować, na podstawie języków, które śledzi w projekcie (z typem Autodebugger). Niemniej jednak możesz ręcznie wybrać debuger w konfiguracji debugowania (kliknij Uruchom > Edytuj konfiguracje) lub w oknie dialogowym, które pojawia się po kliknięciu Uruchom i dołącz debuger do procesu systemu Android.

Istnieje kilka najczęściej używanych typów debugowania:

• Auto: wybierz ten typ debugowania, jeśli chcesz, aby Android Studio automatycznie wybierało najlepszą opcję dla debugowanego kodu. Na przykład, jeśli masz w projekcie dowolny kod C lub C++, Android Studio automatycznie używa typu debugowania Dual. W przeciwnym razie Android Studio stosuje typ debugowania Java.

• Java: Ten typ debugowania jest najbardziej odpowiedni, jeśli chcesz debugować tylko kod napisany w Javie lub Kotlinie. Jednocześnie debuger Java po prostu pomija wszelkie punkty przerwania lub zegarki, które wstawiasz do kodu natywnego.

• Natywny (dostępny tylko z kodem C/C++): Ten typ debugowania jest idealny, jeśli chcesz używać tylko LLDB do debugowania kodu. Podczas korzystania z tego typu debugowania widok sesji debugera Java nie będzie dostępny. Domyślnie LLDB przegląda tylko twój kod natywny i pomija punkty przerwania w kodzie Java. Dlatego jeśli chcesz debugować kod Java, powinieneś przełączyć się na typ debugowania automatycznego lub podwójnego. Ponadto debugowanie natywne działa tylko na urządzeniach, które spełniają następujące wymagania:

1. Urządzenie musi obsługiwać tryb run-as: Aby sprawdzić, czy urządzenie obsługuje tryb run-as, dodaj następujące polecenie w powłoce ADB połączonej z twoim urządzeniem:

run-as your-package-name pwd

Tutaj musisz zastąpić nazwę swojego pakietu nazwą pakietu swojej aplikacji. Jeśli urządzenie obsługuje tryb run-as, polecenie powinno powrócić bez żadnych błędów.

2. Urządzenie ma włączone ptrace: Aby sprawdzić, czy ptrace jest włączone, dodaj następujące polecenie w powłoce ADB, która jest połączona z twoim urządzeniem:

sysctl kernel.yama.ptrace_scope

Jeśli ptrace jest włączone, polecenie będzie miało wartość 0 lub pokaże nieznany błąd klucza. Jeśli ptrace nie jest włączone, wypisze każdą wartość inną niż 0.

• Podwójny (dostępny tylko z kodem C/C++): Podwójny typ debugowania jest idealny dla tych, którzy muszą przełączać się między debugowaniem zarówno Javy, jak i kodu natywnego. Android Studio dołącza zarówno debuger Java, jak i LLDB do procesu aplikacji, jeden dla debugera Java i jeden dla LLDB, dzięki czemu możesz śledzić punkty przerwania w kodzie Java i natywnym bez konieczności ponownego uruchamiania aplikacji lub zmiany konfiguracji debugowania.

Podczas debugowania kodu C/C++ możesz także ustawić specjalne typy punktów przerwania, zwane punktami obserwacyjnymi, które mogą opóźnić proces aplikacji, gdy aplikacja wchodzi w interakcję z określonym blokiem pamięci.

Aby wyświetlić wszystkie punkty przerwania i skonfigurować typy punktów przerwania, przejdź do widoku punktów przerwania po lewej stronie okna debugowania. Okno Punkty przerwania pozwala włączyć lub wyłączyć każdy punkt przerwania z listy punktów przerwania. Jeśli punkt przerwania jest wyłączony, Android Studio nie wstrzymuje aplikacji po napotkaniu tego punktu przerwania. Możesz skonfigurować punkt przerwania, aby był najpierw wyłączony i włączyć go po trafieniu innego punktu przerwania. Możesz także zmienić, czy punkt przerwania powinien być wyłączony po trafieniu. Aby ustawić punkt przerwania dla dowolnego wyjątku, po prostu wybierz Punkty przerwania wyjątku na liście punktów przerwania.

Debuguj ramki okien

W oknie Debuger funkcja Frames umożliwia sprawdzenie ramki stosu, która spowodowała trafienie bieżącego punktu przerwania. Dzięki temu możesz uzyskać dostęp do ramki stosu i zbadać ją, a także sprawdzić listę wątków w aplikacji na Androida. Aby wybrać wątek, użyj menu rozwijanego wyboru wątku i wyświetl jego ramkę stosu. Kliknięcie elementów w ramce spowoduje wyświetlenie źródła w edytorze.

Sprawdź zmienne

W oknie Debuger funkcja Variables umożliwia śledzenie zmiennych, gdy system zatrzyma aplikację w punkcie przerwania i wybierzesz ramkę z okienka Ramki. Panel Zmienne umożliwia również ocenę wyrażeń ad hoc przy użyciu metod statycznych i zmiennych dostępnych w wybranej ramce. Funkcja Watches zapewnia podobne możliwości, z wyjątkiem tego, że wyrażenia zawarte w funkcji Watches są zachowywane między sesjami debugowania. Należy dodać zegarki dla zmiennych i pól, do których często uzyskujesz dostęp, lub zapewnić przydatny stan dla bieżącej sesji debugowania. Aby dodać zmienną lub wyrażenie do listy Zegarki, wykonaj następujące proste kroki. Po rozpoczęciu debugowania przejdź do funkcji Zegarki i kliknij Dodaj. W wyświetlonym polu tekstowym wpisz nazwę zmiennej lub wyrażenia, które chcesz obserwować, i naciśnij klawisz Enter. Aby usunąć pozycję z listy zegarków, wybierz ją i po prostu kliknij Usuń. Można również zmienić kolejność elementów na liście zegarków, wybierając element, a następnie klikając w górę lub w dół.

Co więcej, podczas debugowania kodu C/C++ można również ustawić punkty obserwacyjne, wybierając konkretną zmienną lub, mówiąc bardziej szczegółowo, wybierając blok pamięci, który system alokuje do tej zmiennej, a nie do samej zmiennej. Różni się to od dodawania zmiennej do funkcji Zegarki, która

umożliwia zobaczenie wartości zmiennej, ale nie pozwala na odroczenie procesu aplikacji, gdy system zostanie uruchomiony lub zmieni jej wartość w pamięci. Niemniej jednak, aby ustawić punkt obserwacyjny, należy spełnić następujące wymagania:

Po pierwsze, docelowe urządzenie fizyczne lub emulator powinien używać procesora x86 lub x86_64. Jeśli twoje urządzenie używa procesora ARM, musisz wyrównać granicę adresu zmiennej w pamięci do 4 bajtów dla procesorów 32-bitowych lub 8 bajtów dla procesorów 64-bitowych. Możesz wyrównać zmienną w swoim natywnym kodzie, określając __attribute__((aligned(num_bytes))) w spowolnieniu zmiennej, tak jak poniżej: int my_counter __attribute__((aligned(8))) Należy również pamiętać, że gdy Twoja aplikacja wychodzi z funkcji, a system zwalnia jej lokalne zmienne z pamięci, musisz ponownie przypisać wszystkie punkty obserwacyjne, które utworzyłeś dla tych zmiennych, aby ich nie utracić.

Wyświetl i zmień format wyświetlania wartości zasobów

W trybie debugowania można wyświetlić wartości zasobów i ustawić inny format wyświetlania zmiennych w kodzie Java. Po otwarciu karty Zmienne i wybraniu ramki przejdź do listy Zmienne i kliknij prawym przyciskiem myszy w dowolnym miejscu wiersza zasobów, aby wyświetlić listę rozwijaną. Z listy rozwijanej wybierz Wyświetl jako i wybierz format, którego chcesz użyć. Dostępne formaty zależą od typu danych wybranego zasobu. Standardowa lista formatów powinna zazwyczaj zawierać co najmniej jedną z następujących opcji:

• Klasa: Wyświetl definicję klasy

• toString: Wyświetl format ciągu

• Obiekt: Wyświetl definicję obiektu

• Tablica: Wyświetlaj w formacie tablicy

• Znacznik czasu: wyświetlaj datę i godzinę w następujący sposób: rrrr-mm-dd gg:mm:ss

• Auto: Android Studio wybiera najlepszy format na podstawie typu danych

• Binarny: Wyświetlaj wartość binarną za pomocą zer i jedynek

• MeasureSpec: wartość przekazana od rodzica do wybranego dziecka

• Hex: Wyświetlaj jako wartość szesnastkową

• Prymitywne: Wyświetlaj jako wartość liczbową przy użyciu prymitywnego typu danych

• Integer: Wyświetla wartość liczbową typu Integer

Informacje o konfiguracjach uruchamiania/debugowania

Konfiguracje uruchamiania/debugowania określają elementy, takie jak instalacja aplikacji, uruchamianie i funkcje testowania. Możesz ustawić konfigurację do jednorazowego użytku lub zapisać do dalszego użytku. Po zapisaniu możesz uzyskać dostęp do tej konfiguracji z listy rozwijanej Wybierz konfigurację uruchamiania/debugowania na pasku narzędzi. Android Studio zapisuje tam wszystkie konfiguracje w ramach projektu.

Domyślna konfiguracja uruchamiania/debugowania

Gdy po raz pierwszy uruchamiasz projekt, Android Studio tworzy domyślną konfigurację uruchamiania/debugowania dla głównego działania na podstawie szablonu Android App. Tak więc, aby uruchomić lub debugować swój projekt, zawsze musisz mieć przynajmniej jedną konfigurację

uruchamiania/debugowania określoną w taki sposób. Z tego powodu zaleca się, aby nie usuwać, ale raczej zachować domyślną konfigurację. Wszelkie konfiguracje uruchamiania/debugowania i modyfikacje szablonu będą miały zastosowanie tylko do bieżącego projektu. Aby otworzyć okno dialogowe Uruchom/Debuguj konfiguracje, wybierz opcję Edytuj konfiguracje, a okno dialogowe Uruchom/debuguj konfiguracje powinno pojawić się natychmiast. Możliwe jest również udostępnianie konfiguracji uruchamiania/debugowania (nie szablonu) za pośrednictwem systemu kontroli wersji. Okno dialogowe przedstawia domyślne szablony w lewym panelu w folderze Domyślne i łączy zdefiniowane konfiguracje według typu szablonu powyżej folderu Domyślne. Możesz zmienić rozmiar okna dialogowego, aby wyszukać ukryte elementy. W tym samym oknie dialogowym możesz także:

• Utwórz nowe konfiguracje uruchamiania/debugowania

• Edycja konfiguracji uruchamiania/debugowania

• Edytuj domyślne szablony

• Konfiguracje sortowania i grupowania

• Utwórz nową konfigurację uruchamiania/debugowania

Ponadto można ustanowić nowe konfiguracje uruchamiania/debugowania w oknie dialogowym Konfiguracje uruchamiania/debugowania, oknie projektu lub Edytorze kodu. Ale nowa konfiguracja powinna opierać się na domyślnym szablonie. Okno dialogowe Konfiguracje uruchamiania/debugowania wyświetla konfiguracje uruchamiania/debugowania oraz dostępne szablony domyślne. Nową konfigurację można rozpocząć bezpośrednio z szablonu lub z kopii innej konfiguracji, a następnie zmodyfikować wartości pól zgodnie z wymaganiami projektu. Alternatywnie możesz kliknąć prawym przyciskiem element w oknie Projekt, aby automatycznie utworzyć konfigurację specyficzną dla tego elementu. Na przykład możesz kliknąć prawym przyciskiem myszy plik Java działania i nacisnąć Uruchom, jeśli chcesz uruchomić określone działanie. W zależności od elementu, Android Studio stosuje aplikację Android, Android Instrumented Tests lub domyślny szablon Android JUnit, aby utworzyć konfigurację. Jeśli jednak utworzysz konfigurację poza oknem dialogowym Uruchom/Debuguj konfiguracje, konfiguracja będzie traktowana jako tymczasowa, chyba że ją zapiszesz. Domyślnie możesz mieć do pięciu tymczasowych konfiguracji w projekcie, zanim Android Studio zacznie je usuwać jedna po drugiej. Jeśli chcesz zmienić to ustawienie domyślne, otwórz okno dialogowe Uruchom/debuguj konfiguracje i kliknij folder Domyślne. Tutaj możesz po prostu wpisać dowolną wartość w polu Limit konfiguracji tymczasowej. Jak już wspomniano, w przypadku kodu C i C++ Android Studio używa debugera LLDB. Oprócz normalnego interfejsu użytkownika (UI) Android Studio, okno debugera ma kartę LLDB, która umożliwia dostęp i wprowadzanie poleceń LLDB podczas debugowania. Możesz wprowadzić te same polecenia, których Android Studio używa do wyświetlania informacji w interfejsie debugera, a także wykonywać inne dodatkowe operacje. Dlatego, aby dodać katalogi symboli, polecenia startowe LLDB, w zakładce Debugger musisz wiedzieć, jak używać przycisków podobnych do poniższych:

• Dodaj + : Dodaj katalog lub polecenie

• Usuń - : Wybierz katalog lub polecenie, a następnie kliknij ten przycisk, aby usunąć element

• W górę [stzalka w górę] : Wybierz katalog lub polecenie, a następnie kliknij ten przycisk, aby przenieść element w górę listy

• W dół [strzałka w dół] : Wybierz katalog lub polecenie, a następnie kliknij ten przycisk, aby przenieść pozycję w dół listy

Po wybraniu typu debugowania:

• Java: debuguj tylko kod Java

• Automatycznie: pozwól Android Studio wybrać najlepszy typ debugowania dla Twojego projektu

• Natywny: Debuguj natywny kod C lub C++

• Podwójny: Debuguj Javę i kod natywny w dwóch oddzielnych sesjach debugowania

Możesz teraz przejść do włączania dodatkowych komponentów do swojego projektu, zaczynając od komponentów, o których właśnie wspomnieliśmy: katalogów symboli i poleceń LLBD.

Katalogi symboli

Jeśli chcesz dodać pliki symboli, aby udostępnić debugerowi informacje w języku C lub C++ wygenerowane poza Android Studio, możesz dodać jeden lub więcej katalogów symboli. Android Studio preferencyjnie rejestruje wszystkie pliki w tych katalogach niż pliki generowane przez wtyczkę Androida dla Gradle. Debuger szuka katalogów symboli od góry do dołu, w kolejności, aż znajdzie to, czego potrzebuje. Przeszukuje również rekurencyjnie pliki w katalogu. Tak więc, aby zoptymalizować listę i zaoszczędzić czas, możesz umieścić najczęściej używane katalogi na górze listy. Gdy określisz katalog wysoko w drzewie, przeszukanie wszystkich podkatalogów może zająć więcej czasu. Ale jeśli dodasz bardzo konkretny katalog, wyszukiwanie zajmie mniej czasu. Musisz znaleźć odpowiednią równowagę między szybkością a znajdowaniem plików potrzebnych do debugowania. Na przykład, jeśli masz katalog, który zawiera podkatalogi dla różnych interfejsów binarnych systemu Android (ABI), możesz dodać katalog dla określonego ABI lub dla wszystkich interfejsów ABI. Przeszukiwanie katalogu wyższego poziomu może zająć więcej czasu, ale jest też bezpieczniejsze, jeśli zdecydujesz się na debugowanie na innym urządzeniu. Pamiętaj, że nie ma potrzeby dodawania katalogów zawierających pliki symboli Gradle, ponieważ debugger używa ich automatycznie.

Polecenia uruchamiania LLDB

Dodaj polecenia LLDB, które chcesz uruchomić, zanim debuger dołączy do procesu. Na przykład możesz zidentyfikować ustawienia środowiska, jak pokazano w następującym poleceniu:

settings set target.max-memory-read-size 2048

LLDB uruchamia polecenia w kolejności od góry do dołu.

Polecenia dołączania postów LLDB

Dodaj polecenia LLDB, które musisz uruchomić zaraz po dołączeniu debugera do procesu. Ilustrować:

process handle SIGPIPE -n true -p true -s false

LLDB uruchamia polecenia w kolejności od góry do dołu.

Rejestrowanie: kanały docelowe

Kanały docelowe są najczęściej stosowane do określania opcji dziennika LLDB. Android Studio ustawia domyślne opcje w oparciu o Twoje wymagania, więc nie jest zbyt wolne, ale zawiera informacje potrzebne do rozwiązywania problemów. Dziennik może być również często wymagany w przypadku raportów o błędach Android Studio. Domyślną funkcją jest lldb process:gdb-remote packets. Jednak możliwa jest również zmiana ustawienia domyślnego, aby zebrać więcej informacji. Na przykład następujące opcje dziennika zbierają informacje o określonej platformie:

Platforma procesów lldb: pakiety gdb-remote

Android Studio umieszcza dzienniki urządzenia w następującej lokalizacji, gdzie ApplicationId to unikalny identyfikator aplikacji używany w skompilowanym pliku manifestu APK i identyfikujący Twoją aplikację na urządzeniu oraz w sklepie Google Play:

/data/data/Identyfikator Aplikacji/lldb/log

Podobnie, gdy wielu użytkowników uzyskuje dostęp do urządzenia, umieszcza dzienniki w następującej lokalizacji, gdzie AndroidUserId jest unikalnym identyfikatorem użytkownika na urządzeniu:

/data/user/AndroidUserId/ApplicationId/lldb/log

Zakładka Różne

Karta Różne służy głównie do określania opcji logowania, instalacji, uruchamiania i wdrażania:

• Logcat: Wyczyść dziennik przed uruchomieniem . Wybierz tę opcję, jeśli chcesz, aby Android Studio usunęło wszystkie dane z poprzednich sesji z pliku dziennika przed uruchomieniem aplikacji. Domyślnie ta opcja nie jest zaznaczona.

• Opcje instalacji: Pomiń instalację, jeśli pakiet APK nie uległ zmianie. Po wybraniu Android Studio nie wdroży ponownie pliku APK, jeśli wykryje, że nie jest on zmodyfikowany. Jeśli chcesz, aby Android Studio wymusiło instalację pakietu APK, nawet jeśli nie uległ on zmianie, usuń zaznaczenie tej opcji, ponieważ domyślnie pozostaje zaznaczona.

• Opcje instalacji: Wymuś zatrzymanie działania aplikacji przed uruchomieniem aktywności . Jeśli ta opcja zostanie wybrana, gdy Android Studio wykryje, że nie musi ponownie instalować pakietu APK, ponieważ nie uległ on zmianie, wymusi zatrzymanie aplikacji do uruchomienia z domyślnej aktywności programu uruchamiającego. Jeśli ta opcja nie jest zaznaczona, Android Studio nie wymusi zatrzymania aplikacji. Ta opcja działa w połączeniu z poprzednią opcją, która reguluje, czy pakiet APK jest zainstalowany, czy nie. Lepiej jest, aby oba pola Opcje instalacji pozostawiły je jako domyślne, chyba że wyraźnie chcesz wymusić instalację za każdym razem.

W niektórych przypadkach może być konieczne odznaczenie tej opcji. Na przykład, jeśli piszesz silnik metody wprowadzania (IME), wymuszenie zatrzymania aplikacji odznacza ją jako bieżącą klawiaturę, co może nie odpowiadać Twoim planom.

MONITOR URZĄDZENIA Z ANDROIDEM

Android Device Monitor to samodzielne narzędzie, które oferuje graficzny interfejs użytkownika dla kilku usług debugowania i analizy aplikacji Android. Narzędzie Monitor nie wymaga instalacji zintegrowanego środowiska programistycznego i zawiera następujące narzędzia: Dalvik Debug Monitor Server, Traceview, Systrace, Hierarchy Viewer, Pixel Perfect i narzędzie Network Traffic. Aby uruchomić samodzielną aplikację Device Monitor w Android Studio 3.1 i niższych, w wierszu poleceń w katalogu android-SDK/tools/ należy wstawić następujące polecenie:

monitor

Następnie możesz połączyć narzędzie z podłączonym urządzeniem, wybierając je z panelu Urządzenia. Ale jeśli masz problemy z przeglądaniem funkcji lub okien, spróbuj edytować za pomocą funkcji Resetuj perspektywę z paska menu. Przejdźmy teraz do głównych komponentów Android Device Monitor, o których wspomniano wcześniej.

Serwer monitorowania debugowania Dalvik (DDMS)

To narzędzie jest przestarzałe. Zamiast tego należy użyć Android Profiler (od Android Studio 3.0 i nowszych) do profilowania procesora, pamięci i manipulacji siecią w aplikacji. Narzędzia Android Profiler udostępniają dane w czasie rzeczywistym, aby pomóc Ci rozpoznać, w jaki sposób Twoja aplikacja wykorzystuje procesor, pamięć, sieć i potencjał baterii. Wspaniałe funkcje zapewniane przez zaawansowane profilowanie obejmują:

• Ustawianie osi czasu zdarzenia we wszystkich oknach profilera

• Śledzenie liczby przydzielonych obiektów w programie Memory Profiler

• Aktywacja zdarzeń związanych z odśmiecaniem pamięci w programie Memory Profiler

• Dostarczanie informacji o wszystkich przesłanych plikach w Network Profiler

Pamiętaj, że wszystkie te funkcje są domyślnie dostępne na urządzeniu z systemem Android 8.0 lub nowszym. Aby włączyć zaawansowane profilowanie, wystarczy wybrać opcję Uruchom i edytuj konfiguracje. Tutaj kliknij moduł swojej aplikacji, a następnie zaznacz Włącz zaawansowane profilowanie. Zaawansowana konfiguracja profilowania spowalnia proces kompilacji, dlatego należy ją włączać tylko wtedy, gdy naprawdę trzeba rozpocząć profilowanie aplikacji. Jeśli chcesz wykonać inne zadania debugowania, takie jak przekazywanie poleceń do podłączonego urządzenia w celu skonfigurowania przekierowania portów, przesyłania plików lub wykonywania zrzutów ekranu, możesz użyć okna Android Debug Bridge (ADB), emulatora systemu Android lub debugera.

Traceview

To narzędzie jest również przestarzałe. Tak więc, jeśli chcesz sprawdzić. pliki śledzenia powstałe w wyniku oprzyrządowania aplikacji za pomocą klasy Debug i rejestrowania nowych śladów metod, wystarczy użyć profilera procesora Android Studio.

Systrace

W przypadku konieczności sprawdzenia natywnych procesów systemowych i zacięcia interfejsu użytkownika spowodowanego przez porzucone ramki, zastosuj systrace z wiersza poleceń lub uproszczone śledzenie systemu w Profiler procesora. Możesz użyć narzędzia Profiler procesora do sprawdzania użycia procesora przez aplikację i aktywności wątków w czasie rzeczywistym podczas interakcji z aplikacją lub możesz przejrzeć szczegóły w zarejestrowanych śladach metod, śladach funkcji i śladach systemu. Konkretne rodzaje danych rejestrowanych i wyświetlanych przez narzędzie CPU Profiler są określane przez wybraną konfigurację nagrywania:

Śledzenie systemu: Przechwytuje szczegółowe elementy, które umożliwiają sprawdzenie interakcji aplikacji z zasobami systemowymi. Ślady metod i funkcji: dla każdego wątku w procesie aplikacji można śledzić, które metody (Java) lub funkcje (C/C++) są implementowane w określonym czasie, a także zasoby procesora CPU, które każda metoda lub funkcja żąda podczas jej wykonywania. Możesz również użyć śladów metod i funkcji do identyfikacji wywołujących i wywoływanych. Wywołujący może być tutaj zdefiniowany jako metoda lub funkcja, która aktywuje inną metodę lub funkcję, a wywoływany to taki, który jest wywoływany przez inną metodę lub funkcję. Możesz użyć tych informacji, aby zobaczyć, które metody lub funkcje są odpowiedzialne za zbyt częste wywoływanie określonych działań obciążających zasoby i zoptymalizować kod aplikacji, aby uniknąć niepotrzebnego obciążenia. Ponadto domyślny widok programu CPU Profiler obejmuje następujące osie czasu:

• Oś czasu zdarzenia: wyświetla różne działania w aplikacji na różnych etapach cyklu życia i wskazuje interakcje użytkownika z urządzeniem, w tym zdarzenia obracania ekranu.

• Oś czasu procesora: przedstawia wykorzystanie procesora przez aplikację w czasie rzeczywistym — jako procent całkowitego dostępnego czasu procesora — oraz całkowitą liczbę wątków, w których działa aplikacja. Oś czasu wyświetla również wykorzystanie procesora przez inne procesy (takie jak procesy systemowe lub inne aplikacje), dzięki czemu możesz je porównać z wykorzystaniem Twojej aplikacji. Możesz sprawdzić historyczne dane o wykorzystaniu procesora, po prostu przesuwając mysz wzdłuż osi poziomej osi czasu.

Oś czasu aktywności wątków: przegląda każdy wątek, który należy do procesu aplikacji i wskazuje jego aktywność na osi czasu, używając kolorów wymienionych poniżej. Po zarejestrowaniu śledzenia możesz wybrać wątek z tej osi czasu, aby sprawdzić jego dane w funkcji śledzenia.

• Zielony: wątek jest aktywny lub gotowy do użycia procesora. Oznacza to, że jest w stanie uruchomionym lub możliwym do uruchomienia.

• Żółty: wątek jest aktywny, ale czeka na operację wejścia/wyjścia, taką jak dyskowe lub sieciowe wejście/wyjście, zanim będzie mógł zakończyć swoją pracę.

• Szary: wątek nie jest aktywny i nie zużywa czasu procesora. Czasami dzieje się tak, gdy wątek wymaga dostępu do zasobu, który nie jest jeszcze dostępny. Albo wątek przechodzi w stan uśpienia dobrowolnego, albo jądro usypia wątek, dopóki wymagany zasób nie stanie się dostępny.

Wyświetlający hierarchię

Ponieważ to narzędzie jest również przestarzałe, należy użyć Inspektora układu, aby sprawdzić hierarchię widoków aplikacji w czasie wykonywania. Inspektor układu w Android Studio umożliwia porównywanie układu aplikacji z innymi standardowymi makietami projektu, wyświetlanie powiększonego lub 3D widoku aplikacji, a także badanie szczegółów jej układu w czasie wykonywania. Jest to szczególnie wygodne, gdy układ jest tworzony w czasie wykonywania, a nie w całości w XML; dlatego może czasami zachowywać się nieoczekiwanie. Ponadto funkcja walidacji układu umożliwia jednoczesne wyświetlanie podglądu układów na różnych urządzeniach i pokazywanie konfiguracji, w tym zmiennych rozmiarów czcionek lub języków użytkownika, co ułatwia przeprowadzanie testów dla różnych typowych problemów z układem. Aby otworzyć Inspektora układu, uruchom aplikację na podłączonym urządzeniu lub emulatorze i otwórz Narzędzia, a następnie Inspektor układu. Zazwyczaj Inspektor układu powinien wyświetlać następujące składniki:

• Drzewo komponentów: hierarchia widoków w układzie

• Wyświetlanie układu: renderowanie układu aplikacji w postaci, w jakiej pojawia się na urządzeniu lub emulatorze, z ograniczeniami układu pokazanymi dla każdego widoku

• Pasek narzędzi Inspektora układu: Narzędzia dla Inspektora układu

• Atrybuty: atrybuty układu dla wybranego widoku

Pixel Perfect

Podobnie jak inne narzędzia, Pixel Perfect również został wycofany. Jeśli potrzebujesz pracować na makietach projektowych, upewnij się, że używasz Inspektora układu.

Narzędzie ruchu sieciowego

Podobnie jak inne narzędzia, narzędzie ruchu sieciowego jest również przestarzałe. Jeśli chcesz sprawdzić, jak i kiedy Twoja aplikacja przesyła dane przez sieć, zalecamy zamiast tego użycie Network Profiler. Network Profiler zapewnia aktywność sieciową w czasie rzeczywistym na osi czasu, pokazując

dane przekazywane i odbierane, a także aktualną liczbę połączeń. Pozwala to sprawdzić, jak i kiedy aplikacja manipuluje danymi i odpowiednio zoptymalizować kod źródłowy. Mogą pojawić się wątpliwości, dlaczego w ogóle należy rejestrować i profilować aktywność sieciową aplikacji. To jest zupełnie niepotrzebne. Zamiast tego pomyśl o tym – gdy Twoja aplikacja wysyła żądanie do sieci, urządzenie musi korzystać z energochłonnego radia komórkowego lub Wi-Fi, aby przesyłać i odbierać pakiety. Radiotelefony wykorzystują moc do przesyłania danych i wykorzystują dodatkową moc do włączania się i czuwania. Korzystając z Network Profiler, będziesz mógł szukać częstych, krótkich skoków aktywności sieciowej, co oznacza, że Twoja aplikacja wymaga częstego włączania radia lub pozostawania w stanie czuwania przez dłuższy czas, aby zarządzać wieloma krótkimi żądaniami blisko siebie. Ten wzorzec pokazuje, że możesz zoptymalizować swoją aplikację pod kątem lepszej wydajności baterii przez grupowanie żądań sieciowych, minimalizując w ten sposób liczbę włączania się radia w celu wysłania lub odebrania danych. Pozwala to również radiotelefonom przełączyć się w tryb niskiego zużycia energii, aby oszczędzać baterię w długich przerwach między żądaniami grupowymi. Aby otworzyć Network Profiler, przejdź do okna narzędziowego i kliknij Profiler. Pamiętaj, aby wybrać urządzenie i proces aplikacji, które chcesz profilować, z paska narzędzi Android Profiler. Jeśli podłączyłeś urządzenie przez USB, ale nie możesz go znaleźć na liście, upewnij się, że masz włączone debugowanie USB. Ponadto, jeśli chcesz wybrać część osi czasu, przejrzeć listę wysłanych żądań sieciowych i otrzymanych odpowiedzi lub sprawdzić szczegółowe informacje o wybranym pliku, należy najpierw włączyć zaawansowane profilowanie.

WAŻNE SKRÓTY I TECHNIKI PRZYCISKÓW

Android Studio ma różne skróty klawiaturowe do wielu typowych czynności. W tej sekcji wymienimy wiele wartości domyślnych, a także specyficzne dla niektórych skrótów klawiaturowych operacji. Oprócz map klawiszy, które tutaj zobaczysz, możesz także wybrać niezbędne funkcje z gotowych map klawiszy, a nawet samodzielnie utworzyć niestandardową mapę klawiszy. Należy pamiętać, że ponieważ Android Studio jest oparte na IntelliJ IDEA, dodatkowe skróty można znaleźć w dokumentacji referencyjnej mapy klawiszy IntelliJ IDEA

Przedstawione tutaj domyślne skróty klawiaturowe zawierają opcje systemów operacyjnych Windows/Linux i Mac.

Ogólny

1. Zapisz wszystko – Control+S/Command+S

2. Synchronizuj – Control+Alt+Y/Command+Option+Y

3. Edytor maksymalizacji/minimalizacji - Control+Shift+F12/Control+Command+F12

4.Dodaj do ulubionych – Alt+Shift+F/Option+Shift+F

5. Sprawdź bieżący plik z bieżącym profilem - Alt+Shift+I/Option+Shift+I

6.Schemat szybkiego przełączania – Control+`(backquote)/Control+`(backquote)

7. Otwórz okno ustawień – Control+Alt+S/Command+, (przecinek)

8. Otwórz okno dialogowe struktury projektu - Control+Alt+Shift+S/Command+; (średnik)

9. Przełączaj się między zakładkami a oknem narzędzi - Control+Tab/Control+Tab

Nawigacja i wyszukiwanie w Studio

1. Przeszukaj wszystko (w tym kod i menu) - naciśnij dwukrotnie Shift / naciśnij dwukrotnie Shift

2.Znajdź – Control+F/Command+F

3.Znajdź następny – F3/Command+G

4. Znajdź poprzednie – Shift+F3/Command+Shift+G

5. Zamień – Control+R/Command+R

6. Znajdź akcję – Control+Shift+A/Command+Shift+A

7. Wyszukaj według nazwy symbolu - Control+Alt+Shift+N/Command+Option+O

8. Znajdź klasę – Control+N/Command+O

9.Znajdź plik (zamiast klasy) – Control+Shift+N/Command+Shift+O

10.Znajdź ścieżkę – Control+Shift+F/Command+Shift+F

11. Otwórz wyskakujące okienko struktury pliku – Control+F12/Command+F12

12. Poruszaj się między otwartymi zakładkami edytora – Alt+Strzałka w prawo lub Strzałka w lewo/Control+Strzałka w prawo lub Control+Strzałka w lewo

13. Przejdź do źródła - F4 lub Control+Enter/F4 lub Command+strzałka w dół

14. Otwórz bieżącą zakładkę edytora w nowym oknie – Shift+F4/Shift+F4

15. Wyskakujące okienko z ostatnio otwieranymi plikami – Control+E/Command+E

16. Wyskakujące okienko ostatnio edytowanych plików - Control+Shift+E/Command+Shift+E

17. Przejdź do lokalizacji ostatniej edycji - Control+Shift+Backspace/Command+Shift+Delete

18.Zamknij kartę aktywnego edytora - Control+F4/Command+W

19.Powrót do okna edytora z okna narzędzia – Esc/Esc

20.Ukryj aktywne lub ostatnio aktywne okno narzędzia – Shift+Esc/Shift+Esc

21. Przejdź do wiersza – Control+G/Command+L

22. Otwarta hierarchia typów – Control+H/Control+H

23.Otwórz hierarchię metod – Control+Shift+H/Command+Shift+H

24. Otwórz hierarchię połączeń – Control+Alt+H/Control+Option+H

Przeglądanie układów

1.Powiększ/pomniejsz – Control+plus lub Control+minus/Command+plus lub Command+minus

2. Dopasuj do ekranu – Control+0/Command+0

3. Rzeczywisty rozmiar – Control+Shift+1/Command+Shift+1

Narzędzia projektowe: Edytor układu

1. Przełącz między trybami projektowania i schematu – B/B

2. Przełącz między trybami portretowym i poziomym – O/O

3.Przełącz urządzenia – D/D

4.Wymuś odświeżenie – R/R

5. Przełącz panel błędów renderowania –E/E

6. Usuń ograniczenia – Usuń lub Control+klik/Delete lub Command+klik

7.Powiększ – Control+plus/Command+plus

8. Oddal – Control+minus/Command+minus

9. Przybliż, aby dopasować – Control+0/Command+0

10.Pan - Przytrzymaj spację+kliknij i przeciągnij/Przytrzymaj spację+kliknij i przeciągnij

11. Przejdź do XML – Control+B/Command+B

12.Wybierz wszystkie komponenty – Control+A/Command+A

13.Wybierz wiele komponentów – Shift+klik lub Control+klik/Shift+klik lub Command+klik

Narzędzia projektowe: Edytor nawigacji

1. Powiększ – Control+plus/Command+plus

2. Oddal – Control+minus/Command+minus

3. Powiększ, aby dopasować – Control+0/Command+0

4. Pan – przytrzymaj spację+kliknij i przeciągnij/przytrzymaj spację+kliknij i przeciągnij

5. Przejdź do XML – Control+B/Command+B

6. Przełącz panel błędów renderowania –E/E

7. Grupuj w zagnieżdżony wykres – Control+G/Command+G

8. Przełączaj się między miejscami docelowymi – Tab lub Shift+Tab/Tab lub Shift+Tab

9. Wybierz wszystkie miejsca docelowe – Control+A/Command+A

10.Wybierz wiele miejsc docelowych – Shift+klik lub Control+klik/Shift+klik lub Command+klik

Pisanie kodu

1. Wygeneruj kod (gettery, settery, konstruktory, hashCode/equals, toString, nowy plik, nowa klasa) - Alt+Insert/Command+N

2.Omijanie metod – Control+O/Control+O

3. Metody realizacji – Control+I/Control+I

4.Surround z (if … else/try … catch) - Control+Alt+T/Command+Option+T

5. Usuń linię przy daszku – Control+Y/Command+Delete

6. Zwiń/rozwiń bieżący blok kodu – Control+minus lub Control+plus/Command+minus lub Command+plus

7. Zwiń/rozwiń wszystkie bloki kodu – Control+Shift+minus lub Control+Shift+plus/Command+Shift+minus lub Command+Shift+plus

8. Powiel bieżącą linię lub zaznaczenie – Control+D/Command+D

9. Podstawowe uzupełnianie kodu – Control+Spacja/Control+Spacja

10. Inteligentne uzupełnianie kodu (filtruje listę metod i zmiennych według oczekiwanego typu) - Control+Shift+Spacja/Control+Shift+Spacja

11. Kompletna instrukcja – Control+Shift+Enter/Command+Shift+Enter

12. Szybkie przeglądanie dokumentacji – Control+Q/Control+J

13.Pokaż parametry dla wybranej metody – Control+P/Command+P

14. Przejdź do deklaracji (bezpośrednio) – Control+B lub Control+klik/Command+B lub Command+klik

15.Przejdź do wdrożeń – Control+Alt+B/Command+Option+B

16. Przejdź do super-metody/super-klasy – Control+U/Command+U

17. Otwórz szybkie wyszukiwanie definicji – Control+Shift+I/Command+Y

18. Przełącz widoczność okna narzędzia projektu – Alt+1/Command+1

19. Przełącz zakładkę – F11/F3

20. Przełącz zakładkę z mnemonikiem – Control+F11/Option+F3

21. Skomentuj/odkomentuj z komentarzem linii – Control+//Command+/

22. Skomentuj/odkomentuj z komentarzem bloku – Control+Shift+//Command+Shift+/

23.Wybierz kolejno rosnące bloki kodu – Control+W/Option+Up

24. Zmniejsz aktualny wybór do poprzedniego stanu – Control+Shift+W/Opcja+Dół

25. Przejdź do początku bloku kodu – Control+[/Option+Command+[

26. Przejdź do końca bloku kodu – Control+]/Option+Command+]

27. Wybierz początek bloku kodu – Control+Shift+ [/Option+Command+Shift+[

28. Wybierz koniec bloku kodu – Control+Shift+]/Option+Command+Shift+]

29. Usuń do końca słowa – Control+Delete/Option+Delete

30. Usuń na początek słowa – Control+Backspace/Option+Delete

31. Zoptymalizuj importy – Control+Alt+O/Control+Option+O

32. Szybka naprawa projektu (pokaż działania intencji i szybkie poprawki) – Alt+Enter/Option+Enter

33.Kod reformatowania – Control+Alt+L/Command+Option+L

34. Linie z automatycznym wcięciem – Control+Alt+I/Control+Option+I

35. Wcięcie/odcięcie linii – Tab lub Shift+Tab/Tab lub Shift+Tab

36. Inteligentne łączenie linii – Control+Shift+J/Control+Shift+J

37. Inteligentny podział linii – Control+Enter/Command+Enter

38. Rozpocznij nową linię – Shift+Enter/Shift+Enter

39. Następny/poprzedni podświetlony błąd – F2 lub Shift+F2/F2 lub Shift+F2

Buduj i uruchamiaj

1.Buduj – Control+F9/Command+F9

2. Zbuduj i uruchom – Shift+F10/Control+R

3. Zastosuj zmiany i uruchom ponownie działanie – Control+F10/Control+Command+R

4. Zastosuj zmiany w kodzie – Control+Alt+F10/Control+Shift+Command+R

Debugowanie

1. Debugowanie – Shift+F9/Control+D

2. Przejdź dalej – F8/F8

3. Wejdź do – F7/F7

4. Inteligentne wejście do – Shift+F7/Shift+F7

5. Wyjdź – Shift+F8/Shift+F8

6. Biegnij do kursora – Alt+F9/Option+F9

7. Oblicz wyrażenie – Alt+F8/Option+F8

8. Wznów program – F9/Command+Option+R

9. Przełącz punkt przerwania – Control+F8/Command+F8

10. Wyświetl punkty przerwania – Control+Shift+F8/Command+Shift+F8

Refaktoryzacja

1.Kopiuj – F5/F5

2.Ruch – F6/F6

3. Bezpieczne usuwanie – Alt+Delete/Command+Delete

4. Zmień nazwę – Shift+F6/Shift+F6

5.Zmień podpis – Control+F6/Command+F6

6.Inline — Control+Alt+N/Command+Option+N

7. Metoda wyodrębniania – Control+Alt+M/Command+Option+M

8.Wyodrębnij zmienną – Control+Alt+V/Command+Option+V

9. Pole wyodrębniania – Control+Alt+F/Command+Option+F

10. Stała ekstrakcji – Control+Alt+C/Command+Option+C

11. Wyodrębnij parametr – Control+Alt+P/Command+Option+P

Kontrola wersji/historia lokalna

1. Zatwierdź projekt w VCS – Control+K/Command+K

2. Zaktualizuj projekt z VCS – Control+T/Command+T

3.Wyświetl ostatnie zmiany – Alt+Shift+C/Option+Shift+C

4. Otwórz wyskakujące okienko VCS – Alt+` (cytat wsteczny)/Control+V

Nie, przejrzymy skróty klawiszowe i polecenia Android SDK, które składają się z wielu pakietów wymaganych do tworzenia aplikacji. Tutaj wymienimy najważniejsze dostępne narzędzia wiersza poleceń, uporządkowane według pakietów, w których są dostarczane. Możesz zainstalować i edytować każdy pakiet za pomocą Menedżera SDK w Android Studio lub narzędzia wiersza poleceń Menedżera SDK. Wszystkie pakiety są domyślnie dostępne w katalogu Android SDK, do którego można uzyskać dostęp w następujący sposób: W Android Studio kliknij Plik i otwórz Strukturę projektu, w której znajduje się wszystko5. Jednocześnie możesz mieć wiele wersji narzędzi do kompilacji, aby zbudować swoją aplikację dla różnych wersji Androida.

• apkanalyzer: zapewnia wgląd w skład pakietu APK po zakończeniu procesu kompilacji.

• avdmanager: umożliwia tworzenie i zarządzanie AVD z wiersza poleceń.

• lint: Wywołuje narzędzie do skanowania kodu, które może pomóc w zidentyfikowaniu i naprawieniu problemów z jakością strukturalną kodu.

• Retrace: W przypadku aplikacji skompilowanych przez R8 funkcja retrace dekoduje zaciemniony ślad stosu, który odwzorowuje oryginalny kod źródłowy.

• sdkmanager: umożliwia przeglądanie, instalowanie, aktualizowanie i odinstalowywanie pakietów Android SDK.

Oprócz wyżej wymienionych, istnieją również pewne narzędzia SDK Build Tools, które znajdują się w: android_sdk/build-tools/version/. Są one wymagane głównie do tworzenia aplikacji na Androida. Większość narzędzi w tym pakiecie jest wywoływana przez narzędzia do budowania i nie jest przeznaczona dla Ciebie. Jednak przydatne mogą być następujące narzędzia wiersza polecenia:

• aapt2: analizuje, indeksuje i kompiluje zasoby systemu Android do formatu binarnego zoptymalizowanego pod kątem platformy Android i pakuje skompilowane zasoby w jedno wyjście.

• apksigner: podpisuje pakiety APK i sprawdza, czy podpisy APK zostaną pomyślnie zweryfikowane na wszystkich wersjach platform obsługiwanych przez dany pakiet APK.

• zipalign: Optymalizuje pliki APK, zapewniając, że wszystkie nieskompresowane dane zaczynają się od określonego wyrównania względem początku pliku.

W Android SDK, poza narzędziami do budowania, niektóre narzędzia platformy znajdują się w: android_sdk/platform-tools/. Narzędzia te są aktualizowane dla każdej nowej wersji platformy Android, aby obsługiwać nowe funkcje (a czasami częściej testować i aktualizować narzędzia), a każda aktualizacja jest wstecznie kompatybilna ze starszymi wersjami platformy.

• adb: Android Debug Bridge (ADB) to wszechstronne narzędzie, które umożliwia organizowanie stanu wystąpienia emulatora lub urządzenia z systemem Android. Możesz go również użyć do zainstalowania pakietu APK na urządzeniu.

• etc1tool: narzędzie wiersza polecenia, które umożliwia kodowanie obrazów PNG do standardu kompresji ETC1 i dekodowanie skompresowanych obrazów ETC1 z powrotem do formatu PNG.

• fastboot: Służy do flashowania urządzenia z platformą i innymi obrazami systemu.

• logcat: jest to narzędzie wywoływane przez ADB w celu przeglądania dzienników aplikacji i systemu.

Kolejne ważne narzędzie do budowania systemu Android, które powinno zostać omówione w AAPT2 lub narzędziu do pakowania zasobów systemu Android, którego Android Studio i wtyczka Android Gradle wykorzystują do zbierania i pakowania zasobów aplikacji. AAPT2 analizuje, indeksuje i kompiluje zasoby do formatu binarnego zoptymalizowanego pod kątem platformy Android. Aby zastosować AAPT2 z wiersza poleceń w systemie Linux lub Mac, po prostu uruchom polecenie aapt2. W systemie Windows należy wstawić polecenie aapt2.exe. AAPT2 obsługuje szybszą kompilację danych, umożliwiając kompilację przyrostową. Osiąga się to, dzieląc przetwarzanie zasobów na dwa następujące etapy:

a. Kompiluj: kompiluje pliki zasobów do formatów binarnych.

b. Link: łączy wszystkie skompilowane pliki i pakuje je w jeden pakiet.

Opcje kompilacji

Istnieje kilka kluczowych poleceń, których można używać z poleceniem kompilacji, jak pokazano poniżej:

• o ścieżka: Określa ścieżkę wyjściową dla skompilowanych zasobów. Jest to obowiązkowa flaga, ponieważ musisz określić ścieżkę do katalogu, w którym AAPT2 może wypisać i umieścić skompilowane zasoby.

• dir directory: Określa katalog do skanowania w poszukiwaniu zasobów. Chociaż można użyć tej flagi do zbierania wielu plików zasobów za pomocą jednego polecenia, wyłącza ona zalety kompilacji przyrostowej i dlatego nie należy jej stosować w przypadku dużych projektów.

• pseudo-localize: Służy do generowania pseudo-zlokalizowanych wersji domyślnych ciągów, takich jak en-XA i en-XB.

• no-crunch: Aktywowany, aby wyłączyć przetwarzanie PNG. Możesz użyć tej opcji, jeśli przetworzono już pliki PNG lub jeśli tworzysz kompilacje debugowania, które nie wymagają zmniejszania rozmiaru pliku. Włączenie tej opcji powoduje szybsze wykonanie, ale zwiększa rozmiar pliku wyjściowego.

• legacy: traktuje błędy, które są dopuszczalne podczas używania wcześniejszych wersji AAPT jako ostrzeżenia. Ta flaga jest również odpowiednia dla nieoczekiwanych błędów podczas kompilacji.

• v: Służy do włączania pełnego rejestrowania.

Opcje łącza

Za pomocą polecenia link możesz użyć następujących opcji:

• o ścieżka: określa ścieżkę wyjściową dla pliku APK połączonego zasobu. Jest to obowiązkowa flaga łącza, ponieważ oczekuje się, że określisz ścieżkę do wyjściowego pakietu APK, który może zawierać połączone zasoby.

• plik manifestu: służy do określania ścieżki do pliku manifestu systemu Android do skompilowania. Jest to kolejna obowiązkowa flaga, ponieważ plik manifestu zawiera podstawowe informacje o Twojej aplikacji, takie jak nazwa pakietu i identyfikator aplikacji.

• I: To polecenie zapewnia ścieżkę do pliku android.jar platformy lub innych plików APK, takich jak framework-res.apk, które mogą być przydatne podczas tworzenia funkcji. Ta flaga jest niezbędna, jeśli w plikach zasobów używasz atrybutów z przestrzenią nazw systemu Android (na przykład android:id).

• Katalog: ten katalog określa zasoby, które mają być uwzględnione w pakiecie APK. Możesz również użyć tego katalogu do przechowywania oryginalnych nieprzetworzonych plików.

• Plik R: Stosowany do przekazywania pojedynczego pliku .flat do połączenia przy użyciu semantyki nakładki bez użycia znacznika <add-resource>. Gdy dostarczasz plik zasobów, który nakłada, rozszerza lub modyfikuje istniejący plik, używany jest ostatni podany zasób powodujący konflikt.

• id-pakietu: określa identyfikator pakietu, który ma być używany w aplikacji. Podany identyfikator pakietu musi być większy lub równy 0x7f, chyba że zostanie użyty w połączeniu z parametrem allow-reserved-package-id.

• allow-reserved-package-id: umożliwia użycie zarezerwowanego identyfikatora pakietu. Zarezerwowane identyfikatory pakietów to identyfikatory, które są zwykle przypisywane do bibliotek współdzielonych i mieszczą się w zakresie od 0x02 do 0x7e włącznie. Wstawiając -allow-reserved-package-id, możesz przypisać identyfikatory, które mieszczą się w zakresie zarezerwowanych identyfikatorów pakietów. Powinno to być używane tylko w przypadku pakietów z wersją min-SDK 26 lub niższą.

• katalog java: określa katalog, w którym ma zostać wygenerowany R.java.

• proguard proguard_options: Tworzy plik wyjściowy dla reguł ProGuard.

• proguard-conditional-keep-rules: Tworzy plik wyjściowy dla reguł ProGuard dla głównego pliku dex.

• no-auto-version: Wyłącza automatyczne wersjonowanie zestawu SDK stylu i układu.

• no-version-vectors: Wyłącza automatyczne wersjonowanie wektorów do rysowania. Zastosuj to tylko podczas tworzenia pliku APK za pomocą biblioteki do rysowania wektorów.

• no-version-transitions: Wyłącza automatyczne wersjonowanie zasobów przejściowych. Używaj tego tylko podczas tworzenia pakietu APK za pomocą biblioteki obsługi przejścia.

• brak deduplikacji zasobów: wyłącza automatyczną deduplikację zasobów o identycznych wartościach w zgodnych konfiguracjach.

• enable-sparse-encoding: Służy do włączania kodowania rzadkich wpisów przy użyciu drzewa wyszukiwania binarnego. Jest to szczególnie przydatne do optymalizacji rozmiaru APK, ale kosztem wydajności odzyskiwania zasobów.

• z: Służy do żądania lokalizacji ciągów oznaczonych jako „sugerowane”.

• c config: Zawiera listę konfiguracji oddzielonych przecinkami. Na przykład, jeśli masz zależności od biblioteki wsparcia, która przechowuje tłumaczenia dla wielu języków, możesz filtrować zasoby tylko dla danej konfiguracji językowej, takiej jak angielski lub hiszpański. Zazwyczaj zaleca się zdefiniowanie konfiguracji języka za pomocą dwuliterowego kodu języka ISO 639-1, po którym opcjonalnie następuje dwuliterowy kod regionu ISO 3166-1-alpha-2 poprzedzony małą literą „r” (na przykład en- ros.).

• Preferowana gęstość: Pozwala AAPT2 wybrać najbardziej pasującą gęstość i usunąć wszystkie inne. W aplikacji dostępnych jest kilka kwalifikatorów gęstości pikseli, takich jak ldpi, hdpi i xhdpi. Wybierając preferowaną gęstość, AAPT2 zapisze najbardziej pasującą gęstość w tabeli zasobów i usunie wszystkie inne.

• output-to-dir: Wyprowadza zawartość APK do katalogu określonego przez -o. Jeśli wystąpią jakiekolwiek błędy z tą flagą, możesz je rozwiązać, uaktualniając do Android SDK Build Tools 28.0.0 lub nowszy.

• min-SDK-version: ustawia domyślną minimalną wersję SDK, która ma być używana dla AndroidManifest.xml.

• target-SDK-version: Ustawia domyślną docelową wersję SDK do użycia dla AndroidManifest.xml.

• kod wersji: określa kod wersji do wstawienia do pliku AndroidManifest.xml, jeśli go nie ma.

• compile-SDK-version-name: Identyfikuje nazwę wersji do wstawienia do AndroidManifest.xml, jeśli jej nie ma.

• proto-format: Tworzy skompilowane zasoby w formacie Protobuf. Ma zastosowanie jako dane wejściowe do narzędzia pakietu do tworzenia pakietu aplikacji na Androida.

• nonfinal-ids: Wyniki w R.java z niefinalnymi identyfikatorami zasobów (odwołania do identyfikatorów z kodu aplikacji, które nie są wstawiane podczas kompilacji kotlinc/javac).

• emit-ids path: Emituje plik o podanej ścieżce z listą nazw typów zasobów i ich odwzorowaniami ID. Nadaje się głównie do użycia z –stable-ids.

• stable-ids outputfilename.ext: zawiera plik wygenerowany z opcją –emit-ids, który zawiera listę nazw typów zasobów i przypisanych im identyfikatorów. Ta opcja umożliwia zachowanie stabilności przypisanych identyfikatorów nawet po usunięciu lub dodaniu nowych zasobów podczas łączenia.

• custom-package nazwa_pakietu: Identyfikuje niestandardowy pakiet Java, w ramach którego ma być tworzony R.java.

• extra-packages nazwa_pakietu: Tworzy ten sam plik R.java, ale z różnymi nazwami pakietów.

• add-Javadoc-annotation: Dodaje adnotację JavaDoc do wszystkich wygenerowanych klas Java.

• output-text-symbols path: Tworzy plik tekstowy zawierający symbole zasobów klasy R w określonym pliku. Należy określić ścieżkę do pliku wyjściowego.

• auto-add-overlay: umożliwia dodawanie nowych zasobów w nakładkach bez użycia znacznika <add-resource>.

• rename-manifest-package: Zmienia nazwę pakietu w AndroidManifest.xml.

• rename-instrumentation-target-package: Zmienia nazwę pakietu docelowego dla instrumentacji. Zwykle powinno być używane razem z -rename-manifest-package.

• Rozszerzenie 0: Służy do określania rozszerzeń plików, których nie trzeba kompresować.

• split path:config[,config[..]]: Dzieli zasoby na podstawie zestawu konfiguracji, aby utworzyć inną wersję APK. Należy określić ścieżkę do wyjściowego pliku APK wraz z zestawem konfiguracji.

• v: Służy do włączania zwiększonej szczegółowości danych wyjściowych.

Zmienne środowiskowe

Aby określić takie rzeczy, jak miejsce instalacji pakietu SDK i miejsce przechowywania danych specyficznych dla użytkownika, powinieneś wiedzieć, jak ustawić zmienne środowiskowe dla Android Studio i narzędzi wiersza poleceń. Poniższy przykład ilustruje, jak użyć zmiennej środowiskowej do

uruchomienia emulatora, gdy instalacja SDK została umieszczona w E:\Android\sdk\zamiast w domyślnej lokalizacji $USER_HOME lub $HOME:

$ ustaw ANDROID_SDK_ROOT=E:\Android\sdk\

Emulator $ -i Pixel_API_25

Zobaczmy teraz, jakie są najczęściej używane zmienne środowiskowe dla narzędzi Android SDK:

• ANDROID_SDK_ROOT: Służy do ustawiania ścieżki do katalogu instalacyjnego SDK. Po ustawieniu wartość zwykle się nie zmienia i może być współużytkowana przez wielu użytkowników na tym samym komputerze. ANDROID_HOME, który również wskazuje na katalog instalacyjny SDK, jest trwale przestarzały. Ale jeśli nadal będziesz go używać, automatycznie zostaną zastosowane następujące zasady:

Jeśli ANDROID_HOME jest uwzględniony i ma prawidłową instalację zestawu SDK, jego wartość jest używana zamiast wartości w ANDROID_SDK_ROOT. Ale jeśli ANDROID_HOME nie jest określony, używana jest wartość w ANDROID_SDK_ROOT. Jeśli ANDROID_HOME jest zdefiniowany, ale nie istnieje lub nie zawiera prawidłowej instalacji zestawu SDK, zamiast tego używana jest wartość ANDROID_SDK_ROOT.

• REPO_OS_OVERRIDE: Pamiętaj, aby używać tej zmiennej w systemach Windows, Macosx lub Linux, gdy używasz menedżera SDK do pobierania pakietów dla systemu operacyjnego innego niż bieżący komputer.

Zmienne środowiska konfiguracyjnego Android Studio

Zmienne konfiguracyjne Android Studio składają się z ustawień, które dostosowują lokalizację plików konfiguracyjnych i JDK. Na początek sprawdź następujące zmienne dla ustawień:

• STUDIO_VM_OPTIONS: Określa lokalizację pliku studio.vmoptions. Ten plik zawiera ustawienia, które wpływają na charakterystykę wydajności wirtualnej maszyny Java HotSpot.

• STUDIO_PROPERTIES: Służy do ustawiania lokalizacji pliku idea.properties. Ten plik umożliwia dostosowanie właściwości IDE Android Studio, takich jak ścieżka do wtyczek zainstalowanych przez użytkownika i maksymalny rozmiar pliku obsługiwany przez IDE.

• STUDIO_JDK: Identyfikuje lokalizację JDK, za pomocą którego można uruchomić Android Studio. Po uruchomieniu IDE sprawdza zmienne środowiskowe STUDIO_JDK, JDK_HOME i JAVA_HOME w tej kolejności.

• STUDIO_GRADLE_JDK: Służy do ustawiania lokalizacji JDK, którego Android Studio używa do uruchamiania demona Gradle. Po uruchomieniu IDE najpierw sprawdza STUDIO_GRADLE_JDK. Jeśli STUDIO_GRADLE_JDK nie jest zdefiniowane, IDE zastosuje wartość ustawioną w oknie dialogowym Struktura projektu.

Zmienne środowiskowe emulatora

Domyślnie emulator przechowuje pliki konfiguracyjne w $HOME/.android/, a dane AVD w $HOME/.android/avd/. Możliwe jest zmodyfikowanie ustawień domyślnych poprzez ustawienie następujących zmiennych środowiskowych. Komenda emulatora -avd <avd_name> przeszukuje katalog avd w kolejności wartości w $ANDROID_AVD_HOME, $ANDROID_SDK_HOME/. android/avd/ i $HOME/. android/śr./.

Warto zauważyć, że począwszy od Android Studio 4.2, zmienna środowiskowa ANDROID_SDK_HOME jest przestarzała i została zastąpiona przez ANDROID_PREFS_ROOT.

• ANDROID_EMULATOR_HOME: Służy do ustanawiania ścieżki do katalogu konfiguracji emulatora specyficznego dla użytkownika. W Android Studio 4.1 i starszych domyślna lokalizacja to $ANDROID_SDK_HOME/.android/. Jednak począwszy od Android Studio 4.2, zmienna środowiskowa ANDROID_SDK_HOME jest przestarzała, a domyślną lokalizacją katalogu konfiguracji emulatora jest teraz $ANDROID_PREFS_ROOT/.android/.

• ANDROID_AVD_HOME: Tworzy ścieżkę do katalogu zawierającego wszystkie pliki specyficzne dla AVD, które w większości składają się z bardzo dużych obrazów dysków. Domyślna lokalizacja to $ANDROID_EMULATOR_HOME/avd/. Możesz chcieć określić nową lokalizację, jeśli w domyślnej lokalizacji jest mało miejsca na dysku.

Ponadto emulator systemu Android podczas uruchamiania wymaga następujących zmiennych środowiskowych:

• HTTP_PROXY: Przechowuje ustawienia proxy HTTP/HTTPS (nazwa hosta i port) dla globalnego proxy HTTP. Używa separatora dwukropkowego (:) między hostem a portem. Na przykład możesz ustawić HTTP_PROXY=myserver:1981.

• ANDROID_EMULATOR_USE_SYSTEM_LIBS: Zawiera wartość 0 (domyślnie) lub 1. Wartość 1 oznacza użycie systemowego pliku libstdc++.so zamiast tego, który jest dostarczany z emulatorem. Tę zmienną środowiskową należy ustawić tylko wtedy, gdy emulator nie uruchamia się w systemie Linux z powodu potencjalnego problemu z biblioteką. Na przykład, gdy niektóre biblioteki sterowników Linux Radeon GL wymagają nowszego pliku libstdc++.so.

adb zmienne środowiskowe

• ANDROID_SERIAL: Musisz użyć tej zmiennej, aby podać numer seryjny emulatora, taki jak emulator-5555, do polecenia ADB. Jeśli ustawisz tę zmienną, ale użyjesz opcji -s do zidentyfikowania numeru seryjnego z wiersza poleceń, dane wejściowe wiersza poleceń zastąpią wartość w ANDROID_SERIAL. Poniższy przykład ustawia ANDROID_SERIAL i wywołuje adb install helloworld.apk, który następnie instaluje APK na emulatorze-5555:

set ANDROID_SERIAL=emulator-555

adb install helloWorld.apk

Zmienne środowiskowe ADB Logcat

• ANDROID_LOG_TAGS: tej zmiennej środowiskowej można użyć do ustanowienia domyślnego wyrażenia filtru podczas uruchamiania programu logcat z komputera deweloperskiego. Aby zilustrować przykładem:

set ANDROID_LOG_TAGS=ActivityManager:I MyApp:D *:.

• ADB_TRACE: składa się z rozdzielonej przecinkami listy informacji debugowania do zarejestrowania. Wartości domyślne mogą być następujące: all, adb, sockets, packets, rwx, USB, sync, sysdeps, transport i JDWP. Aby wyświetlić dzienniki ADB dla klientów ADB i serwera ADB, upewnij się, że ustawiłeś ADB_TRACE na all, a następnie wywołaj polecenie ADB logcat, jak pokazano tutaj:

set ADB_TRACE=all

ADB logcat

• ANDROID_VERBOSE: Zawiera rozdzieloną przecinkami listę pełnych opcji wyjściowych używanych przez emulator. Poniższy przykład pokazuje ANDROID_VERBOSE zdefiniowane za pomocą znaczników debug-socket i debug-radio debug:

set ANDROID_VERBOSE=socket,radio

Jak ustawić zmienne środowiskowe

Na koniec zobaczmy, jak ustawić zmienne środowiskowe w oknie terminala i w skrypcie powłoki dla różnych systemów operacyjnych. Należy pamiętać, że ustawienia zmiennych w oknach terminala trwają tak długo, jak okno jest otwarte, podczas gdy ustawienia zmiennych w skryptach powłoki pozostają we wszystkich sesjach logowania.

W systemie Windows: otwórz okno terminala i po prostu wpisz:

set HTTP_PROXY=myserver:1981

Alternatywnie możesz dodać to samo polecenie do skryptu powłoki za pomocą interfejsu użytkownika systemu Windows. Jeśli używasz Maca i Linuksa: podobnie otwórz okno terminala i wpisz:

export HTTP_PROXY=myserver:1981

Podsumowując, w tej części omówiono różne dostępne narzędzia do analizy i projektowania aplikacji. Przyjrzano się wielu dostępnym opcjom badania wydajności Twojej aplikacji z różnych aspektów. Nauczyłeś się korzystać z nowego monitora urządzeń z systemem Android, aby szybko testować pomysły, które później można wbudować w pełnoprawne aplikacje. Na koniec zagłębiłeś się w procedurę debugowania Android Studio i zobaczyłeś, jak używać wielu skrótów klawiszowych i technik przycisków podczas tworzenia aplikacji w Android Studio. Każde z tych narzędzi powinno zapewniać potężną kontrolę i wgląd, które można przećwiczyć w celu tworzenia niezawodnych aplikacji.