Разработка Android-приложений на языке Scala

Катков А.

Полезные ресурсы

http://www.scala-sbt.org

https://github.com/pocorall/scaloid

https://github.com/pfn/android-sdk-plugin

Scala project build definition (Build.scala)

```
val buildSettings = android.Plugin.androidBuild +
      sourceGenerators in Compile <+= (sourceManaged in Compile, version, name, androidBuildType) map { (d,
v, n, bt) =>
        val file = d / "info.scala"
        val target: String = bt match {
          case Some(Release) => "prod"
          case Some(Monkey) => "monkey"
          case _ => "dev"
        }
        IO.write(file, """package me.selfish.android.common
                         |object VersionInfo {
                         |  val version = "%s"
                         |  val name = "%s"
                         |  val target = "%s"
                         |}
                         |""".stripMargin.format(v, n, target))
        Seq(file)
      }
```

Using composition for activities

```
trait SideMenu extends SActivity { this: EventListenerActivity =>

  override def onPostCreate(savedState: Bundle): Unit = {
    super.onPostCreate(savedState)
    if (activateDrawerByDefault) {
      setupDrawerToggle()
      toggle.syncState()
    }
  }


  override def onConfigurationChanged(newConfig: Configuration) {
    super.onConfigurationChanged(newConfig)
    toggle.onConfigurationChanged(newConfig)
  }
}
```

```
private def preloadHeaderGfx(userInfo: UserInfo): Future[(Option[Bitmap], Option[Bitmap])] = {
   (ImageHelper.loadingFileFuture(userInfo.avatarId, Preloaded.bigAvatarWidth, Preloaded.bigAvatarWidth,
     Some(ImageHelper.getAvatarDisplayOptions(Preloaded.bigAvatarMask))) zip
     ImageHelper.loadingFileFuture(userInfo.coverId.get, CoverSize, CoverSize))
 }

 private def refreshData(): Unit = {
   HttpClient.talkToServer[UserInfo, GetUserInfoRequest](
     GetUserInfoRequest(userId), method = HttpMethods.GET) flatMap(result => {
     withDefaultErrorHandlingThroughFuture(result) { sc =>
       preloadHeaderGfx(sc).
         map (sc -> _)
     }
   }) foreach (infoWithPreloadedGfx => {
     infoWithPreloadedGfx match {
       case (userInfo, preloadedImages) => {
         runOnUiThread {
           setPreloadedHeaderGfx(userInfo, preloadedImages)
           initAdapters(userInfo, None)
         }
       }
     }
   })
 }
```

Request class example

```
final case class GetFeedRequest(
                                feedName: String,
                                itemState: Option[FeedItemState.Value],
                                sinceTime: Option[FeedItemUUID],
                                untilTime: Option[FeedItemUUID]
                                 )extends FeedName with SelfishRequest with RequestPagination {

  def path = Feed / feedName
  override def queryParams = (itemState.map"state" -> _.toString()) ::
    sinceTime.map "since" -> _.toString()) :: untilTime.map"until" -> _.toString()) :: Nil).flatten.toMap
}
```

```
def withDefaultErrorHandling[T](response: Either[ServerErrorResponse, T], preErrorHandling: => Unit = {})
      (onSuccess: T => Unit,
       onFailure: PartialFunction[ServerErrorResponse, Unit] = PartialFunction.empty)(implicit context: Context): Unit
= {
    response match {
      case Right(successfulResponse) => onSuccess(successfulResponse)
      case Left(err) => {
        preErrorHandling
        (onFailure orElse {
          case ServerErrorResponse(StatusCodes.Unauthorized, _) => SessionInfo.checkSession()
          case ServerErrorResponse(HttpClient.CustomSessionExpiredStatusCode, _) => {
            runOnUiThread {
                refreshSession();
              }
            }
          }
          case ServerErrorResponse(HttpClient.CustomNetworkErrorStatusCode, _) =>
            EventBus.getDefault.post(ConnectionLost)
          case ServerErrorResponse(code, msg) =>
            runOnUiThread { Toast.makeText(context, s"SERVER ERROR: ${code} - ${msg}", Toast.LENGTH_LONG).show() }
        } : PartialFunction[ServerErrorResponse, Unit])(err)
      }
    }
  }
```

```
private def preloadHeaderGfx(userInfo: UserInfo): Future[(Option[Bitmap], Option[Bitmap])] = {
   (ImageHelper.loadingFileFuture(userInfo.avatarId, Preloaded.bigAvatarWidth, Preloaded.bigAvatarWidth,
     Some(ImageHelper.getAvatarDisplayOptions(Preloaded.bigAvatarMask))) zip
     ImageHelper.loadingFileFuture(userInfo.coverId.get, CoverSize, CoverSize))
 }

 private def refreshData(): Unit = {
   HttpClient.talkToServer[UserInfo, GetUserInfoRequest](
     GetUserInfoRequest(userId), method = HttpMethods.GET) flatMap(result => {
     withDefaultErrorHandlingThroughFuture(result) { sc =>
       preloadHeaderGfx(sc).
         map (sc -> _)
     }
   }) foreach (infoWithPreloadedGfx => {
     infoWithPreloadedGfx match {
       case (userInfo, preloadedImages) => {
         runOnUiThread {
           setPreloadedHeaderGfx(userInfo, preloadedImages)
           initAdapters(userInfo, None)
         }
       }
     }
   })
 }
```

```
private def renderParagraph(acc: String, level: Int, x: JsValue, formatMentions: Boolean = true): String = {
   if (level == 10) acc else {
     x match {
       case d: JsObject => {
         d.fields.get(`Type`) -> d.fields.get(Contents) match {
           case (Some(JsString(Text)), Some(JsString(text))) =>
             acc + text
           case (Some(JsString(Mention)), Some(children @ JsArray(_))) =>
             if (formatMentions) renderParagraph(acc +"<b>", level, children) +"</b>"
               else renderParagraph(acc ,level, children)
           case (_, Some(children @ JsArray(_))) =>
             renderParagraph(acc ,level, children)
           case (Some(JsString(Br)), _) => "</ br>"
           case _ => ""
         }
       }
       case JsArray(elements) =>
         acc + " " + elements.map(renderParagraph"", level + 1, _).trim).mkString" ")
       case _ => ""
     }
   }
 }
```

```
def renderFormattingTree(tree: FormattingTree[FormattingNode]): List[JsObject]= {

  val rootElement = tree.getRootElement

  def renderLoop(node: FormattingNode): List[JsObject] = {
    val children: List[FormattingNode] = tree.getChildren(node)

    JsObject(Map(Type -> JsString(Text), Contents -> JsString(node.text) ::
      children.map(child => {
        child match {
          case s: StyleFormattingNode =>
            JsObject(Map(Contents -> JsArray(renderLoop(child).toVector)) ++ child.getTypeEntryMap())
          case m: MentionFormattingNode =>
            JsObject(Map(Contents -> JsArray(renderLoop(child).toVector)) ++ child.getTypeEntryMap())
          case t: FormattingNode => JsObject(Map(Type -> JsString(Text), Contents -> JsString(t.text)))
        }
      })
  }

  renderLoop(rootElement)
}
```