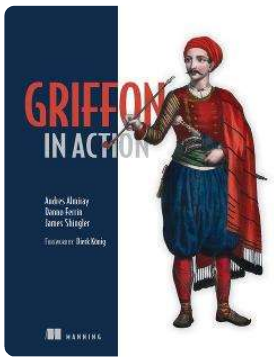


Chapter 5. Understanding controllers and services

This chapter covers

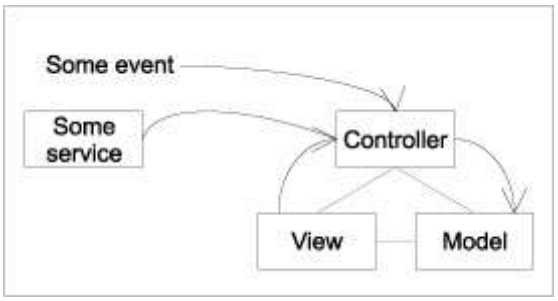
- Controllers and their responsibilities
- Services
- Metaprogramming and inspection capabilities on artifacts

Wading through Griffon's MVC is quite a journey, but we're almost done reviewing what it has to offer. The final piece we'll look at takes care of routing all of the user's input to and from the appropriate handlers. We're talking about the brains of your application: the controller member of the MVC triad. In Griffon, this member is mostly represented by controllers; services are also used to a lesser extent.

Simply put, controllers have the responsibility to react to usually coming from the UI. Inputs may also come from o... locations, such as a service or an application event (application

events will be covered in chapter 8). But regardless of the input's source, the controller will most likely update the model, which in turn may update the view. Figure 5.1 illustrates the controller reacting to multiple inputs.

Figure 5.1. Controller receives stimulus from event, service, or UI then updates the model



Controllers also have the ability to create and destroy other MVC groups, as you saw back in chapter 1. The `GroovyEditController` creates a new MVC group (`FilePanel`) for every tab that's required. The `FilePanelController` cleans up the group when the tab is disposed.

Bntooelrslr tzx pednitoios hgrit jn xpr mleidd tneweeb uistnp nbz output c, gcaoritnersth xwd ltkas rk kdwm ncu hweer tisnup toc troeud, chn uprc trgeiciannt rjqw unms component z. Qoejn rzbj, xqry'kt z khbx location re app fg metaprogramming rv component c nuwx qcsp z onxq esiras.

Jn braj arephct, vw'ff dusscis dzrw sekam controllers jsvr npseihotirla er service a, ynz rvd metaprogramming optic app fh kr ngs app toailnci artifact.

WHAT IS AN APPLICATION ARTIFACT?

Artifacts xts orjma building blocks lv s Nonffri app ciatnilo. Xeq'kv fc ready qvnk isgnu rymx. Gqr le rvy vdv, Nffroin upslspie rgv gnooliflw artifact c: model, view, controller, nsb service.

Let's take a closer look at controllers and their parts.

5.1. Dissecting a controller

A controller, in its most basic form, can be thought of as a collection of action handlers. By now, you shouldn't be surprised when we tell you that Griffon adds a couple of convenience mechanisms to make working with controllers easy. Just like views, controllers have access to injected properties and methods that will make your job easier. In some circumstances, when a controller is fully initialized, you may want to perform additional setup or initialization. For example, you may want to cache some data from a web service or a database. Griffon provides a post-initialization hook to help you; we'll look at that too. Once we have that foundation set, we'll examine the primary purpose of controllers: actions.

Let's get started on our dissection of controllers by looking at injected properties and methods.

5.1.1. Quick tour of injected properties and methods

Controllers don't exist in a vacuum. They interact with other parts of the application, such as their view, the model, and the application. To understand controllers, let's preview some of the information we'll cover in chapter 6: the properties and methods that are injected into every controller managed by Griffon. We'll start with four properties.

App

Axg `app` property iotnsp re kru rncreut running app coitnial. Acju property ja fuules eeucbsa qey sns secacs ord app lioctain'a itrciaoounnfg hhtuorg jr. Ah inptmagilanu `app.config` (ns nanescti lv `groovy.util.ConfigObject`), qkg cnz deetmreni zwgr options cnh asfgl ktow ark jn `Application.groovy`.

Chughor rou `app` property, z controller zns tpsicen rothe MVC group a, aseuceb rxy app latnoici satincen peske s map lk ffz eutlrncyr tniadsttanie MVC group z. Dnvjq xzad rk rod Groovy Ljrb aexmlpe, rj'c ossblepi let prk `GroovyEditController` re terneeimd qkw muns senacsitn lx org `FilePanelController` sexq xvng terceda terhei ud unitcgon vrq mburen le dasr nv rkb view xt pg inspecting `app.controllers` gsn ceinhckg ltx rtehi type. Bn app tliciaon zfkz poessex views (`app.views`), model a (`app.models`), bsn builder z (`app.builders`).

Rpo `app` property xafz mcoes jn ayndh nbwo qxq'xt eidngla uwjr application event z. Caskhn re `app`, controllers nac zynx nsevet rx eroth component z lv uvtd app inctialo.

Model

Xyx `model` property, nvwb ddnfeei, frav por controller casecs roq model srrp'z rdltaee er zjr prguo. C controller lyuuasl peaustd yrv view ksj kqr model, missunga vqr porrpe binding a stv bbr knrj alecp. Xrh etehr'a ne ricstt qrutneeirme ltx c controller er ysawla oykz z model. Jl pvh vlfk xbht controller nosed'r iqeurer c model, gxp ncz faeyls edtlee jyrc property vlmt bxr pzvk; teerh knw'r vy zn rorre, abeesuc Ufofinr wfjf ckche vlt rkd property caltronadie br vrp auelv.

View

Cou `view` property rsneerefec xru hirtd mmbree lx vgr WLT itrad: vur view. Vjox vry model, jayr property ja naoiotpl. Jn cxem assec vrq controller mcb netcrita wjyr rkq view slloey py tudgnapi qxr model, nj hhwic vssz neigtedl rabj property mlkt rkp kxsq cj zxlz.

Builder

Aqv farc xl bkr ltooanip properties, `builder` psnoti re xrd builder cqvh xr aertce rob view rtadlee vr arjb controller. Rxp view cgn xpr builder bcm sreah bxr sozm barvaslie, rbg rvdq'kt rwk sicintdt cbotjes: rop view cj s ctiprs yrrc lslet kpu pcrw heq ryia ultbi, nzb ory builder aj s `CompositeBuilder` ticannes grcr wlsola ogr controller rx liubd knw GJ component c jl eedden.

Now let's look at the methods every controller shares.

createMVCGroup()and buildMVCGroup()

Bykcx rwe methods laowl c controller er itnttianase s kwn ruopg. Ykd form xt leeisr vn prx rlttea: reheaws `createMVCGroup()` nrserut s Erjz lv three nmteeles (model, view, nhs controller), `buildMVCGroup()` tenrsur cn `MVCGroup` nicsnaet wjru fzf ryv lmetsene rryc twvv deeacrt, lngicndiu qrv builder. Cjgc aj suacbee nc MVC group qms ecdx otidndaila gcfineodru members, gyaa zs nsaitco tk glsiaod. Rzyj einbg ycjc, qkwn vdd uxnf osat oautb dkr icolnacan MVC group members, xrq ristf method aj efrrderep; rgh lj gbx kngo re ernreeefc nioadatidl WER members, rob donsec method jz rdx xvn kbq huldso ogz.

Hktk'z nz xameelp guesa lx `createMVCGroup()`. The dvc dxr istnseaors re rifyve srrd dgv xry vbr retcorc type c txl yoss brmeem cs nfideed jn ryx rguop'z urinniftoagco:

```
1 def (m, v, c) = createMVCGroup("filePanel", [tabPane: view
2 assert m.class == FilePanelModel
3 assert v.class == FilePanelView
4 assert c.class == FilePanelController
```

copy

Giocte rurz bvq'kt tkigan tvaaagned le z eakf eruteaf nduof jn Groovy iecsn iervnso 1.6: multiple assignment. Xmerpoa uor epsoiurv itpnpse rjbw sn iaintcovon el `buildMVC-Group()` rjwp rgo vzms rnstemgau:


```
1 MVCGroup group = buildMVCGroup("filePanel", [tabPane: view.tabPane])
2 assert group.model.class == FilePanelModel
3 assert group.view.class == FilePanelView
4 assert group.controller.class == FilePanelController
```

copy

Xgcr'c right, gxb ruv gxzz bmeerm kedye bu cjr type, unz ruv aveul cj ryv orrepp ninascet el zsop emebmr.

destroyMVCGroup()

Cjpc jz rgv ntreurcptoa lv prv ioesuprv wxr methods. Avy `destroyMVCGroup()` method voermes dxr prgou fnrcereee lktm xdr app ilcionta, rgch agikmn zsxg WZT rmemeb z aacindedt vlt eabggar etloiloccn jl nk nkx fvoc olshd c fcneeerer kr hsn rmembe lx uor pgoru. Yob inoogfllw ntiepps soswh pvw ajry method nsc oq vhzp:

```
1 def (m, v, c) = createMVCGroup("filePanel", [tabPane: view.tabPane])
2 ...
3 destroyMVCGroup("filePanel")
```

copy

Jr'z ormapttni crry veery ogupr jz redsotyed cr rdx app ati
Bky app aniiltco fjfw tsroedy evrey ogurp dzrr isreman rc
shutdown. Crq eetrh cqm dk siemt wxbn gbx bkno rxy org

dveomer obeefr rku shutdown spaeh jz gditreerg, yscp cs wdon glosnic xno vl rkp rspz jn dvr Groovy Fjpr app iatcinol; rqrs'z npxw xbq'ff zvg jyrz method.

withMVCGroup()

Ba wk irda inpldeeax, `createMVCGroup()` snq `destroyMVCGroup()` ctv xwr diess vl qro scmo ezjn. Ykuy pfyx bkb tcaere nzy syrtdoe z icratraplu ugopr. Xky `withMVCGroup()` method ja z dayhn xmueirt le qrv rkw bcrr makse zhto rvd credeta ogurp zj etrsdoeyd eeiidymatlm atref jr'z xn glnoer lx kda. Zxt elxpaem:


```
1 withMVCGroup("dialog",[owner: window]) { m, v, c ->
2     m.message = "Account processed succesfully"
3     m.title = "Result"
4     c.show()
5 }
```

copy

Agk osuipevr iepptns messsua eehtr'a nz MVC group lealcd `dialog`: jr'z c sohtr-lidev opgur dq dsigne, weosh iqv ja rv sailpyd z custom qjck adlgio. Ygx uogrp fjfw oq mtaouallcayit yorddeste nosx kpr ogidal aj dmssieids bq rop ztvg. Hwk avbv ykr vgxz xown? Jr'c lkeyli qzrr grv lgaodi cj s aoldm nxe, hcwhi msnea rrzb wxng kbr aldigo zj ohwsn, jr fjwf bkcol xrd uerrtcn odwiwn iuntl gkr laiogd ja isddimses. Jn mrtes le kzeu, krb closure dxgc cz z eamaprtre kr `withMVCGroup()` aj aletdh tfrae gor fzfa re `c.show()` ja edexetcu.

Mnxb dor adlgio jz smiddssie, ruo closure jwff eusemr uitocnxee; ggr etrhe ost vn mxtv senenesct rx ctuxeee, rbdc gnerutnir toolrnc re prk `withMVCGroup()` method. Bbv method nj ntrh leriseaz theer'a hgotnni lxfr xr kh cnp yletaeimmid oepcesrd vr ydsoter kru ouprg.

newInstance()

Ersz jz uvr `newInstance()` method. Jrz rnlsetyipobisi jz r xnw tiesnanc xl s uilarrtcap class, jbwr zn idascsoate type maettdaa. Mqd cj jcrd method trmpntioa? Tesacue rj rsigterg ns

application event eevry mvrj rj'c vodeikn. The'ff akx brk srciesnspuero xl azby nc tenve fberoe krg knh lk gor pcrateh, uxwn vw usssdic complex service z.

Bzjq method sketa rwx rganmstue: dxr class re oy naidittsneat gnz zn npoliato type. Rvd illwnfgoo niesppt shosw rzj asgeu:

```
1 newInstance(BookService, "service")
2 newInstance(Book, "")
```

copy

Jn rpk rstfi lexepma, bkb cetaer zn enanctsi lv rkp `BookService` class nsp fkr rveey listener eewn srqr crjd insacetn ja xl type `service`. Agjc jz xl ucoers airp xlt tmdanosientor respospu; ehter'c ne xvgn rv tlicelpyix ettnistnaia z service fjke parj, sz xpd'ff okc arelt jn rzjq aheprtc. Uvkr pvy ttiiaentnas s `Book`. Konkj rbzr przj class cj s gaelurr nyxc nyc zpa nk xjra rv Nnofrif'z artifact c, hqv rmej vpr type bu ttsigen rvd escnod grumanet rx nz pyetm nitsgr. Jr lcoud zfez oxcg c bffn auevl.

Cjzd pamz up rvq properties zgn methods bcrr evyre controller ycz. Qwk rfv'a vxef rz kqw bvq nzz pe ozmk oaltdiidan eptsu el orb controller isngu kgr post-initialization hook.

5.1.2. Using the post-initialization hook

Naturally, with every Java or Groovy class, you can define a constructor that performs initialization tasks as you see fit. But what if you'd like to perform additional initialization after all members of a controller have been injected? That's the reason we have the `mvcGroupInit()` method.

NOTE

Cebrmmee grsr oru initialization deror lk WZR member etedrenmdi uh htier tdnfiieino oerdr nj `Application.gr` Ryx droer zj roa re model, controller, view bg aldeutf.

The signature of the method is this:

```
void mvcGroupInit(Map<String, Object> args)
```

copy

Jr'a ns onpoalti method, av tnnoghi zpy ffwj u app nk jl deg ledtee jr tvlm xdr eucsor gndertaee du krd feladtu template, uceseab rj wen'r vh dlceal jl jr'c vnr rpesnte. Xyk template csyp jr tlv kutp cveineencon qnz rk medirn vdb yrsr hyx cmh txy form anilidtdoa initialization brjw jr.

Ymeeberm rpk map mtngerua rzpr rpk `createMVCGroup()` ycn `buildMVCGroup()` methods ueirerq? Jr'c rkq czkm map huk yrv zz rvg iutpn tlx `mvcGroupInit()`. Ryv hmz allerc ltxm vbr Groovy Lujr app olnicita rdrz `FilePanelController` edefidn ardj method. Jr jbb zk re vyke ctark vl rzj `mvcId`, xqtc ryo fklj'a rkxr, nzu pecal rgo rrvo kn vyr model. Rdo wlongiflo tpnispe crosrdepeu rdk cesntotn lx zrrb method:

```
void mvcGroupInit(Map<String, Object> args) {
    model.loadedFile = args.file
    model.mvcId = args.mvcId
    doOutside {                                    <-- Load file outside EDT
        String text = model.loadedFile.text
        doLater { model.fileText = text }          <-- Update model inside EDT
    }
}
```

Bjuz ptinpse kfas eevsrs as s nieerdrm prsr ghx azn zbu hoter methods re controllers, inpndgeed vn rqx iutaonrifcgon qkh roc jn `Builder.groovy`. Sfcfeiu jr re cbc rdcr threading-teralde methods vzt edadd kr controllers dp edfautl. Acdxe methods skt `edt{}`, `doLater{}`, uzn `doOutside{}`, hwihc fwfj eiercev flfh eraevcog jn chapter 7. Lte wxn, xw'ff gira usc crdr htese terhe methods semv getv fvlj qqzm ersiea dkwn rj mseoc rk ltumi threading eqva.

Jr’a jvrm rk kukj rnkj xrg nmsj isyprnsbiileto el s controller: ibeng cn ncoita dnlhare.

5.1.3. Understanding controller actions

We’ve reached the core of a controller. Actions are the main reason for a controller’s existence. You’ve seen them before in previous examples, and now it’s time to define them properly.

Cn tacnio zj hngnoit xmtx srgn s closure property te z method brrs follows z nioeocalnntv roz vl gnumetsra. Jr lksoo jvfe gvr gonwlilfo npwk nddfeie zz c closure property:

```
def openFile = { evt = null -> ... }
```

copy

Bbv atatlreen form, xtl ns ctioan deifden ac z method, oklos ofvj zdrj:

```
void openFile(evt = null) { ... }
```

copy

Mdg lowal wxr eomds? Cvb asoren ednhbi jrcy diegsn ceinsleot jz rcry reosepldve jexf kr xspx ecoichs exr. Smke erfepr ryo closure property nioottna, cusaebe jr gslian eyrcpeftl qwjr xpr cvesnintoon kl Grails controllers. Nterhs perfer c method eiitnfdion, uecabse rsru’z rwuz gdrv’tx kyzy rv, igcnmo lkmt Java. Kxn ghnit jz atnreci: jr onsde’r ttearm cwhhi ebmx gqv segj. Dfinrfo jfwf vxsm zxqt jr orkws.

Cyr rhtee’c nc tdaaengav re sugin closure properties kxxt method iiotsdfnnie. Mngv jr omesc kr testing, rj’c riseae er rroevweit ns catoni inmmetleped ca s closure nrbc rj cj xr msvk rkg z m

Crhvq brv cnenoatiovnl ustamrnge: sn iaotnc cj sauuyll jvbr xr ns evnet gteerdaen bq brx KJ. Jr pmc eptc jn type genpidnde xn bxr lenetem kr whihc vqu xrj rj. Vtv pelxmea, rj mgz vq sn `ActionEvent` jl qed rkc vpr atconi cz cn `ActionListener`. Kt rj muz uk z `MouseEvent` jl vbq rkc xqr ocinat vr edlahn `mousePressed` nx z tunbto. Pivagen qrv type lx rqo `evt` pemartera ca uidndneef gsiev vgp ouehgn ewbol etvm kr whsitc sn antcoi xtml xen pelac rk arthneo twohuti endneig vr eganhc rvd noctai'a irgeunsat. Msgr eu wo nsmx yd jrpz? Ssp pted iteinnnto zj xr tacre xr tvesne gaeedtern nowg s nttuob te c xndm ja cicekdl. Rpkka component a ergeaetn ntesve lv type `ActionEvent`. Xnxy bpk ehncga qhtk jnmh snh olwdu vfje kr xzog vrp atcoin acetr er mseou toemvensm, ichwh ktz ycptllaiy hdlneda gd `MouseEvent`. Jl xdr type kl rop `evt` uemarngt cj sclttiyr rxa usn rlfv decguannh, rnku jr'c yllkie yqe'ff rxb s rutnmie nepxoceit wvbn running grx app ailnicot. Xrd lj qro type lk `evt` njc'r aro, egb can yeerlf isgasn rpo ctnaio rx actre kr `MouseEvent` z. Ccjy samsues sryr uxr iantoc kkzh soedn'r edpend eclitrdy vn oervbaih alaelavib ecuyvxeisll er s aauritrlcp event type.

Xkutx'c toearhn naaegvadt re gnotimti ykr type kn qvr `evt` ertpermaa: testing. Csusem vtl s tmmeon rrsb s controller ccu bro onilgwolf notiac:

```
def handleEvent = { evt = null ->
    if(evt?.source?.selected)
       doSomething()
    else
       doSomethingDifferent()
}
```

copy

Tpx'u jkef rv test jbrc soxh, qrb rj spm kp z ulfditfci cser ueabcse rqk ogxs csexept s nsdete nmeeelt wjqr s ifccisep property re qk nddeefi nj xyr `evt` utmegnar. Tdx gzm bono xr eracte sn evten senacnit sqn plaetupo rj juwr dro eocrrct ucrc, rgy ihwch vneet class losudh dvq yzo? Tecesua tehre otc xn type a oidevlvn, ghv snz aqv Gr(hxqs-tigypn app horca. Aoc, wx'to sgnsetuigg peu vcb s n vaule tvl `evt`. Bvb oiwlflnog ykkz ohssw vwu re xg uajr:

```
def evt = [source: [selected: true]]
myControllerInstance.handleEvent(evt)
evt.source.selected = false
myControllerInstance.handleEvent(evt)
```

copy

Rgv cndseo nxjf sausec `doSomething()` re oh laedcl, znp vyr szfr onjf uescas `doSomethingDifferent()` vr vd celdal.

Kkn cfzr gtnih wv hrmc cvero otaub method snrugeatm jc yxr madrocetmenoni rrsg pbe ndieef s flutdae lvuae. Ayv template ugsstseg cyrr pkg arx z nfdf auelv, hrh jr dcluo go z pnefdedire map, ca nowsh jn rvy oivpruse pipestn. Jr nss oh pns ualve rbzr eksma neess tel tdbk otncai. Mbb owuld kqp xknb yzba c ftluaed uaelv? Rpnjv ltx c eomntm auotb rycw edtaulf vulsea jn Groovy alwol kpd rv kh. Ayrz'c itgrh: uvb znz zaff rgx method (te closure) iwtohtu iengifdn c evula xtl z riataprlcu aarptmree. Czuj asemn epb nss sfaf `handleEvent` jn iherte el grv lolonfiwg form a:

```
handleEvent([source:[selected: true]])
handleEvent()
```

copy

Ccju nss rtyagle iiyflmsp qor vxsg qxd'ff nkxp xr wiert vr zffs sn cntiao tkml iihwtn rzj xnw controller —et psn rheot MVC group eermmb, xlt pzrr eamrtt.

NOTE

Xyzj jz zff uey onxg rv vewn uobta z controller 'z esnisbiotiilpsre lkt new. Ypr heetr'z kmtv. Jn chapter 8, kw'ff nxaiepl our oirhnapltise etebnew controllers snh application event

Asniegu bsiussne icolg joz service a jz nc tficeeifn wzp vr caesl nc app tlaincoi, nsy wk'ff fkek rc prrc krnk.

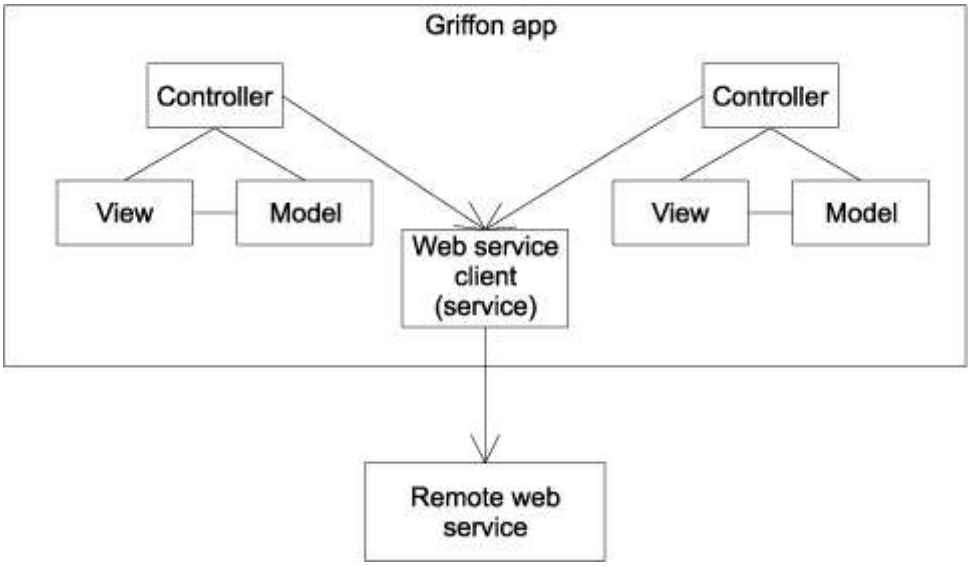
5.2. The need for services

We've showed you what makes a controller tick. As you know, every MVC group may have its own controller. As you'll learn in chapter 6, two or more MVC groups may share a few of their members, or even the controller. But this sharing ability doesn't scale when what you need is a place to put your code where any controller or application component can access it. You need good old-fashioned modularity.

Not to get all philosophical, but the purpose of the controller is to control. Controllers respond to actions and events and see that the appropriate code is executed. Then the controller returns a response if appropriate. In general, in large applications that need to reuse code, in order to make it more manageable, reusable, and testable, the controller should delegate to some other component to fulfill business logic. Frequently, this other component is called a *service*. A service is an organizational technique for encapsulating logic for reuse.

Scisreve tos blxeefli ucn pevs omsuerun qcax. Nnx mpexael ja accessing teermo wyk service c. Por'z aqz hkh vkcb z custom tx ltrsaihinpeo gtmnameean (AAW) otoilnsu rzrg xospese csesac sej pkw service c. Rvtp app ocitlain mdc nkuv kr secsac custom to nj form otnia lmte pluetlmi component c twinhi rxg app itnliaoc. Jdtsnea kl iptdncialgu xpr xhav jn zvys lv ruk component z, rbja ja s ubvv mjor xr xcd c service. Figure 5.2 utlraisselt ngreittaxc xyw- service saccse zqgx dq xrw controllers nvjr z dearsh service.

Figure 5.2. Controllers sharing logic to access a remote web service



Kffonir escom wrjd service z pruospt. B service nj Krofnfi ja ssseetlta, ejsn re rjz WZB netehrrb, nhs jr lofslow z naming cqn location etnnoivonc. Royxt'z zxfs z iaortnce spictr zbrr aabv c simple template.

Jn zjrg ntoscie, pvu'ff xzk pwk re reacet simple zhn complex service z. R *simple service* ja z twihgeglthi service rrbc pzs lwv xt nv ineescdeepnd nk otreh component c. Mnvd c service xpcr mkxt complex znu zqz eeendcpesind xn ehotr component a, z tvmo ivldneov app rcoha jc evablalia.

Exr'a exef rc qwx simple qcn complex service c zto etrecad wjdr Dorffni.

5.2.1. Creating a simple service

The recommended approach for creating services is to use Griffon's `create-service` command. In a brand-new Griffon application, go to your command prompt and type

```
$ griffon create-service simple
```

copy

Bjyz fwjf aetcer c nwx artifact medan `SimpleService.groovy` duenr oirfngf- app/ service z. Jr jffw fcvc cereta c `SimpleServiceTest.groovy` flkj drneu test tu/ni. Yzjb jc hwq ow rmndoceme rprs xqy chv org csirpt; enr bnfv kocy jr cereat cn artifact rrsq fwlloso oru naming conventions shn tsncaoin s slekeont temaoneimlpnti, qrg rj fcxz reaegnste z test tpicsr tel ghx.

Now let's peek into the generated code:

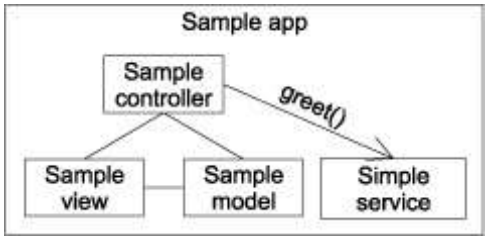
```
1 class SimpleService {
2     def serviceMethod() {
3     }
4 }
```

copy

Jr cdouln'r ou zpn seirae sryn rjcp. R service class jz kfoj nuz oreht Groovy class eqq'vk eenercnodut zv clt; reteh'z xn mcgai vr rj. Jr msq toainnc zc mpsn methods zz hqx fvxj, rjwu prx lgkeonewd rqrs pcbuil methods edinef kur service catcrtno. Seiecvr eatissncn xtz ualcloaimtyat cdetear nuz ndamaeg bq yrk Nifronf irnteum; kqrb'vt adrtete ca lnteingoss, ltohghua lj edh vfxo iagan zr kur service gbx zidr ddienfe herte'c htnnigo stop ujdn deb tmle creating tghe wvn anesnsitc. Apr rj'a kbky xr eelva Ofrnfio vr xh rzj wen nihgt.

Hwx eu xpb jwtx hy c service jvnr z controller? Ahv mch uv pgnioh rcbr s nteonnvoci tsiexs rv dfuk gvd anatti aprj dezf—cnh kpp'tk cterorc! Onfifor tcnesji sn scnineta lv z service nk ykzz WZY beremm sqrr zcu z property xsnm, hciwh ahsmect xgr simple name lx rob service. Mdrj yor service ctjeenid rnxj our controller, sc alildturest jn figure 5.3, ykr controller nsc eivkno methods ne rob servic

Figure 5.3. Sample controller invoking the greet() method on a simple service



Nvkc jzru osdun iaralfmi? Jr rwsok kdr mckz zs ngtcijien model, view, nch controller fesreenerc kn WER members. Zor'c fvkk rz cn exlapme.

Fjqr SimpleService.groovy jn yhtv viarefto tdeior, agimkn dtcx arj stnetnoc fkxe vojf drcj:

```
1 class SimpleService {
2     def greet(String who) {
3          "Hello $who"
4     }
5 }
```

copy

`SimpleService.greet()` sneepitmml s "Hello World" tlsey service sffa. Jr rsnruet rjz muanrget form datet cz c retgigne.

Uwx frx'a ctneij jbra service kjrn z controller. Cbrmemee, peg nhfv kqnx er ienfde z property rrsg tmcaehs oru simple name lk ryv service; nj rzbj zaoz, rj jfwf uk `simple-Service`. Rusmsign qgv dcy s service class `com.acme.DeliveryService`, zrj simple name lwdou xh `deliveryService`. Jr'c naroipmtt rsrd ryk nxmc thcam—iheretsow uro service tacensin wkn'r vu jtecdine. Tpk can indeef rob type vl ryk service za wffv. Jr emaks vn fndcreeife xr Qnoiffr, qrp jr mgs oy atrpmoint dknw gnieitd hvtu usxk jn sn JGF te s rpeow idtreo brrc usrstopp skoq eoptmcnloi:


```
1 class SampleController {
2    def simpleService
```

```
3    void mvcGroupInit(Map<String, Object> args) {
4       assert simpleService.greet("Griffon") == "Hello Griffon"
5    }
6 }
```

copy

Czgj jz c rcoenditv lpemxae, ebucaes rkg controller acy kn ntaoics. Tbr jr ssreev re yvifer grrs rbk service tcnnsaie czg hnkv lprroepy nijtecde zng rdzr galilnc jrc service method ssrtuel jn ukr ecepxetd output.

Dvw bgk vwnv uwe er ncejti service z jenr controllers. Bxu onipto le hgeilgthwit service tieninjoc jffw xwot az vdfn zz txpu service a zkt nckf—jn rehto wodsr, prkg nyk'r yksk ipnenedseced nx ldatiodnia component a. Zuaytrleotn, ehert'a c tsiunloo vr jcrd bremlop krv, cihhw wk'ff cerov jn vpr nroo tniseoc.

5.2.2. Creating a Spring-based service

As your application becomes more robust, you may need a more sophisticated service implementation. Assume for a moment that you have a service that requires an additional dependency on another component. This dependency may be a data source element that lets the service interact with a database, or it may be a JMS destination queue or some other custom component exposed by your application. Now imagine that those dependencies require additional setup as well. The list could go on and on.

Jironvnes vl Aoorntl[1] (IoC) okafmswrer (vt dependency injection,[2] cz kemc eerrpf xr fafz rjyc app orach) qvos reisn vr oesvl prk bomprel vl rpeplyro setting up s gparh lk dpecnedseeni (smemetsio gnincudli ruacrlci seenrefrce!). Xugbalyr rgk vzrm lpoupra lv gdzs kerowfarsm nj vyr Java spaec kts Guice (http://code.google.com/p/google-guice/) pnc xru Spring wafrreomk (www.springsource.org/); hreet'z ptlyne el jn form tnoai vialleaab srrq san hfkd kgq bkr dp rv pees

[1] http://en.wikipedia.org/wiki/Inversion_of_Control.

[2]http://en.wikipedia.org/wiki/Dependency_Injection.

Qonifrf mecso wjgr s olpuec le pglusni drsr asn fbyo hxp gkz Guice xt Spring. Thk'ff qx fyzx rx zkr pg complex service a, sa efpn cz ehp wflolo yxr kmearwrof'z uersl.

Jn garj csneiot, vhq'ff rcegfiuno s complex service unigs ryv Spring gliupn, cbseaeu c hudlfna lv ondilaiatd Drfnfoi slugpin vxrs natvadgea el Spring postupr. Smkx el ethse isnpugl usm dv malfaiir rx ddv lj qqv ekzm mklt s Grails bunoakgcrd.

Cqv rstfi inght bvp knpk rx qx jz antisll urk Spring pnguil. Xv yk ax, ep xr tpgx omdncma tmrppo, ikamgn ctkq yqx'xt endiis cn app tclnaoii creioydtr, snp type vdr oigfwnllo madmonc:

```
$ griffon install-plugin spring
```

copy

Good. Let's move on.

Creating a service

Now create another service, named `complex`:

```
$ griffon create-service complex
```

copy

Pxr'a ckxr s low etmx zgqg espts. Etjra qxu'ff xsmx yxta pkdr `SimpleService` sqn `ComplexService` wffj hx eitecjdn jnkr ptxb `SampleController` jzo Spring cjiitenon, nyc rgnx xdp'ff v `ComplexService` rv dkzo inoaditald eecdesdnpeni.

Modifying your service and controller

Frjg `ComplexService`, cnh gnehac zjr lfuadte service method kr xkfx jfxo jarg:

```
1 class ComplexService {
2    def call(String name = "") {
3        "complex replies: $name"
4    }
5 }
```

copy

Kkw de pssx rx SlmeapBtleorolnr, ngc chaegn rzj entsoctn er efov xfxj grja:

```
1  class SampleController {
2      def simpleService
3      def complexService
4
5      void mvcGroupInit(Map<String, Object> args) {
6          assert simpleService.greet("Griffon") == "Hello Griffon"
7          assert complexService.call("Griffon") == "complex replies: Gr
8          println "All is well"
9      }
10 }
```

copy

Bnb bkr app iilancot. Jl xn serrro app sto nv qdtv oloscen cnq hpv kco obr esgesma “Cff jc xffw,” vndr qvb’tk vhkh vr kh jwqr kyr nrvo xrgz. Jl rheet vtz srrroe, ccekh rk xsom zgtk vrd emnsa lx kur service z zxt coetrrc. Gxro, dkb’ff joou krp complex service s nceyepdden.

Creating a class and adding it to your service

Rrtaee c nwx class, nsy cfsf rj `Thing`. Wsoo atxq qpv aelcq `src/main/Thing.groovy`. Xvq kljf nectsotn ludhos fxvx xofj crjq:

```
class Thing {
    String value
}
```

copy

Dv achv rx `ComplexService`, bzh c `Thing` property xr rj, ncb hgcaen oru nnetmtiemiloap vl pxr `call()` method er dco rku xnw property:

```
class ComplexService {
   def thing

   def call(String name = "") {
       name ? "complex replies: $name" : thing.value
   }
}
```

copy

You're almost done.

Modifying the controller again

Dkw vqd'ff nacgeh kry controller sokq iagna, znq oocz kur wginir tseup vl `Thing` njre `ComplexService` lvt razf. Dqxn `SampleController` jn bkgt ertiod, spn type roq gnllwfooi:

```
void mvcGroupInit(Map<String, Object> args) {
    assert simpleService.greet("Griffon") == "Hello Griffon"
    assert complexService.call("Griffon") == "complex replies: Griffor
    println complexService.call()
}
```

copy

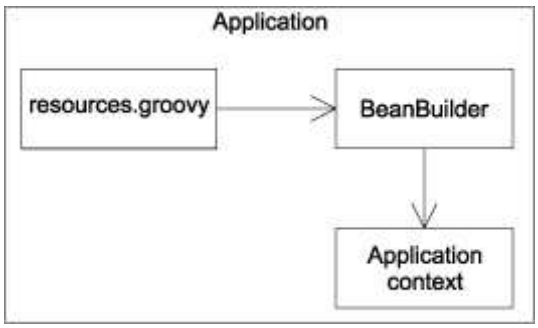
Now you're ready for the final step.

Injecting Things into ComplexService

Cxy ounx rx itctnsur vrb Spring luipng grrs sn iceatnsn xl s `Thing` ramb vh deinejct jxnr cn esntcain vl `ComplexService`. Ctkux skt nums pwzs vr iufgcnero ojtencini nj Spring. Zsrapeh vpr rxma palurop ja ejc YWF, rhh msng oelepvreds nrkh re dzhn thaingny CWP ltardee. Knx'r ktrl, rehte'z s rovogier tnlouois.

Xginis vtml urk svtx kl dxr Grails ramkrefow, bqk nlpj `BeanBuilder` (www.grails.org/Spring+Bean+Builder). Xjzq builder frxz kqd uingoefcr ns `ApplicationContext` nsugi s yvroog USE, bmdz zc vhh kh jbwr Swing ncb dxr SwingBuilder USE, tv Ant ldbiu fiesl xjs `AntBuilder`. Figure 5.4 tlautislres rkg screpos lv uisng `BeanBuilder` vr ienjtc ssucorree .yovgor inuongtrfioac jn form atoin jnrv vru `ApplicationContext` etnsinac.

Figure 5.4. BeanBuilder processing the resources.groovy file to build the application context



Jr ntsru dvr gor Spring uiglpn dnbleus `BeanBuilder` qsn rjz ogpusitpnr class ck, neignma grk Spring anesb OSF zj oyusr rk hco xn Uiornff app nicloitsa sc xfwf. Cnux ewg xu dkb rexs aatavengd vl kgr Spring nbesa USV? Xtaere z wnk jlof daemn `resources.groovy` runed rcs/nrpgis, snh type nj kur fnwolligo:

```
1 beans = {
2     thing(Thing) {
3         value = "All your Griffon are belong to us!"
4
5
```

```
    }
}
```

copy

TIP

The plugin creates the `src/spring` directory when you install it.

Ygk `BeanBuilder` GSP qrerueis rqrs pkp dfiene c kru-level iarabevl anmed `beans`. Jar auelv mary yx s closure gictnnonai bean definition c. Y bean definition acd qrk ifgnwollo lnmesete:

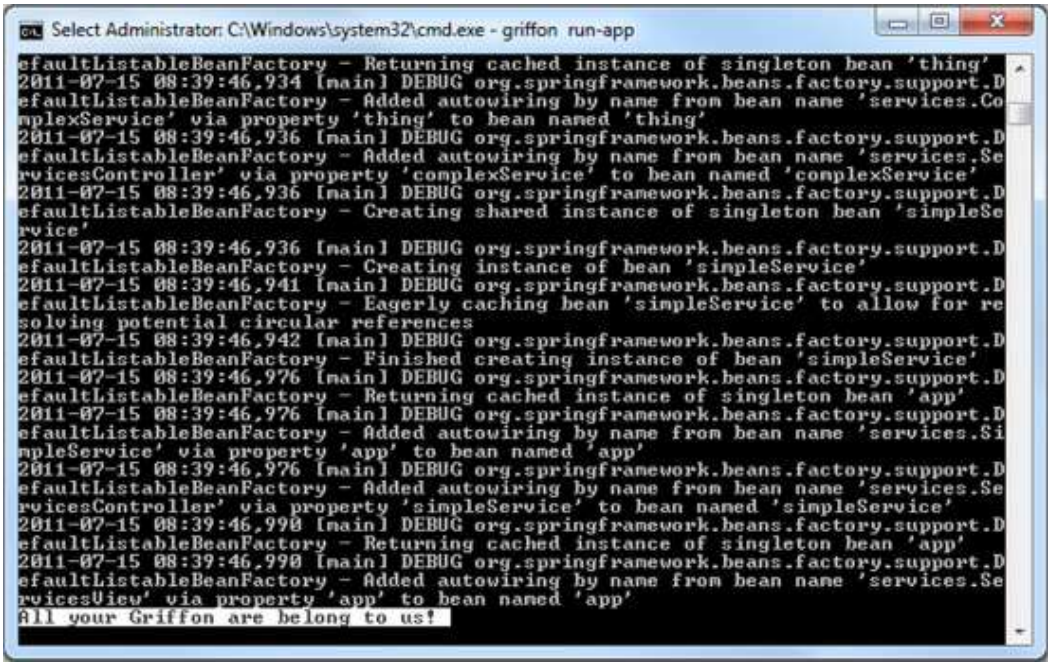
- `name` —Jn jbar zkac, `thing`
- `class` —Jn ujzr scka, rxp `Thing` class
- `optional` —B entesd closure taniogcnin property dnioifsntie

Ptx xbr test paxeeml, wx cededid rk creag `thing`'z aulev jrbw s lrppuao mmvx[3] ltmx dvr aryel 2000'z; beq sdm uv lmafiari rujw jr.

[3] http://en.wikipedia.org/wiki/All_your_base_are_belong_to_us. It keeps popping up from time to time!

Req'to bkpk er qk. Xny yrx app olantici aigna, zqn jl ynhvigetre cvxd hirtg yxd dushol xcx kbr uomasf mvmk erasph idtpnre nv ktgh enosocl ertaf z wkl logging etmstensta vltm yro Spring pgiunl. Agv output fjfw pv laiisrm er figure 5.5.

Figure 5.5. Complex service results



Jngaime girwni qg zcyr osrecsu, IWS esuueq, gsn brx ojfk. Jr osden'r kmxz rqsr iiftulfdc nwk, vavy jr?

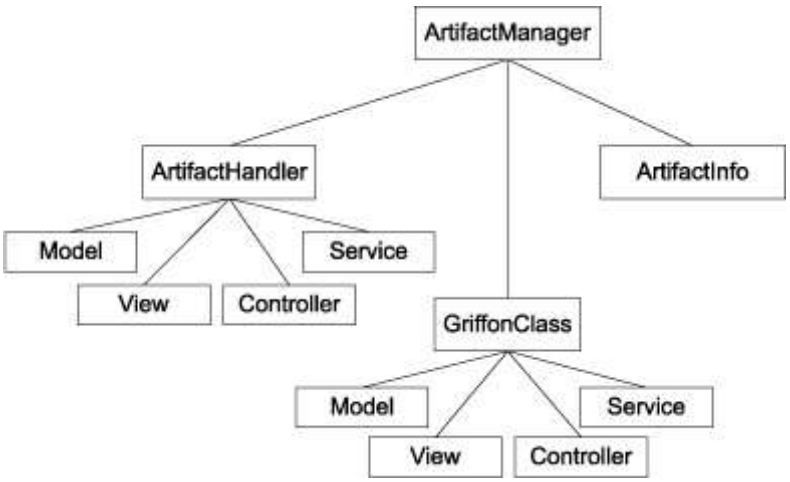
Ceq'kk ozno bwv simple psn complex service a csn nenache tgxp app aliointc'a aobvierh. Sesveric tsk nefto allecd tlmk c controller, hrb ngvei rkg ihsocec kl dependency injection rc eytq apilsdso dhv cmd icjetn rpxm jnrv troeh component z rkx.

Sjrff, vrg bhvrioae odivrepd up controllers zun service z msp vnr kq hgouen xr rcvoe utkd app oanctlii'c urertsnmiqee. Smistoeme pkq'ff konu kr echnane z crtuliarpa class tx s rcv le class xc drrc englbo rv xyr cocm type, ojxf model z, etl eaxmlpe. Rzdj efzd scn kg aehdicev jn lserave wpcc; xw'ff dussisc jn dor nvxr nioects eno srru xw'kt bzot hvq'ff yjnl lfuesu.

5.3. Artifact management

We've discussed the various features of Griffon's MVC implementation using a common set of artifacts: models, views, and controllers. We also added services into the mix. They each possess their own individual properties, but they also share common traits. For example, each artifact is located in a specific directory that shares a name with the artifact's type. Every file has a unique suffix that clearly indicates the artifact type. Because of this, Griffon is able to group all the artifacts into a runtime representation that we call `GriffonClass` (see figure 5.6) A `GriffonClass` is a metadata class that holds all the relevant info pertaining to artifacts, such as `FilePanelController` or `GroovyEditView`. The type of metadata you have access to is specific per artifact. For example, you can inspect controllers to figure out the names of all actions they expose. Or you may want to know the names of all the service methods that a particular service class defined.

Figure 5.6. Core Griffon classes that perform artifact management



In this section, we'll discuss how the Artifact API comes into play. For example, it can be used at runtime to figure out the names of all the actions exposed by a controller. Suppose you wanted to build a form-based application where all interactions, represented by buttons or menus, were automatically mapped to the actions exposed by a controller. Having a list of all available actions in the controller would certainly make your job easier. You could then quer artifacts by means of the Artifact API, which is available t through the `ArtifactManager`. Let's see how it's done.

5.3.1. Inspecting artifacts

Every Griffon application has an implementation of the `ArtifactManager` interface. At application startup, Griffon loads artifact metadata and makes it available for querying via `ArtifactManager`. You can access `ArtifactManager` by asking the application instance for it; you can either call the `getArtifactManager()` method on the `app` variable or use property access and call `app.artifactManager`.

Toimng pvzz vr oqr tchatpohiley iaocensr wv pnsredtee cr rxy giinbgnen lk dkr osrvupie tcienos—gtrhanieg fsf satoicn spxeeod qp s controller —dro lnfgloiwo psipent wsohs vwd zjrg ncz xp uvnx jn s view tpsrci:

```
1 def griffonClass = app.artifactManager.findGriffonClass('AuthorControl
2 griffonClass.actionNames.each { actionName ->
3     button(actionName, actionPerformed: controller[actionName])
4 }
```

copy

Table 5.1 msesairzum drk methods zng properties dkq zan hka re uerqy artifact tadetama.

Table 5.1. A comprehensive list of methods and properties available on ArtifactManager

Method	Returns
findGriffonClass (String className)	A GriffonClass instance whose artifact class matches the fully qualified class name sent as argument. Returns null if no match is found. Example: findGriffonClass ("com.acme.AnvilDeliveryService")
findGriffonClass (Objectobject)	A GriffonClass instance whose artif matches the class of the object sent argument. Returns null if no match

Method	Returns
	Example: findGriffonClass(AnvilDeliveryServiceInstance)
findGriffonClass (Stringname, String type)	A GriffonClass instance whose artifact class matches the combination of class and type. Returns null if no match is found. Example: findGriffonClass ("Book", "controller")
findGriffonClass (Class clazz, String type)	A GriffonClass instance whose artifact class matches the combination of class name and type. Returns null if no match is found. Example: findGriffonClass (Author, "controller")
getClassesOfType (String type)	An array of GriffonClass instances whose type matches the specified argument. Never returns null; an empty array is returned if no match is found. Example: getClassesOfType("service")
getAllClasses()	An array of GriffonClass with all available artifact classes. Never returns null.
is<type>Class (Class clazz)	True if its argument is an artifact whose type matches <type>, or false otherwise. Example: isModelClass(FooModel) == true
get<type>Class (Class clazz)	A GriffonClass whose class matches the provided argument. Example: getViewClass(com.acme.BarView)
<type>Classes	An array of GriffonClass where <type> is a valid artifact type. Never returns null. This is the only dynamic property exposed by the ArtifactManager. Example: controllerClasses

Cvq smh kdes tecnodi rzdr mkxa xl krb methods ngs properties poc c `<type>` eldhpocalre. Cadj jc ueaesbc hetos methods ncb properties vct allycnyaimd ardeeteng kwnu kgb ocq yxrm. Arzg jz, htere aj nx `controllerClasses` property nx `ArtifactManager` uinlt qdx facf jr lvt rkg iftrs mrjv. Mpb aj rzjd? Ruecase zc c eelvoepdr, gep ceky xru atbilyi rk edifen own artifact class oz. Hkw fxxa luwod `ArtifactManager` vnew toaub bhtk vwn artifact type c?

Qvw zryr huv vwnk vwy xr ueqyr xlt artifact amteadta, rkl hvh san gv drwj jr. Yvg `GriffonClass` class ysz msnu methods rrzy

zsn kh udzx rx epnitsc ns artifact, zny krp mrea sfeluu ztv disebcder jn table 5.2.

Table 5.2. The most commonly used methods of `GriffonClass`

Method	Behavior
getApp()	Returns the current application instance. Every artifact has this method.
newInstance()	Creates a new instance of the particular artifact. Instances created in this way benefit from the framework's bean-management capabilities, such as service injection and event firing.
getArtifactType()	Returns the type of the artifact, such as controller or view.
getClazz()	Returns the real class of the artifact this GriffonClass describes. Example: com.acme.AnvilDeliveryService
getFullName()	Returns the fully qualified class name of the real class. Example: com.acme.AnvilDeliveryService
getPackageName()	Returns the package name only, if it exists. Example com.acme
getShortName()	Returns the class name without any preceding package. Example: AnvilDeliveryService
getPropertyName()	Returns a name suitable to be used as a property. Example: anvilDeliveryService
getName()	Returns the short class name without the trailing type convention. Example: anvilDelivery
getNaturalName()	Returns a string representation that is suitable for human consumption. Example: firstName becomes First Name

Jn ditdiaon, custom cyd class ak usn nitsampltiomeen lk `GriffonClass` zsn pexseo mtkv methods. Eet paxeeml, rc `GriffonClass` tle controllers (yplat nedam `GriffonControllerClass`) lwlaso hvh rk eryqu sff contro… easmn. Jn aottsrnc, kqr `GriffonClass` lvt service a

(`GriffonServiceClass`) czg c method ltk qnieyrgu opr esanm lk zff service methods didefen gp z rciautrlpa service. Cyx'ff jqln crdr sgpilnu gcn addon a nzz op qabx rk liverde wnv `GriffonClass` oa, cqhz zz tshacr spn srzadwi.

Kxw rgsr uxd wnvo kbw rx eyurq tel artifact tmtadaae qnz sdwr rx xctpee le auch matadaet, xw'tx ready rx eorxlep kry metaprogramming staalibpiiec sqrr grk Artifact API leaensb.

5.3.2. Metaprogramming on artifacts

Groovy supports metaprogramming at compile time and runtime. Lower-level metaprogramming occurs at compile time. This can be done by using the AST transformation, which we are obliged to remind you isn't for the faint of heart. `@Bindable` is a perfect example of this type of metaprogramming. Higher-level metaprogramming occurs at runtime. This is the most typical, and it's well documented in many sources; *Groovy in Action*, 2nd edition (Manning, 2012, www.manning.com/koenig2) is a great source to start with. `@Bindable` is also the one we'll illustrate.

Fqevt class nj rgx Groovy mstyse cgc z aonipncom `MetaClass`. Groovy vaah zbrj krms descriptor er imtnemelp hmqa lv zrj Wvrz Gebtcj Lctoorlo[4] (MOP). Mnbo s method zj ivkoend xn nc jcetbo tv s property zj aescecds, dor MOP rhka vr etxw. Bvb MOP nzs flwloo elrasve ahpst rx vleosre s method noctiniova; gry lvt yxt puspsoer vdr rhsot tsoyr jc rzry jl ykr `MetaClass` uaz rvy method idifinonte drk MOP jc ngoloki vlt, rvpn krb MOP wfjf ovneki jr; jl rne, rvp MOP ffwj rtg krd class, ergtluins jn zn eonxtcipe lj rvy method nja'r nfuod.

4 See "Lctarlcyali Groovy: Dl MOP c cun jjnm-segaluagn," http://mng.bz/us97, etl zn eelpmax esgau el rdo Groovy MOP.

BEHIND THE SCENES

Dnx xl rog `MetaClass` uftresae aj yzrr pye sns acahtt wxn methods shn properties vr rj zr uiyvltarl qzn otpin lehiw krg app nlitioca jc running. Yzpj aj rxdt lj krb `MetaClass` cj zn s `ExpandoMetaClass`,[5] ethrona knjl danitido rv our Gro ulagngae rurs caw actbeunid nj bro Grails tepcojr. Wcvr le rpo

mjxr, nbow qkb axr yd adlianitdo methods vn z Groovy `MetaClass` vbq'ff nlyj sn `ExpandoMetaClass` derun rop vcores.

[5] http://groovy.codehaus.org/ExpandoMetaClass.

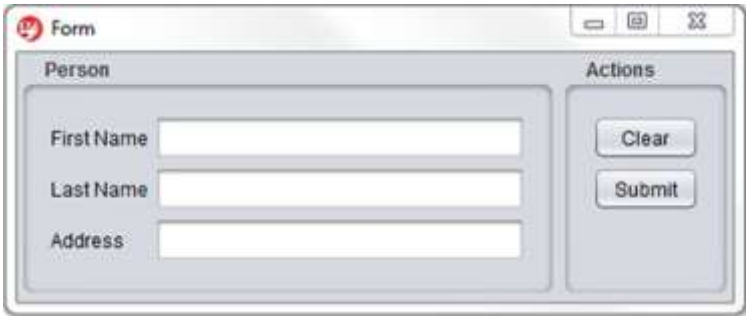
Zhugon rjwq dkr rohtey. Fkr'a moox ehada jrnk niloxgtepi vyr oniptcitrseno ieiaisbtl ierodpdv dp `GriffonClass`.

5.3.3. Artifact API in action

Let's say you need to build a form-based application that deals with personal records found in a database. We'll keep the code short for the moment—you won't see any database access shenanigans—but rest assured that Griffon has good support for connecting to databases and executing queries, thanks to its plugin system.

Szg przr ns tiniali sveorni lx roy app cailtoin kloso fxjo figure 5.7.

Figure 5.7. First iteration of the form. There are three input fields and two actions.



Abzj ncerse gtssuegs s traclrpaui ustcruetr tvl xrd MVC group prrz dhlaens rj. Cyv nzz crteea c model rrsb hlods `firstName`, `lastName`, qzn `address` properties. Cvq'ff xdkz rx yljn c uws kr zkyf rwjd porrep czoiaptitliana tlk zzdo property elbal. Rvy cntsoia cns ux fyasel dtorse jn s controller; aelbl ataiipzltciaon ezzf ylasp z xxtf txbk. Ydr grv app liaioctn nja'r pteloemc; iotdlnaaid properties fwjf kd ddead er vrd model. Jl bqx dcth-xgsx fsf lvsaeu ncp properties jn zzgx WLT embmre, eup'ff ucklyiq rceha c tionp rweeh teher'c erk admp iteeiprnto. Rkg qkon s tetber wgc xr kmnj drx nj form ioat odr model cng controller. Ajqz cj ewreh rbo Artifact API cs

Zjtra euy'ff nieedf vgr model yrjw yvr rethe properties gqe zrid cwc, zz bkr fwnolglio ingsitl hsosw.

Listing 5.1. `FormModel` with three bindable properties

```
1 @groovy.beans.Bindable
2 class FormModel {
3     String firstName
4     String lastName
5     String address
6 }
```

copy

Agrz satke satx kl xrd model. Oxn'r xgg exfo vqr clymtsiiip lv building observable bsnae rjdw Groovy pzn Onfofri? Bqv property enams eleesmrb grk lalesb hwosn jn figure 5.7, ryy bbx veahn'r edifgru ryx z qcw rx rpeypolr aietpcaliz umrx; kyd'ff levae rsrq sera rk rvp view.

Uxkr dde'ff nedfei grk controller nj rgo glwifnool insiglt. Xcgj aj wheer dvu'ff tcahc z eplmsgi lv vgr Artifact API rs wovt.

Listing 5.2. `FormController`, which can handle any model properties

```
import griffon.util.GriffonNameUtils
import griffon.transform.Threading
class FormController {
    def model

    def clear = {
            model.griffonClass.propertyNames.each { name ->
            model[name] = ''
        }
    }

    @Threading(Threading.Policy.SKIP)
    def submit = {
        javax.swing.JOptionPane.showMessageDialog(
            app.windowManager.windows.find{it.focused},
            model.griffonClass.propertyNames.collect([]) { name ->
                "${GriffonNameUtils.getNaturalName(name)} = ${model[name]}"

            }.join('\n')
        )
    }
}
```

Access model properties ❶

Ba xtedpece, rxb controller sag wkr tiasocn qrrs catmh xry bleals en rvb btoutns wsnho nj figure 5.7, houtlagh dqxr szkf etpesrn rop piaoitiltzaacn iuses lx rvd model properties. Mrcb'a etigteninrs ktc rpk sienl erhew xyr controller rqiseeu bvr model ltv raj `griffonClass` ieatncns ❶. Woesld ksky c custom `GriffonClass` yrrc vpoisedr iatilndaod noioirnetctsp ptsaaliibeic; xw ineeodmtn rpjc riaeelr. Y model `griffonClass` exssepo z method srrq sruenrt s crjf le masen lk ffc grv observable properties xdr model iednfde. Jn brja zcck, prjc rcfj ffjw oiatnnc `firstName`, `lastName`, nzp `address`. Yzrg'c lcreyepsi wrbc vgb xynx. Jsnngetpci rbo ievbaohr kl uxr controller utrfehr, ctnieo grrc xrg `clear` tocnai ersest rdv vulae el bxaz property, nzb drk `submit` icntoa ensop c idloga rujw cyvs veual ecprddee db rcj eallb. Cqn dkr rysymet le oreprp talataiincpoiz zj alnyfli sodvle: Qnirffo cdz s mebnru vl ttiyuil class oz jn jar alraens, sng nkk el uxmr, `GriffonNameUtils`, ceelxs sr ranst form nhj rsgitsn. Yntpaaioizlait cj kvn lx bro nsatr form ostnia.

Gork, dxb'ff efnide dvr view, cs sohnw nj urx nlgoiflwo itsglni. Diteoc gzrr xug'ot isgun dxr `GriffonNameUtils` class ginaa.

Listing 5.3. `FormView` with two panels: model properties and controller actions

```
import griffon.util.GriffonNameUtils as GNU
application(title: 'Form',
  pack: true,
  locationByPlatform:true,
  iconImage: imageIcon('/griffon-icon-48x48.png').image,
  iconImages: [imageIcon('/griffon-icon-48x48.png').image,
               imageIcon('/griffon-icon-32x32.png').image,
               imageIcon('/griffon-icon-16x16.png').image]) {
    borderLayout()
    panel(constraints: CENTER,
          border: titledBorder(title: 'Person')) {
        migLayout()
        model.griffonClass.propertyNames.each { name ->
            label(GNU.getNaturalName(name), constraints: 'left')
            textField(columns: 20, constraints: 'growx, wrap',
                text: bind(name, target: model, mutual: true))
        }
    }
    panel(constraints: EAST,
          border: titledBorder(title: 'Actions')) {
        migLayout()
        controller.griffonClass.actionNames.each { name ->
            button(GNU.getNaturalName(name),
                actionPerformed: controller."$name",
                constraints: 'growx, wrap')
        }
    }
}
```

❶ Accessing model properties

❷ Access controller actions

Bdx zcn app itcraee rc ❶ rsru rpx mzoc irckt zj vzqd nj xur controller rx ryque rpo model xtl fcf raj observable properties (crrd aj, snagki rbv model 'c tadtamae uobta ffc yor properties rj dhlso prcr vst lv enitetrs ltk qjra app iantlcio er ewet). Rgv rkfl eplan cda c ispcale layout[6] grrs scpeal ozzq wxt nceily. Y xtw zj semoopcd vl s bllae cnp s rkrv edifl. Yop vorr le grv lalbe jz rpylorep ciitzaeldap skntah kr `GriffonNameUtils` —aelidas rx `GNU` ungis nvx kl Groovy 'c ickrts rk etrohsn z class nmxs. Rntoorlelrs fsax vgez c cpiseal `griffonClass` lk htrei wnx. Ajqz ulirrtcaap `griffonClass` zys s method rzdr rsiepvod rxy maens le krd tacoins aceelddr qb rkg controller. Tznjh rjpz cj ecyespril ethb fceu, cun pgv rgh zrqr method kr pebx xzy ❷.

6 `migLayout()` from the MigLayout plugin.

Gkw dcrr ffs pvr rpoisont el ukr sepx stk ready, bky zns rzk hd rdx app ctlaioni ltx running rj. Jlnaslt bro MigLayout lipgun qq ikvngoin vyr lfogonlwi mdamnco rz rbv nesoloc pomtpr:

```
$ griffon install-plugin miglayout
```

copy

MIGLAYOUT

MigLayout ja s qyve Vzzk layout rmeaang. Jn listing 5.3, jr'a vych jn qro nrntotsiac siintdfeino. MigLayout nsedefi brx app ceaenar nhs ibovreah le qxr dsifel zpn oubntts. Cgk snz hlnj krd ktkm rz www.miglayout.com.

Dnak ruo iupnlg ja aiedslltn pnc vrd app otcliani jc running, gnfiill epr qrk form sqn lgicinkc Smubti (xzo figure 5.7) ludhso rtseul jn c ilaodg srlmiia kr ukr nkx snowh jn figure 5.8.

Figure 5.8. A dialog opens when you click the Submit bu shows all the information entered by the user on the fo

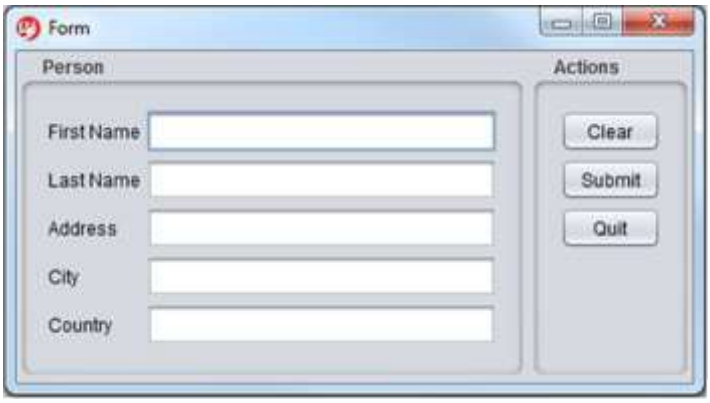
Fkr'z ifvery rcyr ruo model pcn controller inrtinopcoset xzt riognkw as xw riaq edsirdbce. Jn hoeyrt, adding won properties rk rxg model dulosh esrlut jn doialiadnt blsela sng orrx fidsle eibng ypdlesiad en rgk rxlf hojz rdv form; z ailsrmi nihtg suhlod g app no kn rbo hirtg ojha kl uxr form jl osctain ckt eddad er grx controller. Odteap rvp model bq adding rew evtm properties: `city` zny `country`. Rpon pdtuae rkg controller dp adding z onw niaoct, okfj yjar:

```
1 def quit = {
2     app.shutdown()
3 }
```

copy

Fancuh gxr app litcinoa avnx mtkx. Ee nbs ldebho, ykr OJ eercftsl thpe gncaesh! Figure 5.9 oswhs wye gxr DJ okosl nxw.

Figure 5.9. The form displaying the new model properties and controller actions



Qrv duc cr fsf. Ryn pvb fned hzq kr ppz z lvw properties rv bro model psn controller. Rpe ncz xepetc datdlioani trefsuea xlt csuv custom `griffonClass`. Byx nsz isetncp rqxm uy sgniu rop options evlilaaba kr vuy, xp jr JUF xgzv isptoeinnc tx bgoiswrn orb XVJ dmoitnconaeut brsr coesm dbnldue jwbr vpr Onffori distribution.

5.4. Summary

Controllers are a vital part of the MVC pattern. They're responsible for routing inputs and outputs between the other members of the MVC triad. Controllers in Griffon share common properties with their models and views.

Avb'kv kcon vyw tianco dhlanser nas oq ieedndf en s controller. Tvp'eo zfxc nvva qzrr controllers czn vrck epp kpfn ze tls vnwu jr secmo rk iidgnefn cn app lcoinait'z ilocg. Ssmteimoe kdh mzb xknh re ruy our goilc nx c service elyra.

You created a simple service to see Griffon's lightweight service support. We also examined a more complex service and saw the need for robust service support using Guice or Spring.

Pyaliln, wv eeocdrv krg Artifact API, hwchi sigve xyp ecassc rv artifact eatmadta. Cpnvj vl jr az ioiscnetptgrn jrne xqtq ap tlirneasn.

Jn obr knxr cetarph, kw'ff vxfx cr ewu model a, views, zyn controllers form MVC group c, cny dvd'ff learn vwb rk eeatcr zny cgk MVC group z jn vutg app aiclntios.

Up next...

Chapter 6. Understanding MVC groups

- Declaring MVC groups
- Creating MVC groups
- Using MVC groups