Dependency Injection in Scala

Dave Gurnell

underscore

AKA

99 Ways To DI

Cake

Guice

Constructors

Macwire

Cake

Thin Cake

Reader/ReaderT

Free

Functions

XML (LOL)

Traits

Cake

Guice

Constructors

Macwire

Thin Cake

Reader/ReaderT

Free

Functions

XML (LOL)

Traits

What is DI?

#1

Build app from components

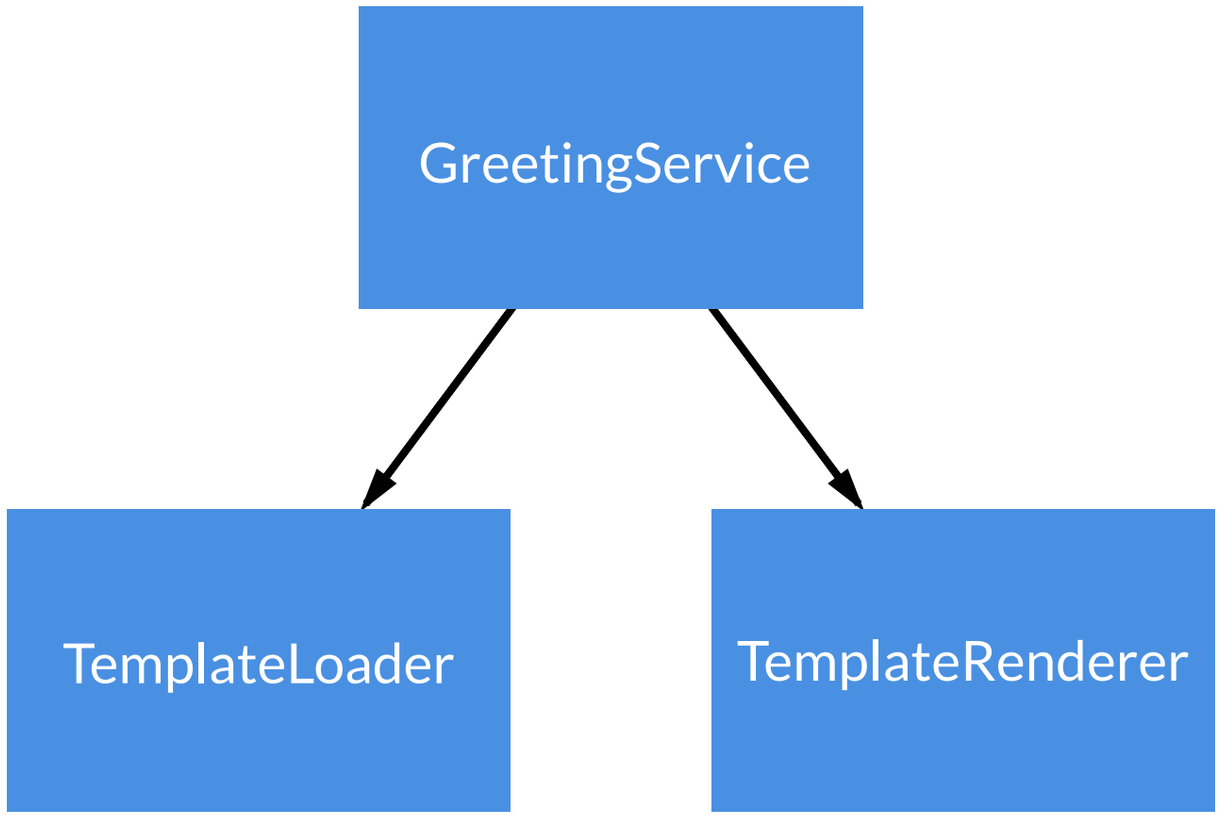
#2

Swap parts out for testing

#3

Do it all at compile time!

Give me an example!


GreetingService
TemplateLoader
TemplateRenderer

```
object TemplateLoader {
  def apply(id: TemplateId): Template =
    ???
}

object TemplateRenderer {
  def apply(id: Template, params: Params): String =
    ???
}

object GreetingService {
  val loader = TemplateLoader
  val renderer = TemplateRenderer

  def greet(id: TemplateId, params: Params): String =
    renderer(loader(id), params)
}
```

```
"greeting service" should {
  "render a greeting" in {
    // Test body:
    val params   = Params(Map("name" -> "Dave"))
    val actual   = GreetingService.greet(Greeting, params)
    val expected = "Test greeting for Dave"

    // Postconditions:
    actual should be(expected)
  }
}
```

```
trait TemplateLoader {
  def apply(id: TemplateId): Template
}

class S3TemplateLoader extends TemplateLoader {
  def apply(id: TemplateId): Template =
    ???
}

class FakeTemplateLoader(template: Template)
    extends TemplateLoader {
  def apply(id: TemplateId): Template =
    template
}
```

```
object GreetingService(
  val loader: TemplateLoader = ???
  val renderer: TemplateRenderer = ???

  def greet(id: TemplateId, params: Params): String =
    renderer(loader(id), params)
}
```

Constructor-based DI

```
class GreetingService(
  loader: TemplateLoader,
  renderer: TemplateRenderer
) {
  def greet(id: TemplateId, params: Params): String =
    renderer(loader(id), params)
}
```

```
object App {
  val loader = new S3TemplateLoader()
  val renderer = new MustacheTemplateRenderer()
  val greetings = new GreetingService(loader, renderer)
}
```

```
"greeting service" should {
  "render a greeting" in {
    // Preconditions:
    val template = Template("Test greeting for {{name}}")
    val loader = new FakeTemplateLoader(template)
    val renderer = new MustacheTemplateRenderer()
    val greetings = new GreetingService(loader, renderer)

    // Test body:
    val params = Params(Map("name" -> "Dave"))
    val actual = greetings.greet(Greeting, params)
    val expected = "Test greeting for Dave"

    // Postconditions:
    actual should be(expected)
  }
}
```

```
trait Fixtures {
  val template = Template("Test greeting for {{name}}")
  val loader = new FakeTemplateLoader(template)
  val renderer = new MustacheTemplateRenderer()
  val greetings = new GreetingService(loader, renderer)
}

"greeting service" should {
  "render a greeting" in new Fixtures {
    // Test body:
    val params = Params(Map("name" -> "Dave"))
    val actual = greetings.greet(Greeting, params)
    val expected = "Test greeting for Dave"

    // Postconditions:
    actual should be(expected)
  }
}
```

```
object App {
  val loader = new S3TemplateLoader()
  val renderer = new MustacheTemplateRenderer()
  val greetings = new GreetingService(loader, renderer)
}
```

```
trait Fixtures {
  val template = Template("Test greeting for {{name}}")
  val loader = new FakeTemplateLoader(template)
  val renderer = new MustacheTemplateRenderer()
  val greetings = new GreetingService(loader, renderer)
}
```

Gotcha

```scala
trait Components {
  val config: Config = Config.read(configuration) match {
    case Left(err)  => configReadError(err)
    case Right(cfg) => cfg
  }

  // Low-level config:
  val urls: Urls = Urls(config)
  val timeouts: Timeouts = new Timeouts
  val metrics: Metrics = Metrics(config)

  // Email modules:
  val emailer: Emailer = config.notification.email match {
    case config: TestEmailerConfig => new TestEmailer(config)
    case config: MailgunEmailerConfig => new MailgunEmailer(config, wsClient)
    case config: SendGridEmailerConfig => new SendGridEmailer(config)
  }

  // Database modules:
  val dbConfig: PostgresDatabaseConfig = new PostgresDatabaseConfig(config)
  val mapDatabase: MapDatabase = new SlickMapDatabase(dbConfig)
  val notificationDatabase: NotificationDatabase = new SlickNotificationDatabase(dbConfig)
  val surveyDatabase: SurveyDatabase = new SlickSurveyDatabase(dbConfig)
  val uploadDatabase: UploadDatabase = new SlickUploadDatabase(dbConfig)
  val userDatabase: UserDatabase = new SlickUserDatabase(dbConfig)

  // Redis cache modules:
  val redisClient: RedisClient = new RedisClient(config)
  val forgotCache: ForgotCache = new RedisForgotCache(redisClient)
  val sessionCache: SessionCache = new RedisSessionCache(redisClient)
  val signupCodeCache: SignupCodeCache = new RedisSignupCodeCache(redisClient)
  val unsavedSurveyCache: UnsavedSurveyCache = new RedisUnsavedSurveyCache(redisClient)

  // S3 access:
  val s3 = S3(config)

  // Service modules:
  val mapService: MapService = new MapService(mapDatabase)
  val mapboxService: MapboxService = MapboxService(config)
  val newsletterService: NewsletterService = new NewsletterService(config, wsClient, timeouts)
  val notificationService: NotificationService = new NotificationService(notificationDatabase, emailer, urls)
  val organizationService: OrganizationService = new OrganizationService
  val uploadService: UploadService = UploadService(uploadDatabase, s3, config, urls)
  val reportService: ReportService = new ReportService(uploadService, mapboxService, urls)
  val signupCodeService: SignupCodeService = new SignupCodeService(signupCodeCache, timeouts)
  val surveyService: SurveyService = new SurveyService(surveyDatabase, mapService, notificationService, unsavedSurveyCache)
  val userService: UserService = new UserService(userDatabase, forgotCache, organizationService, notificationService, newsletterService, timeouts)
  val authService: AuthService = new AuthService(sessionCache, forgotCache, userService, signupCodeService, organizationService, notificationService, newsletterService, timeouts)

  // Controller modules:
  val uploadJson: UploadJson = new UploadJson(uploadService)
  val authController: AuthController = new AuthController(authService, timeouts, urls)
  val mapController: MapController = new MapController(mapService, authService, timeouts, urls)
  val newsletterController: NewsletterController = new NewsletterController(newsletterService, authService, timeouts, urls)
  val notificationController: NotificationController = new NotificationController(notificationService, authService, timeouts, urls)
  val organizationController: OrganizationController = new OrganizationController(organizationService, authService, timeouts, urls)
  val reportController: ReportController = new ReportController(surveyService, authService, reportService, timeouts, urls, wsClient)
  val signupCodeController: SignupCodeController = new SignupCodeController(signupCodeService, authService, timeouts, urls, actorSystem)
  val surveyController: SurveyController = new SurveyController(surveyService, authService, timeouts, urls)
  val userController: UserController = new UserController(userService, authService, timeouts, urls)
  val uploadController: UploadController = new UploadController(uploadService, surveyService, authService, timeouts, uploadJson, urls)
  val preFlightController: PreFlightController = new PreFlightController
```

```
  // Redis cache modules:
  val redisClient: RedisClient = new RedisClient(config)
  val forgotCache: ForgotCache = new RedisForgotCache(redisClient)
  val sessionCache: SessionCache = new RedisSessionCache(redisClient)
  val signupCodeCache: SignupCodeCache = new RedisSignupCodeCache(redisClient)
  val unsavedSurveyCache: UnsavedSurveyCache = new RedisUnsavedSurveyCache(redisClient)

  // S3 access:
  val s3 = S3(config)

  // Service modules:
  val mapService: MapService = new MapService(mapDatabase)
  val mapboxService: MapboxService = MapboxService(config)
  val newsletterService: NewsletterService = new NewsletterService(config, wsClient, timeouts)
  val notificationService: NotificationService = new NotificationService(notificationDatabase, emailer, urls)
  val organizationService: OrganizationService = new OrganizationService
  val uploadService: UploadService = UploadService(uploadDatabase, s3, config, urls)
  val reportService: ReportService = new ReportService(uploadService, mapboxService, urls)
  val signupCodeService: SignupCodeService = new SignupCodeService(signupCodeCache, timeouts)
  val surveyService: SurveyService = new SurveyService(surveyDatabase, mapService, notificationService, unsavedSurveyCache)
  val userService: UserService = new UserService(userDatabase, forgotCache, organizationService, notificationService, newsletterService, timeouts)
  val authService: AuthService = new AuthService(sessionCache, forgotCache, userService, signupCodeService, organizationService, notificationService, newsletterService, timeouts)

  // Controller modules:
  val uploadJson: UploadJson = new UploadJson(uploadService)
  val authController: AuthController = new AuthController(authService, timeouts, urls)
  val mapController: MapController = new MapController(mapService, authService, timeouts, urls)
  val newsletterController: NewsletterController = new NewsletterController(newsletterService, authService, timeouts, urls)
  val notificationController: NotificationController = new NotificationController(notificationService, authService, timeouts, urls)
  val organizationController: OrganizationController = new OrganizationController(organizationService, authService, timeouts, urls)
  val reportController: ReportController = new ReportController(surveyService, authService, reportService, timeouts, urls, wsClient)
  val signupCodeController: SignupCodeController = new SignupCodeController(signupCodeService, authService, timeouts, urls, actorSystem)
  val surveyController: SurveyController = new SurveyController(surveyService, authService, timeouts, urls)
  val userController: UserController = new UserController(userService, authService, timeouts, urls)
  val uploadController: UploadController = new UploadController(uploadService, surveyService, authService, timeouts, uploadJson, urls)
  val preFlightController: PreFlightController = new PreFlightController
  val versionController: VersionController = new VersionController

  // HTTP modules:
  val httpFilters: Seq[EssentialFilter] = new Filters(metrics).filters
  val httpErrorHandler: ErrorHandler = new ErrorHandler(environment, configuration, sourceMapper, Option(router), metrics)
  val httpRequestHandler: RequestHandler = new RequestHandler(router, httpErrorHandler, httpConfiguration, metrics, httpFilters : _*)

  // Router:
  val router: Router = new Routes(
    httpErrorHandler,
    versionController,
    surveyController,
    reportController,
    mapController,
    uploadController,
    userController,
    notificationController,
    authController,
    signupCodeController,
    organizationController,
    newsletterController,
    preFlightController,
    prefix = ""
  )
}
```

Google Guice

```
import com.google.inject._

@ImplementedBy(classOf[S3TemplateLoader])
trait TemplateLoader { ... }

@ImplementedBy(classOf[MustacheTemplateRenderer])
trait TemplateRenderer { ... }

class GreetingService @Inject() (
  loader: TemplateLoader,
  renderer: TemplateRenderer) { ... }

object App {
  val injector = Guice.createInjector()
  val greetings = injector
        .getInstance(classOf[GreetingService])
}
```

Gotcha

It does everything at runtime!

Macwire

```
object App {
  import com.softwaremill.macwire._

  val loader = wire[S3TemplateLoader]
  val renderer = wire[MustacheTemplateRenderer]
  val greetings = wire[GreetingService]
}
```

```scala
object App {
  import com.softwaremill.macwire._

  val loader = new S3TemplateLoader()
  val renderer = new MustacheTemplateRenderer()
  val greetings = new GreetingService(loader, renderer)
}
```

```
val config = Config.read(configuration) match {
  case Left(err)  => configReadError(err)
  case Right(cfg) => cfg
}

// Low-level config:
val urls: Urls = wire[Urls]
val timeouts: Timeouts = wire[Timeouts]
val metrics: Metrics = wire[Metrics]

// Email modules:
val emailer: Emailer = config.notification.email match {
  case config: TestEmailerConfig => wire[TestEmailer]
  case config: MailgunEmailerConfig => wire[MailgunEmailer]
  case config: SendGridEmailerConfig => wire[SendGridEmailer]
}

// Database modules:
val dbConfig = wire[PostgresDatabaseConfig]
val mapDatabase = wire[SlickMapDatabase]
val notificationDatabase = wire[SlickNotificationDatabase]
val surveyDatabase = wire[SlickSurveyDatabase]
val uploadDatabase = wire[SlickUploadDatabase]
val userDatabase = wire[SlickUserDatabase]

// Redis cache modules:
val redisClient = wire[RedisClient]
val forgotCache = wire[RedisForgotCache]
val sessionCache = wire[RedisSessionCache]
val signupCodeCache = wire[RedisSignupCodeCache]
val unsavedSurveyCache = wire[RedisUnsavedSurveyCache]

// S3 access:
val s3 = S3(config)

// Service modules:
val mapService = wire[MapService]
val mapboxService = wire[MapboxService]
val newsletterService = wire[NewsletterService]
val notificationService = wire[NotificationService]
val organizationService = wire[OrganizationService]
val uploadService = wire[UploadService]
val reportService = wire[ReportService]
val signupCodeService = wire[SignupCodeService]
val surveyService = wire[SurveyService]
val userService = wire[UserService]
val authService = wire[AuthService]

// Controller modules:
val uploadJson = wire[UploadJson]
val authController = wire[AuthController]
val mapController = wire[MapController]
val newsletterController = wire[NewsletterController]
val notificationController = wire[NotificationController]
val organizationController = wire[OrganizationController]
val reportController = wire[ReportController]
val signupCodeController = wire[SignupCodeController]
val surveyController = wire[SurveyController]
val userController = wire[UserController]
val uploadController = wire[UploadController]
val preFlightController = PreFlightController
```

```
// Email modules:
val emailer: Emailer = config.notification.email match {
  case config: TestEmailerConfig => wire[TestEmailer]
  case config: MailgunEmailerConfig => wire[MailgunEmailer]
  case config: SendGridEmailerConfig => wire[SendGridEmailer]
}

// Database modules:
val dbConfig = wire[PostgresDatabaseConfig]
val mapDatabase = wire[SlickMapDatabase]
val notificationDatabase = wire[SlickNotificationDatabase]
val surveyDatabase = wire[SlickSurveyDatabase]
val uploadDatabase = wire[SlickUploadDatabase]
val userDatabase = wire[SlickUserDatabase]

// Redis cache modules:
val redisClient = wire[RedisClient]
val forgotCache = wire[RedisForgotCache]
val sessionCache = wire[RedisSessionCache]
val signupCodeCache = wire[RedisSignupCodeCache]
val unsavedSurveyCache = wire[RedisUnsavedSurveyCache]

// S3 access:
val s3 = S3(config)

// Service modules:
val mapService = wire[MapService]
val mapboxService = wire[MapboxService]
val newsletterService = wire[NewsletterService]
val notificationService = wire[NotificationService]
val organizationService = wire[OrganizationService]
val uploadService = wire[UploadService]
val reportService = wire[ReportService]
val signupCodeService = wire[SignupCodeService]
val surveyService = wire[SurveyService]
val userService = wire[UserService]
val authService = wire[AuthService]

// Controller modules:
val uploadJson = wire[UploadJson]
val authController = wire[AuthController]
val mapController = wire[MapController]
val newsletterController = wire[NewsletterController]
val notificationController = wire[NotificationController]
val organizationController = wire[OrganizationController]
val reportController = wire[ReportController]
val signupCodeController = wire[SignupCodeController]
val surveyController = wire[SurveyController]
val userController = wire[UserController]
val uploadController = wire[UploadController]
val preFlightController = PreFlightController
val versionController = VersionController

// HTTP modules:
val httpFilters = wire[Filters].filters
val httpErrorHandler = wire[ErrorHandler]
val httpRequestHandler = wire[RequestHandler]

// Router:
val router = wire[Routes]
```

Gotchas

Gotcha #1

```
object App {
  val foo = new Foo(bar, baz)
  val bar = new Bar()
  val baz = new Baz()
}

App.foo
```

```
object App {
  val foo = new Foo(bar, baz)
  val bar = new Bar()
  val baz = new Baz()
}

App.foo
// OK!
```

```
object App {
  val foo = new Foo(bar, baz)
  val bar = new Bar()
  val baz = new Baz()
}
```

```
App.foo.methodThatUsesBarOrBaz
// NullPointerException
//    at ...
//    at ...
```

```
object App {
  lazy val foo = new Foo(bar, baz)
  lazy val bar = new Bar()
  lazy val baz = new Baz()
}

App.foo.methodThatUsesBarOrBaz
// OK!
```

Gotcha #2

```scala
class Foo(b: Bar)
class Bar(f: Foo)

object App {
  lazy val foo: Foo = new Foo(bar)
  lazy val bar: Bar = new Bar(foo)
}

App.foo
```

```
class Foo(b: Bar)
class Bar(f: Foo)

object App {
  lazy val foo: Foo = new Foo(bar)
  lazy val bar: Bar = new Bar(foo)
}

App.foo
// StackOverflowException
//    at ...
//    at ...
```

```scala
class Foo(b: => Bar)
class Bar(f: => Foo)

object App {
  lazy val foo: Foo = new Foo(bar)
  lazy val bar: Bar = new Bar(foo)
}

App.foo
// OK!
```

Gotcha #3

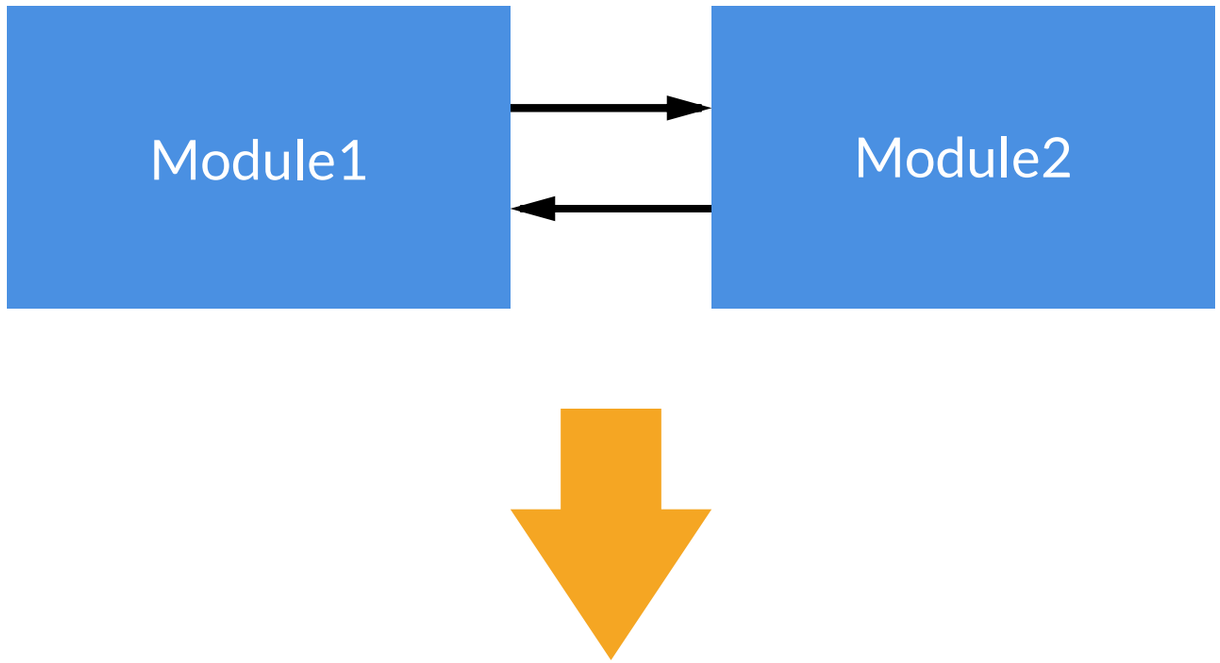
```
class Foo(b: => Bar) { val x = b }
class Bar(f: => Foo) { val y = f }

object App {
  lazy val foo: Foo = new Foo(bar)
  lazy val bar: Bar = new Bar(foo)
}

App.foo
```

```scala
class Foo(b: => Bar) { val x = b }
class Bar(f: => Foo) { val y = f }

object App {
  lazy val foo: Foo = new Foo(bar)
  lazy val bar: Bar = new Bar(foo)
}

App.foo
// StackOverflowException
//    at ...
//    at ...
```

```
class Foo(b: => Bar) { def x = b }
class Bar(f: => Foo) { def y = f }

object App {
  lazy val foo: Foo = new Foo(bar)
  lazy val bar: Bar = new Bar(foo)
}

App.foo
```

```
class Foo(b: => Bar) { lazy val x = b }
class Bar(f: => Foo) { lazy val y = f }

object App {
  lazy val foo: Foo = new Foo(bar)
  lazy val bar: Bar = new Bar(foo)
}

App.foo
```

ERMAGERD!!!!!


Module1
Module2
Module1
Module2
Module3

Trait-based DI

```scala
trait TemplateLoader {
  def load(id: TemplateId): Template
}

trait TemplateRenderer {
  def render(template: Template, params: Params): String
}
```

```scala
trait GreetingService {
  self: TemplateLoader with TemplateRenderer =>

  def greet(id: TemplateId, params: Params): String =
    render(load(id), params)
}
```

```
object App extends GreetingService
  with S3TemplateLoader
  with MustacheTemplateRenderer

App.greet(...)
```

Naming things!

```scala
trait TemplateLoader {
  def load(id: TemplateId): Template
}

trait TemplateRenderer {
  def render(template: Template, params: Params): String
}
```

```scala
trait GreetingService {
  self: TemplateLoader with TemplateRenderer =>

  def greet(id: TemplateId, params: Params): String =
    render(load(id), params)
}
```

```scala
trait TemplateLoader {
  def apply(id: TemplateId): Template
}

trait TemplateRenderer {
  def apply(template: Template, params: Params): String
}
```

```scala
trait GreetingService {
  self: TemplateLoader with TemplateRenderer =>

  def greet(id: TemplateId, params: Params): String =
    apply(apply(id), params)
}
```

“Thin” Cake Pattern

```
trait TemplateLoaderModule {
  def loader: TemplateLoader
}
```

```
trait S3TemplateLoaderModule
    extends TemplateLoaderModule {
  lazy val loader: TemplateLoader =
    new S3TemplateLoader()
}
```

```
trait GreetingServiceModule {
  self: TemplateLoaderModule
      with TemplateRendererModule =>

  lazy val greeter: GreetingService =
    new GreetingService(loader, renderer)
}
```

```
object App extends GreetingServiceModule
  with S3TemplateLoaderModule
  with MustacheTemplateRendererModule
```

Hang on a minute…

```
object App extends GreetingServiceModule
  with S3TemplateLoaderModule
  with MustacheTemplateRendererModule
```

```
object App {
  val loader = new S3TemplateLoader()
  val renderer = new MustacheTemplateRenderer()
  val greetings = new GreetingService(loader, renderer)
}
```

“Full” Cake Pattern

```
trait TemplateLoaderModule {
  trait TemplateLoader {
    def load(id: TemplateId): Template
  }

  def loader: TemplateLoader
}
```

```
trait S3TemplateLoaderModule extends TemplateLoaderModule {
  class S3TemplateLoader extends TemplateLoader {
    ...
  }

  lazy val loader = new S3TemplateLoader()
}
```

```
trait GreetingServiceModule {
  self: TemplateLoaderModule
      with TemplateRendererModule =>

  trait GreetingService {
    def greet(id: TemplateId, params: Params): String =
      renderer(loader(id), params)
  }

  lazy val greeter: GreetingService =
    new GreetingService(loader, renderer)
}
```

```
object App extends GreetingServiceModule
  with S3TemplateLoaderModule
  with MustacheTemplateRendererModule
```

Gotcha

```
object App extends GreetingServiceModule
  with S3TemplateLoaderModule
  with MustacheTemplateRendererModule

val loader = App.loader
```

```
object App extends GreetingServiceModule
  with S3TemplateLoaderModule
  with MustacheTemplateRendererModule

val loader = App.loader
// loader: App.TemplateLoader = ...
```

Bakery of DOOM

In Summary...

Cake

Guice

Constructors

Macwire

Cake

Thin Cake

Reader/ReaderT

Free

Functions

XML (LOL)

Traits

Constructors

Guice

Macwire

Cake

Thin Cake

Reader/ReaderT

Free

Functions

Constructors

Guice

Macwire

Thin Cake

Constructors

Guice

Macwire

Thin Cake

We didn’t cover…

Functions

Reader/ReaderT

Free

References

DI approaches we covered here…

Krzysztof Pado - Dependency injection in Play Framework using Scala
http://www.schibsted.pl/blog/dependency-injection-play-framework-scala

Macwire User Guide (project README)
https://github.com/adamw/macwire

Google Guice User Guide
https://github.com/google/guice/wiki/Motivation

Adam Warski - Using Scala traits as modules, or the "Thin Cake" Pattern
http://www.warski.org/blog/2014/02/using-scala-traits-as-modules-or-the-thin-cake-pattern

Mark Harrison - Cake Pattern in Depth
http://www.cakesolutions.net/teamblogs/2011/12/19/cake-pattern-in-depth

Andrew Rollins - The Cake Pattern in Scala - Self Type Annotations vs. Inheritance
http://www.andrewrollins.com/2014/08/07/scala-cake-pattern-self-type-annotations-vs-inheritance

Functional approaches…

Mark Seemann - Dependency injection is passing an argument
http://blog.ploeh.dk/2017/01/27/dependency-injection-is-passing-an-argument

Mark Seemann - Partial application is dependency injection
http://blog.ploeh.dk/2017/01/30/partial-application-is-dependency-injection

Mark Seemann - Dependency rejection
http://blog.ploeh.dk/2017/02/02/dependency-rejection

The Reader monad…

Jason Arhart - Scrap Your Cake Pattern Boilerplate: DI Using the Reader Monad
http://blog.originate.com/blog/2013/10/21/reader-monad-for-dependency-injection/

The Free monad…

Pere Villega - On Free Monads
http://perevillega.com/understanding-free-monads

Pere Villega - Free Monads using FreeK
http://perevillega.com/freek-and-free-monads