Google Guice

základní seznámení s frameworkem

Lukáš Kolísko
lkolisko@gmail.com

26.11.2007

Cíle prezentace

Cílem prezentace je ukázat, že...

- v jednoduchosti je síla
- Java aplikace může fungovat bez XML
- dependency injection není "žádná magie"

Cíle prezentace

Cílem prezentace je ukázat, že...

- v jednoduchosti je síla
- Java aplikace může fungovat bez XML
- dependency injection není "žádná magie"

...mezi tím Vás mimochodem seznámit s **Google Guice**

Cíle prezentace

Cílem prezentace je ukázat, že...

- v jednoduchosti je síla
- Java aplikace může fungovat bez XML
- dependency injection není "žádná magie"

...mezi tím Vás mimochodem
seznámit s **Google Guice**
a pokuď možno neuspat...

To DI, or not to DI...

Hollywoodský princip

"Don't call us, We'll call you"

Základní princip

Objekty definují svoje závislosti pouze na základě argumentů konstruktorů, statických factory metod nebo setterů, které jsou volány na nově vytvořené instanci.

Hollywoodský princip

"Don't call us, We'll call you"

Základní princip

Objekty definují svoje závislosti pouze na základě argumentů konstruktorů, statických factory metod nebo setterů, které jsou volány na nově vytvořené instanci.

DI kontejner

DI kontejner vytváří instance objektů a zajišťuje injekci deklarovaných závislostí.

Varianty řešení DI

Varianty řešení DI

Field injection

Závislosti jsou kontejnerem injektovány přímo do **instančních nebo statických proměnných** třídy.

Varianty řešení DI

Field injection

Závislosti jsou kontejnerem injektovány přímo do **instančních nebo statických proměnných** třídy.

Setter injection

DI kontejner vytvoří novou instanci pomocí **bezparametrického konstruktoru** a závislosti jsou nastaveny pomocí **setterů**.

Varianty řešení DI

Field injection

Závislosti jsou kontejnerem injektovány přímo do **instančních nebo statických proměnných** třídy.

Setter injection

DI kontejner vytvoří novou instanci pomocí **bezparametrického konstruktoru** a závislosti jsou nastaveny pomocí **setterů**.

Constructor injection

DI kontejner vloží závislosti jako argumenty **konstruktoru** nebo parametry **statické factory metody**.

Výhody a nevýhody

- **Field injection**
  - ⊕ kompaktní syntaxe (výhoda pro slajdy)
  - ⊖ nelze navázat další akci
  - ⊖ testovatelnost
- **Setter injection**
  - ⊕ není field injection
  - ⊕ testovatelnost
- **Constructor injection**
  - ⊕ možnost použití final
  - ⊕ jistota "plné" inicializace
  - ⊕ přehlednost
  - ⊖ volitelná injekce
  - ⊖ nelze použít v případě instancí vznikajících mimo DI kontejner
  - ⊖ nadtřída musí znát všechny závislosti podtřídy
  - ⊖ méně pohodlné testování

Google Guice [juice]

"Lightweight" dependency injection framework pro Javu 5+

- **Kdo ?**
  - Bob Lee [crazyboblee] - software enginner, Google, Inc.
  - Kevin Bourrillion [kevinb9n] - senior sofware engineer, Google, Inc.
- **Proč ?**
  Projekt vznikl v rámci vývoje systému AdWords ve společnosti Google. Cílem bylo dosažení vyšší modularity kódu pro zjednodušení testovatelnosti celého systému.
- **Kde ?**
  http://code.google.com/p/google-guice
- **Licence ?**
  Apache License, Version 2.0

Džusík s Vodkou...

- **Služba**

```
public interface LahevVodky {
  public void otevri();
  public Vodka nacti() throws Exception;
  public void zavri();
}
```

- **Klient**

```
public class Programtor {
  public void dejSi(){...}
  public boolean vesely(){...}
}
```

Džusík s Vodkou...

- **Služba**

```
public interface LahevVodky {
  public void otevri();
  public Vodka nacti() throws Exception;
  public void zavri();
}
```

- **Klient**

```
public class Programtor {
  public void dejSi(){...}
  public boolean vesely(){...}
}
```

Factory pro LahevVodky

```
public class Lihovar {
  private static LahevVodky instance;
  private Lihovar(){}
  public static LahevVodky getInstance(){
    return instace;
  }
  public static void setInstance(LahevVodky other){
    instance=other;
  }
}
```

Klient pro factory

```
public class Programator {
 public void dejSi(){
   Vodka vodka=Lihovar.getInstance();
   vodka.otevri();
   ...
 }
}
```

Vložení závislosti **@Inject** - Constructor based

```
public class Programator {
  private final LahevVodky lahev;

  @Inject
  public Programator(LahevVodky lahev){
    this.lahev=lahev;
  }

  public void dejSi(){
    vodka.otevri();
    vnitrniStav=lahev.nacti();
    ...
  }}
```

Vložení závislosti **@Inject** - Setter based

```
public class Programator {
  private LahevVodky lahev;
  public Programator(){}

  @Inject
  public setLahev(LahevVodky lahev){
    this.lahev=lahev;
  }

  public void dejSi(){
    vodka.otevri();
    ...
  }}
```

Definice modulu

- **Modul**
  - programovým způsobem popisuje vazby mezi jednotlivými objekty a vlastnosti těchto vazeb
  - modul:aplikace je ve vztahu M:N
  - moduly mohou vytvářet strom

```
public class AppModule implements Module {
  public void configure(Binder binder){
 binder.bind(LahevVodky.class)
 .to(Finlandia.class)
  }
}
```

Definice modulu

- **Modul**
  - programovým způsobem popisuje vazby mezi jednotlivými objekty a vlastnosti těchto vazeb
  - modul:aplikace je ve vztahu M:N
  - moduly mohou vytvářet strom

```
public class AppModule implements Module {
  public void configure(Binder binder){
 binder.bind(LahevVodky.class)
 .to(Finlandia.class)
  }
}
```

Definice modulu

- **Modul**
  - programovým způsobem popisuje vazby mezi jednotlivými objekty a vlastnosti těchto vazeb
  - modul:aplikace je ve vztahu M:N
  - moduly mohou vytvářet strom

```
public class AppModule implements Module {
  public void configure(Binder binder){
 binder.bind(LahevVodky.class)
 .to(Finlandia.class)
  }
}
```

Definice modulu

- **Modul**
  - programovým způsobem popisuje vazby mezi jednotlivými objekty a vlastnosti těchto vazeb
  - modul:aplikace je ve vztahu M:N
  - moduly mohou vytvářet strom

```
public class AppModule implements Module {
  public void configure(Binder binder){
 binder.bind(LahevVodky.class)
 .to(Finlandia.class)
  }
}
```

Získání instance z kontejneru

```
public class Aplikace {
  public static void main(String[] args){
    Injector injector=
    Guice.createInjector(AppModule.class);
    Programator programator=
    injector.getInstance(Programator.class);
    ...
  }
}
```

Získání instance z kontejneru

```
public class Aplikace {
  public static void main(String[] args){
    Injector injector=
    Guice.createInjector(AppModule.class);
    Programator programator=
    injector.getInstance(Programator.class);
    ...
  }
}
```

Získání instance z kontejneru

```
public class Aplikace {
  public static void main(String[] args){
    Injector injector=
    Guice.createInjector(AppModule.class);
    Programator programator=
    injector.getInstance(Programator.class);
    ...
  }
}
```

Explicitní definice závislosti **@ImplementedBy**

Anotace **@ImplementedBy** explicitně definuje, který typ bude použit jako implementace daného rozhraní v případě injekce závislosti na místo interface.

```
@ImplementedBy(Finlandia.class)
public interface LahevVodky {
   ...
}
```

Implicitní binding

Pokud není specifikována explicitní vazba, pak Guice hledá implementaci vazby za běhu (JIT binding).

- provider
- převedená konstanta
- @ImplementedBy
- @ProvidedBy
- třída

```
class RuskyProgramator{
  @Inject
  public RuskyProgramator(Putinka lahev){
    ...
  }
}
```

Implicitní binding

Pokud není specifikována explicitní vazba, pak Guice hledá implementaci vazby za běhu (JIT binding).

- provider
- převedená konstanta
- @ImplementedBy
- @ProvidedBy
- třída

```
class RuskyProgramator{
  @Inject
  public RuskyProgramator(Putinka lahev){
    ...
  }
}
```

Rozhraní služby má více různých implementací. Různí klienti potřebují specifické implementace, ale nechceme zavést přímou vazbu na danou implementaci.

Anotace vazeb - konfigurace

Definice anotace vazby:

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.FIELD, ElementType.PARAMETER})
@BindingAnnotation
public @interface Ruska {}
```

Konfigurace s použitím anotace:

```
...
bind(LahevVodky.class).annotatedWith(Ruska.class)
  .to(Putinka.class)
bind(LahevVodky.class).annotatedWith(Finska.class)
  .to(Finlandia.class)
...
```

Použití anotace vazby

```
@Inject @Ruska LahevVodky lahev;
```

nebo

```
@Inject
void setLahev(@Finska LahevVodky lahev){
   ...
}
```

Parametrizovatelné pojmenované vazby - **@Named**

Pokud nechceme z nějakého důvodu definovat novou anotaci, pak lze použít parametrizovatelnou annotaci **@Named**.

```
import static com.google.inject.name.Names.*;

bind(LahevVodky.class)
  .annotatedWith(named("Ruska"))
  .to(Putinka.class)

@Inject @Named("Ruska") LahevVodky lahev;
```

V Guice lze definovat vlastní parametrizovatelné vazby.

- Potřebujeme mít možnost získat libovolný počet instancí daného typu.
- Potřebujeme vytvářet objekty, které mají závislosti, ale nelze je anotovat.
- Potřebujeme lazy inicializaci instancí.
- Typ injektované instance závisí na nějaké podmínce známé až během výpočtu.

```
lahev=(víkend)?new Putinka():new Finlandia();
```

- Potřebujeme mít možnost získat libovolný počet instancí daného typu.
- Potřebujeme vytvářet objekty, které mají závislosti, ale nelze je anotovat.
- Potřebujeme lazy inicializaci instancí.
- Typ injektované instance závisí na nějaké podmínce známé až během výpočtu.

```
lahev=(víkend)?new Putinka():new Finlandia();
```

- Potřebujeme mít možnost získat libovolný počet instancí daného typu.
- Potřebujeme vytvářet objekty, které mají závislosti, ale nelze je anotovat.
- Potřebujeme lazy inicializaci instancí.
- Typ injektované instance závisí na nějaké podmínce známé až během výpočtu.

```
lahev=(víkend)?new Putinka():new Finlandia();
```

- Potřebujeme mít možnost získat libovolný počet instancí daného typu.
- Potřebujeme vytvářet objekty, které mají závislosti, ale nelze je anotovat.
- Potřebujeme lazy inicializaci instancí.
- Typ injektované instance závisí na nějaké podmínce známé až během výpočtu.

```
lahev=(víkend)?new Putinka():new Finlandia();
```

- Potřebujeme mít možnost získat libovolný počet instancí daného typu.
- Potřebujeme vytvářet objekty, které mají závislosti, ale nelze je anotovat.
- Potřebujeme lazy inicializaci instancí.
- Typ injektované instance závisí na nějaké podmínce známé až během výpočtu.

```
lahev=(víkend)?new Putinka():new Finlandia();
```

Providery - definice

- jeden z hlavních konceptů Guice
- slouží jako vnitřní factory na objekty
- závislosti provideru mohou být injektovány
- providery lze skládat (např. Scopes)

```
class Lihovar implements Provider<LahevVodky> {
  @Inject
  Lihovar(Cukrovar cukrovar){...}
  LahevVodky get(){...}
}
```

- provider metody - @Provides, ProviderMethods.from(Object providers)

Providery - definice

- jeden z hlavních konceptů Guice
- slouží jako vnitřní factory na objekty
- závislosti provideru mohou být injektovány
- providery lze skládat (např. Scopes)

```
class Lihovar implements Provider<LahevVodky> {
  @Inject
  Lihovar(Cukrovar cukrovar){...}
  LahevVodky get(){...}
}
```

- provider metody - @Provides, ProviderMethods.from(Object providers)

Providery - definice

- jeden z hlavních konceptů Guice
- slouží jako vnitřní factory na objekty
- závislosti provideru mohou být injektovány
- providery lze skládat (např. Scopes)

```
class Lihovar implements Provider<LahevVodky> {
  @Inject
  Lihovar(Cukrovar cukrovar){...}
  LahevVodky get(){...}
}
```

- provider metody - @Provides, ProviderMethods.from(Object providers)

Providery - definice

- jeden z hlavních konceptů Guice
- slouží jako vnitřní factory na objekty
- závislosti provideru mohou být injektovány
- providery lze skládat (např. Scopes)

```
class Lihovar implements Provider<LahevVodky> {
  @Inject
  Lihovar(Cukrovar cukrovar){...}
  LahevVodky get(){...}
}
```

- provider metody - @Provides, ProviderMethods.from(Object providers)

Providery - konfigurace

- **Binderem**

```
bind(LahevVodky.class)
  .toProvider(Lihovar.class);
```

- **Anotací @ProvidedBy**

```
@ProvidedBy(Lihovar.class)
interface LahevVodky {...}
```

- **Modulem**

```
Object obj=new ObjectSProviderMetodami();
binder.install(ProviderMethods.from(obj))
```

Providery - konfigurace

- **Binderem**

```
bind(LahevVodky.class)
  .toProvider(Lihovar.class);
```

- **Anotací @ProvidedBy**

```
@ProvidedBy(Lihovar.class)
interface LahevVodky {...}
```

- **Modulem**

```
Object obj=new ObjectSProviderMetodami();
binder.install(ProviderMethods.from(obj))
```

Providery - konfigurace

- **Binderem**

```
bind(LahevVodky.class)
  .toProvider(Lihovar.class);
```

- **Anotací @ProvidedBy**

```
@ProvidedBy(Lihovar.class)
interface LahevVodky {...}
```

- **Modulem**

```
Object obj=new ObjectSProviderMetodami();
binder.install(ProviderMethods.from(obj))
```

Providery - konfigurace

- **Binderem**

```
bind(LahevVodky.class)
  .toProvider(Lihovar.class);
```

- **Anotací @ProvidedBy**

```
@ProvidedBy(Lihovar.class)
interface LahevVodky {...}
```

- **Modulem**

```
Object obj=new ObjectSProviderMetodami();
binder.install(ProviderMethods.from(obj))
```

Providery - více instancí

```
@Inject Provider<LahevVodky> lihovar;
...
LahevVodky jedna=lihovar.get();
LahevVodky druha=lihovar.get();
...
```

- Pokud neposkytneme vlastní provider, Guice vytvoří provider automaticky.

Providery - více instancí

```
@Inject Provider<LahevVodky> lihovar;
...
LahevVodky jedna=lihovar.get();
LahevVodky druha=lihovar.get();
...
```

- Pokud neposkytneme vlastní provider, Guice vytvoří provider automaticky.

Providery - více instancí

```
@Inject Provider<LahevVodky> lihovar;
...
LahevVodky jedna=lihovar.get();
LahevVodky druha=lihovar.get();
...
```

- Pokud neposkytneme vlastní provider, Guice vytvoří provider automaticky.

Providery - různé instance

```
class Vecerka implements Provider<Vodka> {

  @Inject @Ruska Provider<LahevVodka> ruske;
  @Inject @Finska Provider<LahevVodka> finske;

  public LahevVodky get(){
    return (vánoce)?ruske.get():finske.get();
  }
}
```

Providery - různé instance

```
class Vecerka implements Provider<Vodka> {

  @Inject @Ruska Provider<LahevVodka> ruske;
  @Inject @Finska Provider<LahevVodka> finske;

  public LahevVodky get(){
    return (vánoce)?ruske.get():finske.get();
  }
}
```

Providery - různé instance

```
class Vecerka implements Provider<Vodka> {

  @Inject @Ruska Provider<LahevVodka> ruske;
  @Inject @Finska Provider<LahevVodka> finske;

  public LahevVodky get(){
    return (vánoce)?ruske.get():finske.get();
  }
}
```

Providery - různé instance

```
class Vecerka implements Provider<Vodka> {

  @Inject @Ruska Provider<LahevVodka> ruske;
  @Inject @Finska Provider<LahevVodka> finske;

  public LahevVodky get(){
    return (vánoce)?ruske.get():finske.get();
  }
}
```

Inicializace objektu při startu

V implicitním případě jsou objekty vytvářeny až v okamžiku, kdy jsou vyžadovány jiným objektem v rámci definované závislosti. Toto chování lze změnit následujícím způsobem:

```
bind(StartupTask.class).asEagerSingleton();
```

Konstanty

Guice umožňuje injektovat konstatní hodnoty pro následující typy:

- Primitivní typy
- Objektové wrappery pro primitivní typy
- Řetězce
- Výčty
- Třídy

Definice a injekce konstanty

- **Klíč konstanty**
  - anotace - např. @AlcoholVol
- **Konfigurace konstaty**
  - **Určení typu z argumentu vazby**

```
bindConstant()
  .annotatedWith(AlcoholVol.class).to(38);
```

  - **Explicitní určení typu konstanty**

```
bind(int.class)
  .annotatedWith(AlcoholVol.class).toInstance(38);
```

  - **Implicitní konverze podle typu v místě injekce**

```
bindContstant()
  .annotatedWith(AlcoholVol.class).to("38");
```

- **Použití konstanty**

```
@Inject @AlcoholVol int vol;
```

Definice a injekce konstanty

- **Klíč konstanty**
  - anotace - např. @AlcoholVol
- **Konfigurace konstaty**
  - **Určení typu z argumentu vazby**

```
bindConstant()
  .annotatedWith(AlcoholVol.class).to(38);
```

  - **Explicitní určení typu konstanty**

```
bind(int.class)
  .annotatedWith(AlcoholVol.class).toInstance(38);
```

  - **Implicitní konverze podle typu v místě injekce**

```
bindContstant()
  .annotatedWith(AlcoholVol.class).to("38");
```

- **Použití konstanty**

```
@Inject @AlcoholVol int vol;
```

Definice a injekce konstanty

- **Klíč konstanty**
  - anotace - např. @AlcoholVol
- **Konfigurace konstaty**
  - **Určení typu z argumentu vazby**

```
bindConstant()
  .annotatedWith(AlcoholVol.class).to(38);
```

  - **Explicitní určení typu konstanty**

```
bind(int.class)
  .annotatedWith(AlcoholVol.class).toInstance(38);
```

  - **Implicitní konverze podle typu v místě injekce**

```
bindContstant()
  .annotatedWith(AlcoholVol.class).to("38");
```

- **Použití konstanty**

```
@Inject @AlcoholVol int vol;
```

Definice a injekce konstanty

- **Klíč konstanty**
  - anotace - např. @AlcoholVol
- **Konfigurace konstaty**
  - **Určení typu z argumentu vazby**

```
bindConstant()
  .annotatedWith(AlcoholVol.class).to(38);
```

  - **Explicitní určení typu konstanty**

```
bind(int.class)
  .annotatedWith(AlcoholVol.class).toInstance(38);
```

  - **Implicitní konverze podle typu v místě injekce**

```
bindContstant()
  .annotatedWith(AlcoholVol.class).to("38");
```

- **Použití konstanty**

```
@Inject @AlcoholVol int vol;
```

Definice a injekce konstanty

- **Klíč konstanty**
  - anotace - např. @AlcoholVol
- **Konfigurace konstaty**
  - **Určení typu z argumentu vazby**

```
bindConstant()
  .annotatedWith(AlcoholVol.class).to(38);
```

  - **Explicitní určení typu konstanty**

```
bind(int.class)
  .annotatedWith(AlcoholVol.class).toInstance(38);
```

  - **Implicitní konverze podle typu v místě injekce**

```
bindContstant()
  .annotatedWith(AlcoholVol.class).to("38");
```

- **Použití konstanty**

```
@Inject @AlcoholVol int vol;
```

Scope v Guice

Scope

Scope specifikuje způsob práce s vytvořenými instancemi z hlediska jejich životního cyklu v Guice.

Implicitní scope

Implicitně nemá vazba definovaný žádný scope.Pro každou vazbu je vytvořena nová instance.

- vytvoř
- injektuj
- zapomeň

Podporované scopes

- **No scope** - nová instance pro každou injekci (prototype)
- **Singleton**
  - jediná instance
  - @Singleton, Scopes.SINGLETON
- **Request**
  - instance v attributes requestu
  - @RequestScoped, ServletScopes.REQUEST
- **Session**
  - instance v session
  - @SessionScoped, ServletScopes.SESSION

Request, Session scope - GuiceFilter, GuiceServletContextListener, InjectedHttpServlet

Podporované scopes

- **No scope** - nová instance pro každou injekci (prototype)
- **Singleton**
  - jediná instance
  - @Singleton, Scopes.SINGLETON
- **Request**
  - instance v attributes requestu
  - @RequestScoped, ServletScopes.REQUEST
- **Session**
  - instance v session
  - @SessionScoped, ServletScopes.SESSION

Request, Session scope - GuiceFilter, GuiceServletContextListener, InjectedHttpServlet

Podporované scopes

- **No scope** - nová instance pro každou injekci (prototype)
- **Singleton**
  - jediná instance
  - @Singleton, Scopes.SINGLETON
- **Request**
  - instance v attributes requestu
  - @RequestScoped, ServletScopes.REQUEST
- **Session**
  - instance v session
  - @SessionScoped, ServletScopes.SESSION

Request, Session scope - GuiceFilter, GuiceServletContextListener, InjectedHttpServlet

Podporované scopes

- **No scope** - nová instance pro každou injekci (prototype)
- **Singleton**
  - jediná instance
  - @Singleton, Scopes.SINGLETON
- **Request**
  - instance v attributes requestu
  - @RequestScoped, ServletScopes.REQUEST
- **Session**
  - instance v session
  - @SessionScoped, ServletScopes.SESSION

Request, Session scope - GuiceFilter, GuiceServletContextListener, InjectedHttpServlet

Podporované scopes

- **No scope** - nová instance pro každou injekci (prototype)
- **Singleton**
  - jediná instance
  - @Singleton, Scopes.SINGLETON
- **Request**
  - instance v attributes requestu
  - @RequestScoped, ServletScopes.REQUEST
- **Session**
  - instance v session
  - @SessionScoped, ServletScopes.SESSION

Request, Session scope - GuiceFilter, GuiceServletContextListener, InjectedHttpServlet

Konfigurace scope

- **Konfigurace pomocí modulu**

```
binder.bind(LahevVodky.class)
  .to(Finlandia.class)
  .in(Scopes.SINGLETON);
```

- **Konfigurace pomocí anotace**

```
@Singleton
public class Finlandia implements LahevVodky {
  public void otevri(){
    ...
  }
}
```

Uživatelsky definované scopes

- **Definice anotace scope**

```
@Retention(RUNTIME) @Target({TYPE}) @ScopeAnnotation
pulic @interface VodkaScoped {}
```

- **Implementace scope**

```
static final Scope VODKA_PARTY_SCOPE = new Scope {
  public <T> Provider<T> scope(Key<T> key,
    final Provider<T> creator){
    ...
  }
}
```

- **Konfigurace scope**

```
binder.bindScope(VodkaScoped.class,
    VODKA_PARTY_SCOPE);
```

Uživatelsky definované scopes

- **Definice anotace scope**

```
@Retention(RUNTIME) @Target({TYPE}) @ScopeAnnotation
pulic @interface VodkaScoped {}
```

- **Implementace scope**

```
static final Scope VODKA_PARTY_SCOPE = new Scope {
  public <T> Provider<T> scope(Key<T> key,
    final Provider<T> creator){
    ...
  }
}
```

- **Konfigurace scope**

```
binder.bindScope(VodkaScoped.class,
    VODKA_PARTY_SCOPE);
```

Uživatelsky definované scopes

- **Definice anotace scope**

```
@Retention(RUNTIME) @Target({TYPE}) @ScopeAnnotation
pulic @interface VodkaScoped {}
```

- **Implementace scope**

```
static final Scope VODKA_PARTY_SCOPE = new Scope {
  public <T> Provider<T> scope(Key<T> key,
    final Provider<T> creator){
    ...
  }
}
```

- **Konfigurace scope**

```
binder.bindScope(VodkaScoped.class,
    VODKA_PARTY_SCOPE);
```

Uživatelsky definované scopes

- **Definice anotace scope**

```
@Retention(RUNTIME) @Target({TYPE}) @ScopeAnnotation
pulic @interface VodkaScoped {}
```

- **Implementace scope**

```
static final Scope VODKA_PARTY_SCOPE = new Scope {
  public <T> Provider<T> scope(Key<T> key,
    final Provider<T> creator){
    ...
  }
}
```

- **Konfigurace scope**

```
binder.bindScope(VodkaScoped.class,
    VODKA_PARTY_SCOPE);
```

Development stages

Development stages

Umožňují měnit chování Guice na základě prostředí, ve kterém je aplikace spouštěna.

- **DEVELOPMENT** (implicitní)
  - Rychlý start aplikace + detailnějšími informace o vzniklých chybách
- **PRODUCTION**
  - zachycení maxima chyb konfigurace při startu aplikace.
- **TOOL**
  - Stage pro IDE
  - Vytvoření metadat bindingů, nevytváří se žádné instance

Development stages

Development stages

Umožňují měnit chování Guice na základě prostředí, ve kterém je aplikace spouštěna.

- **DEVELOPMENT** (implicitní)
  - Rychlý start aplikace + detailnějšími informace o vzniklých chybách
- **PRODUCTION**
  - zachycení maxima chyb konfigurace při startu aplikace.
- **TOOL**
  - Stage pro IDE
  - Vytvoření metadat bindingů, nevytváří se žádné instance

Development stages

Development stages

Umožňují měnit chování Guice na základě prostředí, ve kterém je aplikace spouštěna.

- **DEVELOPMENT** (implicitní)
  - Rychlý start aplikace + detailnějšími informace o vzniklých chybách
- **PRODUCTION**
  - zachycení maxima chyb konfigurace při startu aplikace.
- **TOOL**
  - Stage pro IDE
  - Vytvoření metadat bindingů, nevytváří se žádné instance

Development stages

Development stages

Umožňují měnit chování Guice na základě prostředí, ve kterém je aplikace spouštěna.

- **DEVELOPMENT** (implicitní)
  - Rychlý start aplikace + detailnějšími informace o vzniklých chybách
- **PRODUCTION**
  - zachycení maxima chyb konfigurace při startu aplikace.
- **TOOL**
  - Stage pro IDE
  - Vytvoření metadat bindingů, nevytváří se žádné instance

AOP interceptory

- Guice podporuje method interceptory používající AOP Alliance API
- method interceptory se registrují v modulech
- definice pointcuts implementacemi Matcher API (předpřipravené typ, annotace, AND, OR, NOT)
- přímo v jádře frameworku

```
binder.bindInterceptor(
  any(),
  annotatedWith(Transactional.class),
  new TransactionInterceptor()
);
```

Při replikaci je objekt serializován a přesunut mezi dvěma uzly. Jak zajistíme získání referencí na služby na cílovém uzlu ?

Statická injekce závislostí

- **Klient**

```
@SessionScoped
class Client {
  @Inject static Service service;
  ...
}
```

- **Konfigurace**

```
public void configure(Binder binder){
  ...
  binder.requestStaticInjection(Client.class);
  ...
}
```

  - Statickou injekci je třeba explicitně vyžádat

Integrace s jinými frameworky (1)

přímo v Guice

- **JNDI**
  - provider pro lookup v JNDI
- **Spring**
  - provider pro lookup Spring beans na základě jména
  - binding všech beans ze Spring factory jako @Named("...")
  - použití interceptorů ze Springu
- **JMX**
  - registrace všech beans v injectoru jako MBeans serveru
- **Struts2**
  - implementace object factory

Integrace s jinými frameworky (2)

3rd party moduly

- **JPA** (Hibernate)
  - Warp Persist, Dynamic Finders
    http://www.wideplay.com/guicewebextensions2
- **OSGi**
  - Peabery - patch Guice - podpora injekce OSGi služeb
    http://code.google.com/p/peaberry
- **DWR**
  - DWR-Guice integration
    http://dev.priorartisans.com/tim/dwr-guice

Shrnutí hlavních vlastností GUICE

- Využití možností Javy 5+ - anotace, generika
  - kompaktnější zápis metadat
  - statická kontrola
  - refaktoring
  - žádné XML
- Typová konverze pro konstanty
- AOP interceptory
  - AOP Alliance API
  - Matcher API
- Development stages
- Integrace s jinými frameworky

Použité zdroje informací

- **Google Guice** - domovská stránka projektu
  `http://code.google.com/p/google-guice`
- **Bob Lee and Kevin Bourrillion** - interview a prezentace Google Developer Day, JavaOne, Google Tech Talks, TheServerSide
- **The Spring Framework - Reference Documentation** http://static.springframework.org/spring/docs/2.5.x/reference

Zeptejte se . . .

Děkuji Vám za Vaši pozornost...

Lukáš Kolísko

lkolisko@gmail.com